COMMUNICATIONS IN SCIENCE AND TECHNOLOGY
Homepage: cst.kipmi.or.id

Concise convolutional neural network model for fault detection

Muhammad Firdausi*, Shafiq Ahmad

Industrial Engineering Department, King Saud University, Riyadh 12662, Saudi Arabia

Article history:
Received: 7 March 2022 / Received in revised form: 19 June 2022 / Accepted: 26 June 2022

Abstract

Fault detection is an urgent need for maintenance to obtain the optimal scheduling of production activities, improve system reliability, and reduce operation and maintenance costs. Many studies published in recent years focus on machine learning models to detect any system anomalies in line with the era of big data and the fourth industrial revolution (Industry 4.0). Say, a working condition of bearing can be monitored and then any fault can be detected using the vibration analysis of bearing acceleration data. Most of the published works are presented based upon the knowledge of signal processing in which the result depends heavily on feature extraction. It becomes a challenge then to apply a machine learning algorithm directly to the raw acceleration data as it has been successfully applied to raw data in other science and engineering domains. In this article, a concise Convolutional Neural Networks-based deep learning model is proposed for bearing fault detection. The proposed model was concise with 98% less number of parameters compared to other well-known models. It produced 21.21% and 7.03% better accuracy and fault detection rate, respectively. The model was also tested in different operating parameter environments and still gave an excellent result. Since the proposed concise architecture of the model needed short training time, it is deemed suitable for application on manufacturing floor where the pace of production moves fast and the change of the production machine configuration likely occurs.

Keywords: Fault detection; Ball bearing; Deep learning; Convolutional neural network; vibration

1. Introduction

Industrial machines, which can consist of hundreds of parts are expected to have high availability time. To make the most of it and reduce operational costs, any possible unexpected situations should be anticipated and the machine condition should be monitored [1]. Early detection in an emerging harmful problem is essential for anticipating machine idle time to save the time and cost from taking corrective actions for any unscheduled maintenance [2].

Rotating machinery is widely used in domestic and industrial applications. As one of the fundamental types of mechanical systems, its reliabilities affect the entire system [3]. Based on several surveys conducted by the IEEE Industry Application Society, bearing fault is the most common fault type and contributes to more than 50% of all machine failures [4]. Since bearings are typically working in a tough working environment, they are prone to fail during operation. If the defect is not detected in time, it may cause unexpected downtime of the machinery and even lead to catastrophic damage. Therefore, bearing health monitoring is deemed essential for the safe and reliable operation of the machinery and production [5].

A huge amount of vibration data from rolling bearing operations can be collected thanks to the development of advanced sensing technologies and computing systems [6]. As the data are generally collected faster than diagnosticians can analyze it. There is an urgent need for diagnosis methods that can effectively analyze the massive data and automatically provide the accurate diagnosis results. This kind of method is called as intelligent fault diagnosis method in which artificial intelligence techniques are used for distinguishing machinery health conditions [7].

This huge vibration data can be analyzed thoroughly to obtain the condition of the machine, thanks to the advancement of Machine Learning methods. Multiple Restricted Boltzmann Machine (RBM) units were stacked to build a Deep Belief Networks (DBN) in [8] to analyze the vibration data of induction motors. The Fast Fourier Transform (FFT) was used for transforming the input signal into the frequency domain due to DBN modeling difficulty in functioning the input units' correlation. To improve the diagnosis efficiency, a modified-t-distributed stochastic neighbor embedding (M-tSNE) was developed for reducing the input dimension. They applied their method to the artificially-generated fault bearing vibration signal by Electro-Discharge Machining (EDM). The model of 3 hidden layers containing 400 hidden layer units each was trained for 500 epochs to get the accuracy result of 93.18% before feature reduction and 96.36% after feature reduction. However, many trials are still needed to estimate the feature dimension reduction size of M-tSNE, which is an obvious factor for the accuracy improvement in their model.

The emergence of Convolutional Neural Network (CNN), which is motivated by the visual cortex [9] is marked a starting era of successful machine learning [10]. As a subset of the

* Corresponding author.
Email: 438106660@student.ksu.edu.sa
https://doi.org/10.21924/cst.7.1.2022.746

machine learning domain, CNN-based deep learning architectures have been tremendously successful in many practical applications in which the main domain is in computer vision [11]–[14]. Several research works were going toward Transfer Learning that used a popular pre-trained CNN model in computer vision and deployed it in the domain of fault detection. [15] made the use of AlexNet architecture comprising five convolution layers and three fully-connected layers to predict bearing health conditions. They fed the model with features extracted by Ensemble Empirical Mode Decomposition (EEMD) and enveloped decomposition and generated 2-Dimension time-frequency images by wavelet transform. The total parameters of AlexNet were approximately 60 million trainable parameters. ResNet-50 architecture was employed for bearings and centrifugal pump fault detection by [16]. The ResNet-50 architecture consisted of 51 layers and 23 million trainable parameters in which they trained the model from layer 49 until the last layer with the machine fault datasets. The result was comparable with the state-of-theart deep learning application for fault detection in which they altered the raw vibration data into RGB images with 3 Dimension matrix. However, each of the red, green, and blue elements produced by their methods was the same to each other.

This research work aims to develop a concise CNN-based deep learning model for bearing fault diagnosis to make the implementation in a real-world situation simple. The input required for the model was designed as 50 by 50 input array, which reduced the computation process and provided a fast training process [35] with a flexibility to accommodate up to 3 channels input. One channel belonging to raw data and the other 2 accompanying channels were calculated based on a basic statistical formula. We took the benefit of more channels input since it gave not only a better fault detection ability but also a more stable training process. Simplicity in deep learning implementation on fault detection aims to slightly reduce the sole dependency on the signal processing experts who need an extensive training in different subjects. Hence, we proposed a concise deep learning model with a simple form of inputs.

2. Materials and Methods

This section briefly presents the vibration signal, parts of the CNN-based deep learning model, and the input for the architecture.

2.1. Vibration signal

Vibration signal from a bearing is measured by accelerometer and may be used as an indicator in machines that have some problems in quality with the bearing and as the first indication of incoming need for repairment or replacement after running for a long period. Bearings could act as the excitation sources, producing time varying forces that cause system vibration. In some cases, these forces are the result of the imperfections of the bearings [17].

The readings from an accelerometer sensor give decimal values varying in time. When the raw vibration signal is plotted, given the period and sample rate of measurement for the x-axis and acceleration for the y-axis, the appearance is shown in figure 1 taken from a normal bearing of the KAT datacenter in Paderborn University [18]. Whatever the condition of a bearing and the plot is, the readings from the accelerometer are always as a decimal number and this condition makes the bearing fault detection with deep learning suitable. Figure 1 illustrates the first ten data points of a signal as a red dashed rectangle.

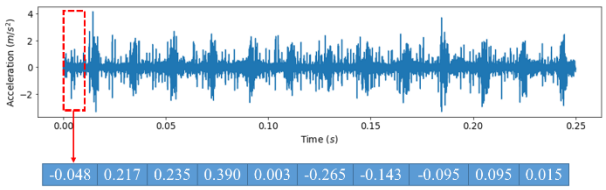


Fig. 1 First ten data points of a signal

2.2. Convolutional layer

The basic idea in a convolutional layer is to apply a small filter kernel to input to learn features. Each kernel contains the learnable weights and will be updated by the backpropagation algorithm to reduce the loss. In this work, the initialization of weights followed the initialization [19] and was done in the PyTorch framework. The activation unit followed each filter to finally generate output features. Inputs for the kernel were called as the input local region and an identical kernel with specific weights convolved the input from beginning to the end; therefore, a kernel resulted in one output channel in the next layer. The number of channels of a layer determined the depth of that layer.

A convolutional layer works by multiplying weights in a kernel with an input local region and it is repeated until the end of the input layer. The process is described as follows:

$$y^{l(i,j)} = K_i^l * x^{l(r^j)} = \sum_{(j'=0)}^{W-1} K_i^l(j') x^{l(j+j')} \quad (1)$$

Where:
K_i^l = the weights of the i-th kernel in layer l
$x^{(l(r^j))}$ = j-th local region in layer l
* = the dot product
W = kernel width
$K_i^l(j')$ = the j'-th weights of a kernel.

The index j represents the index of a point in a local region. Local region itself refers to a region in input array facing the kernel. Therefore, the j added with backtick, j‘ represents index in the kernel facing the local region. The notation r stands for region in l(r j). This region spans from index j to j+j‘. In the summation process, the index j+j‘will change according to j‘. For instance, the kernel with width of 3 and moving one step right (the second convolutional operation), will have index ranging from 1 to 3 (the 0 is the first index in this operation). A visual example of first convolutional operation is depicted in figure 2.

The kernel slides throughout the input until end and an output is produced. The first dot product as seen in figure 2 is calculated as follows:

$$K_1^1(0); K_1^1(1); K_1^1(2) = 0.322;\ 0.150;\ 0.103$$
$$x^{(1(1+0))}, x^{(1(1+1))}, x^{(1(1+2))} = 0.217;\ -0.290;\ 0.349$$

$$y^{(1(1,1))} = 0.322 \times 0.217 + 0.150 \times 0.290 + 0.103 \times 0.349 = 0.062$$

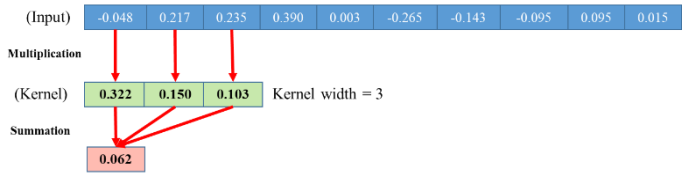


Fig. 2 Convolutional operation

2.3. Pooling layer

In a CNN model, pooling layers are generally placed after convolutional layers. A pooling kernel is used to compress output from a convolutional layer to reduce the dimensionality of the output. The main advantage of the pooling layer is to help the CNN layer's output to be resistant against the small input changes. This advantage is very useful for revealing a feature whether it is present in input data. The most commonly used approach in pooling operation is max-pooling, reporting the maximum value within a local region input for the pooling kernel and outperforming other types of pooling [20]. The max-pooling operation is described as follows:

$$p^{l(i,j)} = max_{(j-1)W+1 \le t \le jW} \{y^{l(i,t)}\} \quad (2)$$

Where:

$y^{l(i,t)}$ = the value of t-th neuron in the i-th frame of layer l, $t \in [(j\text{-}1)W\text{+}1, jW]$

W = the width of the pooling region

$p^{l(i,j)}$ = the corresponding value of the neuron in layer l of the pooling operation

A visual example of convolutional operation is depicted in figure 3.

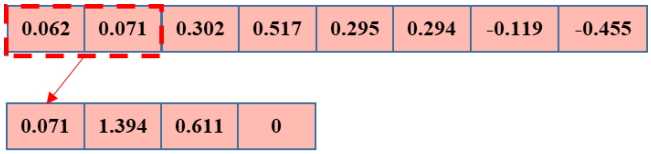


Fig. 3. Max-Pooling Operation

2.4. Activation Operation

In recent neural networks training, the default recommendation activation function for the hidden layer is to use the rectified linear unit or ReLU [21], as defined by the activation function g(z) = max{0, z}. A visual example of convolutional operation is depicted in figure 4.

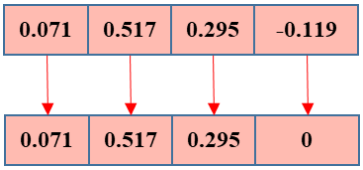


Fig. 4 ReLU activation function of max-pooling output

2.5. Fully-Connected Layer

A fully connected layer receives input as a flat n×1 array form and generates output as a linear representation of the input. The linear expression of the fully-connected layer is:

$$y = xW + b \quad (3)$$

where:

x: the input(s) for a neuron

W: the weight(s) corresponding to the connection between source and destination neurons

b: the bias of the corresponding y

The linear transformation of a fully connected layer from the input layer to the output layer is called as feedforward. Here, we provided a simple feed-forward calculation of a fully connected layer consisting of 1 input layer, 1 hidden layer, and 1 output layer. The weights here were randomly initialized.

Table 1. Data instance for Fully-Connected Layer

Observation	x_1	x_2	Category
1	0.04	0.42	Normal
2	0.5	0.37	Fault 1
3	1	0.54	Fault 2

Given a fully connected (FC) layer that had input neurons called x1 and x2 and their values, two neurons in the hidden layer were called as h1 and h2, and three output neurons were called as Normal (y1), Fault 1 (y2), and Fault 2 (y3). The number of output neurons was the same as the number of categories in the dataset. Then, five bias neurons were added to the network, called as b1 and b2 for the hidden layer; and b2, b4, b5 for the output one. Each neuron connection contained weight and its values were randomly initialized, from W1 to W12. To obtain the value for two neurons in the hidden layer (h1 and h2) first, we picked data from table 1, observation 1 with a health condition and we calculated it based on Equation 3:

h1 = x1×w1 + x2×w2 + b1 = 0.04×-2.5 + 0.42×0.6 + 1.6 = 1.752

h2 = x1×w3 + x2×w4 + b2 = 0.04×-1.5 + 0.42×0.4 + 0.7 = 0.808

Every time a value is produced for a neuron, an activation function is applied to that value before succeeding calculation. The activation function transforms an output value into the input value for the next layer. This function determines whether a neuron is active and enables a CNN model to adapt to the non-linearity of the data. ReLU activation function is applied to the values of the neurons in the hidden layer. Therefore, the output of h1 and h2 is presented as follows:

a (1(1,1)) = max{0, 1.752} = 1.752

a (1(2,1)) = max{0, 0.808} = 0.808

Then, we calculated the value for three output neurons y1, y2 and y3 in the output layer with the linear expression of formula 3:

y1 = h1×w5 + h2×w6 + b3 = 1.3872

y2 = h1×w7 + h2×w8 + b4 = 0.0032

y3 = h1×w9 + h2×w10 + b5 = 0.1352

In this example, to determine what the fully connected (FC) layer predicts based on two inputs x1 = 0.04 and x2 = 0.42 it was firstly to calculate the probability output value after applying the activation function. In the hidden layer, we used ReLU for the activation function but for the output layer, ReLU

could lead a network to stop learning because it could always produce value outputs of 0 and no gradient at all for updating the weights [22]. For activation function in the output layer, softmax function is a better choice to work on classification task [21] for representing the probability distribution over n different classes and a penalty of a prediction could be calculated. The formula for softmax is presented as follows:

$$Softmax(x_i) = \frac{exp(x_i)}{\sum_j exp(x_j)} \quad (4)$$

Where:
x_i: the value of neuron i
x_j: all neuron values in layer j
exp: exponential function with the base e = 2.71828

The probability of each prediction obtained from the softmax activation function is presented as follows:

$$Softmax(y_1) = \frac{exp(y_1)}{exp(y_1)+exp(y_2)+exp(y_3)}$$
$$= \frac{exp((1.387)}{exp(1.3872)+exp(0.0032)+exp(0.1352)}$$
$$= 0.6508$$

With the same way of calculation, we obtained:
Softmax(y2) = 0.1631
Softmax(y3) = 0.1861

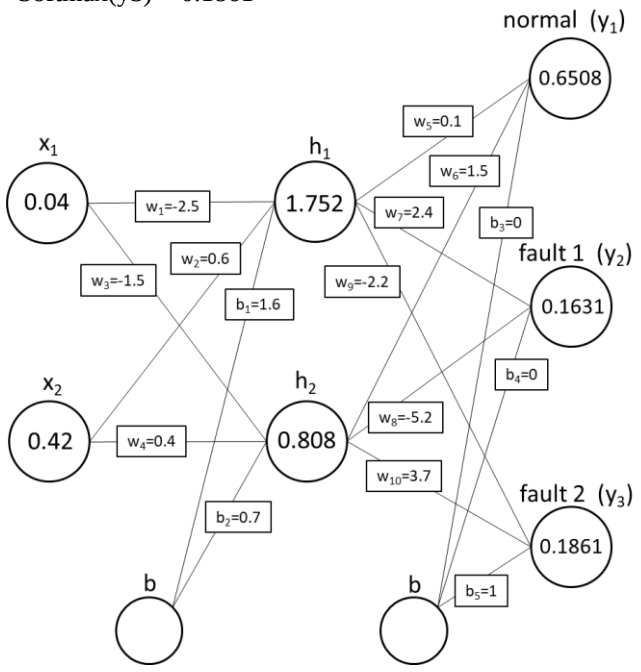


Fig. 5 Fully-connected layer with feed-forward

Figure 5 shows the complete FC calculation result. The final prediction made by the networks was decided by the largest value from the last activation function. Therefore, with input x1 = 0.04 and x2 = 0.42 the model prediction was normal condition with the probability of 0.6508. If we supplied the network with the inputs of observation 2 and observation 3 from table 1, we ended up with all predictions of three observations as shown in table 2.

Table 2. Predicted probability of instance data

Observation	x_1	x_2	Category	Probability
1	0.04	0.42	Normal	0.6508
2	0.5	0.37	Fault 1	0.5036
3	1	0.54	Fault 2	0.5761

2.6. Backpropagation for training the model

Figure 5 shows how a feed-forward neural network predicts a target by giving a specific input. In fact, the true probabilities of all category of the specific input, with x1 = 0.04 and x2 = 0.42 are [normal = 1; fault 1 = 0; fault 2 = 0] because we know that the input belongs to normal condition.

There is a discrepancy between the predicted probabilities generated by the network and the true probabilities and the predicted values are quite different from the real values. The way a network to predict correctly is by having good weight values for a specific dataset. Hence, we needed to repeatedly train the model by tweaking the weights and biases until the output values were nearly similar to the target values. The training process mostly involves a backpropagation (BP) algorithm to fit a neural network model with the training data. BP computes the gradient of the loss function (in this work we used cross-entropy loss) with respect to the weights and biases of every neuron connection. The algorithm aims to tweak the weights, so the model can learn how to map the specific inputs to outputs. The steps of backpropagation calculation are presented as follows:

1) Calculating the total loss of the network. First, we calculated the cross-entropy (CE) for each prediction. The reason behind cross-entropy loss was that it heavily penalized a wrong prediction enabling a network to take a bigger step to minimize the loss. The formula for CE is shown as follows:

$$-\sum_{c=1}^{M} Observed \times log(Predicted_c) \quad (5)$$

Where:
W = number of categories
Observed = true probability
Predicted = predicted probability
log = natural logarithm (base e = 2.718)

Therefore, CE calculation for all categories is:
CEnormal = -1 × log(Predictednormal) + -0 ×log(Predictedfault1) + -0 ×log(Predictedfault2) = 0.4295
CEfault1 = -0 × log(Predictednormal) + -1 ×log(Predictedfault1) + -0 ×log(Predictedfault2) = 0.6859
CEfault2 = -0 × log(Predictednormal) + -0 ×log(Predictedfault1) + -1 ×log(Predictedfault2) = 0.5514
Total loss =CEnormal +CEfault1 +CEfault2 = 0.4295 + 0.6859 + 0.5514 = 1.6668 and the information of all CEs and loss is shown in table 3.

Table 3. Total network loss

Observation	x_1	x_2	Category	Probability	CE
1	0.04	0.42	Normal	0.6508	0.4295
2	0.5	0.37	Fault 1	0.5036	0.6859
3	1	0.54	Fault 2	0.5761	0.5514

2) Calculating the effect of change or derivative of total loss with respect to (wrt.) weights and biases in the output layer by backward pass. Doing this needs to revisit the formula of total loss and depict some figures. First, the total loss is defined as:

Total CE loss = CEnormal + CEfault1 + CEfault2

Each CE has its formula and for calculation example, we picked the normal category.

CEnormal = −1 × log(Predictednormal) + −0 × log(Predictedfault1) + −0 × log(Predictedfault2) = −log(Predictednormal)

The term Predictednormal (pn) is defined as the output from the softmax function of a normal neuron in the output layer. Hence:

$$P_n = Softmax(y_1) = \frac{exp(y_1)}{exp(y_1)+exp(y_2)+exp(y_3)}$$

Lastly, y1 was obtained from Equation 3, i.e. y1 = h1 × w5 +h2 ×w6 +b3. Because the aim of BP is to tweaking weights and biases to minimize the total loss, we first searched the derivative of total loss wrt. b3:

$$\frac{\partial CEloss}{\partial b_3} = \frac{\partial CE_{normal}}{\partial b_3} + \frac{\partial CE_{fault1}}{\partial b_3} + \frac{\partial CE_{fault2}}{\partial b_3} \quad (6)$$

First, we calculated the derivative of CEnormal wrt. b3.

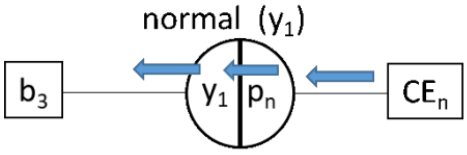


Fig. 6. Finding how much CEnormal change wrt. b3

Figure 6 visually depicts the flow of derivative of CEnormal (CEn in the figure) wrt. b3. In the middle, we had pn and y1 representing the predicted probability of normal condition and input value for output neuron with label normal, respectively. By applying the chain rule we had:

$$\frac{\partial CE_{normal}}{\partial b_3} = \frac{\partial CE_{normal}}{\partial p_n} + \frac{\partial CE_{p_n}}{\partial y_1} + \frac{\partial CE_{y_1}}{\partial b_3} \quad (7)$$

Now we calculated one-by-one all terms on the right-hand side of Equation 7.

$$\frac{\partial CE_{normal}}{\partial p_n} = \frac{\partial}{\partial p_n} - log(p_n) = \frac{-1}{p_n} \quad (8)$$

The derivative of $\frac{\partial p_n}{\partial y_1}$ or the softmax activation function is:

$$p_n \times (1 - p_n) \quad (9)$$

Lastly, we calculated the derivative of the last term

$$\frac{\partial y_1}{\partial b_3} = \frac{\partial}{\partial b_3} h_1 \times w_5 + h_2 \times w_6 + b_3 = 0 + 0 + 1 = 1$$

Hence, we were able to calculate the derivative of Equation 7.

$$\frac{\partial CE_{normal}}{\partial b_3} = \frac{-1}{p_n} \times p_n \times (1 - p_n) \times 1 = 0.6508 - 1 = -0.3492 \quad (10)$$

Second, we calculated the derivative for CEfault1 wrt. b3

$$\frac{\partial CE_{f1}}{\partial b_3} = \frac{\partial CE_{f1}}{\partial p_{f1}} \times \frac{\partial p_{f1}}{\partial y_1} \times \frac{\partial p_{y_1}}{\partial b_3}$$
$$\frac{\partial CE_{f1}}{\partial p_{f1}} = \frac{-1}{p_{f1}}$$
$$\frac{\partial p_{f1}}{\partial y_1} = -p_n \times p_{f1}$$
$$\frac{\partial p_{y_1}}{\partial b_3} = 1$$

$$\frac{\partial CE_{f1}}{\partial b_3} = \frac{-1}{p_{f1}} \times -p_n \times p_{f1} \times 1 = p_n \quad (11)$$

Therefore, derivative of CEfault1 wrt. b3 was pn. To calculate pn for ∂CEf/∂b3, we used x1 and x2 from respective observation i.e. observation of fault 1 (x1 = 0.5; x2 = 0.37) and plug the input into the network. Hence we obtained pn = 0.2606

Third, we calculated the derivative for CEf ault2 wrt. b3

$$\frac{\partial CE_{f2}}{\partial b_3} = \frac{\partial CE_{f2}}{\partial p_{f2}} \times \frac{\partial p_{f2}}{\partial y_1} \times \frac{\partial p_{y_1}}{\partial b_3}$$
$$\frac{\partial CE_{f2}}{\partial p_{f2}} = \frac{-1}{p_{f2}}$$
$$\frac{\partial p_{f2}}{\partial y_1} = -p_n \times p_{f2}$$
$$\frac{\partial p_{y_1}}{\partial b_3} = 1$$

$$\frac{\partial CE_{f2}}{\partial b_3} = \frac{-1}{p_{f2}} \times -p_n \times p_{f2} \times 1 = p_n \quad (12)$$

To calculate pn for ∂CEf2/∂b3, we used x1 and x2 from respective observation i.e. observation of fault 2 (x1 = 1; x2 = 0.54) and plug the input to the network. Hence we obtained pn = 0.2119. We solved all the derivatives for total loss wrt. b3. Recalling equation (6), (10), (11), (12):

$$\frac{\partial CEloss}{\partial b_3} = \frac{\partial CE_{normal}}{\partial b_3} + \frac{\partial CE_{fault1}}{\partial b_3} + \frac{\partial CE_{fault2}}{\partial b_3}$$

$$= -0.3492 + 0.2606 + 0.2119 = 0.1233$$

Table 4. Weights and biases update of Output Layer

Parameter	Before BP*	Gradient	After BP*
b_3	0	0.1233	-0.00123
b_4	0	-0.12132	0.001213
b_5	1	-0.002	1.00002
w_5	0.1	0.123342	0.098767
w_6	1.5	0.123342	1.498767
w_7	2.4	-0.12133	2.401213
w_8	-5.2	-0.12132	-5.19879
w_9	-2.2	-0.00202	-2.19998
w_{10}	3.7	-0.00202	3.70002

The slope of 0.1233 was to update the value of bias b3 with a certain learning rate. The learning rate is a parameter set to determine the step size of each iteration in backpropagation toward a minimum total loss. To calculate the new value of b3 with a learning rate η of 0.01 is presented as follows:

Step size = slope × η = 0.1233 × 0.01 = 0.001233

New b3 = b3 – step size = 0 - 0.001233 = -0.001233

In the same way, we could calculate the new value for b4, b5, w5, w6, w7, w8, w9, and w10. All new values regarding weights and biases in the output layer are shown in table 4.

3) Calculating the derivative of total loss wrt. weights and biases in the hidden layer. We continued the backward pass process for w1, w2, w3, w4, b1, and b2. The backward pass for w1 is shown as follows:

To update the weights and biases in the hidden layer, we used the old weights and biases in the output layer before updating with BP. The main idea of BP in the hidden layer was the same as in the output layer. We calculated the derivative of total loss wrt. weights and biases in the hidden layer. The derivative is written as:

$$\frac{\partial CE_{loss}}{\partial w_1} = \frac{\partial CE_{loss}}{\partial ReLU_{h1}} \times \frac{\partial ReLU_{h1}}{\partial input_{h1}} \times \frac{\partial input_{h1}}{\partial w_1} \quad (13)$$

The process of finding the derivative was similar to the previous calculation but slightly different because the output of neurons in the hidden layer contributed to the output of multiple neurons in the last layer. The connection of neuron h1 with neuron y1, y2, and y3 implied that the output of ReLU in h1 could affect the total loss, as presented as follows:

$$\frac{\partial CE_{loss}}{\partial ReLU_{h1}} = \frac{\partial CE_n}{\partial ReLU_{h1}} + \frac{\partial CE_{f1}}{\partial ReLU_{h1}} + \frac{\partial CE_{f2}}{\partial ReLU_{h1}} \quad (14)$$

First, we calculate $\frac{\partial CE_n}{\partial ReLU_{h1}}$:

$\frac{\partial CE_n}{\partial ReLU_{h1}} = \frac{\partial CE_n}{\partial y_1} \times \frac{\partial y_1}{\partial ReLU_{h1}}$

$\frac{\partial CE_n}{\partial y_1}$ can be solved by:

$\frac{\partial CE_n}{\partial y_1} = \frac{\partial CE_n}{\partial p_n} \times \frac{\partial p_n}{\partial y_1} = \frac{-1}{p_n} \times p_n \times (1 - p_n) = p_n - 1$

To calculate $\frac{\partial y_1}{\partial ReLU_{h1}}$, we recall Equation 3:

$y_1 = ReLU(h_1) \times w_5 + ReLU(h_2) \times w_6 + b_3$

Hence, $\frac{\partial y_1}{\partial ReLU_{h1}}$ is equal to w_5, therefore:

$\frac{\partial CE_n}{\partial ReLU_{h1}} = (0.6508 - 1) \times 0.1 = -0.03492$

Second, we calculated ∂CEf1/∂ReLUh1. Following the previous step, we obtained:

$\frac{\partial CE_{f1}}{\partial ReLU_{h1}} = \frac{\partial CE_{f1}}{\partial y_2} \times \frac{\partial y_2}{\partial ReLU_{h1}}$

$\frac{\partial CE_{f1}}{\partial y_2} = \frac{\partial CE_{f1}}{\partial p_{f1}} \times \frac{\partial p_{f1}}{\partial y_2} = p_{f1} - 1$

$\frac{\partial y_2}{\partial ReLU_{h1}} = \frac{\partial}{\partial ReLU_{h1}} Relu(h_1) \times w_7 + Relu(h_2) \times w_8 + b_4 = w_7$

$\frac{\partial CE_{f1}}{\partial ReLU_{h1}} = (0.5036 - 1) \times 2.4 = -1.1912$

Third, ∂CEf2/∂ReLUh1 was equal to:

$\frac{\partial CE_{f2}}{\partial ReLU_{h1}} = (p_{f2} - 1) \times w_9 = 0.9325$

Put all together:

$\frac{\partial CE_{loss}}{\partial ReLU_{h1}} = -0.03492 + (-1.1912) + 0.9325 = -0.2936$

Now, we needed to find out ∂ReLUh1/∂inputh1 and then ∂inputh1/∂w1 for each weight:

$\partial Relu_{h1} = max(0, input_{h1})$

$$\frac{\partial ReLU_{h1}}{\partial input_{h1}} \begin{cases} 0, & \text{if } input_{h1} < 1 \\ 1, & \text{if } input_{h1} > 1 \end{cases} = 1 \ (since \ input_{h1} = 1.752) \quad (15)$$

$\frac{\partial input_{h1}}{\partial w_1} = \frac{\partial}{\partial w_1} x_1 \times w_1 + x_2 \times w_2 + b_1 = x_1 = 0.04$

Put all together (from Equation 13):

$\frac{\partial CE_{loss}}{\partial w_1} = -0.293610.04 = -0.01174$

The slope of -0.01174 was used to update the w1, so: Stepsize = slope×η = −0.01174×0.01 = −0.0001174

Neww+ 1 = w1–stepsize = −2.5–(−0.0001174) = −2.4998826

With the same steps, we could update weights and biases in the hidden layer, w2, w3, w4, b1, and b2. All updates are shown in table 5.

We finished updating all weights and biases, calculated the classification probability of all input and then determined the total loss. We have updated all of our weights and biases through BP one time. Before the update, the net loss was 1.6668. After our one BP example, the total loss then became 1.644101. Running BP a bunch of times led to net loss toward 0 and made the net capable of well predicting the training data. However, if we continued the BP process a large number of times, it could lead the network to have a low loss value in predicting the data it is used to but predict poorly the data that has never been seen and it is called as overfitting.

Table 5. Weights and biases update of hidden layer

Parameter	Before BP*	Gradient	After BP*
b_1	1.6	-0.2936	1.60294
b_2	0.7	0.48888	0.69511
w_1	-2.5	-0.01174	-2.4998
w_2	0.6	-0.1234	0.60123
w_3	-1.5	0.01956	-1.5002
w_4	0.4	0.20534	0.39795

*BP=Backpropagation

Table 6. Total network loss after update

Observation	x_1	x_2	Category	Probability	CE*
1	0.04	0.42	Normal	0.646766	0.435771
2	0.5	0.37	Fault 1	0.656898	0.656898
3	1	0.54	Fault 2	0.551432	0.551432
			Total Loss		1.644101

2.7. Dropout

To prevent the network from overfitting, we applied a method called as dropout as proposed by [23]. The dropout method works by deactivating neurons randomly along with their connections based on some probabilities p during training. This method proves to prevent neurons from fitting too much to training data. During the training phase, the weights and biases being updated are the active neurons only. During testing the network with new data, dropout is no longer applied. Based on experiments by Srivastava et. al. [23], a network trained with dropout commonly had much better generalization ability on classification problems during test time. A dropout example from the previous fully connected network with probability p=0.3 is presented as follows:

Say, we randomly deactivated neuron in hidden layer from Figure 5 and for instance the deactivated neuron was h1; therefore, the values of y1, y2 and y3 are presented as follows:

y1 = h2 ×w6 +b3 = 1.212
y2 = h2 ×w8 +b4 = −4.2016
y3 = h2 ×w10 +b5 = 3.9896

2.8. Proposed Model

The proposed model for this research is based on Convolutional Neural Network (CNN) taking raw signal data as input without any pre-processing. CNN can extract any relevant features from the data for the prediction task. The model architecture is motivated by early successful model in document recognition by LeCun et al [32]. The model comprises two convolutional layers and is ended with fully-connected layer. These subsequent convolutional (conv) layers

intersected with pooling layer after each conv layer detect salient features that differ between normal and faulty bearing. Conv layer learns multiple features in parallel for a given input and it is common for a conv layer to learn from 32 to 512 filters to get their features [33]. The number of features (or output) from a conv layer, called as feature map, in this architecture is set to 32 for the first conv layer, and 64 for the second one are inspired by VGG-model [13] where the authors used smallest size of filter possible to capture the features in the beginning, and went bigger afterward.

In our CNN model (figure 7), there are two main parts. The first part is the convolution part and the second one is the fully connected layers. We had two convolutional layers each of which was followed by the max-pooling layer. The activation function used for both convolutional layers was ReLU. The usage of maxpooling layers ensures that the most important features are selected. All functions of the proposed model are described in table 7.

The size of the kernel was set as 4 on all the convolution and pooling layer plus a padding of 1 to benefit from generalization capabilities of even-sized kernels at little computational cost [34]. The stride of 1 in convolutional layer to catch fine features from the data and stride of 2 in pooling layer is to sufficiently reduce the dimension of the input data.

Table 7. Model's layers description

Function	Name	Description
F(1)	2D Convolution	i=3; o=32; k=4; s=1; p=1
F(2)	Max-Pooling	k=4; s=2
F(3)	2D Convolution	i=32; o=64; k=4; s=1
F(4)	Max-Pooling	k=4; s=2
F(5)	Flatten	size= 5184×1
F(6)	Fully-Connected	i=5184; o=256; p = 0.2
F(7)	Fully-Connected	i=256; o=3
i=input channel	o=output channel	k=kernel size
s=stride	p=padding	

3. Results and Discussion

We separated the whole dataset into two different parts: the training data set and the testing dataset. The training dataset aims to be a way for the model to learn the vibration data until it can classify the normal bearing and the faulty one. The learning process of the model is to repeatedly see the same training dataset as much as a hyperparameter called as epoch is set. Hyperparameters setting for the training phase are shown in table 8.

Table 8. Hyperparameters setting for training

Hyperparameter	Values
Batch Size	128
Learning rate	0.001
Epochs	100

We used a random train-test split of 80%-20% respectively and then reported the average prediction accuracy. The training dataset refers to a dataset used for training the model with feedforward and backpropagation repeatedly until the number of epochs is reached. During the training phase, the model was fed with the training dataset multiple times until the loss score was lowered. The test dataset aimed to know the prediction ability of the model after training on the dataset that the model has not seen before; it is called as generalization. In the test phase, the model was fed by the test dataset and did the feedforward but not the back pass or backpropagation. Therefore, there were no parameters updated during the test phase.

For this dataset, the signals of 256,000 data points were clipped at the beginning and the ending by 3000 data points to avoid noise disturbance [24]. Then, 250,000 data points per signal were reshaped into the smaller signals of shape 50 × 50 2D arrays, which resulted in 100 smaller signals from each original signal. The considered input shape was based on approximation on how many a bearing rotated in a second. In our setting, the operating parameters of the test rig were in the speed of 1500 revolutions per minute, load torque of 0.7 Nm, and radial force of 1000 N. The speed of a bearing was 1500 revolutions per minute leading to 100 revolutions in 4 seconds. Hence, a signal containing 2500 data points is represented as a bearing revolution.

To recap we had 29 bearings coming with 3 health conditions, 20 times of signal measurement for each bearing, and smaller 50 × 50 2D arrays signals from each measurement counted into 59,317 signals in total in which 47,453 signals belonged to the training dataset and the remaining 11,864 signals belonged to test dataset. Each row in the figure represented a single signal and each column was the 2500 data points of each signal (features). The figure was clipped in the middle to accommodate the page width (represented as three

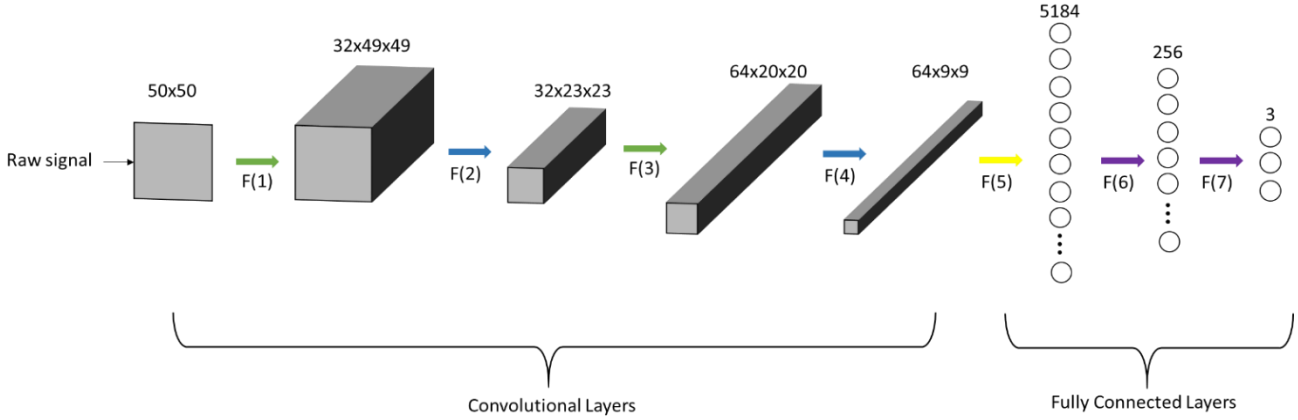


Fig. 7. Proposed model for fault diagnosis

red bold dots). Therefore, in total, we had 59,317 rows and 2501 columns (include the Condition column). The Condition column consisted of IR, normal, and OR, which stand for bearings condition of inner ring fault, normal, and outer ring fault, respectively.

As the problem's nature was classification, we used Crossentropy loss [25] for the loss function, as we provided in Chapter 4. The whole network was trained for 100 epochs with a batch size of 128 on a Google colaboratory GPU machine.

We fed the model with raw data and calculated the loss and accuracy of the model. The accuracy is a ratio of correct prediction for all classes to the total observations and in our case, is defined as:

$$Accuracy = \frac{TP_1 + TP_2 + TP_3}{TP_1 + TP_2 + TP_3 + FP_{all}} \quad (16)$$

where:

TP1,2,3 = True Prediction of class 1, 2, and 3

FPall = False Prediction from all class

The 100 epochs training process with raw data input took a time of 27 minutes and 38 seconds with a loss of 0.00014514. In line with the results of the training phase, the accuracy on the training dataset maxed out was at 99.6% with a loss of 0.0170.

To know more about the effect of input data on the training result, we added some additional channels to the input. Originally, we had the input of raw acceleration data in the shape of 50 × 50 data points. Then, we made a new channel called as mean channel and median channel. The establishment of the two new channels was by making use of a sliding window with a length of 10 as a filter with a shift by the length of 1. For every given sample of raw signal data, the filter scanned through the whole sample data from the front to the end. The size of the mean and median windows depended on the size of the single original input data. Here, the decision of the windows size of 10 was to fairly accommodate the size of the original input data of 2500. In other words, the size of 0.4% of the original data is adequate for the mean and median windows. These two parameters provided a balance combination to represent data, the mean for measuring the central tendency and the median was to make the addition input insensitive to the outlier data point. The statistical parameters were chosen based on the computational cost and their advantages. For the sake of simplicity, the example of generating mean and median channels from raw data with a sliding window of length 3 is depicted in figure 9.

The three different channels could provide us several combinations of input i.e. input of raw channel, raw plus mean channels, raw plus median channels, and all three channels. In summary, we trained the model with four different inputs and recap the loss at the end of the training, the time needed for running 100 epochs in table 9.

It was found out that an additional channel could make the training process longer but give an improvement in the training phase - in terms of lower loss and better accuracy. The combination of raw signal and its median resulted in better loss and more accuracy than the combination of raw signal and its mean. Therefore, we could assess that median of signal presented a better feature of bearing fault. Next, the loss score of the whole training process is depicted in figure 8.

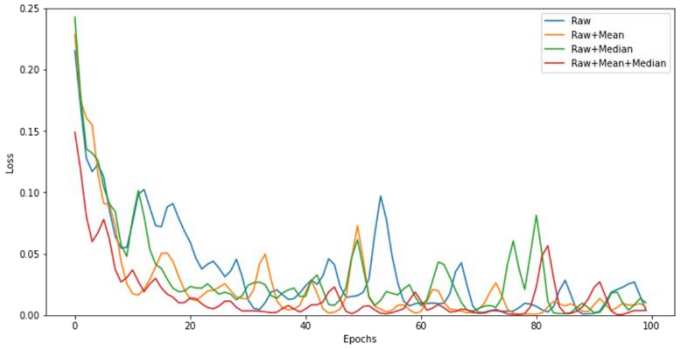


Fig. 8. Loss across the different inputs

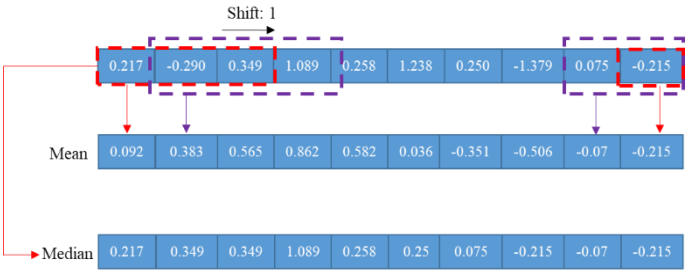


Fig. 9. Mean and median input channel generator

Table 9. Training results of different input combination

Combination	Training Loss	Time	Test loss & Accuracy
Raw Channel	0.000145	27 mins 38 secs	0.02214; 99.47%
Raw + Mean Channels	0.00029	28 mins 2 secs	0.01614; 99.62%
Raw + Median Channels	0.000023	29 mins 3 secs	0.01379; 99.72%
Raw + Mean + Median Channels	0.004996	29 mins 15 secs	0.00692; 99.83%

3.1. False Alarm Rate (FAR)

To augment accuracy as an evaluation metric, we also calculated the FAR metrics, which is the ratio of falsely predicting positive observations to all observations in actual positive class. An example of this metric answer is of all bearings that predicted fault, how many are not fault. To calculate the FAR metrics, first, we established a confusion matrix containing the prediction and the true class label in a single matrix. To establish the confusion matrix, we employed the model trained with 3 channels input and recorded the predicted class and the ground truth class. The confusion matrix is shown in figure 10. TN is True Negative or the model correctly predicts a bearing as normal bearing, where FN (False Negative) is the opposite, and the model predicts a bearing as normal but it is faulty. Likewise, TP is True Positive where the model correctly predicts a bearing as faulty bearing and FP is False Positive where a bearing is falsely classified as a faulty bearing which the true condition of the bearing is normal.

The labels of N, IR, and OR in figure 10 stand for Normal bearing, Inner Fault bearing, and Outer fault bearing, respectively. The green and red rounded rectangles indicate both the two true values, True Positive and True Negative (TP and TN), and two false values, False Positive and False Negative (FP and FN) respectively. FP means that the model predicts the input signal as IR or OR; however, the actual condition of bearing is N. Here, we considered a misprediction of ground truth from IR predicted as OR and vice versa as a true positive since the main objective of fault detection is to distinguish a fault from a normal one. This is the primary concern in practical applications for the operators on-site [26]. The FAR metric is calculated by the following formula:

$$False\ Alarm\ Rate = \frac{FP}{FP+TN} \tag{17}$$

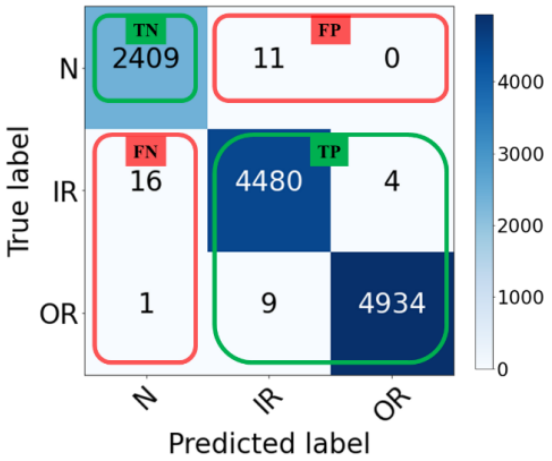


Fig. 10. Confusion matrix of the model

3.2. Fault Detection Rate (FDR)

The last metric to consider the performance of the fault detection model is Fault Detection Rate (FDR), which is calculated based on faulty data. In literature, FDR is called as Recall or sensitivity [27]. In general, the higher the FDR score, the better a model. FDR is the opposite of FAR, which is a ratio of correctly predicted positive observations over all observations in positive class. The question to be answered by the FDR metric is about, of all positive observations, how many percent a model can predict fault bearing from a dataset. To calculate FDR, we used the same confusion matrix as to calculate FAR. The formula for FDR is presented as follows:

$$False\ Detection\ Rate = \frac{TP}{TP+FN} \tag{18}$$

The two metrics calculation along with the summarized confusion matrix is provided in table 10.

Table 10. Model result

		True bearing condition		
		Fault	Normal	Total
Model Prediction	Fault prediction	9427	11	9438
	Normal prediction	17	2409	2426
	Total	9444	2420	11864

FAR: FP / (FP+TN) = 11 / 2420 = 0.004545 = 0.4545%
FDR: TP / (TP+FN) = 9427 / 9444 = 0.99819 = 99.819%
Accuracy: Correct / All observations = 11836 / 11864 = 99.6544%

3.3. Result in different datasets

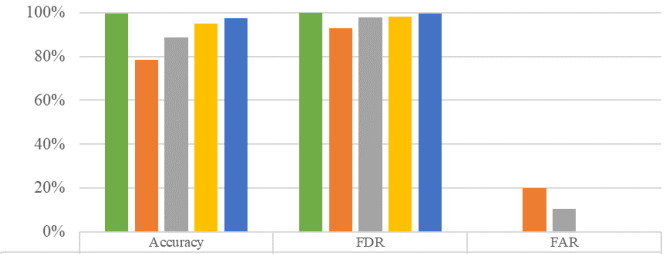
This section presents the test of the trained model to detect the fault from other datasets with different operating parameters. The steps are the same as training and testing the model and the result is shown in table 11. The combination code includes N: speed (rpm); M: load torque (Nm); F: radial force (N) where the details of the combination refers to [18]

Our proposed model achieved a satisfactory result for accuracy and FDR scores of above 99% in all operating parameter combinations. However, in an environment with lower parameter values of speed, load torque, and radial force, it was found that the model architecture encountered slight difficulty when it predicted a real normal bearing. Given the base result for comparison is the parameter combination of speed: 1500 rpm; load torque: 0.7 Nm; and radial force: 1000N (combination number 4), FAR scores showed that a lower value of radial force deteriorates more followed by load torque and speed. The ability of the model to predict a real normal bearing as normal for lower parameter values of radial force, load torque, and speed, decreased from 0.45% to 1.699%, 1.171%, and 1.123%, respectively.

3.4. Result Comparison

Finally, we compared the result between the proposed model and other works from previous authors working with the same dataset. Other works such as [28] employing Multi-Layer Perceptron (MLP) and Deep Belief Network (DBN); TrainingInterference for CNN (TICNN) [29]; Wasserstein distance guided representation learning for domain adaptation (WDGRL) and triplet loss guided adversarial domain adaptation method (TLADA) [30]. The complete comparison is depicted in figure 11.

Our proposed model achieved a better result in terms of accuracy, FDR, and FAR with a maximum value of 21.21% better accuracy, and 7.03% better FDR. Note that work by [30] employing WDGRL and TLADA was done with a smaller dataset in which they achieved a better FAR value; however, it is obvious that a model tested with a bigger dataset will have more probability to error than with a smaller dataset although our accuracy and FDR are still better.



	Accuracy	FDR	FAR
This work	99.65%	99.82%	0.45%
MLP	78.44%	92.79%	20.10%
DBN	88.60%	97.85%	10.57%
WDGRL	95.07%	98.20%	0.00%
TLADA	97.33%	99.55%	0.05%

Fig. 11. Result comparison

4. Conclusions

The main objective of this work is to develop a machine learning model for fault detection. We proposed a deep learning

model with a concise architecture that had some impressive results compared with the previous works. The proposed model can analyze the raw acceleration data directly and requires almost no knowledge about digital signal processing to process the input data. However, a trade-off relationship between input's channel number, training time, and prediction results is evident, as the more input channel would make the training time longer yet yield better results. Thus, it is important to understand the relationship and to utilize the most suitable input for a specific condition.

In addition, to demonstrate the impact of the proposed research model, we highlighted the key areas, which we have investigated based on the available scientific literature. Authors in [31] did a comprehensive survey and they found that most of the models provided in the literature were being trained in a single operation parameter, whereas in this research work, we have demonstrated the ability of the proposed model to predict across different operating parameters, as a significant contribution.

Furthermore, the proposed model is presented in a concise architecture and the proposed architecture will be easy to implement in real-world applications by practitioners. In comparison with several well-known CNN-based architectures like AlexNet [11] containing approximately 60 million trainable parameters, VGG-16 Net [13] with 138 million parameters, ResNet [14] with 23 million parameters, GoogLe Net [12] comprising seven million parameters, our proposed model contained only 1.3 million parameters and still provided the considerably satisfying results. Time for training the CNN model from scratch was rather long (even up to six days of training just for 90 epochs) for several deeper architectures, such as AlexNet, VGG-16 Net, ResNet, and GoogleLe Net. The proposed concise architecture, which in practice needed no more than 30 minutes of training time from scratch for 100 epochs, is more likely fit in the needs on manufacturing floor where the pace of production moves fast.

A further enhancement of the model development is to explore more about the generalization ability, which is one of the most challenging tasks for a machine learning model [21]. The generalization ability of a model means that the model can perform its ability well even on data that have not been seen before. In the domain of machine fault detection, it would be a model, which can detect a fault in different machines, not the same as the model was trained for.

References

1. O. Janssens, V. Slavkovikj, B. Vervisch, K. Stockman, M. Loccufier, S. Ver-stockt, R. V. d. Walle, and S. V. Hoecke, *Convolutional Neural Network Based Fault Detection for Rotating Machinery*, J. Sound Vib. 377 (2016) 331-345.
2. L. Wen, X. Li, L. Gao, and Y. Zhang, *A New Convolutional Neural Network-Based Data-Driven Fault Diagnosis Method*, IEEE Trans. Ind. Electron. 65 (2018) 5990–5998.
3. C. Lu, Y. Wang, M. Ragulskis, and Y. Cheng, *Fault Diagnosis for Rotating Machinery: A Method based on Image Processing*, PloS one, 11 (2016) 1-22.
4. J. Tao, Y. L. Liu, and D. L. Yang, *Bearing Fault Diagnosis Based on Deep Belief Network and Multisensor Information Fusion*, Shock Vib. 2016 (2016) 1-9.
5. Z. Y. Chen and W. H. Li, *Multisensor Feature Fusion for Bearing Fault Di-agnosis Using Sparse Autoencoder and Deep Belief Network*, IEEE Trans Instrum Meas. 66 (2017) 1693–1702.
6. R. Zhao, D. Wang, R. Yan, K. Mao, F. Shen, and J. Wang, *Machine Health Monitoring Using Local Feature-Based Gated Recurrent Unit Networks*, IEEE Trans. Ind. Electron. 65 (2018) 1539–1548.
7. F. Jia, Y. G. Lei, J. Lin, X. Zhou, and N. Lu, *Deep neural networks: A promising tool for fault characteristic mining and intelligent diagnosis of ro-tating machinery with massive data*, Mech Syst Signal Process. 72–73 (2016) 303–315.
8. S.-Y. Shao, W.-J. Sun, R.-Q. Yan, P. Wang, and R. X. Gao, *A Deep Learning Approach for Fault Diagnosis of Induction Motors in Manufacturing*, Chin. J. Mech. Eng. 30 (2017) 1347–1356.
9. K. Fukushima, *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*. Biol. Cybernetics, 36, (1980) 193–202.
10. Y. Le Cun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard and W. Hubbard, *Handwritten digit recognition: applications of neural network chips and automatic learning*, IEEE Commun. Mag. 27 (1989) 41–46.
11. A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, Adv Neural Inf Process Syst. 25 (2012) 1-9.
12. C. Szegedy, et al., *Going Deeper with Convolutions*, Proc. IEEE Conf. Com-put. Vis. Pattern Recognit. Boston, MA, USA, 2014, pp. 1-9.
13. K. Simonyan and A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 3rd International Conference on Learning Representations. San Diego, CA, USA, 2015, pp. 1-14.
14. K. He, et al., *Deep Residual Learning for Image Recognition*. Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Las Vegas, NV, USA, 2015, pp. 770-778.
15. X. Shi, Y. Cheng, B. Zhang, and H. Zhang, *Intelligent fault diagnosis of bearings based on feature model and Alexnet neural network*, Int. Conf. Progn. Health Manag. Detroit, MI, USA, 2020, pp. 1–6.
16. L. Wen, X. Li, and L. Gao, *A transfer convolutional neural network for fault diagnosis based on ResNet-50*, Neural Comput. Applic. 32 (2020) 6111–6124.
17. T. Harris, Boca Raton, USA: John Wiley Sons, Inc., 2001.
18. C. Lessmeier, J. K. Kimotho, D. Zimmer, and W. Sextro, *Condition Monitor-ing of Bearing Damage in Electromechanical Drive Systems by Using Motor Current Signals of Electric Motors: A Benchmark Data Set for Data-Driven Classification*, PHM Society European Conference. 3 (2016) 1-17.
19. K. He, et al., *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, Proc. IEEE Int. Conf. Comput. Vis., Santiago, Chile. 2015, pp. 1026-1034.
20. V. Suarez´-Paniagua and I. Segura-Bedmar, *Evaluation of pooling operations in convolutional architectures for drug-drug interaction extraction*, BMC Bioinformatics. 19 (2018) 39-47.
21. I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. USA: MIT Press, 2016.
22. R. Grosse, CSC321 Lecture 8 Optimization. Canada: CS University of Toronto, 2021.
23. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, J. Mach. Learn Res. 15 (2014) 1929–1958.
24. R. Magar, L. Ghule, J. Li, Y. Zhao, and A. B. Farimani, *FaultNet: A Deep Convolutional Neural Network for Bearing Fault Classification*. IEEE Access, 9 (2021) 25189–25199.
25. Z. Zhang and M. R. Sabuncu, *Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels*, 32nd Int. Conf. Neural Inf. Process. Syst., NY, USA. 2018, pp. 8792–8802.
26. X. Xue and J. Zhou, *A hybrid fault diagnosis approach based on mixed-domain state features for rotating machinery*, ISA Trans. 66 (2017) 284–295.
27. D. G. Altman and J. M. Bland, *Diagnostic tests. 1: Sensitivity and specificity*, BMJ-Brit. Med. J. 308 (1994) 1552.
28. L. Hou, R. Jiang, Y. Tan, and J. Zhang, *Input feature mappings-based deep residual networks for fault diagnosis of rolling element bearing with complicated dataset*, IEEE Access. 8 (2020) 180967–180976.
29. Y. H. Chen, G. L. Peng, C. H. Xie, W. Zhang, C. H. Li, and S. H. Liu, *ACDIN: Bridging the gap between artificial and real bearing damages for bearing fault diagnosis*, Neurocomputing. 294 (2018) 61–71.
30. X. Wang and F. Liu, *Triplet Loss Guided Adversarial Domain Adaptation for Bearing Fault Diagnosis*, Sensors, 20 (2020) 1-19.
31. J. Jiao, M. Zhao, J. Lin, and K. Liang, *A comprehensive review on convolutional neural network in machine fault diagnosis*. Neurocomputing, 417 (2020) 36–63.

32. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-Based Learning Applied to Document Recognition*, Proc. of the IEEE. 86 (1998) 2278–2324.
33. J. Brownlee, *Deep Learning for Computer Vision: Image Classification, Ob-ject Detection and Face Recognition in Python*. Independently Published, 2019.
34. S. Wu, G. Wang, P. Tang, F. Chen, and L. Shi, *Convolution with even-sized kernels and symmetric padding*. Adv. Neural Inf. Process. Syst. 32 (2019) 1-12.
35. C. R. Atmaja Perdana, H. Adi Nugroho, and I. Ardiyanto, *Comparison of text-image fusion models for high school diploma certificate classification*, Commun. Sci. Technol. 5 (2020) 5–9.