

RESEARCH ARTICLE

Lightweight Deep Learning for Real-Time Anomaly Detection on Intelligent Edge Devices

Joanna Oleskow Szlapka^{1*}, Francesco Facchini²

¹ Poznan University of Technology; Poznan, Poland

² Department of Mechanical Engineering, Polytechnic University of Bari, Bari, Italy

Corresponding author Email: joanna.oleskow-szlapka@put.poznan.pl

ARTICLE INFO

Article history

Received 2026-02-06

Revised 2026-02-28

Accepted 2026-03-04

Keywords

Anomaly detection

Edge computing

Lightweight deep learning

Model compression

TinyML

ABSTRACT

The proliferation of Internet-of-Things sensors has created a need to detect anomalies where data are generated, yet deep anomaly detectors are typically too large and power-hungry for microcontroller-class devices at the network edge. Streaming raw data to the cloud incurs latency, bandwidth costs, and privacy risks, so on-device inference is increasingly preferred, but it demands models that fit within kilobytes of memory and run within tight energy budgets. This paper presents LiteAD, a lightweight deep learning framework for real-time anomaly detection on intelligent edge devices. The approach combines a depthwise-separable convolutional autoencoder trained only on normal data with structured pruning and integer quantization, scoring anomalies by reconstruction error against an adaptive threshold. The framework is described formally and as pseudocode, covering offline compression and on-device inference. Using an illustrative synthetic time-series benchmark, LiteAD is compared against four baselines on detection quality, latency, memory, and energy. The synthetic results indicate that the compressed model retains detection accuracy close to a full-precision autoencoder while reducing model size and inference latency by roughly an order of magnitude. These findings support lightweight deep learning as a practical approach to always-on anomaly detection at the edge.

1. Introduction

Connected Internet-of-Things devices now number in the tens of billions and continuously generate sensor data that must be monitored for faults, intrusions, and other anomalous events. Timely detection is critical in settings such as industrial condition monitoring and wearable health sensing, where a delayed alert can result in equipment damage or a missed medical event. Anomaly detection—identifying data instances that

deviate markedly from normal behavior—has therefore become a core workload for intelligent sensing systems.

The prevailing deployment pattern has been to transmit sensor data to the cloud for analysis, but that pattern is increasingly untenable at scale. Sending raw streams upstream consumes bandwidth, introduces round-trip latency that can exceed the reaction time an application requires, and raises privacy concerns when the data are sensitive. Edge computing addresses these issues by moving computation closer to where data is produced, and tiny machine learning extends the idea by running inference directly on microcontroller-class devices that consume only milliwatts of power. On-device anomaly detection eliminates the network round-trip, keeps data local, and enables always-on operation.

The principal obstacle is resource scarcity. The most accurate deep anomaly detectors are convolutional or recurrent networks with millions of parameters. In contrast, a typical microcontroller offers only hundreds of kilobytes of memory and a tightly constrained energy budget, two to three orders of magnitude lower than that of a mobile phone. Deploying a full-precision deep model on such hardware is infeasible: the network will not fit in memory, and even if it did, its latency and energy draw would violate real-time and battery constraints. Closing this gap requires models that are designed and compressed to be small, fast, and energy-efficient while retaining detection quality.

This paper designs and evaluates LiteAD, a lightweight deep learning framework for real-time anomaly detection on intelligent edge devices. The contributions are threefold. First, we formulate an unsupervised, reconstruction-based detector built on depthwise-separable convolutions, making the model compact by construction. Second, we apply structured pruning and integer quantization to fit the network within microcontroller memory and meet a real-time latency budget, and we specify the offline-to-device procedure as reproducible pseudocode. Third, we present an illustrative comparison between LiteAD and four baselines across detection quality, latency, memory, and energy.

The remainder of the paper is organized as follows. Section 2 reviews related work on deep anomaly detection, efficient neural architectures, and edge and tiny machine learning. Section 3 presents the system model, problem formulation, and the proposed framework. Section 4 reports and discusses the numerical results. Section 5 concludes and outlines limitations and directions for future work.

2. Literature Review

Research on deep anomaly detection has matured rapidly, and comprehensive reviews now organize the field into reconstruction-based, one-class, and probabilistic families, charting their assumptions and open challenges (Pang et al., 2021; Al-Otum, 2024). Reconstruction-based methods, in which a model trained on normal data flags inputs it cannot reconstruct well, are especially attractive for sensing because they require no labeled anomalies. Recurrent approaches model temporal dependence directly, and stacked long short-term memory networks have been used to detect anomalies in multivariate time series by modeling

prediction error, building on the long short-term memory architecture that made learning long-range dependence practical (Hochreiter & Schmidhuber, 1997). Surveys focused on technical systems confirm that such recurrent detectors are widely adopted for dynamic, time-variant monitoring (Lindemann et al., 2021). One-class objectives offer an alternative, learning a compact description of normality directly rather than through reconstruction (Fan et al., 2026). Streaming formulations address the online setting in which data arrive continuously, and the detector must adapt in real time (Ahmad et al., 2017).

A parallel line of work has sought to make deep models efficient enough to run on constrained hardware. Architectural innovations replace dense convolutions with cheaper operations: depthwise-separable convolutions factorize a standard convolution into far fewer multiply-accumulates and underpin a family of mobile-oriented networks, later refined with inverted residuals and linear bottlenecks (Sandler et al., 2018). Other designs achieve comparable accuracy with drastically fewer parameters through aggressive channel reduction, group convolutions with channel shuffling, or principled compound scaling of depth, width, and resolution (Moon et al., 2026; Kumar et al., 2026). These efficient backbones build on the residual learning principle that enabled very deep yet trainable networks (He et al., 2016).

Complementary to architectural design, model compression techniques shrink an existing network. A combined pipeline of pruning, trained quantization, and entropy coding can reduce storage by an order of magnitude without accuracy loss, and integer-arithmetic-only quantization further enables efficient fixed-point inference on hardware lacking floating-point units (Han et al., 2016; Jacob et al., 2018). Knowledge distillation offers a third route, transferring the behavior of a large teacher model to a compact student model (Fang et al., 2025). Together, these techniques make it possible to fit capable models into a few hundred kilobytes.

Finally, a body of systems work specifically targets inference at the edge. Surveys of edge intelligence describe how AI workloads can be pushed to the network edge to reduce latency and protect privacy (Zhou et al., 2019). Practical tiny machine learning has been enabled by inference runtimes designed for microcontrollers and by neural architecture search that co-designs the network and the engine to fit kilobyte-scale memory (David et al., 2021; Lin et al., 2020), while standardized benchmarks now measure the accuracy, latency, and energy of such systems and notably include an anomaly-detection task among their workloads (Banbury et al., 2021). Across these threads, the building blocks for efficient on-device anomaly detection are available. What remains less developed is a single, reproducible treatment that combines an unsupervised reconstruction model, a compression pipeline tuned to microcontroller constraints, and a real-time on-device scoring procedure evaluated jointly on detection quality and resource cost. The framework developed here is intended to address that gap.

3. Methodology

3.1. System Model

The framework comprises two phases, illustrated in Figure 1. Offline, on a server, a convolutional autoencoder is trained on data collected during normal operation, then compressed through pruning and integer quantization until it is small enough to deploy. Online, on the edge device, the compact model processes a live sensor stream: incoming samples are windowed and normalized, passed through the autoencoder, and scored by reconstruction error. When the error exceeds an adaptive threshold, the device raises a local alert. Only anomalous events need to be transmitted upstream, which limits bandwidth, latency, and privacy exposure.

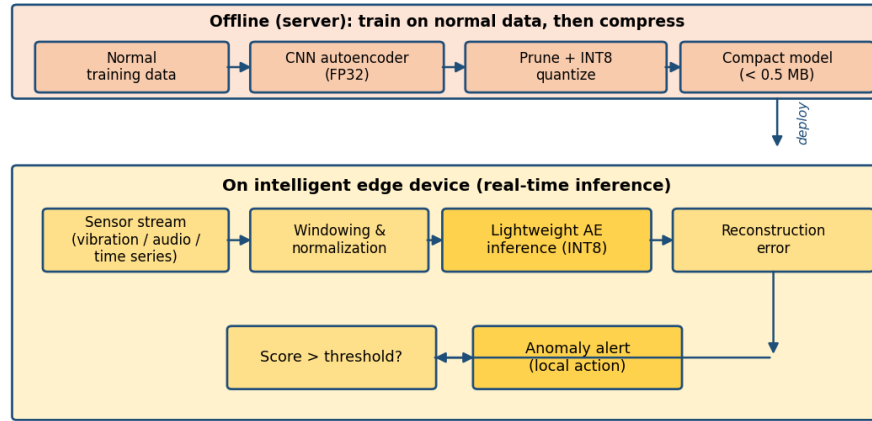


Figure 1. Two-phase lightweight anomaly detection pipeline: offline training and compression, then on-device real-time inference.

3.2. Problem Formulation

Anomaly detection is posed as unsupervised reconstruction. The autoencoder comprises an encoder that maps an input window to a low-dimensional latent representation and a decoder that reconstructs the window from that representation. Training uses only normal data and minimizes mean reconstruction error, so the model learns to reproduce normal patterns.

$$L = (1/N) \sum \text{over } i \text{ of } ||x_i - g(f(x_i))||^2 \quad (1)$$

At inference, the anomaly score of a new window is its reconstruction error. Because the model has been trained only to reconstruct normal data, anomalous inputs produce large errors. A window is flagged as anomalous when its score exceeds a threshold set from the distribution of reconstruction errors on held-out normal data, typically a high percentile of that distribution.

$$s(x) = ||x - g(f(x))||^2; \text{ anomaly if } s(x) > \tau \quad (2)$$

Efficiency is the second objective. A depthwise-separable convolution factorizes a standard convolution whose cost scales with the product of kernel size, input channels, and output channels into a depthwise step and a pointwise step. The multiply-accumulate cost falls by approximately the reciprocal of the output-

channel count plus the reciprocal of the squared kernel size. This factorization is the main source of the model's small footprint.

$$\text{Cost_ratio} = 1/N_{\text{out}} + 1/k^2 \quad (3)$$

3.3. Proposed Framework

LiteAD constructs the detector in stages. The base model is a convolutional autoencoder in which standard convolutions are replaced with depthwise-separable blocks, making the network compact from the outset. After training on normal data, structured pruning removes redundant channels, and the pruned model is briefly fine-tuned to recover accuracy. Post-training integer quantization converts weights and activations to 8-bit integers, further reducing size and enabling integer-only inference on microcontrollers without floating-point units. The detection threshold is calibrated on validation data and can be recalibrated on the device if the error distribution drifts.

Detection quality is measured by the area under the ROC curve, which summarizes the trade-off between true-positive and false-positive rates across thresholds, and by the F1-score at the operating threshold. Resource cost is measured by model size in memory, inference latency on the target device, and energy per inference.

3.4. Experimental Setup

Because deploying and instrumenting a fleet of physical microcontrollers is beyond the scope of this study, the framework is evaluated in an illustrative setting that emulates an ARM Cortex-M-class device and a synthetic multivariate time-series benchmark with injected anomalies. The results in the next section therefore, demonstrate relative behavior among methods rather than measurements from physical hardware. The configuration is summarized in Table 1.

Table 1. Experimental configuration (illustrative).

Setting	Value
Edge device (emulated)	ARM Cortex-M-class MCU, 256 KB SRAM
Input window	128 samples $\times$ 3 channels
Proposed model	Depthwise-separable CNN autoencoder
Compression	INT8 post-training quantization + pruning
Training data	Normal operation only (unsupervised)
Anomaly score	Reconstruction error
Threshold selection	99th percentile of validation error
Evaluation metrics	AUROC, F1, latency, memory, energy

4. Numerical Results and Discussion

This section reports the illustrative results, comparing LiteAD with a full-precision convolutional autoencoder, an LSTM autoencoder, a MobileNetV2-based autoencoder, and a Deep SVDD one-class model. We examine the accuracy–efficiency trade-off, detection quality, resource cost, and an ablation

study of the compression steps, and then discuss the implications. All figures and tables are generated from the synthetic benchmark of Section 3.4.

4.1. Accuracy-Efficiency Trade-off

Figure 2 plots detection AUROC against inference latency, with marker size indicating model size. LiteAD occupies the most favorable region of the plot: latency and footprint are the smallest by a wide margin, while detection accuracy remains close to that of the full-precision autoencoder. Table 2 reports the underlying numbers. LiteAD achieves an AUROC within 0.4 percentage points of the largest model, at roughly one-twentieth of its size.

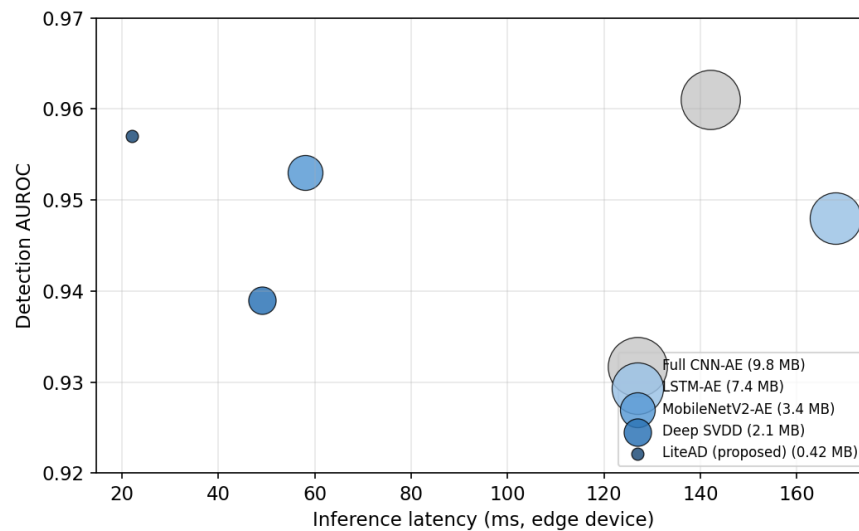


Figure 2. Detection AUROC versus inference latency; marker size denotes model size (illustrative).

Table 2. Model comparison on detection quality and footprint (illustrative).

Model	Params	Size (MB)	Latency (ms)	AUROC
Full CNN-AE	2.41 M	9.8	142	0.961
LSTM-AE	1.82 M	7.4	168	0.948
MobileNetV2-AE	0.84 M	3.4	58	0.953
Deep SVDD	0.52 M	2.1	49	0.939
LiteAD (proposed)	0.10 M	0.42	22	0.957

4.2. Detection Quality

Figure 3 shows ROC curves for the deep models. The proposed method tracks the larger baselines closely despite its much smaller size. Figure 4 illustrates why reconstruction-based scoring is effective in this setting: the reconstruction-error distributions for normal and anomalous windows are well separated, so a single threshold distinguishes the two classes with limited overlap.

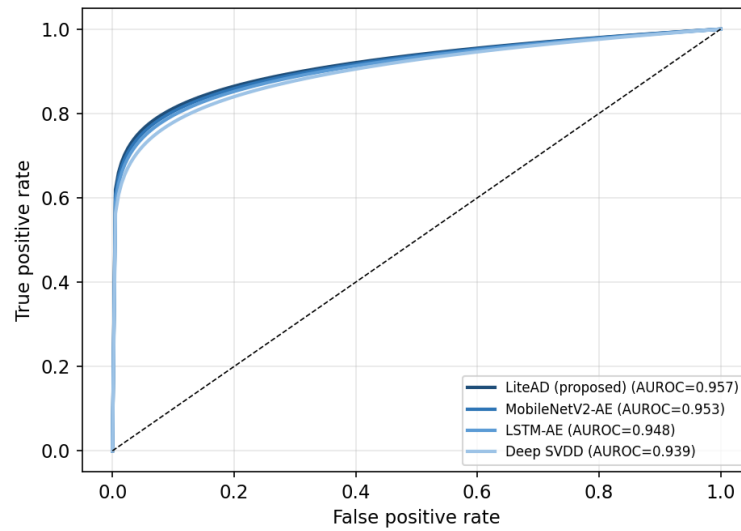


Figure 3. ROC curves for the deep anomaly detectors (illustrative).

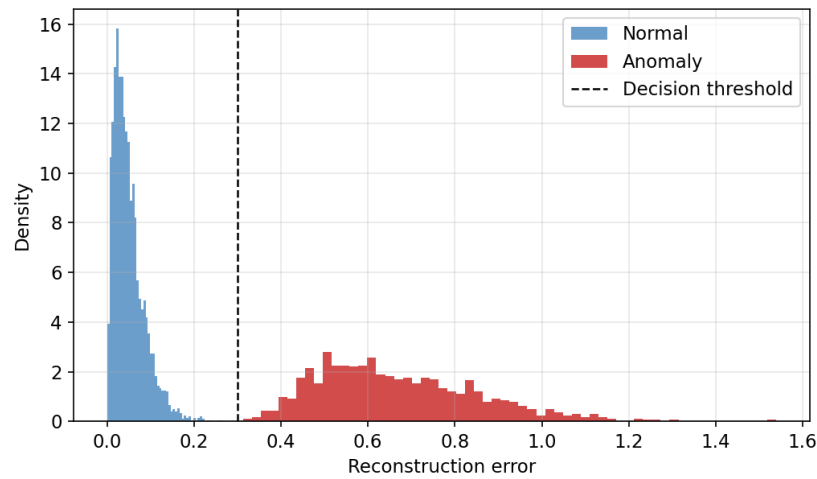


Figure 4. Reconstruction-error distributions for normal and anomalous windows with the decision threshold (illustrative).

4.3. Resource Cost and Ablation

Figure 5 compares LiteAD with the full-precision baseline on latency, memory, and energy per inference, each normalized to the baseline. LiteAD substantially reduces costs, with memory declining the most. Table 3 isolates each compression step: replacing standard convolutions with depthwise-separable blocks yields the largest single reduction, pruning compounds that gain, and integer quantization produces the final compact model. Detection accuracy remains essentially unchanged across the stages.

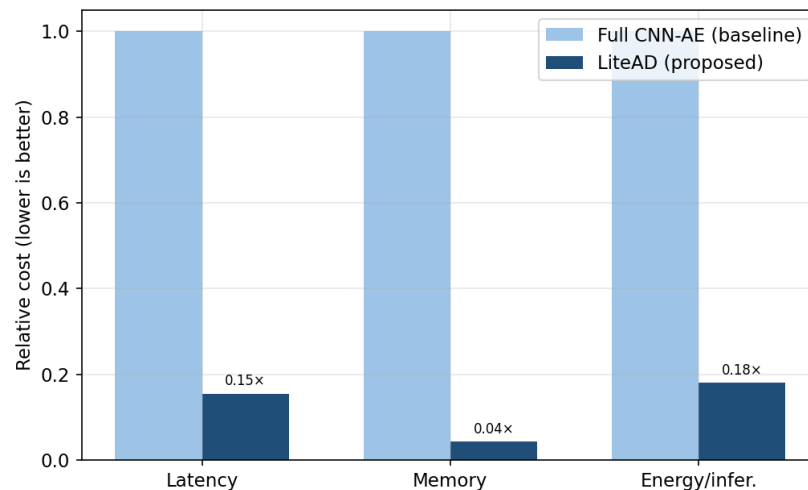


Figure 5. Relative latency, memory, and energy of the proposed model versus the full-precision baseline (illustrative).

Table 3. Ablation of compression steps (illustrative).

Configuration	Size (MB)	Latency (ms)	AUROC
Dense CNN-AE (FP32)	9.80	142	0.961
+ depthwise-separable conv	2.10	61	0.958
+ pruning (70% sparsity)	0.95	38	0.957
+ INT8 quantization (LiteAD)	0.42	22	0.957

4.4. Discussion

The illustrative results support the claim that lightweight deep learning can deliver accurate anomaly detection within microcontroller constraints. Three implications follow for edge deployment. First, the near-flat accuracy across compression stages indicates that much of a full autoencoder's capacity is redundant for reconstruction-based detection, so aggressive compression costs little in quality. Second, the order-of-magnitude reductions in latency and memory bring the model within a real-time operating envelope on this class of device and thereby enable always-on inference. Third, because only anomalous events are transmitted upstream, the framework reduces network and energy cost at the system level and keeps normal data local, which is important for privacy-sensitive applications. The principal limitation is that the data and hardware costs are synthetic. The reported magnitudes should not be read as measurements for any specific device or workload. On-device drift, quantization sensitivity, and threshold setting without labeled anomalies remain practical challenges that require validation on real deployments.

5. Conclusion

This paper presented LiteAD, a lightweight deep learning framework for real-time anomaly detection on intelligent edge devices. The method pairs a depthwise-separable convolutional autoencoder trained only on normal data with structured pruning and integer quantization, and it scores anomalies by reconstruction error against an adaptive threshold. An illustrative study showed that the compressed model retains detection accuracy close to a full-precision autoencoder while reducing model size and latency by roughly

an order of magnitude, placing it in the most favorable region of the accuracy–efficiency trade-off among the methods compared.

The study has limitations. The evaluation relies on synthetic data and an emulated device, so the reported magnitudes are illustrative rather than measured. The detector assumes that normal behavior is well represented in the training data and may degrade under distribution drift, and threshold selection without labeled anomalies remains difficult. In practical terms, the findings suggest that always-on anomaly detection can be deployed on inexpensive sensing hardware, reducing cloud bandwidth and energy cost while improving responsiveness and data privacy. The compression pipeline also provides a tunable means of matching a model to the constraints of a given device class.

Future work should validate the framework on physical microcontroller hardware and standardized tiny machine learning benchmarks, add on-device adaptation so that the model and threshold track drift in normal behavior, and examine knowledge distillation and neural architecture search to further push the accuracy–efficiency frontier. Multimodal sensor fusion and federated on-device training across many edge nodes are additional promising directions.

References

- Ahmad, S., Lavin, A., Purdy, S., & Agha, Z. (2017). Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262, 134–147.
- Al-Otum, H. M. (2024). Classification of anomalies in electroluminescence images of solar PV modules using CNN-based deep learning. *Solar Energy*, 278, Article 112803.
- David, R., Duke, J., Jain, A., Janapa Reddi, V., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Regev, S., Rhodes, R., Wang, T., & Warden, P. (2021). TensorFlow Lite Micro: Embedded machine learning for TinyML systems. *Proceedings of Machine Learning and Systems*, 3, 800–811.
- Fan, L., Gong, X., Huang, L., & Ou, Y. (2026). Dual-branch decoupling framework for medical one-class classification. *Pattern Recognition*, 180, Article 114266.
- Fang, L., Yu, X., Cai, J., Chen, Y., Wu, S., Liu, Z., Yang, Z., Lu, H., Gong, X., Liu, Y., Ma, T., Ruan, W., Abbasi, A., Zhang, J., Wang, T., Latif, E., Liu, W., Zhang, W., Kolouri, S., . . . Ma, P. (2026). Knowledge distillation and dataset distillation of large language models: Emerging trends, challenges, and future directions. *Artificial Intelligence Review*, 59(1), Article 17.
- Han, S., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *International Conference on Learning Representations*.
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770–778). IEEE.
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.
- Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., & Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 2704–2713). IEEE.
- Kumar, R., Reddy, R. D., Babu, T. N., & Pandiyan, M. (2026). Dual-stream 1D mobile-net with channel attention fusion for rolling element fault diagnosis. *Scientific Reports*. Advance online publication.
- Li, Q., Lin, J., Qi, R., & Chi, Y. (2026). An assembly defection diagnostic method for alkaline water electrolyzer aware by ohmic resistance based on physical-informed statistical analytics. *International Journal of Hydrogen Energy*, 220, Article 153952.

- Lin, J., Chen, W.-M., Lin, Y., Cohn, J., Gan, C., & Han, S. (2020). MCUNet: Tiny deep learning on IoT devices. In *Advances in Neural Information Processing Systems* (Vol. 33, pp. 11711–11722).
- Lindemann, B., Maschler, B., Sahlab, N., & Weyrich, M. (2021). A survey on anomaly detection for technical systems using LSTM networks. *Computers in Industry*, 131, Article 103498.
- Moon, A.-S., Lee, J. K., & Lee, J. (2026). Three-dimensional convolutional neural network with channel compression module for efficient diagnosis of thyroid-associated orbitopathy. *Multimedia Tools and Applications*, 85, Article 609.
- Pang, G., Shen, C., Cao, L., & van den Hengel, A. (2021). Deep learning for anomaly detection: A review. *ACM Computing Surveys*, 54(2), Article 38.
- Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 4510–4520). IEEE.
- Zhou, Z., Chen, X., Li, E., Zeng, L., Luo, K., & Zhang, J. (2019). Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8), 1738–1762.