# Machine learning to identify dynamic properties of railway track components

Kaewunruen, Sakdirat; Sresakoolchai, Jessada; Stittle, Henry

*DOI:*
10.1142/S0219455422501097



*Document Version*
Peer reviewed version

*Citation for published version (Harvard):*
Kaewunruen, S, Sresakoolchai, J & Stittle, H 2022, 'Machine learning to identify dynamic properties of railway track components', *International Journal of Structural Stability and Dynamics*, vol. 22, no. 11, 2250109. https://doi.org/10.1142/S0219455422501097

# Machine learning to identify dynamic properties of railway track components

Sakdirat Kaewunruen*, Jessada Sresakoolchai, and Henry Stittle

School of Engineering, University of Birmingham, Birmingham B15 2TT, UK.

Corresponding author e-mail: s.kaewunruen@bham.ac.uk

## Abstract

Investigating the condition of rail track components is important for track maintenance and developing a greater understanding of track design. Railway inspection can be destructive and non-destructive approaches. In the railway industry, the non-destructive approaches are preferred because they retain the track in operation thus significantly reducing the cost of fault testing. One of the non-destructive approaches is using machine learning which applied in this study. Field measurements and advanced analysis of results are used to extract track properties. This study creates, tunes and examines the validity of different machine learning techniques. The aim is to extract the dynamic track properties from the in-field measurements without needing the intermediary steps, saving both time and effort. Contributions of this study are demonstrating that machine learning techniques have the potential to save cost and time for railway inspection. Moreover, the accuracy is satisfied. The following models are produced: Linear Regression, K-Nearest Neighbours, Gradient Boosting and a Convolutional Neural Network. We observe the limitations of linear regression and tune the remainder, producing three models with low errors.

Keywords: Machine Learning, Dynamic Properties, Track Components, Non-Destructive Approach

**Declarations**

Funding: The project has received the financial support from the European Commission, Grant No 691135.

Conflicts of interest/Competing interests: The authors declare that they have no conflict of interest.

Availability of data and material: Not applicable

Code availability: The authors generate the code themselves.

## 1. Introduction

Everyday use of rail track combined with greater engine speeds, the need for heavier axle loads and more frequent transportation leads to the deterioration of many current track infrastructures. As such, efficient monitoring of track conditions is essential to reduce the cost of maintenance in addition to ensuring the highest levels of safety. Damaged tracks are known to cause train derailment [1], hence the need for easy yet precise in situ testing is imperative. There are two kinds of testing available; destructive and non-destructive. Non-destructive is favoured as this significantly reduces the cost of testing, particularly in the case when no fault is found. Currently, there are a number of these non-destructive techniques which include ultrasound [2], modal testing [3], eddy current inspection [4], Magnetic Particle Inspection [5] and many more. These methods all incur their respective costs, such as the equipment needed to perform an ultrasound or time spent to analyse results.

Modal testing [3] is the basis of our machine learning methods. Our models will utilise the same field measurements with the goal of predicting the results instead of calculating them

through experimental modal analysis, and finite element modelling. To improve the efficiency of rail track testing in terms of cost and time, we propose a method using machine learning algorithms which will once be created and optimised, only incur the costs of obtaining the frequency response function (FRF) from the section of rail to be inspected via an instrumented impact hammer. This gives the potential for a mobile phone application, which could for example receive the FRF and input to the model, incurring minimal costs. The basis of these models are from the following equation where $m_1$ and $m_2$ represent the mass of the rail and sleeper (kg), $k_1$ and $c_1$ represent stiffness (N/m) and damping (Ns/m) coefficients of the rail and $k_2$ and $c_2$ represent stiffness (N/m) and damping (Ns/m) coefficients of ballast supporting system.

$$H_{11}(f)=\frac{\sqrt{\left[k_1+k_2-4\pi^2 f^2 m_1\right]^2+\left[2\pi f\left(c_1+c_2\right)\right]^2}}{\left[\left(k_1-4m_1\pi^2 f^2\right)\left(k_2-4m_2\pi^2 f^2\right)-4\pi^2 f^2\left(k_1m_1+c_1c_2\right)\right]^2+4\pi^2 f^2\left[k_1c_2+k_2c_1-\left(m_1\left(c_1+c_2\right)+c_1m_2\right)4\pi^2 f^2\right]^2} \quad (1)$$

The FRF will plot frequency on the x-axis and the y axis:

$$H(f)=H_{11}(f)*f^2*0.02^{-1} \quad (2)$$

This equation is derived from the Fast Fourier Transform (FFT) method using a 2DOF model where $H_{11}$ represents FRF, f represents the frequency of samples, and H(f) represents the function of samples when processed using FFT. This may be used to estimate the FRF with a curve fitting error of just 3% [3]. As such it is reasonable to attempt the application of machine learning to extract the dynamic properties of the track from an FRF.

The aims of this study will therefore be to develop machine learning models to predict the dynamic track properties. Models will be tuned to ensure the accuracy. Different models will be compared to evaluate and find the best model.

## 2. Literature Review

Artificial intelligence is one of the most in-demand aspects of computer science, with applications widespread amongst all fields. Artificial intelligence has replaced many human jobs in a society already and has proved to be more effective and efficient in many cases [6]. It has surpassed humans in very detailed tasks for example Google's DeepMind has conquered many different games, including some which are not solvable such as poker [7]. Whilst machine learning, in particular, may not always be possible, it typically works very well for problems with well-defined inputs and outputs where large datasets are available to train our models on [8]; this fits well for our problem.

Machine learning has previously been used in track defect detection. However, this has been performed on visual inspection classification tasks with varied success. Wu, Yunpeng et.al. produced a groundbreaking UAV-Based visual inspection method which correctly classified surface defects with a 93.75%-94.26% accuracy [9]. Another application of machine learning is real-time visual detection of hexagonal bolts or lack of. This model used two neural networks and cross validates their outputs to avoid false positives. As such this model has an accuracy of 99.6% in the detection of visible bolts and 95% of missing bolts [10]. This model was able to apply visual inspection to a high precision however, the scope of their problem was limited to bolt detection only. Due to the nature of our proposed problem, we will aim to predict with a high degree of accuracy and to detect faults which visual inspection may not be able to see. The research gap is machine learning techniques that popularly used are visual inspection that has the

limitation of light and speed of data collection. In this study, we aim to use FRF to detect track properties which is more cost-efficient and faster.

The first machine learning technique to be explored is linear regression and whether this provides us with reliable predictions. If so then our problem will be solved and further algorithm creation would not be necessary. Linear regression however is one of the simplest techniques and as such comes with many limitations. One main limitation is the assumption of linearity between the dependent and independent variables [11]. It is fair to assume in our case this assumption would be too simple and therefore fail. Our data will not be linearly separable. We realise this model will most likely not be successful is important to start here. Creating this model will be low cost, it may take 10% of the time but provide up to 90% of the results [12]. The baseline model here will provide trivially attainable results quickly; this will provide a useful benchmark for more complex models to surpass.

The next algorithm we will consider is the k-nearest neighbours (KNN) algorithm. Unlike linear regression, KNN does not make the linearity assumption [13]. This supervised algorithm works based on the principle that similar data points will be close to each other in the output space [14]. In the context of our problem, this would mean that similar-looking FRFs would typically have similar dynamic properties. We could conjecture that the reverse case would also be true (similar dynamic properties would produce similar FRF) due to our equation (1). We may have some instances where the dynamic properties are vastly different; however, the FRFs may not be. As such we may suspect the KNN algorithm is a suitable step up from linear regression.

Our KNN model therefore should produce some good results but it also has limitations for certain data such as data with low variation. The next algorithm to be explored is gradient boosting. The aim of gradient boosting is to make an ensemble model of weak learning methods,

where each method focuses on examples which the previous weak learners incurred high errors in predicting [15]. This model therefore can easily obtain very high accuracy in making predictions. We expect this model to perform better and improve on the KNN in multi-output regression, in fact, it is possible gradient boosting will be the best performing model. With supervised learning on a tabular-dataset gradient boosting will often outperform even deep learning models [16].

The final model is a convolutional neural network (CNN), this is a deep learning approach which will hopefully be a contender for the best model. Typically a CNN is used for complex inputs, for example, speech or image recognition, usually to great effect, even outperforming humans at these tasks [17]. Specifically, due to the nature of our data, we will use a 1D CNN, as the input data here has only one dimension. We wish to use a CNN due to the ability of the convolutional layer to perform feature extraction. Data can be input from FRFs in the range of 0-1000 Hz and typically these graphs will have two distinct local maxima, see Figure 1. These features are perfect candidates to be highlighted by CNN.

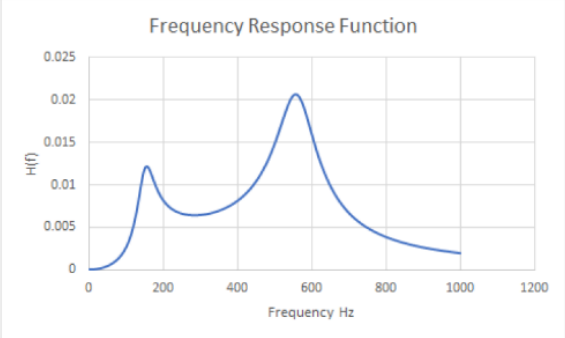


Figure 1. Example Frequency Response Function

CNN is used for complex inputs, for example, speech or image recognition, usually to great effect, even outperforming humans at these tasks [17]. Specifically, due to the nature of our data, we will use a 1D CNN, as the input data here has only one dimension. We wish to use a CNN

due to the ability of the convolutional layer to perform feature extraction. Data can be input from FRFs in the range of 0-1000 Hz and typically these graphs will have two distinct local maxima, see Figure 1. These features are perfect candidates to be highlighted by CNN.

As we obtain these graphs from equation (1), the nature of these peaks and troughs, or lack of in some cases, is due to the dynamic properties of the track so using a CNN may produce satisfying results in predicting these.

Once these models are created we will need to evaluate them. If our models were non-stochastic this would be a simple process. We could obtain the error or accuracy of the tests and compare. However, with many machine learning algorithms, we find the training of them is stochastic, meaning the results are not always consistent due to randomness in the models. For a neural network, this would be the starting weight of the network [18]. We will need to produce these results multiple times if we wish to compare different models. These models may take a long time to run however we require an acceptable sample size of results. We choose to do 50 calibrations for our tests [19].

Once this data is obtained for each model, we will need to perform statistical hypothesis tests to compare our results. A typical way to compare the mean of two distributions would be a t-test [20]. A standard t-test comes with assumptions. It assumes that the probability distribution follows a normal distribution and the equality of the variance of the distributions [20]. Another consideration is whether to use a paired or unpaired t-test [21], we find our data is unpaired and as such must use an appropriate test. We will therefore use an unpaired, non-parametric t-test. Therefore a Wilcoxon-Mann-Whitney test is proposed. This suits our criteria and performs better for smaller datasets as is the situation here [22]. A final consideration to be taken into account is

how the tuning of the models affects our comparisons; this will be discussed further in their creation.

Machine learning can be performed in many languages such as Python, Java, R, JavaScript, or Scala [23]. We choose to conduct ours in Python, as it is considered the most ideal language for general machine learning, this is due to the large set of libraries that can be applied to conduct machine learning [24]. As well as the presence of these libraries their ease of use is unmatched in other languages [24]. We use R to perform statistical analysis. R has a plethora of statistical analysis techniques, easy-to-use methods and great customisation [25].

## 3. Data generation and processing

Dynamic track properties are a simple yet effective parameter for determining if the rail track is defected. The dynamic track properties can be simulated using the 2DOF system [26] which is a widely-used technique. This was confirmed by Kaewunruen and Remennikov [26] who found that FRF could be represented by a 2DOF system. They compared field trial data, finite element model, and 2DOF model using equation (1). They found that the 2DOF mass-spring model was good enough and practical to represent field testing data. We will define acceptable ranges for our $k_1$ as 850-1050 MN/m, $k_2$ as 140 to 270 MN/m, $c_1$ as 1 to 23 kNs/m and $c_2$ as 140 to 270 kNs/m as these are in the acceptable range of industry standards [3]. Extraction of these properties would therefore lead to an effective method for inspections of the current track quality. We must generate data within this range that accounts for working data and also data with variables that fall below. Another consideration is the track with a broken fastening system, this would give a $k_2$ and $c_2$ of zero.

To obtain these properties we use equation (2) to generate our own frequency response function graphs to be an input of models. This is achieved through the use of the random()

function within Python; we multiply this random value from zero inclusive to one exclusive by the range needed, then add the minimum value required. Using this technique on all our variables and incrementing f (frequency) from 0 to 1000 we can generate our own graph.

Inputting two-dimensional data into our models would add unnecessary complexity. Since our frequency range is between 0 to 1000 in all the data, it is possible to omit this information from the models. We accomplish this by inputting our H(f) values, from equation (2), in order. This way we can input two-dimensional data in a one-dimensional form. After these values, we can add on our important variables $k_1$, $c_1$, $k_2$ and $c_2$.

This means each data has 1000 features, this may be fine for some algorithms however, it may make others needlessly complex. As such we can also produce a simplified version of this data where H(f) is only stored for every 10th frequency. This is particularly important for our tuning process which can be very time-consuming. In the same vein, we will explore using an even larger amount of data once the models are tuned. Using the large dataset before tuning would be ideal although, it would add complexity to the problem; switching to the large dataset after tuning should still give a noticeable improvement to the already tuned results.

We will also add artificial noise to our data. In practice frequency response functions will have noise, so for the models to work in-field, it will be important to include. The addition of noise has been shown to help reduce overfitting [27]. It has even been shown in some cases that noise injection can reduce overfitting to a greater extent than early stopping [28].

Finally, we will use normalization techniques on our output variables. This can be important in preventing problems with your model specifically when the variables are magnitudes different [29]. In our data this is the case; we find our $k_1$ and $k_2$ values are rough of the magnitude 108 to

109 whereas $c_1$ and $c_2$ are around 105. We can use a min-max scaler on these to transform each output variable to be between 0 and 1.

## 4. Algorithm Creation

We have decided on our four models, linear regression, k-nearest neighbours, gradient boosting, and a convolutional neural network. To develop an understanding of how each of these could perform for our data we choose to create simplified models for each of these. We ignore many potential hyperparameters at first before deciding if we wish to explore and tune them.

### *4.1. Simplified models*

#### *4.1.1. Linear Regression*

We will be using the linear regression model from the scikit-learn Python library. Our linear regression model will use the default parameters given by scikit-learn. This model would need four output variables and although linear regression does not naturally support this it can be simply altered to create four different linear regression models, with one model for each output.

We split our data into 70% training data and reserve 30% for testing, running this model 50 times to obtain the results with the assignment of training and test data at random. This prevents us from basing our results on a rare split of data where our predictions would be better than average. This random split of test to training data is repeated across all our models. We find that after 50 trials the average mean squared error is 0.0943, with a standard deviation of 0.0046. These results are not terrible and as such the more complex models have the potential to be extremely accurate.

#### *4.1.2. K-Nearest Neighbours*

Next, we have the KNN model. For this, we choose only two of the parameters. For the number of k neighbours, the model uses to make predictions we choose three and the weights metric to be distance, instead of uniform. Uniform weights give each neighbour an equal weighting whereas distance increases influence for closer neighbours. After 50 trials the average mean squared error is 0.0142, with a standard deviation of 0.0008. This is a considerable improvement over the linear regression model.

### *4.1.3. Gradient Boosting*

With our gradient boosting model the default parameters given by scikit-learn were retained. As with linear regression, this algorithm was designed for one output, so we adapt it as a multi-output regressor, fitting one model per output. The default parameters here are a learning rate of 0.01, a subsampling of 1, 100 estimators and a max depth of 3. This produced an average mean squared error of 0.0142 and a standard deviation of 0.0007, from over 50 samples. To more decimal places, gradient boosting has a lower error than the KNN model.

We perform a Wilcoxon rank-sum test to compare the distributions of KNN and gradient boosting here. This test is unpaired, two-sided, and using a confidence level of 95%. Our null hypothesis is that the location shift in the distributions is equal to 0, with the alternative hypothesis being that is not equal to 0. Our p-value for this test comes out to be 0.743, meaning the null hypothesis is retained. That is to say, the difference of error in these models is not statistically significant and it would be unfair for us to tune one of these models without also tuning the other.

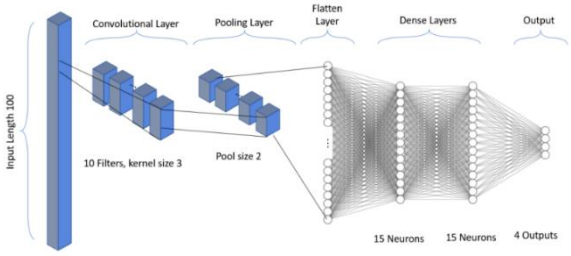


Figure 2. Simple Convolutional Neural Network Architecture

### *4.1.4. Convolutional Neural Network*

Finally, CNN is created, for this, we must choose filter size, pool size, number of hidden layers and neurons. We do not look into optimal values for these to be fair to the other models and choose some small values for these hyperparameters, which results in the architecture shown in Figure 2.

This architecture results in an average mean squared error of 0.0112, with a standard deviation of 0.0039. This is a lower mean squared error however, the variance is significant than our gradient boosting and the KNN models. Applying the Wilcoxon rank-sum test gives p-values of $3.242 \times 10^{-8}$ and $3.012 \times 10^{-8}$ when compared with KNN and gradient boosting respectively. With a confidence level of 95%, our p-value falls below 0.05, in both cases and we should reject the null hypothesis meaning there has been a statistically significant shift in the distributions.

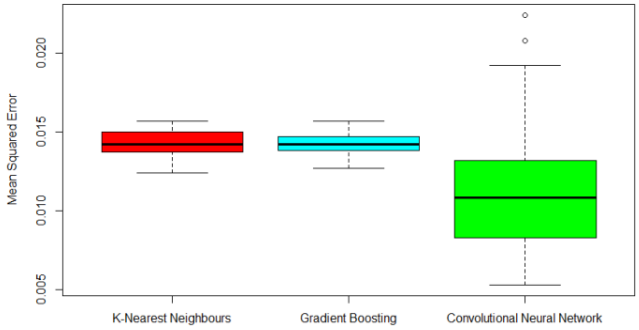

Figure 3. Simplified Models' Mean Squared Errors

#### *4.1.5. Summary*

In conclusion and as expected our simplified models demonstrated that the linear regression model performed much worse than the others. Our KNN and gradient boosting methods were very similar to the CNN performing better than the rest; however, the variance was high. From these results, we shall discard our linear regression model and use hyperparameter tuning on the remainders, with the goal of producing three highly accurate models for further comparisons.

### *4.2. Model Tuning*

A hyperparameter is a set parameter that we choose for our models, for example, the number of hidden layers in a neural network, or the learning rate of a model. We wish to find a tuple of hyperparameters which produce optimal accuracy for each model; this process is called hyperparameter tuning. Due to constraints on time and computing power, we do not conduct a fully exhaustive search on all hyperparameters. For each model, we choose certain important hyperparameters to optimize and reasonable ranges to test them across. We choose to conduct grid searches on our hyperparameters for KNN and gradient boosting. A grid search enumerates all combinations of the chosen subspace of hyperparameters [30], the downside of this method is that with numerous parameters tuning can be very time-consuming. Due to a large number of hyperparameters and long training time, for the CNN, we make use of a hyperband search. Hyperband is an optimised version of random search; it will randomly select models to tune for only a small amount of epochs, using the results it will select candidates to be trained for more epochs. This method can provide over an order of magnitude speedup over competitor algorithms.

### *4.2.1. K-Nearest Neighbours*

For the KNN model, we choose to look at four parameters in particular. Namely the leaf size, the number of neighbours used, the weights metric and the power parametric. We use the default algorithm parameter in scikit-learn which means the model will try to choose the best algorithm itself for selecting the neighbours. The number of neighbours we use in the algorithm is very important and can have a drastic impact on the overall error of the model [31]. The weights metric references either the weight of the nearest neighbours based on how far away they are from the target or the weighting of each of the nearest neighbours equally. Finally, our power metric which is responsible for how the distance to the neighbours is calculated. We choose either l1, the Manhattan distance or l2, the Euclidean distance. The Manhattan distance between two points a and b in n-dimensional space can be defined as:

$$d(a,b) = \sum_{i=1}^{n} (b_i - a_1) \tag{3}$$

The Euclidean distance between two points a and b in n-dimensional space can be defined as:

$$d(a,b) = \sqrt{(b_1 - a_1)^2 + (b_2 - a_2)^2 + \ldots + (b_n - a_n)^2} \tag{4}$$

The average mean squared error of this tuned model is calculated as 0.0142, with a standard deviation of 0.0007. This average mean squared error is the same as our simple model. We perform a Wilcoxon rank-sum test between the simple and tuned KNN models. This gives a p-value of 0.6884, meaning the location shift in the distributions was not statistically significant. Although our mean has not improved the variance of the model was, giving more consistent results.

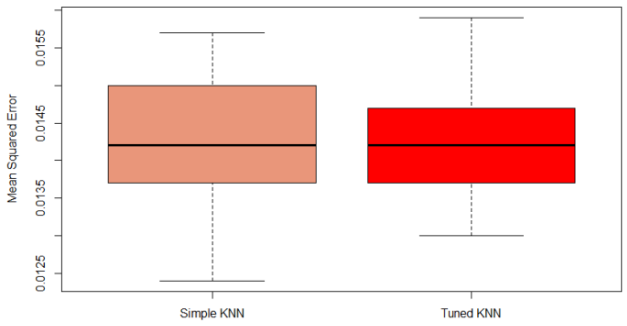


Figure 4. K-Nearest Neighbours Tuning Mean Squared Errors

### *4.2.2. Gradient Boosting*

For the gradient boosting model we tune four parameters; the number of estimators, the learning rate, the maximum depth, and the subsample size. The number of estimators determines the number of boosting stages performed. We choose to investigate 100 to 1000 estimators in steps of 100 in our hyperparameter space. With more time or computing power we could investigate numbers between this range or much larger numbers. For the learning rate, we choose either 0.01 or 0.001. Given enough data and trees, a lower learning rate would be better. However, since we are limited, this is an important feature to tune. The maximum depth can be altered to change the depth of the trees. A deeper tree can be useful for making accurate predictions, however, a high depth may lead to overfitting so this is an important balance to optimise. We select from a maximum depth of 1 to 4. Lastly, the subsampling is tuned; this determines the fraction of samples that are used to fit each weak learner. We choose from either 0.5, 0.75 or 1. Values less than one indicate an increase in the bias of each weak learner however, the overall variance of the model is reduced.

We find the optimal values from the hyperparameter space to be 1000 estimators, a maximum depth of 4, a learning rate of 0.01, and a subsampling of 0.5. Using these parameters we find a

tuned average mean squared error of 0.0093 with a standard deviation of 0.0005, over 50 fittings of the model. This is a clear improvement over the simplified model detailed in Figure 5.

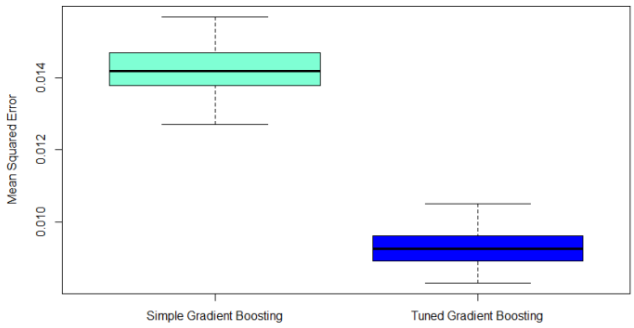


Figure 5. Gradient Boosting Tuning Mean Squared Errors

### *4.2.3. Convolutional Neural Network*

The CNN has a large number of hyperparameters that could be tuned so it will be required to limit these to a select few, especially as the CNN can take a considerable amount of time to train. Firstly we choose the learning rate of our Adam optimiser to be either 0.01 or 0.001. If too low our loss function will not improve fast enough, if too high the loss function will diverge and not settle at an optimum. Next, we look at the convolutional layer and determine the kernel size of our filters in addition to how many filters to use. We look at a range of 5 to 50 filters in steps of 5, also searching over the same range for our kernel size. Due to time constraints, we choose to examine only hidden layers that have the same number of neurons as each other. We search over 1 to 5 hidden layers of 10 to 50 neurons each in steps of 5.

We must also consider two other important hyperparameters however, these will not be adjusted in the training; The number of epochs we train the model on and batch size. Before tuning we will look at our simplified model in order to estimate when overfitting occurs, this will be different for every model, and as such a general rule is required. From Figure 6 we can see the overfitting of the model starts between the 200th and 250th epoch when the error of the training

data starts to decrease as the test error increases. The epochs we tune the models on therefore will be 250. The model uses a batch size of 32, this is a common batch size to use and although on the lower side, potentially hindering accuracy, it will drastically help the computational complexity of the tuning [32].

The tuned CNN deployed the following hyperparameters: a training rate of 0.01, 35 filters, a kernel size of 30, and 4 hidden layers with 50 neurons. This gives the architecture shown in Figure 7. This tuned CNN results in an average mean squared error of 0.0134 with a standard deviation of 0.0265 over 50 samples. The average mean squared error of the simple model is 0.0112, which is actually less than the tuned model. Figure 8 explains how this is possible; the tuned model had outliers. The tuned CNN had a maximum mean squared error of 0.0922, however, the median value was just 0.0046; less than half of the median of the simple model (0.0109). Figure 9 clearly shows that without the outliers, the tuned model performed much better. During training these outlier models stop improving almost instantly, Figure 7 shows this process. The non-outlier tuned CNNs follow a similar pattern to Figure 6. The cause of the outliers may well be a local minima reached in the mean squared error, this is a well-documented problem that can occur in neural networks [33]. In the next stage, we will experiment with large datasets, this extra data may reduce the presence or severity of outliers.

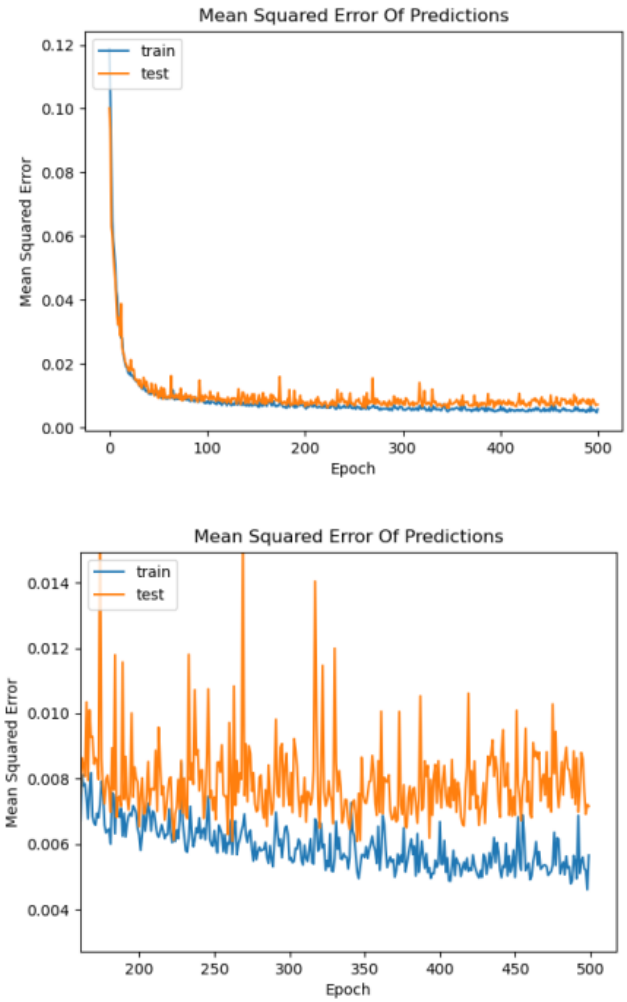


Figure 6. Fitting of Simplified CNN

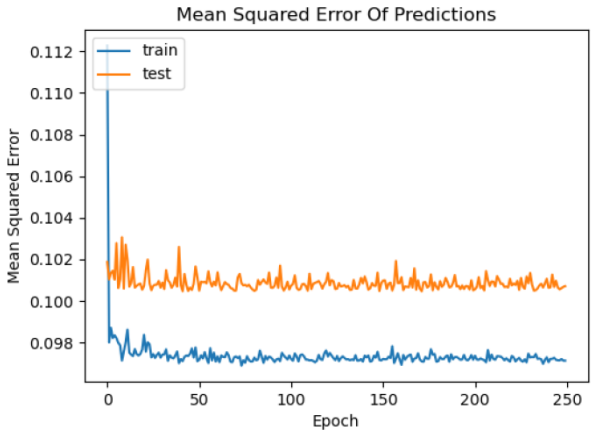


Figure 7. Fitting of an Outlier of the Tuned CNNs

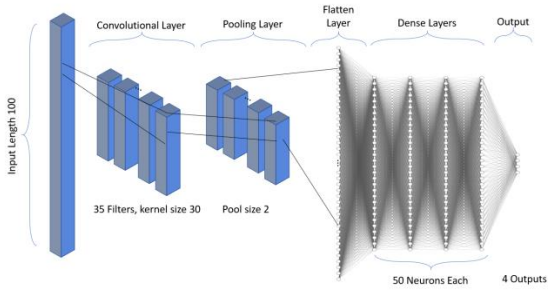


Figure 8. Tuned CNN Architecture

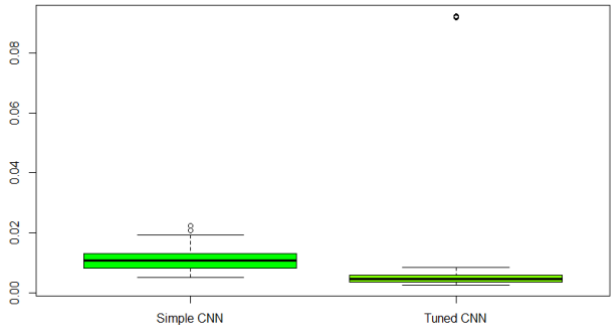


Figure 9. Convolutional Neural Network Tuning Mean Squared Errors

## 5. Results

When analysing the results, it important to maintain the unseen data as the validation data. This is because the model will show bias towards the training data and also to the test data we used in tuning the models. The parameters of models, for example, network weights, will be biased towards the training data and the hyperparameters may show bias towards the test data. In addition to this, with now tuned models, we may experiment with much larger datasets. Satisfying the need for new unseen data, we propose a new dataset. Previously the models were tuned with 2000 data; now we will generate a dataset with 9000 sets of data. All three of our models will be assumed to perform better with more data. In the case of gradient boosting and the CNN, the training time will be increased however, prediction time will stay the same. The

KNN model will take both longers to train and longer to make a prediction, this is a trade-off we must consider when adding more data.

**Table 1.** Average mean squared error of model using a small and large dataset

| | Small Dataset Average MSE | SD | Large Dataset Average MSE | SD |
|---|---|---|---|---|
| KNN | 0.1420 | 0.0007 | 0.0078 | 0.0003 |
| Gradient Boosting | 0.0093 | 0.0005 | 0.0077 | 0.0002 |
| CNN | 0.0134 | 0.0265 | 0.0037 | 0.0021 |

The large dataset models' mean squared errors come from 30 trains of the models, compared to the 50 of the smaller dataset, due to the greatly increased training time.

**Table 2.** Wilcoxon rank-sum Test between small and large dataset

| | Test Statistic w | P-Value | Location Shift Present |
|---|---|---|---|
| KNN | 1,500 | 8.78E-14 | Yes |
| Gradient Boosting | 1,500 | 8.69E-14 | Yes |
| CNN | 1,451 | 3.45E-12 | Yes |

Table 2 shows the results of the Wilcoxon rank-sum test between the small and large dataset versions of the tuned models. This is useful to have as these results will show whether the large dataset has a statistically significant impact on our models.

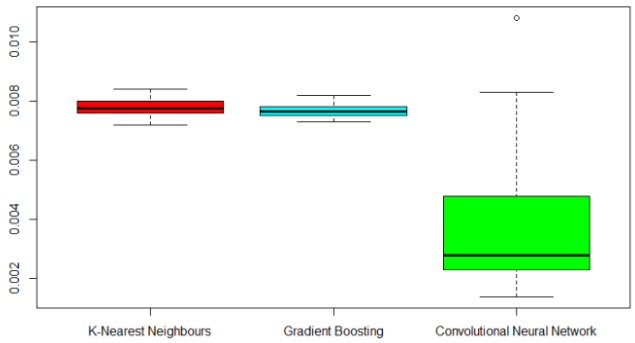


Figure 10. Large Dataset Models' Mean Squared Errors

**Table 3. Time taken to make a single prediction**

| | Prediction Time (s) |
|---|---|
| KNN | 0.0112 |
| Gradient Boosting | 0.0049 |
| CNN | 0.0156 |

Table 3 shows the average time taken to make a singular prediction of an unknown input, this is an average value from 30 trains of the model. This was all completed on one computer and results should therefore not be used as a rule for all systems.

## 6. Discussion

### *6.1. Explanation of Results*

The larger dataset leads to a significant improvement in results. The models showed a decrease in the average and standard deviation of the mean squared error. A Wilcoxon rank-sum test was performed on these distributions for the mean squared error of each model, comparing the training on the small and large datasets. In all cases, the test showed a statistically significant location shift for these distributions. As such, it is clear the larger dataset models are superior in terms of accuracy.

The larger dataset results show a clear best model, the CNN has less than half the average mean squared error of the other models when predicting the test data, at just 0.0037. The main downside of the model is the comparatively high value of the standard deviation. Considering this we calculate the maximum value of the mean squared error of the CNN from our tests is 0.0108, compared with the minimum values of KNN and gradient boosting of 0.0072 and 0.0073 respectively. The larger dataset CNN outliers' errors have seen a significant reduction in magnitude. Although there is some overlap in terms of error, the CNN is clearly a better performing model. A Wilcoxon rank-sum test between our KNN and gradient boosting models results in a p-value of 0.135. Statistically, we cannot say one is better than the other in terms of their respectively incurred mean squared errors.

Some outliers are present in the CNN results, this is something that should be considered carefully if this model is used in practice. The outliers, in this case, are not as drastic as with the smaller dataset however, they still have a large impact on the mean; the median mean squared error of the larger dataset CNN is 0.0028. If this model is applied in the future the particular train of the model should be picked cautiously, to avoid selecting an outlier model. A solution could be to experiment with larger batch sizes in the CNN model. This may reduce the chance of outliers appearing and increase the consistency of the model.

Another consideration was the time taken in generating a prediction. This was a concern, particularly for the KNN model due to the large number of computations required to find the nearest neighbours [34]. We find that although the KNN has the largest time taken to produce a prediction with the large dataset, this time is still minimal and acceptable. All of the models took only a fraction of a second to make predictions with the CNN taking the most time.

### *6.2. Achievements*

This study proposed machine learning techniques as a fast and inexpensive method to perform non-destructive track testing. As proof of concept simple models were generated encompassing different types of machine learning. The linear regression model was ruled out as a sufficient solution with a much higher error than the other models.

Subsequent hyperparameter tuning of the models resulted in varying levels of success. The KNN model improved invariance but did not show a significant change in error. The gradient boosting and CNN models did however show an improvement in their respective errors.

With the models now tuned it is feasible to use the larger dataset and retrain. This results in a significant improvement across our three tuned models. The CNN model has the least error. Unlike the KNN model, the memory usage and prediction time of the CNN model does not increase with the amount of data used in training.

Considering the other models, we propose the tuned CNN, using the large dataset to train on, as a solution to the original problem. This model is capable of identifying dynamic track properties with a very small error and without incurring the cost of expensive equipment or time spent analysing results. This has considerable potential for in-situ testing of track properties and defects, which can easily be equated to the condition of the track; a necessary task performed by members of the rail industry and researchers alike.

### *6.3. Limitations and Further Research*

The data used in the training and validation of the models were produced by the fast fourier transform equation, equation (1), and (2). This method's error is approximately 3%, compared to field data. The models' errors in prediction are therefore amplified. Although these errors are small it is something that could be refined. This would require the collection of large datasets from the field via an instrumented impact hammer. In addition, data with a large range of stiffness and damping would be needed to create a model capable of future predictions.

The models were not tuned on the larger datasets, indeed the tuning of the models took a significant amount of processor time with the smaller dataset. With much greater computing power, it is possible to tune the models with the larger dataset. This process might give even better hyperparameters leading to further reduced error. In the same vein, the hyperparameters could have been searched without steps within their ranges or over a larger range; in addition to the possibility of a search for some unused hyperparameters. A recommendation to reduce the time spent tuning on larger datasets would be to perform hyperparameter tuning using a different method; for example the parameter-setting-free harmony search algorithm [35] or a random search.

Machine learning concepts are not completely unheard of in rail track maintenance. However, many are classifying algorithms that are based on surface-level defects [36]. The proposed models can detect unseen faults through the identification of the dynamic track properties that these image-based classifiers may miss. One limitation is that some faults may cause issues for trains but have little impact on the specific properties of the track we are interested in; an example of this could be deformed track geometry [37].

An unexplored option that could produce improved results is an ensemble model. For our problem, a possible ensemble would be a set of individually trained regressors with predictions taken in combination. Ensemble models are often more accurate than any one component of the ensemble [38]. As we have produced three tuned models this study could lay a solid groundwork for the creation of such an ensemble model.

## 7. Conclusion

The main aim of this study is to produce accurate machine learning models for the predictions of the dynamic properties of track components from a frequency response function.

After ruling out linear regression, three algorithms were produced to solve this problem with varying levels of success. Various techniques were used to reduce the error in the models and the final models perform much better than our simplified versions. Whilst all algorithms have a reasonable level of error, as detailed in Section 5 the CNN was superior to the rest with an average mean squared error of just 0.0037. As such, the aims of this study outlined in Section 1 have been met and this study can be considered successful.

The CNN model has the highest potential utilization for in situ testing with a very low monetary cost and the ability to receive information about the track properties almost instantly. The model could now be adopted into a simple application that could be used on a phone or similar device capable of receiving vibrations from the track. The monetary cost of this would be minimal in comparison to many current techniques.

If the engineering feat of receiving the frequency response function on board a moving train is accomplished, there would be further potential applications of these models. Facilitating access to remote or hard-to-reach tracks also. This would open a path for track testing to be performed effortlessly and constantly across the globe in a way that would revolutionise track maintenance.

Moreover, the railway inspection can be done quickly as the speed of the operation with little cost. Therefore, the inspection can be done regularly to allow the sign of defect can be detected in advanced.

## Acknowledgment

The authors also wish to thank the European Commission for the financial sponsorship of the H2020-RISE Project no.691135 "RISEN: Rail Infrastructure Systems Engineering Network", which enables a global research network that addresses the grand challenge of railway infrastructure resilience and advanced sensing in extreme environments (www.risen2rail.eu) [39].

## References

1. Liu, X., M.R. Saat, and C.P. Barkan, *Analysis of causes of major train derailment and their effect on accident rates.* Transp. Res. Rec., 2012. **2289**(1): p. 154-163.
2. Vipparthy, S.T., *Inspection of Defects in Rails using Ultrasonic Probe.* 2013.
3. Kaewunruen, S. and A. Remennikov, *Integrated field measurements and track simulations for condition assessment of railway track.* 2005.
4. Song, Z., et al., *Detection of damage and crack in railhead by using eddy current testing.* J Electromagn Anal Appl., 2011. **2011**.
5. Lee, G.B., *Applications of Non-Destructive Testing in Rail Tracks*. 2015.
6. Khanam, S., S. Tanweer, and S. Khalid, *Artificial Intelligence Surpassing Human Intelligence: Factual or Hoax.* Comput J, 2020.
7. Ray, T. *Machine learning goes beyond theory to beat human poker champs*. 2019.
8. Brynjolfsson, E. and T. Mitchell, *What can machine learning do? Workforce implications.* Sci., 2017. **358**(6370): p. 1530-1534.

9. Wu, Y., et al., *A UAV-based visual inspection method for rail surface defects.* Appl. Sci., 2018. **8**(7): p. 1028.

10. Marino, F., et al., *A real-time visual inspection system for railway maintenance: automatic hexagonal-headed bolts detection.* IEEE T SYST MAN CY C., 2007. **37**(3): p. 418-428.

11. Kumar, N. *Advantages and Disadvantages of Linear Regression in Machine Learning*. 2019.

12. Ameisen, E., *Always start with a stupid model, no exceptions*. 2018.

13. Yang, J., Z. Sun, and Y. Chen, *Fault detection using the clustering-kNN rule for gas sensor arrays.* J. Sens., 2016. **16**(12): p. 2069.

14. Harrison, O., *Machine learning basics with the k-nearest neighbors algorithm.* Towards Data Science. September, 2018. **10**.

15. Valiant, L., *Probably Approximately Correct: NatureÕs Algorithms for Learning and Prospering in a Complex World*. 2013: Basic Books (AZ).

16. Shavitt, I. and E. Segal. *Regularization learning networks: deep learning for tabular datasets*. in *Advances in Neural Information Processing Systems*. 2018.

17. Xiong, W., et al. *The Microsoft 2017 conversational speech recognition system*. in *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. 2018. IEEE.

18. Hardesty, L. *Explained: Neural networks*. 2017; Available from: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414.

19. Arsenault, R., et al., *Comparison of stochastic optimization algorithms in hydrological model calibration.* J. Hydrol. Eng., 2014. **19**(7): p. 1374-1384.

20. Kim, T.K., *T test as a parametric statistic.* Korean J. Anesthesiol., 2015. **68**(6): p. 540.

21. Hill, S.A., *Chapter Statistics.* Foundations of Anesthesia: Basic Sciences for Clinical Practice, 2006: p. 207.

22. Fagerland, M.W., *t-tests, non-parametric tests, and large studies—a paradox of statistical practice?* BMC Med. Res. Methodol., 2012. **12**(1): p. 78.

23. Bansal, H. *Best languages for machine learning in 2020!* 2019; Available from: https://becominghuman.ai/best-languages-for-machine-learning-in-2020-6034732dd242.

24. Ratan, V. *Why Python is Ideal for Machine Learning*. 2018 [cited 2020 May]; Available from: https://www.opensourceforu.com/2018/10/why-python-is-ideal-for-machine-learning/.

25. Dalgaard, P., *Introductory statistics with R*. 2008: Springer.

26. Kaewunruen, S. and A.M. Remennikov, *Field trials for dynamic characteristics of railway track and its components using impact excitation technique.* NDT & E International, 2007. **40**(7): p. 510-519.

27. Piotrowski, A.P. and J.J. Napiorkowski, *A comparison of methods to avoid overfitting in neural networks training in the case of catchment runoff modelling.* J. Hydrol., 2013. **476**: p. 97-111.

28. Zur, R.M., et al., *Noise injection for training artificial neural networks: A comparison with weight decay and early stopping.* Med Phys, 2009. **36**(10): p. 4810-4818.

29. Microsoft. *Normalize Data*. 2019; Available from: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/normalize-data.

30. Ghawi, R. and J. Pfeffer, *Efficient Hyperparameter Tuning with Grid Search for Text Categorization using kNN Approach with BM25 Similarity.* Open Comput. Sci., 2019. **9**(1): p. 160-180.

31. Mabayoje, M.A., et al., *Parameter tuning in KNN for software defect prediction: an empirical analysis.* JTSISKOM, 2019. **7**(4): p. 121-126.

32. Radiuk, P.M., *Impact of training set batch size on the performance of convolutional neural networks for diverse datasets.* Information Technology and Management Science, 2017. **20**(1): p. 20-24.

33. Lo, J.T.-H., Y. Gui, and Y. Peng, *The normalized risk-averting error criterion for avoiding nonglobal local minima in training neural networks.* Neurocomputing, 2015. **149**: p. 3-12.

34. Arefin, A.S., et al., *Gpu-fs-k nn: A software tool for fast and scalable k nn computation using gpus.* PloS one, 2012. **7**(8): p. e44000.

35. Lee, W.-Y., S.-M. Park, and K.-B. Sim, *Optimal hyperparameter tuning of convolutional neural networks based on the parameter-setting-free harmony search algorithm.* Optik, 2018. **172**: p. 359-367.

36. Nakhaee, M.C., et al. *The recent applications of machine learning in rail track maintenance: A survey*. in *International Conference on Reliability, Safety, and Security of Railway Systems*. 2019. Springer.

37. Westgeest, F., R. Dekker, and R. Fischer, *Predicting rail geometry deterioration by regression models.* Adv Saf Reliab Risk Manag ESREL, 2011. **146**: p. 926-933.

38. Opitz, D. and R. Maclin, *Popular ensemble methods: An empirical study.* J Artif Intell Res, 1999. **11**: p. 169-198.

39. Kaewunruen, S., J.M. Sussman, and A. Matsumoto, *Grand Challenges in Transportation and Transit Systems.* 2016. **2**(4).