Anomaly Detection Based on Tiny Machine Learning: A Review

Article history

Received:
12th October 2021

Received in revised form:
18th October 2021

Accepted:
28th October 2021

Published online:
11th November 2021

*Corresponding author
mdridzuan@utm.my

Yap Yan Siang[1], Mohd. Ridzuan Ahamd[2*], Mastura Shafinaz Zainal Abidin[3]

[1,2,3]School of Electrical Engineering, Faculty of Engineering, Universiti Teknologi Malaysia, 81310 Skudai, Johor, Malaysia
[1]ysyap@graduate.utm.my, [2]mdridzuan@utm.my,
[3]mastura@fke.utm.my

Abstract

Anomaly detection (AD) is the detection of patterns in data under expected behavior. In an industrial environment, any equipment or system that breaks down will affect productivity. Therefore, Tiny Machine Learning (TinyML) is introduced to address this problem. TinyML can undergo anomaly detection to detect if any equipment did not act within expected behavior and notify the user if an anomaly detection has been detected. Anomaly detection is an unsupervised learning algorithm. It aims to identify the patterns in data that do not follow the expected behavior. By using TensorFlow Lite Micro, the TinyML can be trained to undergo anomaly detection. However, the machine learning algorithm had to be exported from TensorFlow, then TensorFlow Lite, and finally TensorFlow Lite Micro in order to upload the machine learning algorithm into TinyML. This paper highlights the state of the art of the current works on TinyML. Some suggestions on the research direction are also introduced for potential future endeavors.

Keywords: *TinyML, Embedded system, Reliability engineering, Anomaly Detection, Model Compression,*

1. Introduction

Reliability is the main attribute for safe operation in any modern technological system. Reliability analysis focuses on uncertainty in failure incidences and the aftereffects. The aim is to shield the system beyond the uncertainties of its accidental situations [1]. In recent years, reliability engineering has been well-established into a multi-disciplinary scientific discipline that seeks to offer an ensemble of formal methods to inspect the unclear boundaries between system operation and failure. Furthermore, E. Zio [1] also recorded the main questions focusing on reliability engineering. These questions are the main causes of the system's failure, the procedure of measuring and testing the reliability in design, operation, and management, the ways to maintain the system's reliability such as maintenance, fault diagnosis, and prognosis, and the techniques to develop the reliable systems. Engineering maintenance and prognostics play a critical role in modern industries such as aerospace, locomotive, manufacturing, and so forth. The usual engineering maintenance approach is to maintain the functionality of the equipment or system, such as preventive maintenance or scheduled maintenance. However, these approaches could not fulfil the growing demand of economic efficiency, reliability, and safety [2]. Therefore, advanced condition-based

* *Corresponding author. mdridzuan@utm.my*

maintenance (CBM) has been introduced [3]. CBM is also known as prognostic and health management (PHM) [4]. PHM is a method that enables the assessment of the reliability of the equipment, a system under its actual application conditions. The objectives of PHM are improving reliability, enhancing performance ability, and reducing the maintenance cost of the equipment. The remaining useful life (RUL) is the remaining lifetime length of the equipment to operate before complete failure. There are several approaches in RUL, such as model-based approaches and data-driven approaches. Model-based approaches build a physical failure model to deduct an accurate RUL, while data-driven approaches build a less complex model such as a degradation model with historical sensor data.

The development of Machine learning (ML) allows the insight of the world by enabling inference of information and knowledge from a tremendous amount of data that are normally unseen [5]. Hence, ML algorithms can be used to predict future values of phenomena using a generated model and abstract the understanding of underlying phenomena in the form of a model. ML is a computational algorithm that converts empirical data into usable models. Besides that, ML can solve the challenges in reliability engineering and safety analysis by discovering more survival data, accurate insights from accident datasets, or degradation[6]. There are several areas that utilize ML, such as fraud detection, bioinformatics, market and business prediction, and speech recognition [7]. Thus, ML enables the optimization of the system designs by upgrading the efficiency of the systems and the designing of the machines [8]. Furthermore, ML can be categorized into supervised learning, semi-supervised learning, and unsupervised learning, as shown in Figure 1.

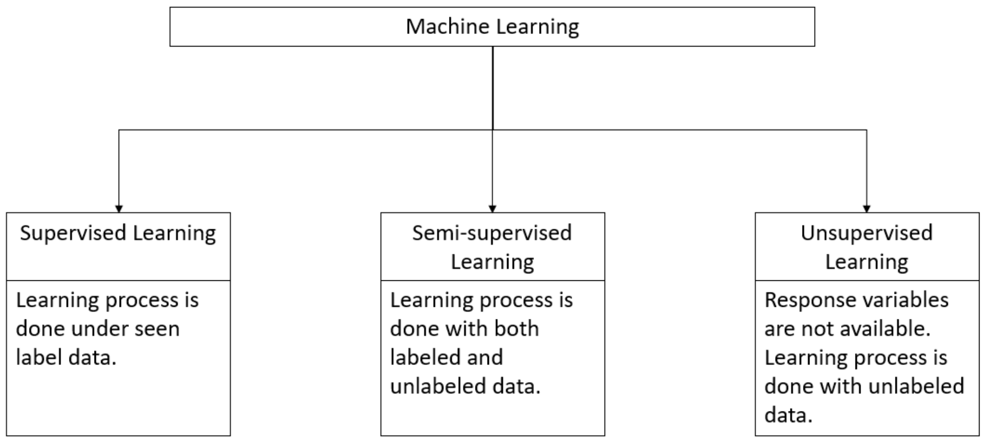
Machine Learning
Supervised Learning
Learning process is done under seen label data.
Semi-supervised Learning
Learning process is done with both labeled and unlabeled data.
Unsupervised Learning
Response variables are not available. Learning process is done with unlabeled data.

Figure 1. Categories of Machine Learning

Cloud machine learning is the first approach to machine learning and following by mobile machine learning and now tiny machine learning (TinyML). TinyML is the deployment or integration of ML in microcontroller Units (MCUs). MCUs are embedded devices that consist of a core processing unit of a small frugal object [5]. TinyML is an innovative approach that aims to bring ML inference to ultra-low-power devices such as under one milliwatt [9]. Moreover, the efficiency of TinyML enables always-on applications, battery-powered, that enable real-time processing and collection of data. Generally, the MCUs had limited memory and onboard compute horsepower. Therefore, the TinyML models should be small enough to fit in the MCUs. However, these tight constraints will limit the number of layers and the size of the input.

2. Literature Review

2.1. Supervised Learning

The equation in Figure 2 represents the ingredient of machine learning. The unknown model function in Figure 2, *f(x)* is the information that the feature vector *X* provides about the label *Y*. In general, machine learning is the estimation of functions that can map the input to obtain the output. Supervised learning means the learning process is done using labeled data. There are two sub-categories of supervised learning, i.e., regression and classification. The regression goal is to predict a continuous value output, while the classification goal is to predict a discrete value output. Linear regression and support vector regression (SVR) are examples of regression machine learning algorithms. Some of the recent development in neural networks are deep neural network (DNN) models, recurrent neural networks (RNN), and convolutional neural networks (CNN).

Classification aims to predict a discrete value output. Classification problem involves an output with one of two or more classes, such as email identification of spam or not spam or multi-class output. However, the number of classes is known in advance and constrained [10]. Classification machine learning algorithm includes k-nearest neighbor (KNN), random forest (RF), and support vector machine (SVM).

2.2. Unsupervised Learning

Unlike supervised learning, the unsupervised learning process is via unlabeled data and aims to explore the feature space and find patterns in the data set. Unsupervised learning is not focusing on predicting the output since there is no output variable. There are two sub-categories of unsupervised learning, i.e., clustering and anomaly detection. Clustering is used to discover the data analysis to find groupings or hidden patterns in the data. One of the famous methods in clustering is the k-means clustering algorithm in image segmentation [11]. K-means

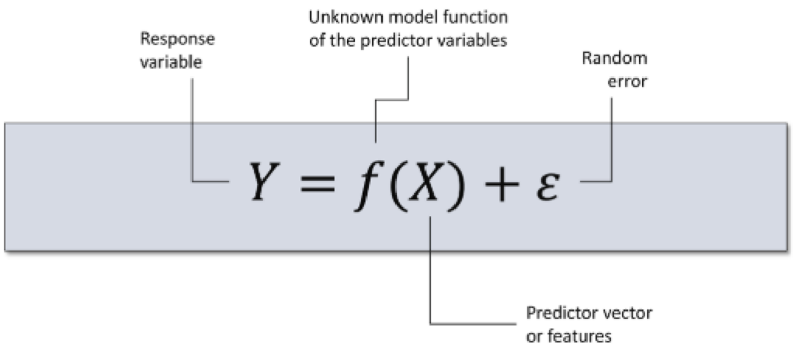
Response variable
Unknown model function of the predictor variables
Random error
$Y = f(X) + \varepsilon$
Predictor vector or features

Figure 2. Ingredient of Machine Learning [6]

clustering algorithm classifies the classes into k numbers. Then it calculates the k centroid and takes the nearest centroid to the cluster. K-means clustering algorithm is an iterative algorithm that lessens the sum of the distance from each object to its cluster centroid.

Besides that, the other sub-category of unsupervised learning is anomaly detection. Anomaly detection aims to identify the patterns in data that do not follow the expected behavior [12]. Many applications utilize anomaly detection such as cyber-intrusion detection, sensor networks, image processing, textual anomaly detection, industrial damage detection, medical anomaly detection, and fraud detection. Besides that, the anomaly detection algorithm consists of self-organizing maps (SOM) and K-mean. SOM approaches are based on neural networks. SOM is used for classifying the data according to the similarity between the data. Besides that, SOM allows about reduction of multidimensional space into two-dimensional space [13]. SOM initialized by setting up a regular grid can be either rectangular or hexagonal arrangement. Furthermore, each unit contains a distinct codebook vector. The weighted average codebook vector will replace the winning unit after each training. The winning unit is the unit with the highest similarity to the presented object [14]. Nevertheless, not only the winning unit will be updated, but all units in the neighborhood will be updated as well. Therefore, the size of the neighborhood also gradually decreases during training.

2.3. Semi-supervised Learning

Semi-supervised learning algorithms are applied in remaining useful lifetime (RUL), identification, and fault detection, which are essential in maintenance planning [6]. Semi-supervised is the mixture of both supervised learning and unsupervised learning. Semi-supervised learning process using both labeled and unlabeled data. Like supervised learning, the sub-categories of semi-supervised learning are semi-supervised regression and classification. Semi-supervised regression (SSR) indicates where the output-valued is real-valued [15]. Semi-supervised classification (SSC) refers to the situation where output is discrete and utilizes a little labeled data and numerous unlabeled data [16]. One of the famous semi-supervised learning is the graph-based model (GBM). The graph structure in GBM can effectively encode the intrinsic physical properties of the complex social relationship in practice [17]. In GBM, a graph was constructed with nodes and edges. Nodes are specified with unlabelled and labelled samples. Furthermore, the edges determine the similarities among the labelled and unlabelled samples [18]. Each sample is represented by a vertex in the weighted graph that calculates the similarity between samples.

2.4. TinyML

Tiny Machine Learning (TinyML) is an innovative approach that involves embedded systems and machine learning. TinyML aims to bring ML inference to ultra-low-power devices such as under one milliwatt. The advantages of introducing TinyML into MCUs are energy efficiency, low cost, system reliability, and data security. MCUs show the ability to reduce energy consumption even working at a maximum workload level and are powered by batteries. Thus, this permits the MCUs to deploy anywhere without the concern of power supply issues. Besides that, the off-the-shelf MCUs price is inexpensive, and this will allow wide deployment of TinyML. Furthermore, TinyML ensures good system reliability and data security since machine learning is not via cloud computing. Thus, this enables all the data on-premises will be confined at the MCUs themselves.

There is a wide selection of machine learning compression algorithms, including model pruning, parameter quantization, knowledge distillation, layer decomposition, and binarization. Model reduction is one of the main challenges in implementing machine learning into microcontrollers. The objective of model reduction is to reduce the size of the neural and fit into the tight constraints of microcontrollers.

2.5. Pruning

Works by [19] presented a model compression that includes several techniques, including pruning, quantization, and Huffman encoding. The operations are applied successively on the neural network model, resulting in reduced model size [20]. The objective of model pruning is to reduce the number of connections in the neural network model by pruning non-informative weights based on some loss function. Besides that, a study by [21] reported an example of the pruning process. The author trains the network to identify which connections are important, prune redundant connections, and retrain the network to fine-tune the weights of remaining connections. Furthermore, Y. Le Chun et al. [22] presented a weight pruning technique called optimal brain damage (OBD). OBD had improved the network speed significantly while reducing the number of parameters. It is noticed that there is a slight increase in recognition accuracy. On the other hand, H. Hu et al. [23] introduced network trimming techniques that prune the unimportant zero activations neurons. The proposed average percentage of zeros (APoZ) is used for measuring the percentage of zero activations of a neuron after ReLU mapping. The zero activation neurons can be identified easily via APoZ regardless of the weights of connections. L.Theis [24] had presented a simple yet greedy pruning known as Fisher pruning. The author utilized the knowledge distillation [25] and Fisher pruning to obtain runtime-efficient architecture for saliency prediction. The speedup is 10x for the same area under the curve (AUC) performance as a state-of-the-art network on the CAT2000 dataset. The author built on the model DeepGaze II [26], the backbone of DeepGaze II is formed by VGG-19 [27], a deep neural network pre-trained for object recognition. Some modifications are made in DeepGaze II. The Molchanov et al. [28] pruning method was applied for gaze prediction. It was noticed that the regularization of the number of FLOPs gives better results. The number of parameters can be further decreasing via pruning: greedy removal of redundant parameters or feature maps known as Fisher pruning. Fisher pruning aims to remove feature maps or parameters which contribute little to the overall performance of the model. The research paper [20] proposed a deep compression which consist of three stage pipeline including pruning, trained quantization and Huffman coding. These process enables the reduction of storage requirement of neural network by 35x to 49x without affecting the accuracy. The author approaches in pruning the network via learning only the important connections. Then the weights are quantized to enforce weight sharing and apply Huffman coding. The research paper aimed to reduce the storage and energy required to run inference on such networks in order to meet the tight constraints of mobile devices. The total size of AlexNet decreased from 240MB to 6.9MB. This enables the implementation of AlexNet into on-chip SRAM.

2.6. Quantization

A network quantization process is where the network size is reduced further by reducing the number of bits needed to represent each weight. The quantization process involves deciding the number of bits needed to represent weights based on an observed [min, max] range of the weight values. There are other methods to quantize the weight efficiently, such as linear quantization, non-linear (log-based) quantization, and k-means clustering, to even single bit quantization for binary-weight neural networks.

In the research paper [29], the authors proposed a quantization scheme that allows inference to be carried out using integer-only arithmetic. This approach is more efficient than floating-point inference on widely available integer-only hardware. The main contribution of the paper is the introduction of a quantization scheme, quantized inference framework, quantized training framework, and implementation of the frameworks. Quantization scheme is the quantization of both weights and activations as 8-bit integers and a few parameters as 32-bit integers. The quantization scheme in the paper was implemented using integer-only arithmetic during inference and floating-point arithmetic during training. The quantization scheme only uses a single set of quantization parameters for all values within each activations array within each weights array. The author had applied quantized training to ResNets [30] and InceptionV3 [31] on the ImageNet dataset. Integer-only quantization outperforms FGQ [32], which utilizes 2 bits for weight quantization. However, the author's integer-only quantization outperforms INQ [33] in run-time improvements. The author had proposed a quantization scheme that relies only on integer arithmetic to approximate the floating-point computations in a neural network. Training that simulates the effect of quantization assist in the restoration of model accuracy to a near-identical level as the original. The author also evaluated quantization in the context of mobile real-time object detection, comparing the performance of quantized 8-bit float models of Mobile Net SSD [33, 34] on the COCO dataset [36]. All the regular convolutions in the SSD prediction layers have been replaced with separable convolutions. This caused the MobileNets to become more computationally efficient.

2.7. Knowledge Distillation

G. Hinton et al. [25] had further developed the model compression proposed in [37] via a different compression technique, i.e., knowledge distillation. Knowledge distillation is an alternative way to transfer knowledge from a large model into a smaller form for ease of deployment. Models are advised to train to generalize well. However, G. Hinton et al. [25] approaches to enable the training of small models to generalize in the same way as the large model. The generalization of a cumbersome model to a small model is through class probabilities. These class probabilities are generated by the cumbersome model as "soft targets" for the small model training.

The lack of example in the transfer set that is used in training the distilled model will not affect the MNIST distillation works. The improvement of a single big net is caused by learning many specialist nets that can discriminate between classes in a highly confusable cluster.

2.8. Layer Decomposition

Single Value Decomposition (SVD) method [38, 39, 40], decomposes a weight matrix $\mathrm{W}_{(m\times n)}$ into three matrices namely $\mathrm{U}_{(m\times m)}$, $\sum_{(m\times n)}$, and $V^T_{(n\times n)}$. The dimension of the matrices can be reduced by ignoring relatively small singular values. In [38], the author had shown how the SVD can be used to compute reliably a number of the basic geometric objects of linear algebra. SVD plays an important role in analyzing square, invertible matrices its full power is realized in the analysis of nonsquare, possibly rank-deficient matrices which arise. Besides that, in [39], SVD was applied to the weight matrices in DNN and restructured the model based on the inherent sparseness of the original matrices. DNN Hidden-Markov-Model (CD-DNN-HMM) [41, 42], the conventional Gaussian Mixture Model (GMM) is replaced by a DNN to evaluate the senone log-likelihood. The implemented CD-DNN-HMMS are initialized from traditional CD-GMM-HMMs. It is crucial for rapid computation and small memory usage in order to obtain a real-time execution. By applying SVD on the weight matrices in DNN, two small matrices will be obtained. There will be no accuracy loss if the restructuring of the DNN model is done via modest parameter reduction. However, if the number of parameters of DNN is greatly reduced, the model can undergo fine-tuning with a new structure to obtain the original accuracy.

In [40], the author had proposed a fast training methodology for learning of Deep Neural Networks (DNNs) via SVD. Besides that, the author utilized unconventional Back Propagation (BP) algorithm to train the models restructured by SVD, which reduces the time complexity than the conventional BP algorithm. Furthermore, the research paper [43] introduced a software accelerator for deep learning models to run on mobile hardware, DeepX. DeepX strengthen the advantages via two inference-time resource control algorithm such as Runtime Later Compression (RLC) and Deep Architecture Decomposition (DAD). Extension of model compression principles in the inference phase can be improved by introducing RLC. Since RLC provides runtime control of memory and computation consumed. The RLC is designed to address significant obstacles. These obstacles include low overhead operation suitable for runtime use, the requirement of retraining, and testing are needed for local test datasets to assess the impact of model architecture changes. A decomposition plan is created, and the unit block of the architecture is identified by DAD. This decomposition plan can maximize resource utilization and achieve user performance goals. The challenges overcome by DAD include potentially prohibitive search space, inference and considering hardware heterogeneity, decomposition and recomposition overhead. Combination of both RLC and DAD enables the DeepX to performs inference across a standard deep learning model with an innovative use of resources.

2.9. Binarization

Binarization is a 1-bit quantization where data only have two possible values, either -1 or +1 [44]. Binarized Neural Networks (BNNs) are deep neural networks that were introduced by Courbariaux et al. [45], which utilizes binary values instead of full precision values in activations and weights [44]. Since BNNs use binary values, thus the computation execution can be done via bitwise operations, which reduce the execution time. Existing methodologies in BNNs such as optimizer, gradient, and weight clipping, batch normalization and, pooling and learning rate. BinaryConnect uses ADAM [46] for CIFAR-10 and vanilla SGD for MNIST. DoReFa-Net and XNOR-Net use ADAM in their experiments, and ABC-Net [47] uses SGD with momentum. The experiment was conducted up to 500 epochs with fine-tuned hyper-parameters for best results. The author noticed that in the early stage of binary models training, more averaging is needed for the optimizer to proceed in the presence of binarization operation. However, in the late stages of training, the exploration power of the optimizer is increased by noisier sources. Bengio et al. [48] introduced a distinct vanilla STE as compared to the common STE variant BinaryConnect, XNOR-Net, and other binary models. The weights can be maintained within the range after clipping of weight via BinaryConnect after gradient updates. Batch normalization (BN) utilizes mini-batch statistics during training, but at inference time, the model is classified as a single data point. Reduction in momentum rate of BN can neutralize the long training effect. Although the effect is tiny, it is consistent. Besides that, Krsihnamoorthi [49] had also reported that better performance of a model can be achieved by handling differently in Batch normalization during the quantized model training process. An equivalent non-binary can be trained with the absence of binarization operations. This is useful since the model is readily available in many cases. This pre-trained model can be utilized for the initialization of values for full-precision proxies in the binary model. Furthermore, the model can be used for STE training and gradient clipping. In [50], the author reported that the work can operate well in terms of accuracy and converges even faster as compared to ResNet-18 [51] and VGG-10 architecture than training the binary models end-to-end. Moreover, a similar result was reported in [52].

To train the binary model with high productivity, there are some recommendations listed by the [50], such as utilizing ADAM for objective optimization, avoiding early stopping, separation of training into two stages, gradient removal and weigh clipping in the first stage, and reduction of averaging rate in the second stage of Batch Normalisation layers. Courbariaux et al. [45] introduced a method to train BNNs via binary weights and activations at run-time. The author uses two different binarization functions based on [55]. These two binarization functions are deterministic and stochastic. Several experiments were conducted based on a different framework Torch7 [61] and Theano [62, 63]. These frameworks Theano and Torch7 were used to test the MLP on MNIST and ConvNet on CIFAR-10. The result obtained is shown in Table 1. BNNs can reduce the memory size and access in the forward pass either at run-time or train time. It allows the replacement of arithmetic operation with a bit-wise operation that may lead to an increase in power efficiency.

Table 1. Classification Test Error Rates of DNNs Trained on MNIST (MLP Architecture without Unsupervised Pretraining), CIFAR-10 (without Data Augmentation), and SVHN [45].

Data set	MNIST	SVHN	CIFAR-10
Binarized activations+weights, during training and test			
BNN (Torch7)	1.40%	2.53%	10.15%
BNN (Theano)	0.96%	2.80%	11.40%
Committee Machines' Array [53]	1.35%	-	-
Binarized weights, during training and test			
BinaryConnect [54]	1.29±0.08%	2.30%	9.90%
Binarized activations+ weight, during test			
Expectation Back Propagation [55]	2.2±0.1%	-	-
Bitwise DNNs [56]	1.33%	-	-
Ternary weights, binary activations, during test			
Hwang & Sung [57]	1.45%	-	-
No binarization (standard results)			
Maxout Networks [58]	0.94%	2.47%	11.38%
Network in Network [59]	-	2.35%	10.41%
Gated pooling [60]	-	1.69%	7.62%

One of the well-known TinyML frameworks is TensorFlow Lite developed by Google. TensorFlow Lite can adapt the TensorFlow model and intended to enable them to run in a mobile and embedded device. This framework consists of two components which are converter and interpreter. The converter is used to port the TensorFlow models to optimized code that can be executed in constrained platforms. Besides that, the interpreter runs the code generated by the converter. Therefore, platforms such as MCUs, smartphones, and embedded Linux can execute the optimized model.

Moreover, TensorFlow Lite Micro (TFLM) is an open-source ML inference framework for running deep-learning models on embedded systems [64]. Robert et al. [64] introduced the model-export workflow from TensorFlow training environment to TensorFlow Lite Exporter and finally to TensorFlow Lite Flatbutter File, which enables TFLM to load the inference model.

3. Conclusion

Anomaly detection is an unsupervised learning algorithm. This algorithm aims to identify the patterns in data that do not follow the expected behavior. Therefore, introducing anomaly detection into TinyML can effectively reduce the broke down of the system. However, in order to meet the tight constraint of the embedded system, model optimization is required. There are several types of model optimization, such as pruning, quantization, knowledge distillation, layer decomposition, and binarization. There is no best model optimization but only suitable model optimization for each application. Industry revolution 5.0 emphasizes cooperation between humans and machines. Therefore, TinyML can ensure the working area for humans is safe by alarming everyone if the equipment is misbehaving.

Acknowledgments

This study was supported by Graduate Research Assistant (GRA). Grant no: R.J130000.7851.5F292.

References

[1] E. Zio, "Reliability engineering: Old problems and new challenges," *Reliab. Eng. Syst. Saf.*, vol. 94, no. 2, pp. 125–141, 2009, doi: 10.1016/j.ress.2008.06.002.

[2] A. Azadeh, S. M. Asadzadeh, N. Salehi, and M. Firoozi, "Condition-based maintenance effectiveness for series–parallel power generation system—A combined Markovian simulation model," *Reliab. Eng. Syst. Saf.*, vol. 142, pp. 357–368, 2015, doi: https://doi.org/10.1016/j.ress.2015.04.009.

[3] J. Lee, F. Wu, W. Zhao, M. Ghaffari, L. Liao, and D. Siegel, "Prognostics and health management design for rotary machinery systems—Reviews, methodology and applications," *Mech. Syst. Signal Process.*, vol. 42, no. 1, pp. 314–334, 2014, doi: https://doi.org/10.1016/j.ymssp.2013.06.004.

[4] J. Wang, G. Wen, S. Yang, and Y. Liu, "Remaining Useful Life Estimation in Prognostics Using Deep Bidirectional LSTM Neural Network," *Proc. - 2018 Progn. Syst. Heal. Manag. Conf. PHM-Chongqing 2018*, pp. 1037–1042, 2019, doi: 10.1109/PHM-Chongqing.2018.00184.

[5] R. Sanchez-Iborra and A. F. Skarmeta, "TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities," *IEEE Circuits Syst. Mag.*, vol. 20, no. 3, pp. 4–18, 2020, doi: 10.1109/MCAS.2020.3005467.

[6] Z. Xu and J. H. Saleh, "Machine learning for reliability engineering and safety applications: Review of current status and future opportunities," *Reliab. Eng. Syst. Saf.*, vol. 211, no. December 2020, p. 107530, 2021, doi: 10.1016/j.ress.2021.107530.

[7] D. K. Bangotra, Y. Singh, and A. Selwal, "Machine learning in wireless sensor networks: Challenges and opportunities," *PDGC 2018 - 2018 5th Int. Conf. Parallel, Distrib. Grid Comput.*, pp. 534–539, 2018, doi: 10.1109/PDGC.2018.8745845.

[8] S. Kotsiantis, "Supervised Machine Learning: A Review of Classification Techniques," *Inform.*, vol. 31, Oct. 2007.

[9] C. R. Banbury *et al.*, "Benchmarking TinyML Systems: Challenges and Direction," 2020, [Online]. Available: http://arxiv.org/abs/2003.04821.

[10] J. Novakovic, A. Veljovi, S. Iiic, Z. Papic, and M. Tomovic, "Evaluation of Classification Models in Machine Learning," *Theory Appl. Math. Comput. Sci.*, vol. 7, no. 1, pp. 39–46, 2017, [Online]. Available: https://uav.ro/applications/se/journal/index.php/TAMCS/article/view/158.

[11] N. Dhanachandra, K. Manglem, and Y. J. Chanu, "Image Segmentation Using K-means Clustering Algorithm and Subtractive Clustering Algorithm," *Procedia Comput. Sci.*, vol. 54, pp. 764–771, 2015, doi: 10.1016/j.procs.2015.06.090.

[12] N. R. Prasad, S. Almanza-Garcia, and T. T. Lu, "Anomaly detection," *Comput. Mater. Contin.*, vol. 14, no. 1, pp. 1–22, 2009, doi: 10.1145/1541880.1541882.

[13] P. E. Novac *et al.*, "Toward unsupervised Human Activity Recognition on Microcontroller Units," *Proc. - Euromicro Conf. Digit. Syst. Des. DSD 2020*, pp. 542–550, 2020, doi: 10.1109/DSD51259.2020.00090.

[14] R. Wehrens, "2.28 - Data Mapping: Linear Methods versus Nonlinear Techniques," S. D. Brown, R. Tauler, and B. B. T.-C. C. Walczak, Eds. Oxford: Elsevier, 2009, pp. 619–633.

[15] G. Kostopoulos, S. Karlos, S. Kotsiantis, and O. Ragos, "Semi-supervised regression: A recent review," *J. Intell. Fuzzy Syst.*, vol. 35, no. 2, pp. 1483–1500, 2018, doi: 10.3233/JIFS-169689.

[16] S. Shaaban, M. Farouk, and H. Ahmed, "Semi-supervised Classification: Cluster and Label Approach using Particle Swarm Optimization," *Int. J. Comput. Appl.*, vol. 160, no. 3, pp. 39–44, 2017, doi: 10.5120/ijca2017913013.

[17] Y. Chong, Y. Ding, Q. Yan, and S. Pan, "Graph-based semi-supervised learning: A review," *Neurocomputing*, vol. 408, pp. 216–230, 2020, doi: 10.1016/j.neucom.2019.12.130.

[18] S. S. Sawant and M. Prabukumar, "A review on graph-based semi-supervised learning methods for hyperspectral image classification," *Egypt. J. Remote Sens. Sp. Sci.*, vol. 23, no. 2, pp. 243–248, 2020, doi: 10.1016/j.ejrs.2018.11.001.

[19] S. Soro, "TinyML for Ubiquitous Edge AI," no. 20, 2021, [Online]. Available: http://arxiv.org/abs/2102.01255.

[20] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, pp. 1–14, 2016.

[21] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *Adv. Neural Inf. Process. Syst.*, vol. 2015-Janua, pp. 1135–1143, 2015.

[22] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal Brain Damage (Pruning)," *Adv. Neural Inf. Process. Syst.*, pp. 598–605, 1990.

[23] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures," 2016, [Online]. Available: http://arxiv.org/abs/1607.03250.

[24] L. Theis, I. Korshunova, A. Tejani, and F. Huszár, “Faster gaze prediction with dense networks and Fisher pruning,” 2018, [Online]. Available: http://arxiv.org/abs/1801.05787.

[25] G. Hinton, O. Vinyals, and J. Dean, “Distilling the Knowledge in a Neural Network,” pp. 1–9, 2015, [Online]. Available: http://arxiv.org/abs/1503.02531.

[26] M. Kümmerer, T. S. A. Wallis, and M. Bethge, “DeepGaze II: Reading fixations from deep features trained on object recognition,” pp. 1–16, 2016, [Online]. Available: http://arxiv.org/abs/1610.01563.

[27] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–14, 2015.

[28] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning convolutional neural networks for resource efficient inference,” *5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc.*, no. 2015, pp. 1–17, 2017.

[29] B. Jacob *et al.*, “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 2704–2713, 2018, doi: 10.1109/CVPR.2018.00286.

[30] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 770–778, 2016, doi: 10.1109/CVPR.2016.90.

[31] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 2818–2826, 2016, doi: 10.1109/CVPR.2016.308.

[32] N. Mellempudi, A. Kundu, D. Mudigere, D. Das, B. Kaul, and P. Dubey, “Ternary Neural Networks with Fine-Grained Quantization,” 2017, [Online]. Available: http://arxiv.org/abs/1705.01462.

[33] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, “Incremental network quantization: Towards lossless cnns with low-precision weights,” *5th Int. Conf. Learn. Represent. ICLR 2017 - Conf. Track Proc.*, pp. 1–14, 2017.

[34] A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” 2017, [Online]. Available: http://arxiv.org/abs/1704.04861.

[35] W. Liu *et al.*, “SSD: Single shot multibox detector,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, 2016, doi: 10.1007/978-3-319-46448-0_2.

[36] T. Y. Lin *et al.*, “Microsoft COCO: Common objects in context,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8693 LNCS, no. PART 5, pp. 740–755, 2014, doi: 10.1007/978-3-319-10602-1_48.

[37] C. Bucilă, R. Caruana, and A. Niculescu-Mizil, “Model compression,” *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, vol. 2006, no. August 2006, pp. 535–541, 2006, doi: 10.1145/1150402.1150464.

[38] V. C. Klema and A. J. Laub, “The Singular Value Decomposition: Its Computation and Some Applications,” *IEEE Trans. Automat. Contr.*, vol. 25, no. 2, pp. 164–176, 1980, doi: 10.1109/TAC.1980.1102314.

[39] J. Xue, J. Li, and Y. Gong, “Restructuring of deep neural network acoustic models with singular value decomposition,” *Proc. Annu. Conf. Int. Speech Commun. Assoc. INTERSPEECH*, pp. 2365–2369, 2013.

[40] C. Cai, D. Ke, Y. Xu, and K. Su, “Fast learning of deep neural networks via singular value decomposition,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8862, pp. 820–826, 2014, doi: 10.1007/978-3-319-13560-1.

[41] D. Yu, P. Seide, and G. Li, “Conversational speech transcription using context-dependent deep neural networks,” *Proc. 29th Int. Conf. Mach. Learn. ICML 2012*, vol. 1, pp. 4–5, 2012.

[42] D. Yu, L. Deng, and G. E. Dahl, “Roles of Pre-Training and Fine-Tuning in Context-Dependent DBN-HMMs for Real-World Speech Recognition,” *Nips ’10*, p. 8, 2010.

[43] N. D. Lane *et al.*, “DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices,” *2016 15th ACM/IEEE Int. Conf. Inf. Process. Sens. Networks, IPSN 2016 - Proc.*, 2016, doi: 10.1109/IPSN.2016.7460664.

[44] T. Simons and D. J. Lee, “A review of binarized neural networks,” *Electron.*, vol. 8, no. 6, 2019, doi: 10.3390/electronics8060661.

[45] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1,” 2016, [Online]. Available: http://arxiv.org/abs/1602.02830.

[46] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–15, 2015.

[47] X. Lin, C. Zhao, and W. Pan, “Towards accurate binary convolutional neural network,” *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, no. 3, pp. 345–353, 2017.

[48] Y. Bengio, N. Léonard, and A. Courville, “Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation,” pp. 1–12, 2013, [Online]. Available: http://arxiv.org/abs/1308.3432.

[49] R. Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A whitepaper,” 2018, [Online]. Available: http://arxiv.org/abs/1806.08342.

[50] M. Alizadeh, J. Fernández-Marqués, N. D. Lane, and Y. Gal, “An empirical study of binary neural networks’ optimisation,” *7th Int. Conf. Learn. Represent. ICLR 2019*, no. 2016, pp. 1–11, 2019.

[51] S. Lu, M. Seo, and R. Lysecky, “Timing-based anomaly detection in embedded systems,” *20th Asia South Pacific Des. Autom. Conf. ASP-DAC 2015*, pp. 809–814, 2015, doi: 10.1109/ASPDAC.2015.7059110.

[52] A. Mishra and D. Marr, “Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy,” *6th Int. Conf. Learn. Represent. ICLR 2018 - Conf. Track Proc.*, pp. 1–15, 2018.

[53] C. Baldassi, A. Ingrosso, C. Lucibello, L. Saglietti, and R. Zecchina, “Subdominant Dense Clusters Allow for Simple Learning and High Computational Performance in Neural Networks with Discrete Synapses,” *Phys. Rev. Lett.*, vol. 115, no. 12, pp. 1–11, 2015, doi: 10.1103/PhysRevLett.115.128101.

[54] M. Courbariaux, Y. Bengio, and J. P. David, “Binaryconnect: Training deep neural networks with binary weights during propagations,” *Adv. Neural Inf. Process. Syst.*, vol. 2015-Janua, pp. 3123–3131, 2015.

[55] Z. Cheng, D. Soudry, Z. Mao, and Z. Lan, “Training Binary Multilayer Neural Networks for Image Classification using Expectation Backpropagation,” no. 2012, pp. 1–8, 2015, [Online]. Available: http://arxiv.org/abs/1503.03562.

[56] M. Kim and P. Smaragdis, “Bitwise Neural Networks,” vol. 37, 2016, [Online]. Available:

http://arxiv.org/abs/1601.06071.
[57] K. Hwang and W. Sung, “Fixed-point feedforward deep neural network design using weights +1, 0, and -1,” *IEEE Work. Signal Process. Syst. SiPS Des. Implement.*, 2014, doi: 10.1109/SiPS.2014.6986082.
[58] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, “Maxout networks,” *30th Int. Conf. Mach. Learn. ICML 2013*, no. PART 3, pp. 2356–2364, 2013.
[59] M. Lin, Q. Chen, and S. Yan, “Network in network,” *2nd Int. Conf. Learn. Represent. ICLR 2014 - Conf. Track Proc.*, pp. 1–10, 2014.
[60] C. Y. Lee, P. W. Gallagher, and Z. Tu, “Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree,” *Proc. 19th Int. Conf. Artif. Intell. Stat. AISTATS 2016*, pp. 464–472, 2016.
[61] R. Collobert, K. Kavukcuoglu, and C. Farabet, “Torch7: A matlab-like environment for machine learning,” *BigLearn, NIPS Work.*, pp. 1–6, 2011, [Online]. Available: http://infoscience.epfl.ch/record/192376/files/Collobert_NIPSWORKSHOP_2011.pdf.
[62] F. Bastien *et al.*, “Theano: new features and speed improvements,” pp. 1–10, 2012, [Online]. Available: http://arxiv.org/abs/1211.5590.
[63] J. Bergstra *et al.*, “Theano: a {CPU} and {GPU} Math Expression Compiler,” *Proc. Python Sci. Comput. Conf.*, no. Scipy, pp. 1–7, 2010.
[64] R. David *et al.*, “TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems,” 2020, [Online]. Available: http://arxiv.org/abs/2010.08678.