ASTES

Advances in Science, Technology and Engineering Systems Journal
Vol. 10, No. 6, 29-41 (2025)
www.astesj.com
Special Issue on Innovation in Computing, Engineering Science & Technology

ASTESJ
ISSN: 2415-6698

StradNet: Automated Structural Adaptation for Efficient Deep Neural Network Design

David Degbor[1], Haiping Xu[*1], Pratiksha Singh[1], Shannon Gibbs[1], Donghui Yan[2]

[1]*Computer and Information Science Department, University of Massachusetts Dartmouth, Dartmouth, MA 02747, USA*

[2]*Mathematics Department, University of Massachusetts Dartmouth, Dartmouth, MA 02747, USA*

ARTICLE INFO

Article history:
Received: 14 October, 2025
Revised: 27 November, 2025
Accepted: 29 November, 2025
Online: 05 December, 2025

Keywords:
Machine learning
Deep neural network (DNN)
Structurally adaptive DNN
Automated network design
Dynamic environments

ABSTRACT

Deep neural networks (DNNs) have demonstrated remarkable success across a wide range of machine learning tasks. However, determining an effective network architecture, particularly the sizes of the hidden layers, remains a significant challenge and often relies on inefficient trial-and-error experimentation. In this paper, we propose an automated architecture design approach based on structurally adaptive DNNs, referred to as StradNet models. Our method begins with training a fully connected DNN (FC-DNN) initialized with standard hidden layer sizes. The structure is then progressively adapted by iteratively pruning weak connections, removing isolated neurons, and fine-tuning the network, producing a series of partially connected DNN (PC-DNN) models that converge toward an effective configuration. Unlike many pruning methods that reduce redundancy only after full training, StradNet integrates structural adaptation directly into the learning process, enabling the network to evolve as it learns. This adaptability makes StradNet well-suited for domains with shifting data distributions and complex, high-dimensional dependencies. Experiments on dynamic environments, such as marine datasets, demonstrate that StradNet produces efficient and scalable models that consistently outperform conventionally pruned FC-DNNs. Overall, this automated strategy enhances the efficiency of DNN design and provides a practical framework for real-world machine learning applications.

1. Introduction

Deep neural networks (DNNs) have achieved remarkable success across a wide range of machine learning applications, including speech recognition, image classification, and natural language processing [1]-[3]. Their ability to approximate highly complex nonlinear functions arises from the presence of multiple hidden layers and large numbers of parameters [4]. These hidden layers allow DNNs to extract hierarchical features from raw input data, capturing both low-level and high-level patterns. For instance, convolutional neural networks (CNNs) learn basic patterns such as edges and textures in their early layers, while deeper layers identify more abstract object features [5]. In contrast, recurrent neural networks (RNNs) and long short-term memory (LSTM) architectures are designed to capture temporal or sequential dependencies, making them effective for language modeling and time-series prediction [6]. In recent years, modern large-scale architectures, such as large language models (LLMs) [7], have pushed the boundaries of deep learning. With billions of parameters, these systems demonstrate extraordinary expressive power, excelling in tasks ranging from text summarization to code generation. These models also exhibit cross-task generalization, transferring knowledge from one domain to another and enabling zero-shot and few-shot learning scenarios that were previously unattainable with smaller networks. Moreover, innovations such as attention mechanisms and transformer architectures have further expanded the capacity of DNNs to model long-range dependencies and capture complex contextual information [8].

Despite these successes, the increasing size and complexity of DNNs present significant challenges. Large models require substantial computational resources and massive training datasets, making their deployment on resource-constrained devices often impractical [9]. Furthermore, a high number of parameters increases the risk of overfitting, where the model memorizes training noise rather than capturing generalizable patterns. Conversely, overly small models may suffer from underfitting, failing to capture important structures in the data. Thus, striking the right balance between model complexity, generalization capability, and computational efficiency is critical for building effective and scalable DNNs [10].

[*]Corresponding Author: Haiping Xu, University of Massachusetts Dartmouth, Dartmouth, MA 02747, Email: hxu@umassd.edu

A key determinant of this balance lies in the number of neurons and connections in the hidden layers, which directly influence both the representational capacity and computational cost of DNNs. Existing methods typically rely on manual hyperparameter tuning or grid/random search, which are computationally expensive, time-consuming, and heavily dependent on domain expertise [11]. To address these challenges, various technical approaches have been proposed to optimize network structures more systematically. Model compression methods, such as quantization and pruning, reduce resource demands while maintaining accuracy [12], [13]. However, traditional pruning methods typically operate within a fixed network structure, limiting their ability to fully exploit structural redundancy. Neural Architecture Search (NAS) frameworks attempt to automate network design by exploring a large architecture space using reinforcement learning, evolutionary algorithms, or gradient-based optimization [14]. While NAS can generate high-performance architectures, its computationally intensive nature may still result in overly parameterized networks that are difficult to deploy efficiently.

Given these limitations, there is a pressing need for adaptive and automated methods that can optimize both the number of neurons and their interconnections. Such approaches would not only improve computational efficiency but also reduce overfitting risks and enhance generalization capabilities. To address this need, this paper introduces structurally adaptive DNNs, referred to as StradNet models. Specifically, this work aims to develop StradNet to dynamically adapt network structures, improving computational efficiency while maintaining competitive accuracy, and automating pruning and tuning processes to handle complex, noisy datasets from dynamic environments. This results in both time savings and scalability improvements over conventional approaches. Our approach begins with training a fully connected DNN (FC-DNN) initialized with standard hidden layer sizes. The network is then progressively adapted through iterative pruning of weak connections, removal of isolated neurons, and fine-tuning of the remaining weights, resulting in lightweight yet high-performing network architectures. Unlike conventional pruning or NAS methods, StradNets dynamically adapt the network structure, enabling more efficient and generalizable partially connected DNN (PC-DNN) models for diverse real-world applications. The main contributions and novelties of this paper are summarized as follows:

- Proposed a systematic approach for identifying and pruning weak connections in DNNs, enhancing both computational efficiency and generalization capability.
- Introduced mechanisms to detect and remove isolated neurons along with their associated connections, simplifying the network structure without compromising accuracy.
- Demonstrated that iterative structural adaptation, combined with network fine-tuning, effectively balances model complexity and predictive performance.
- Demonstrated that efficient network architectures do not require a fixed hidden-layer ratio, and that StradNet adaptation allows models to adjust layer sizes for specific tasks.
- Evaluated the StradNet framework in dynamic environments, such as marine datasets, and demonstrated that it outperforms conventionally pruned FC-DNNs in terms of computational efficiency and scalability.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 presents the proposed automated framework for DNN design. Section 4 describes the procedures for automated structural adaptation of DNNs. Section 5 provides case studies that validate the feasibility and effectiveness of the StradNet architecture. Finally, Section 6 concludes the paper and outlines future research directions.

2. Related Work

In recent years, researchers in deep learning and neural networks have made substantial progress in developing strategies to improve classifier performance. These efforts include making existing approaches not only more accurate but also more resource-efficient, thereby advancing both model architectures and optimization techniques [15], [16]. In machine learning applications, models typically require both learned parameters and hyperparameters to be carefully adjusted. As network sizes grow, manual tuning becomes increasingly difficult, often requiring domain expertise or careful monitoring of experimental results, and if done poorly, it can degrade performance. To address these challenges, researchers have been exploring automated methods that enhance the performance of learning algorithms across a variety of tasks. For example, in [17], the researchers presented an optimization strategy that uses a population-based evolutionary search algorithm to identify promising hyperparameter settings in EEG classification. The optimization process proceeds in a single tuning cycle, where at each stage past evaluation results are compared and leveraged to generate improved parameter settings. Likewise, in [18] the authors described a software-testing-inspired approach to hyperparameter optimization. Presented as a constrained exhaustive method, also known as t-way combinatorial testing, the goal is to evaluate a subset of hyperparameter configurations on the model. In [19], the authors approached hyperparameter tuning from a different perspective by applying asynchronous reinforcement learning to seek optimal configurations. Their methodology employs a master agent to coordinate a group of worker agents that continuously evaluate hyperparameter settings. In [20], the authors introduced a NAS method aimed at mitigating the computational challenges of identifying effective DNN structures. They represented the initial network structure as a root node in a tree and recursively evaluated child nodes using an improved Monte Carlo Tree Search algorithm. In [21], the authors proposed a neural network compression method that removes redundant features from the convolutional layer within a multihead CNN architecture. Their approach computes multi-dimensional principal components on the convolutional layers using a statistically guaranteed hyperparameter optimization scheme. While the aforementioned approaches provide valuable insights into effective DNN design, some methods exhaustively evaluate possible configurations, whereas others attempt a one-shot search process. Unlike these approaches, StradNet employs an adaptive strategy that iterates through multiple cycles to identify an effective hyperparameter configuration, particularly the hidden layer sizes. The process can also terminate early if low accuracy is detected, saving significant time compared to other architecture search algorithms. Techniques such as NAS,

combinatorial methods, and evolutionary algorithms are effective for navigating the complex, high-dimensional hyperparameter space; however, they are typically computationally expensive, and their application to partially connected architectures has not been fully explored. In contrast, a key aspect of our approach is the use of PC-DNNs as lightweight and efficient alternatives to conventional FC-DNN architectures. Accordingly, this work complements existing research that primarily focuses on FC-DNN-based designs.

Research on improving model efficiency through structural adaptation has become an active area of study, leveraging automated pruning techniques to facilitate the transition from FC-DNNs to PC-DNNs. Studies across various pruning strategies show that the automated conversion process can maintain or even improve the accuracy of FC-DNNs [22]. More broadly, automated machine learning (AutoML) integrates algorithms for dynamic model construction and deployment, as well as hyperparameter optimization [23]. Frameworks such as the Dynamic Processing Unit (DPU) presented in [24] show that network compression can take a hybrid approach, where weights are not permanently removed or masked but instead oscillate between active and inactive states depending on the DPU. The authors in [25] explored an Echo State Network (ESN) structure designed to provide simplicity for neural networks running on edge devices. Their approach prunes neurons based on a neuron-importance heuristic combined with iterative fine-tuning. In [26], the researchers presented an activation-profile-based neuron pruning framework that targets neurons during inference and groups them according to similar outputs. In their approach, neurons with outputs that are too small are pruned, while those with sufficiently large outputs are retained. In [27], the authors introduced direct connections from each layer to the output layer, providing multiple substructures to increase network capacity. They then apply a training scheme with L1 regularization to encourage the removal of redundant neurons and facilitate the adjustment of hidden layer sizes. The authors in [28] addressed the structural adaptation of neural networks by constructing uniform or non-uniform subnets. These subnets can be sampled across different epochs to leverage one-shot training and provide a runtime advantage during inference. In [29], the authors presented a DNN-based approach for building budget-constrained models for big data analysis. In their approach, they demonstrated how to eliminate less important features by identifying weak links and neurons, thereby bringing the model cost within a given budget. Building on existing methods for structural adaptation, StradNet introduces a distinct and effective strategy for optimizing network architecture. A key challenge in this domain is the continual growth of network size and the associated computation and storage overhead from weak connections. StradNet addresses this issue by identifying and pruning weights that contribute little to the model's output, thereby improving efficiency without compromising accuracy. Traditional structurally adaptive algorithms often depend on neuron importance scores to guide pruning, a process that can be both tedious and computationally intensive. In contrast, StradNet tracks the number of connections pruned for each neuron and removes a neuron only when all its connections have been eliminated, leading to more compact and efficient PC-DNN architectures. PC-DNNs and structural adaptation have gained increasing attention for their potential to balance model expressiveness and computational efficiency. Previous studies have explored how connectivity patterns affect network performance or specific aspects of partially connected designs. However, automated structural adaptation within PC-DNNs remains a relatively underexplored area. Recent advancements, such as adaptive learning rate algorithms [30], have further enhanced model convergence and generalization, yet structural adaptation in PC-DNNs continues to present promising opportunities for future research.

Previous research on optimizing hyperparameters for dynamically changing environments has explored how neural networks can be applied to tasks requiring adaptability and robustness under shifting conditions. Applications such as oceanography, weather forecasting, stock market analysis, and housing price prediction all involve underlying patterns that fluctuate over time, highlighting the need for flexible models. In such highly volatile domains, it is insufficient to train a model once and rely on it consistently; instead, a model must adapt to complex changes in data and restructure dynamically. In the field of Scientific Machine Learning (SciML), both machine learning and mechanistic modeling techniques have been independently and successfully applied in systems biology [31]. Machine learning excels at uncovering statistical relationships and generating quantitative predictions from data, whereas mechanistic modeling provides a robust framework for capturing domain knowledge and elucidating the causal mechanisms driving dynamic biological processes. In [32], the authors evaluated the impact of diverse marine environments on image classification performance using these methods. They reported that their methodology effectively demonstrates how well models can generalize across a wide range of conditions. The study in [33] examined a weather application, presenting a hybrid machine learning approach for rainfall prediction. They developed a useful machine learning tool capable of identifying potential threats in real time and issuing timely warnings. In [34], the authors proposed an Environmental Graph-Aware Neural Network (EGAN) for modeling and analyzing large-scale, multi-modal environmental datasets. The EGAN framework constructs a spatiotemporal graph representation that integrates physical proximity, ecological similarity, and temporal dynamics, and employs graph convolutional encoders to extract expressive spatial features. In [35], the authors investigated the housing market using DNN predictions. Forecasting the housing market is highly complex and high-dimensional, which can make it challenging for machine learning models to capture underlying patterns. Finally, [36] extended this analysis to the stock market. Despite the availability of years of data, financial markets are heavily influenced by external factors such as geopolitical events and investor sentiment, making accurate predictions difficult. These studies underscore the importance of designing models that can adapt continuously, rather than relying on static training procedures. In our approach, we selected datasets from dynamic environments, including oceanographic and marine settings, where conditions vary seasonally and data are often high-dimensional. Marine factors such as turbidity, depth, light variation, image resolution, camera calibration, and motion blur further increase the challenge of achieving robust generalization. In some applications, only a single training cycle has been used to

capture chaotic patterns in a one-shot manner. While feasible, this approach often struggles in dynamic environments, requiring frequent retraining to maintain reliable predictions. In contrast, our multi-step pruning loop can be fine-tuned and restructured in real time to adapt to new data, providing a general solution for developing efficient PC-DNN models in dynamic environments.

3. An Automated Framework for DNN Design

In deep learning, a network's structure, particularly the configuration of its hidden layers, plays a crucial role in determining accuracy, efficiency, and generalization performance. Unlike generic hyperparameters such as learning rate, batch size, or number of epochs, which primarily control training behavior, structural parameters directly define the capacity, connectivity, and representational power of the network itself. Manually selecting an effective structure is especially challenging because the search space of possible layer sizes, neuron counts, and inter-layer connections is extremely large and combinatorially complex. To address this challenge, we propose an automated framework that adapts hidden layer sizes during training. Our method follows a two-phase approach. In Phase 1, the network undergoes aggressive pruning to remove less significant connections and neurons. If pruning exceeds the optimal point, Phase 2 readjusts by restoring connections and pruning more conservatively. This adaptive procedure ensures convergence toward an efficient structure without extensive manual intervention. The key idea is the progressive transformation of an initialized FC-DNN into a more efficient PC-DNN. By dynamically removing weak connections and isolated neurons, the framework reduces computation during forward and backward propagation, improving efficiency while maintaining accuracy. To emphasize the structural effects, all other hyperparameters are held constant. Figure 1 illustrates the process, beginning with dataset preprocessing, including cleaning and normalization, followed by iterative pruning and refinement until the network stabilizes at a desirable structure.

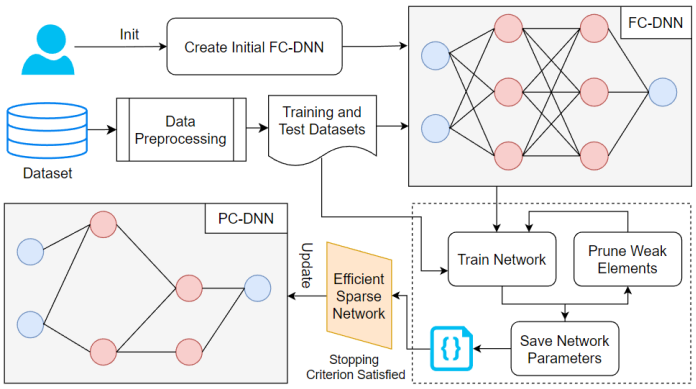


Figure 1: An automated framework for DNN design.

As shown in Figure 1, the dataset is first preprocessed by normalizing input features using the min-max scaling method as in (1). Min-max scaling ensures comparability across features, improves convergence speed, and preserves relative relationships among input values. This procedure rescales all features to the range [0, 1], transforming raw data into a normalized space that the network can effectively interpret while retaining the original distribution.

$$x_k' = \frac{x_k - x_{k\_min}}{x_{k\_max} - x_{k\_min}} \quad (1)$$

where x_{k_min} and x_{k_max} are the minimum and maximum values of input feature k, x_k is the original feature value, and x_k' is the normalized feature value.

After preprocessing, the dataset is divided into training and test sets to enable evaluation on unseen data, typically using a 70:30 or 80:20 split to ensure sufficient data for both sets. The automated structural adaptation process requires user interaction only during network initialization. Hidden layer sizes are set in powers of two. For example, a network can be initialized with two hidden layers of 512 and 256 neurons, respectively, which serve as standard starting sizes. All other hyperparameters (e.g., number of layers, learning rate, loss function) remain fixed for comparability. Once initialized and prepared, the framework iteratively adjusts the network structure through a prune-train-test loop until a predefined stopping criterion is satisfied, resulting in an efficient sparse network as a PC-DNN. During each iteration, the network is trained and evaluated on the split dataset, and the maximum accuracy is tracked to identify the most promising state for subsequent refinements. Although the base network starts with random weights, pruning and retraining preserve previously learned knowledge by saving the network parameters instead of reinitializing them.

To enable iterative and adaptive structural refinement, we construct two-dimensional mask matrices as part of the saved network parameters to continuously track pruned connections. Each weight tensor is paired with a matrix of size $i \times j$, matching its dimensions to record pruning decisions precisely. At initialization, all entries are set to *false*, indicating that no connections have been removed; when a connection is pruned, the corresponding entry is switched to *true*. These mask matrices allow the framework to maintain an explicit and transparent record of which connections have been pruned, enabling reproducible and verifiable updates to the network structure during iterative training. The extent of pruning is governed by the pruning ratio, a tunable hyperparameter that controls pruning aggressiveness: higher ratios prune more aggressively, while lower ratios perform pruning more conservatively. This parameter provides flexibility to explore different strategies and balance computational efficiency against potential accuracy degradation. Pruning decisions specifically target weak elements, including individual connections and entire neurons, that contribute minimally to model performance while consuming computational resources. Retaining such redundant elements increases training time and memory usage while offering negligible performance gains. By systematically eliminating these weak components, the framework concentrates computational resources on the most informative parts of the network, thereby enhancing both training efficiency and learning effectiveness. Overall, our StradNet framework aims to reduce network size and training time by removing weak elements, resulting in compact and efficient PC-DNN models without sacrificing accuracy. Furthermore, by integrating mask-based tracking with controlled pruning, the framework ensures that network adaptation remains fully automated, reproducible, and robust across diverse datasets, initialization conditions, and dynamically evolving data distributions.

4. Automated Structural Adaptation of DNNs

4.1. Pruning Weak Connections in a DNN

The first step in network reduction is to prune weak connections by removing those with the smallest absolute weights, as determined by a predefined minimal pruning ratio, p_{min}. As defined in (2), the minimal pruning ratio p_{min} is the proportion of connections removed from the DNN during pruning relative to the total number of connections before pruning.

$$p_{min} = \frac{N_{\text{removed}}}{N_{\text{total}}} \times 100\% \quad (2)$$

where N_{removed} is the number of pruned connections, and N_{total} is the total number of connections prior to pruning. For example, if p_{min} = 10%, the 10% weakest connections are removed. We refer to this as the *minimal* pruning ratio because, in the next step (described in Section 4.2), removing isolated neurons also requires eliminating their associated connections. Consequently, the actual pruning ratio in each round may exceed p_{min}.

This pruning step reduces the neural network's complexity while maintaining high performance in the resulting PC-DNN. The network to be pruned may be an FC-DNN in the first pruning round or a PC-DNN if pruning has already been applied. To identify weak connections, we sort all active weights in ascending order of magnitude. The pruning threshold is computed from the absolute weight values, since the influence of a connection depends on its magnitude rather than its sign: a large negative weight represents a strong inhibitory effect, while a large positive weight represents a strong excitatory effect. Magnitude-based pruning is effective because small-magnitude weights have minimal impact on downstream activations. In contrast, large-magnitude weights encode more informative features. Removing the weakest weights therefore eliminates low-impact parameters while preserving essential computational pathways of the network. As discussed in Section 3, mask matrices track which connections have been removed.

Algorithm 1 outlines the procedure for removing weak connections. As shown in the algorithm, the pruning threshold t is determined using the minimal pruning ratio p_{min} and the k^{th} smallest element (step 6 and 7), thereby identifying the weakest proportion p_{min} of weights. The algorithm then removes all weights whose magnitudes fall below this threshold. Given the array of weight values R_w and the number k_w of smallest elements to remove, the value t_w is set as the largest of these k_w elements and serves as the pruning cutoff for the entire network in the current iteration. When a weak connection in a weight tensor Θ is flagged for removal, the corresponding entry in the mask matrix is set to *true* (step 10), ensuring that the training process excludes this connection in subsequent updates. After all weak connections are marked, the algorithm returns the updated PC-DNN (step 11). This threshold-based procedure ensures that pruning decisions remain consistent across layers, regardless of their size or scale. Because the threshold is computed directly from the sorted weight magnitudes, the algorithm adaptively adjusts to the distribution of weights in each pruning round. This makes the pruning robust to training dynamics, including changes during fine-tuning or shifts in connection importance. Moreover, by eliminating only the weakest connections at each iteration, the algorithm avoids abrupt structural changes that could destabilize the learning process, allowing the model to gradually converge toward a compact and efficient PC-DNN.

Algorithm 1: Pruning Weak Connections

Input: DNN N with mask matrices, minimal pruning ratio p_{min}
Output: A PC-DNN with weak connections pruned

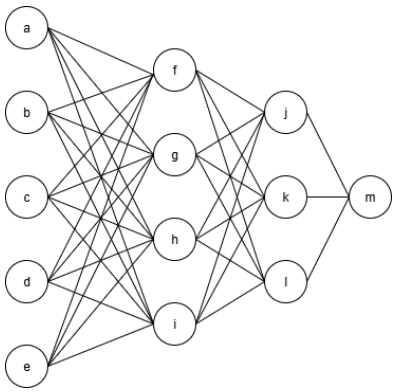
1. Initialize an empty list R_w for remaining (active) weights
2. **for** each weight tensor Θ in N
3. **for** each weight w in Θ with corresponding mask value = *false*
4. Append w to R_w
5. Sort R_w in ascending order
6. Compute $k_w = \lfloor p_{min} * len(R_w) \rfloor$ // the number of weights to prune
7. Set pruning threshold $t_w = R_w[k_w]$ // the k^{th} smallest weight
8. **for** each weight tenor Θ in N
9. **for** each connection c in weight tenor Θ
10. Mark c as *true* (pruned) in the mask matrix if weight $w_c \leq t_w$
11. **return** updated N

Figure 2 shows a sample DNN used to illustrate the pruning of the weakest weight connections. In this example, the network is initialized as an FC-DNN with two hidden layers, containing four and three hidden neurons, respectively. The weights for each connection are displayed in the three weight tensor tables on the left, providing a clear view of the initial parameter distribution. The pruning threshold is determined dynamically based on the current minimal pruning ratio and the weight values in the tables, ensuring that the weakest connections are consistently identified and removed during each pruning round.

	a	*b*	*c*	*d*	*e*
f	0.4	0.5	0.9	0.3	0.7
g	0.7	0.1	0.4	0.6	0.4
h	0.4	0.6	0.5	0.8	0.8
i	0.1	0.9	0.5	0.4	0.6

	f	*g*	*h*	*i*
j	0.2	0.4	0.6	0.5
k	0.2	0.7	0.8	0.5
l	0.1	0.3	0.9	0.8

	j	*k*	*l*
m	0.5	0.2	0.3



(a) Connection weights in the FC-DNN (b) Network structure of the FC-DNN

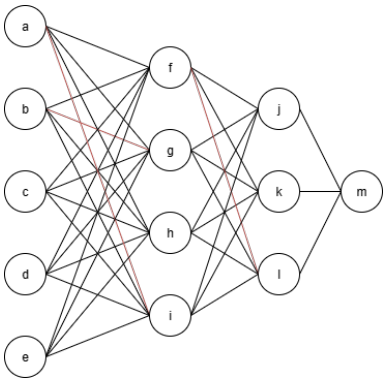
Figure 2: A sample FC-DNN initialized with two hidden layers

Figure 3 illustrates the DNN after pruning weak connections with a minimal pruning ratio of p_{min} = 10%. According to Algorithm 1, k_w is set to 3, and the pruning threshold t_w to 0.1, meaning any connection with a weight less than or equal to 0.1 is flagged for pruning and removed from the neural network. As shown in the figure, all connections flagged for pruning are highlighted in red in both the network diagram and its table representation. After compressing the model's connections by 10%, we can observe which neurons contribute to strong activations and thus influence the network's output. Since no isolated hidden neurons are present after this round of pruning, the same 10% minimal pruning ratio is applied in a second round without removing any neurons. This process can be repeated iteratively until a performance drop is observed.

	a	*b*	*c*	*d*	*e*
f	0.4	0.5	0.9	0.3	0.7
g	0.7	0.1	0.4	0.6	0.4
h	0.4	0.6	0.5	0.8	0.8
i	0.1	0.9	0.5	0.4	0.6

	f	*g*	*h*	*i*
j	0.2	0.4	0.6	0.5
k	0.2	0.7	0.8	0.5
l	0.1	0.3	0.9	0.8

	j	*k*	*l*
m	0.5	0.2	0.3



(a) Connection weights in the PC-DNN (b) Network structure of the PC-DNN

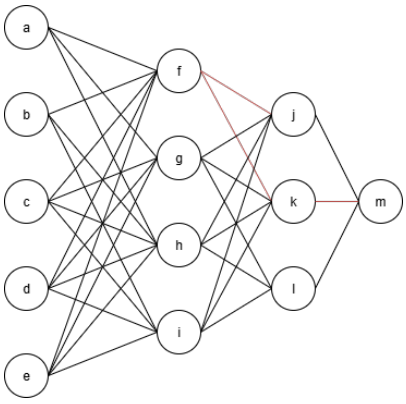
Figure 3: The PC-DNN after pruning weak connections with p_{min} = 10%

Figure 4 shows the updated PC-DNN after retraining. Previously pruned weights are marked as "n/a" in the weight tensor tables, indicating their removal from the network and ensuring that these inactive connections are permanently excluded from all future computations and parameter updates.

	a	*b*	*c*	*d*	*e*
f	1.4	0.6	1.3	0.5	1.7
g	2.7	n/a	1.0	2.0	0.9
h	1.1	1.5	0.7	0.6	0.5
i	n/a	1.9	1.2	2.4	2.2

	f	*g*	*h*	*i*
j	0.3	0.8	0.9	0.5
k	0.3	1.7	2.3	0.7
l	n/a	0.6	1.0	1.3

	j	*k*	*l*
m	2.5	0.3	0.8



(a) Connection weights in the PC-DNN (b) Network structure of the PC-DNN

Figure 4: The PC-DNN after pruning weak connections again with p_{min} = 10%

According to Algorithm 1, in the second round of weak connection pruning, k_w is set to 3, the pruning threshold t_w across all layers is set to 0.3. The computed threshold does not affect any connections in the first weight tensor. However, in the second and third weight tensors, connections (*f*, *j*), (*f*, *k*), and (*k*, *m*) are identified as meeting the pruning threshold and are therefore removed, further simplifying the PC-DNN structure.

4.2. Removing Isolated Neurons and Associated Connections

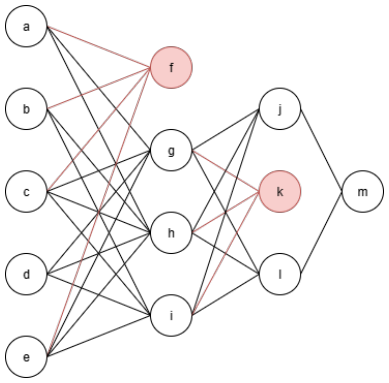
Following the example in Section 4.1, we are left with a PC-DNN containing isolated neurons whose input or output connections have been entirely pruned. We extend the concept of model compression by removing these neurons, which also eliminates their associated incoming or outgoing connections. Figure 5 provides a visual representation of this process, where neurons *f* and *k* are identified as isolated and subsequently removed along with their connections.

A key feature of the model design is that weak connections are removed before weak neurons. This ordering matters because a neuron becomes isolated only after all its input or output connections are pruned. As shown in Figure 3, some pruning rounds produce no isolated neurons, so the neuron-removal step is skipped. When a neuron does lose all incoming or outgoing connections, it is marked as isolated and removed. No pruning ratio is defined for neurons in our approach, because their removal depends solely on the absence of connections. Algorithm 2 summarizes the procedure for detecting and deleting isolated neurons and their associated connections.

	a	*b*	*c*	*d*	*e*
f	1.4	0.6	1.3	0.5	1.7
g	2.7	n/a	1.0	2.0	0.9
h	1.1	1.5	0.7	0.6	0.5
i	n/a	1.9	1.2	2.4	2.2

	f	*g*	*h*	*i*
j	n/a	0.8	0.9	0.5
k	n/a	1.7	2.3	0.7
l	n/a	0.6	1.0	1.3

	j	*k*	*l*
m	2.5	n/a	0.8



(a) Connection weights in the PC-DNN (b) Network structure of the PC-DNN

Figure 5: The PC-DNN with identified isolated neurons

Algorithm 2: Removing Isolated Neurons and Their Connections

Input: DNN *N* with mask matrices
Output: A PC-DNN with isolated neurons and their associated connections removed

1. **repeat**
2. Set *removed* = 0
3. **for** each hidden layer Ψ in *N*
4. **for** each undeleted neuron θ in Ψ
5. Let W_{in} = incoming connections of θ
6. Let W_{out} = outgoing connections of θ
7. **if** all connections in W_{in} are pruned (mask = *true*)
8. Delete neuron θ and all connections in W_{out}
9. Increment *removed* by 1
10. **else if** all connections in W_{out} are pruned (mask = *true*)
11. Delete neuron θ and all connections in W_{in}
12. Increment *removed* by 1
13. **until** *removed* = 0
14. Update *N* and its mask matrices
15. **return** *N*

As shown in Algorithm 2, for an undeleted hidden neuron θ, let W_{in} and W_{out} denote its sets of incoming and outgoing connections, respectively. If all connections in W_{in} or W_{out} are pruned, θ is identified as isolated. In this case, θ is marked for deletion, and all its associated connections are removed from the network (step 8 and 11). All hidden neurons in each hidden layer must be examined for potential removal. To ensure that both existing isolated neurons and any new isolated neurons resulting from the removal of their connections are identified, we define a counter *removed* to record the number of hidden neurons eliminated in each deletion round. This process is repeated until no additional hidden neurons are identified in the current round. At that point, the PC-DNN is updated by removing all identified isolated hidden neurons and their associated connections, and the corresponding mask matrices are updated accordingly (step 14).

4.3. Automated Structural Adaptation Process

As discussed in Section 3, the automated structural adaptation process follows a two-phase pruning strategy. Phase 1 performs coarse pruning across the hidden-layer space, inspired by the lottery ticket hypothesis [37], which suggests that subnetworks within a large model can achieve performance comparable to the full network. Starting from an FC-DNN initialized with an input layer, a number of hidden layers, and an output layer, connections are pruned using a minimal pruning ratio p_{min} (e.g., 10%), and isolated neurons are removed according to Algorithms 1 and 2, respectively. Weights are preserved between iterations to speed convergence, and metrics such as accuracy, neuron count, total weights, and pruning thresholds are tracked. This coarse stage allows the model to rapidly explore a wide range of architectural variations and discard obviously redundant structures. By aggressively pruning early, the network is guided toward a more compact region of the structural space where promising sub-architectures reside. Pruning continues until accuracy drops by more than 1% below the best observed value, at which point the previous best model is restored as the baseline for Phase 2, which applies fine-grained adjustments to finalize the network structure. The duration of Phase 1 depends on the initial hidden layer sizes and the stopping criterion. Larger starting layers generally require more pruning cycles (e.g., 20-30 iterations), whereas well-chosen sizes converge faster (e.g., 5-10 iterations). The stopping criterion balances under-pruning, which leaves the network over-parameterized, and over-pruning, which degrades accuracy. This ensures that Phase 1 terminates at a structurally meaningful point, providing a stable foundation for the more targeted refinements performed in Phase 2.

Phase 2 further refines the network structure established in Phase 1 by applying a smaller pruning ratio, where p_{min} is halved (e.g., 5% in our implementation) to achieve finer-grained structural adjustments. Starting from the best-performing model obtained from Phase 1, Phase 2 executes a single prune-train-test cycle to fine-tune neuron counts and consolidate model stability. Only one iteration is required, as additional pruning at this scale would largely replicate Phase 1 outcomes without significant improvement. Phase 2 serves primarily to polish the architecture rather than reshape it, allowing the network to stabilize around the strongest connections and finalize the distribution of neurons across layers. Algorithm 3 summarizes the complete conversion process from an FC-DNN to a PC-DNN using StradNets. As shown in the algorithm, entering the *while*-loop (Step 10) initiates pruning, during which weak connections and isolated neurons are removed. The network is then retrained without reinitialization, and pruned connections are skipped during feedforward and backpropagation, strengthening the remaining ones. After retraining, accuracy is compared with the maximum observed so far. If accuracy improves, the maximum is updated. If accuracy decreases slightly but remains within tolerance, the process continues. The 1% tolerance margin ensures that natural fluctuations in training are distinguished from genuine performance loss, preventing the network from being pruned beyond its capacity. Once this condition is met in Phase 1 (step 20), the previous state is restored, and Phase 2 begins. A baseline accuracy a is recorded at the start of Phase 2 to measure its refinement. Since Phase 2 consists of only a single prune-train-test cycle, its stopping criterion is met immediately after that cycle. Likewise, if the condition is met in Phase 2 (step 17), the previous state is also restored. At this point, the optimal neural network configuration is finalized and returned to the user (step 19). This two-phase design ensures both global exploration (Phase 1) and local refinement (Phase 2), enabling the framework to produce compact architectures that preserve predictive accuracy while substantially reducing computational cost.

Algorithm 3: Automated Structural Adaptation

Input: FC-DNN N, Train dataset D_{Train}, test dataset D_{Test}
Output: A PC-DNN with optimized hidden layer sizes

1. Scale $D_{Train}$and D_{Test} using min-max normalization
2. Initialize N with standard starting hidden layer sizes
3. Initialize N with random weights
4. **for** each weight tensor Θ in N
5. Create an $i \times j$ mask matrix for Θ, initialized to *false*
6. Train and test N on D_{Train} and D_{Test}, respectively
7. Let a be the current accuracy of N
8. Initialize pruning ratio p_{min} to a predefined value (e.g., 10%)
9. Set *phase* = 1
10. **while** *true*
11. Save the current network N to N_prev
12. Prune weak connections in N according to ratio p_{min}
13. Remove isolated neurons and their remaining connections in N
14. Update N and the mask matrix for each weight tensor in N
15. Train N with saved weights and evaluate accuracy a' on D_{Test}
16. **if** *phase* = 2 // in Phase 2
17. **if** $a' < a - 0.01$ // ≥ 1% decrease from the best accuracy
18. Reload previous network N_prev to N
19. **return** N
20. **else if** $a' < a - 0.01$ // in Phase 1
21. Reload previous network N_prev to N
22. Set *phase* = 2 and $p_{min} = p_{min}$*0.5
23. **else if** $a' > a$, set $a = a'$ // set the best accuracy

In the automated structural adaptation process, the pruning and structural adaptation steps (step 12 and 13) in each iteration have a complexity of $O(N+E)$, where N and E denote the number of neurons and connections, respectively. The $O(E)$ component and the $O(N)$ component correspond to the time complexities of Algorithm 1 and Algorithm 2, respectively. Specifically, Algorithm 1 iterates through each connection in the network, while Algorithm 2 iterates through each node to evaluate and adjust neuron connectivity. Consequently, the overall complexity of the automated structural adaptation introduces only a small additional computational overhead due to structural adjustments, with the training step (Step 15) remaining the major time-consuming component of the process.

5. Case Studies

To evaluate the StradNet architecture, we conducted case studies on real-world machine learning applications. These studies demonstrate that StradNet's autotuning strategies enhance performance while reducing manual intervention. StradNets effectively adapt to complex patterns in noisy datasets, making them suitable for dynamic fields such as autonomous systems, healthcare, oceanography, and finance. In this paper, we focus on marine ecosystem datasets, which are inherently complex and require large neural networks for accurate prediction. Our primary goal is to evaluate the efficiency and scalability of StradNet

relative to the conventional pruning approach, a well-recognized baseline. While comparisons with other adaptive pruning frameworks (e.g., dynamic sparse training [38], NAS-based methods [14], or other modern structured pruning techniques) are valuable, they are beyond the scope of this study due to differences in experimental settings, computational requirements, and model assumptions. In all subsequent experiments, each dataset was divided into 70% training data and 30% testing data. This split was chosen to ensure that each dataset contained sufficient samples to support a stable training process and accurate predictions.

5.1 Automated Two-Phase Pruning Process

In this case study, we apply the StradNet model to predict distant adversarial vessels using 15 input features derived from real-world maritime surveillance data [39]. The adversarial vessels dataset was originally generated to predict the fishing activity of a vessel using its position and behavioral features. In an effort to align the dataset with the objectives of marine research, the labels were repurposed to classify vessels as hostile or non-hostile. The structure of the dataset and the underlying datapoints remained unchanged; only the target labels were adapted to reflect relevant maritime scenarios. The dataset is large, comprising approximately 130,000 records, and was further enhanced using the Synthetic Minority Oversampling Technique (SMOTE) to address class imbalance and better preserve representative category distributions [40]. By applying SMOTE, we generated synthetic samples for the minority class while maintaining the overall feature distribution and diversity of the data. The input features include calendar variables capturing temporal and seasonal patterns; geographical indicators representing vessel detection locations; distance-based metrics estimating proximity to shores and ports; physical attributes such as speed, course, direction, and size; and environmental factors including weather and visibility conditions. Together, these features provide rich contextual information essential for accurately identifying potential adversarial vessels at a distance.

To construct the initial FC-DNN model, we first define its key hyperparameters, including the number of hidden neurons, activation functions, and training parameters. While our approach supports an arbitrary number of hidden layers, we restrict the network architecture to two hidden layers for simplicity and reproducibility. The remaining hyperparameters are set as follows: the Rectified Linear Unit (ReLU) activation function in all hidden layers, a learning rate of 0.01, 12 training epochs, and a batch size of 16. The number of hidden neurons is dataset-dependent; in this experiment, we use a standard configuration with 512 and 256 neurons in the first and second hidden layers, respectively. This configuration provides a balanced trade-off between model capacity and computational efficiency, serving as a strong baseline for evaluating the effects of iterative pruning. The setup aims to demonstrate that progressive pruning of the fully connected network can maintain strong generalization performance throughout the pruning process. Equation (3) defines the computation used to determine the initial total number of weights in the FC-DNN with n layers.

$$totalWeights = \sum_{i=1}^{n-1}(nNeu\_Layer_i * nNeu\_Layer_{i+1}) \quad (3)$$

where $nNeu\_Layer_i$ refers to the number of neurons in layer i, where $1 \leq i \leq n$.

We begin with a minimal pruning ratio p_{min} of 10%, removing 10% of the total weights at each pruning step to gradually simplify the architecture. If model performance drops beyond a predefined accuracy threshold, the model is reverted to a previously saved architecture, and pruning continues at a reduced rate to preserve stability. In this study, a 1% decrease from the best recorded accuracy in Phase 1 is used as the cutoff threshold, at which point the minimal pruning ratio p_{min} is reduced from 10% to 5% in Phase 2 to enable finer-grained structural refinement. Table 1 provides an illustrative example of the automated pruning process used to identify an effective StradNet architecture by progressively removing weak connections and isolated neurons while maintaining overall model generalization.

Table 1: An illustrative example of the automated pruning process.

Total Neurons	*HL1*	*HL2*	*Total Weights*	*Pruning Ratio (%)*	*Threshold*	*Accuracy (%)*
***Phase* 1 (*Coarse Pruning Step*)**						
768	512	256	139,264	N/A	N/A	96.30
765	512	253	125,337	10	0.007	97.10
748	512	236	112,803	10	0.015	97.31
742	512	212	101,523	10	0.022	97.27
704	512	192	91,371	10	0.029	97.65
686	512	174	82,234	10	0.036	97.74
613	454	159	74,010	10	0.045	97.66
569	440	129	66,609	10	0.070	97.85
555	433	122	59,948	10	0.127	97.65
543	422	121	53,953	10	0.206	***98.12***
526	409	117	48,558	10	0.302	97.96
512	401	111	43,702	10	0.413	98.04
495	396	99	39,332	10	0.526	97.97
484	390	94	35,399	10	0.651	98.12
475	384	91	31,859	10	0.785	97.97
463	382	81	28,673	10	0.924	97.93
448	374	74	25,805	10	1.081	97.38
423	363	60	23,225	10	1.217	***96.88***
***Phase* 2 (*Fine-Tuning Step*)**						
448	374	74	25,805	10	1.081	97.38
437	**372**	**65**	**24,514**	**5**	**1.133**	**97.36**

From Table 1, we observe that the pruning process completes its first phase when accuracy drops to 96.88%, 1.24% below the best recorded accuracy of 98.12%. The network is then reverted to its previous architecture and enters the fine-tuning step with a minimal pruning ratio of 5%. The final architecture obtained through this process contains 437 neurons: 372 in the first hidden layer and 65 in the second. This configuration achieves an accuracy of 97.36%, 1.06% higher than the initialized FC-DNN model (96.30%). These results indicate that the StradNet pruning process generalizes well to new data, achieving accuracy comparable to the much larger initial model, and demonstrate that many weights and neurons are unnecessary. In particular, these findings highlight how pruning effectively removes redundant structure while preserving the essential computational pathways needed for accurate prediction. While only the strongest connections contribute significantly to accurate predictions, excessive pruning can be detrimental; removing too many important connections reduces accuracy and impairs prediction quality. This underscores the importance of selecting appropriate pruning ratios to balance compactness and performance. Moreover, the smooth recovery of accuracy after reverting to the

best-performing architecture reinforces the value of maintaining historical model states during the pruning process. This mechanism prevents the framework from drifting into overly aggressive pruning regimes and ensures that structural exploration remains both controlled and reversible. Figure 7 illustrates this effect with a constant minimal pruning ratio of 10%, showing a steady decline in accuracy as the network becomes too small to perform effectively.

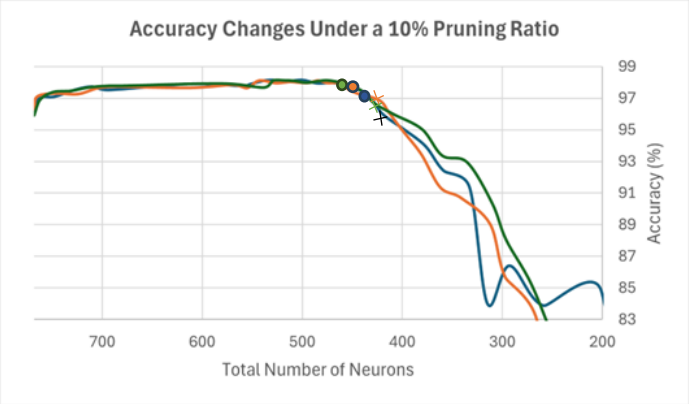


Figure 7: Accuracy vs. total number of neurons under a 10% pruning ratio

In Figure 7, we present three trials of the StradNet workflow, in which weak connections and neurons are progressively pruned from the network. Each trial begins with 768 neurons, consistently arranged as 512 in the first hidden layer and 256 in the second. Accuracy initially improves by approximately 2%, suggesting a reduction in overfitting as the network size decreases. The early pruning stages indicate that the network can still achieve accurate predictions on the test dataset. However, once the network size falls below roughly 450 neurons, accuracy declines sharply. Continuing to prune at a fixed 10% rate beyond this point results in a steady performance decline, which we attribute to the aggressiveness of the pruning strategy and the need for more careful dynamic adjustment. This trend confirms that the crosses in the figure mark the appropriate points for activating the stopping criterion. At these points, the network transitions to fallback architectures, represented by the dots, allowing pruning to continue under a more relaxed strategy.

The fine-tuning step begins by training and testing the reverted network using the stored weight parameters. In this phase, we apply a single prune-train-test cycle with a 5% minimal pruning ratio, after which network modifications are finalized. Figure 8 illustrates the reinstated neural network from Phase 1. At the start of Phase 2, further pruning remains possible at a smaller percentage; accordingly, this diagram reflects pruning 5% of the weakest connections. The resulting performance curve closely resembles that of the 10% pruning diagram, confirming that the observed accuracy decline is consistent across trials. The dots in Figure 8 correspond to the same total-neuron counts shown in Figure 7, while the crosses indicate the configurations returned to the user upon algorithm completion, representing the effective network architectures. As shown in the figure, all three trials demonstrate that accuracy continues to decline when pruning at 5% is repeated. Additional pruning at this rate is therefore unnecessary, as Phase 1 has already revealed the adverse effects of more aggressive 10% pruning. The convergence of results across multiple trials also indicates that the selected 5% pruning ratio provides a stable termination condition, ensuring that the final PC-DNN architecture reflects both structural efficiency and reliable predictive performance.

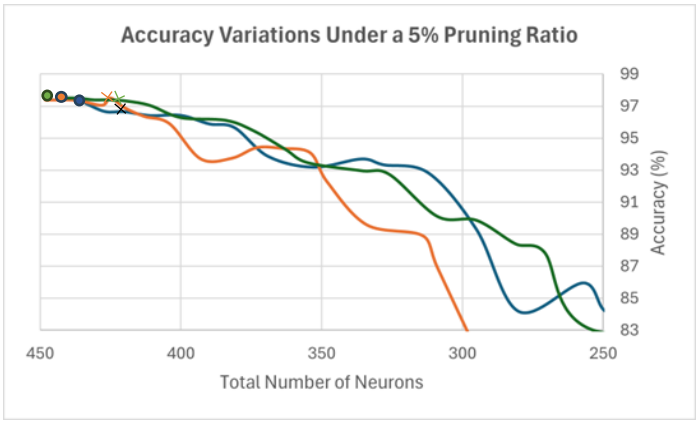


Figure 8: Accuracy vs. total number of neurons under a 5% pruning ratio

Figure 9 illustrates the relationship between the total number of weights in the neural network and its accuracy under a 10% minimal pruning ratio. Across three trials, we observe a positive trend in the early stages of pruning: as the number of weights decreases, the network achieves higher accuracy, suggesting that larger networks may be prone to overfitting. This trend continues until the networks become sufficiently small, around 20,000 total weights, at which point accuracy drops sharply from approximately 98% to 83%. These results are consistent with the accuracy decline observed in Figure 7 when excessive neurons and their associated weak connections are removed. Similarly, the crosses in Figure 9 mark the points at which the stopping criterion is activated, leading into the fine-tuning step. This pattern highlights the balance between removing redundant weights and keeping enough capacity for accurate predictions. The consistent results across trials also show that the stopping rule effectively prevents the network from being pruned too aggressively.

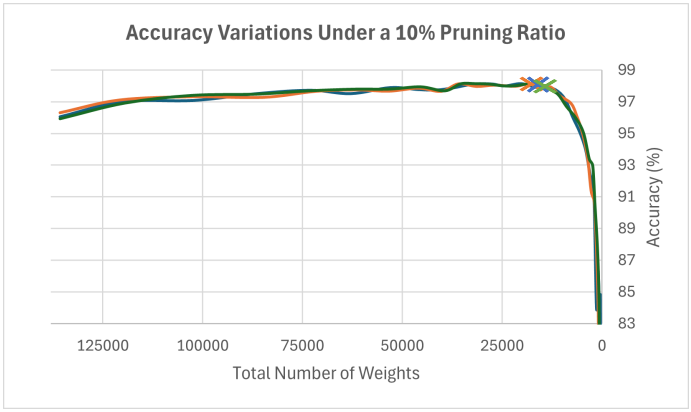


Figure 9: Accuracy vs. total number of weights with 10% pruning ratio

5.2 Neuron Allocation Between Hidden Layers

In conventional DNNs, the hidden layers often follow a pyramid configuration, in which each successive layer contains fewer neurons than the previous one [41]. For instance, in a network with two hidden layers, the size of the first layer is typically determined by factors such as dataset size, number of

input features, number of output neurons, and task complexity. The second hidden layer often contains about half as many neurons as the first, reflecting its role in refining the learned representation into a form suitable for the output layer. Using this conventional architecture as a baseline, the present study investigates whether a specific relationship between the two hidden layers of a PC-DNN can achieve predictive performance comparable to the standard design. To evaluate this, the StradNet model was tested on three datasets, including *Adversarial Vessels*, *Oil Spill*, and *Sonar Classification*. The *Adversarial Vessels* dataset is described in Section 5.1. The *Oil Spill* dataset, derived from satellite imagery and reformatted into tabular form, is used to distinguish between spill and non-spill patches [42]. This dataset serves as our medium-scale application case and features a highly modified structure. It contains 60 attributes and 1,200 records and was augmented using the SMOTE technique to address class imbalance by generating additional samples for the underrepresented class. Controlled noise was also introduced as a form of data augmentation to encourage the network to learn meaningful patterns rather than memorize specific instances. Two augmentation strategies were considered: feature perturbation and output flipping [43], [44]. Feature perturbation involves applying small random variations to input feature values, which enhances the model's robustness and reflects natural variations observed in real-world conditions. Output flipping, on the other hand, simulates label noise by occasionally swapping class labels to mimic annotation errors. Together, the use of SMOTE and controlled noise increased the dataset's variability and complexity, thereby enhancing the network's capacity for robust pattern recognition.

The *Sonar Classification* dataset, which contains acoustic signals recorded at varying frequencies and angles, can be used to evaluate object detection performance in applications such as mine, vessel, and underwater structure identification [45]. Although the dataset documentation does not describe the specific data collection methodology, its close association with sonar technology, a common sensing modality used by marine vessels to detect underwater mines, debris, and other hazards, makes it relevant for our study. This small-scale dataset allows us to evaluate how well the network maintains generalization when trained with limited data and to demonstrate the robustness of the StradNet approach across different dataset sizes. It contains 60 input features and 200 samples. Using a small dataset also highlights the pruning scheme's behavior under a high risk of overfitting. If the pruned network trained on this dataset exhibits a substantial accuracy drop, it would indicate that the model depended heavily on specific features and was overfitted to the limited data. All input features were normalized using the min-max normalization scheme, which scales values into the [0, 1] range to ensure equal contribution of all variables. Each of the 60 features captures distinct signal characteristics, where different angles of incidence yield unique representations that enrich the overall input space.

In this case study, we examine the relationship between the two hidden layers, HL1 and HL2, in the StradNet models. Specifically, we investigate whether the pruning process affects the neuron ratio between these layers, as defined in (4).

$$neuronRatio = \frac{nNeu\_HL2}{nNeu\_HL1} \times 100\% \quad (4)$$

where *nNeu*_HL1 and *nNeu*_HL2 are the numbers of neurons in hidden layers HL1 and HL2, respectively.

In the initial configuration, the neuron ratio between HL1 to HL2 is set to 50%. This setup allows us to determine whether a consistent pattern emerges across the resulting StradNet models. Table 2 presents both the initial and resulting configurations for the three datasets. The experimental results represent the averages of four independent runs, and the last column of the table reports the standard deviation of the accuracy values. Our findings indicate that a neuron ratio of 50% is not always optimal, particularly for large hidden layers, likely due to the increased complexity of the model and the associated tasks. Nevertheless, our approach effectively identifies dataset-specific optimal ratios of hidden neurons between HL1 and HL2.

Table 2: Average Changes in the ratio of neurons between HL1 and HL2

Neuron Ratio	*Total Neurons*	*HL1*	*HL2*	*Total Weights*	*Accuracy*	*Standard Deviation*
Adversarial Vessels						
50	768	512	256	139264	96.30	0.379
17.8	428	363	65	23537	97.41	0.252
Oil Spill						
50	768	512	256	156671	92.65	0.821
41.4	724	512	212	67609	91.03	0.543
Sonar Classification						
50	768	512	256	162304	90.11	0.904
75.3	384	219	165	16365	91.73	0.353

As shown in Table 2, for the *Adversarial Vessels* dataset, the baseline configuration used 512 neurons in HL1 and 256 in HL2. After applying the StradNet approach, the suitable HL2 size was found to be 17.8% of HL1, a much lower ratio than the standard 50%, determined dynamically through iterative structural adaptation. On average, this pruning removed about 340 neurons and approximately 83.1% of the weights, yet achieved slightly higher average accuracy than the baseline, suggesting that smaller, more targeted architectures may outperform larger ones when redundant nodes are removed. For the *Oil Spill* dataset, the suitable HL2 size was about 41.4% of HL1 compared to the baseline's 50%. Only 44 neurons, all from HL2, were pruned, and both configurations achieved similar accuracy, indicating that in some cases the 50% rule still yields desirable performance. For the *Sonar Classification* dataset, the model identified a increased ratio of 75.3%, removing nearly 50% of neurons and around 89.9% of the weights, which increased accuracy by more than 1.6% over the baseline. This suggests that the original network was over-parameterized, and that near-symmetrical layer sizes may benefit certain datasets. The standard deviation values of accuracy, also reported in Table 2, are consistently low, confirming that the results across the four independent runs are stable and reliable. Overall, these findings demonstrate that suitable neuron allocation between HL1 and HL2 is heavily dataset-dependent. The adaptability of the StradNet approach enables it to determine dataset-specific pruning schemes, reducing over-parameterization while improving efficiency and predictive accuracy. These results also highlight that fixed architectural heuristics, such as the common 50% HL1-to-HL2 ratio, are not universally optimal. Instead, effective layer sizing often emerges from data-driven adjustments that reflect the complexity and structure of the underlying task. By automatically discovering these patterns, StradNet reduces the need for manual

trial-and-error design and produces compact architectures that match the requirements of each dataset.

5.3 A Comparative Study on Time Efficiency and Scalability

In most software systems, particularly real-time systems that handle large volumes of sensor data, rapid retraining and fast response times are critical for adapting to changing environments and preventing failures. In some scenarios, delays can be life-threatening. For example, in an industrial plant that relies on alarm algorithms to detect harmful substances, even a short delay of a few minutes could result in equipment damage or, in the worst case, casualties. Similarly, autonomous vehicles, medical monitoring devices, and maritime surveillance systems all depend on timely updates to remain effective in dynamic conditions. Faster retraining and more responsive software can therefore save both resources and human lives in situations where slower systems could lead to costly failures. The above examples highlight the practical importance of efficient model adaptation in real-world operational settings.

This section presents a case study comparing the time efficiency and scalability of StradNet with a conventional DNN pruning approach, in which 10% of the neurons are removed at each step until accuracy decreases by a specified amount (i.e., 1%). For a given dataset *DS*, we record the total time required to construct a suitable DNN architecture with two hidden layers using both approaches. As defined in (5), the total time, T_{DS_total}, is the sum of the times for each pruning round, consisting of three components: the training or retraining time T_{train_i}, the pruning time $T_{pruning\_i}$, and the testing time $T_{testing\_i}$, where $1 \leq i \leq k$ and k is the number of pruning rounds. The primary purpose of model testing is to determine when performance begins to degrade and when the pruning process should stop.

$$T_{DS\_total} = \sum_{i=1}^{k} (T_{train\_i} + T_{pruning\_i} + T_{testing\_i}) \quad (5)$$

In the conventional approach, the structured-pruned DNNs are FC-DNNs, and at each step, the pruned networks are retrained from scratch. Each sample is trained and evaluated using the same model configuration, with hidden layer sizes of 512 and 256. The purpose of this setup is to examine the effects of dataset size reduction on each pruning algorithm. Modifying the neural network structure as the dataset shrinks would prevent a direct comparison between samples using the same algorithm. This study evaluates the time efficiency of StradNet models relative to the conventional pruning approach and demonstrates that StradNet exhibits greater scalability with increasing dataset size. As mentioned previously, our method removes weak connections and isolated neurons via magnitude-based unstructured global pruning, a directed methodology that eliminates neurons and parameters contributing least to overall performance. For this experiment, we selected the Adversarial Vessels dataset, which contains a large number of complex data points, to assess the efficiency of our automated approach.

Figure 10 compares time efficiency and scalability between the StradNet approach (blue line) and the conventional structured pruning approach (orange line) as the number of data points varies. From the figure, we observe a small gap between the two approaches when the number of data points is below approximately 50,000. As the dataset grows beyond 50,000 points, the time difference increases rapidly, reaching about 35 minutes for the conventional approach and 12.5 minutes for StradNet when the number of new data points is around 120,600. These results suggest that StradNet models are particularly well-suited for large-scale datasets, offering significant time savings and improved scalability compared to conventional pruning methods. This outcome is expected because the StradNet approach requires less time to identify a suitable network structure, preserving weight values across pruning iterations and dynamically shortening training epochs as the network achieves accurate predictions. In contrast, conventional networks reinitialize weights after each pruning step, making it harder to capture dataset patterns and effectively forcing the network to "start over" during each iteration. Additionally, as demonstrated in previous experiments, StradNet produces a more concise and efficient partially connected network structure, further contributing to faster response times and better scalability for growing datasets.

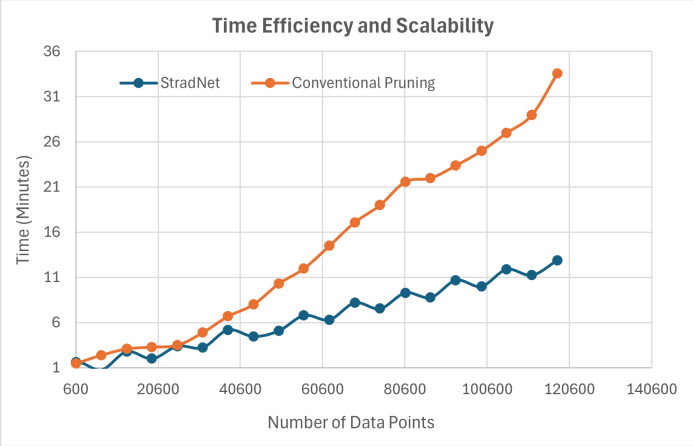


Figure 10: Time efficiency and scalability under varying dataset sizes

These improvements become increasingly important in real-time or continuously updated systems, where models must be retrained frequently to reflect new information. In such environments, even small reductions in retraining time accumulate into substantial long-term savings. Furthermore, the stability of the StradNet pruning strategy ensures that model performance does not degrade as data volume increases, enabling it to maintain both accuracy and responsiveness under demanding operational conditions.

6. Conclusions and Future Work

As DNNs continue to advance and achieve better performance on benchmark tests and practical applications, they also exhibit increasing complexity, storage requirements, and energy consumption. In this paper, we presented StradNet, an efficient and scalable framework for converting an FC-DNN into a reliable PC-DNN capable of generalizing and performing well on unseen data. This approach effectively reduces the resource demands of large DNNs while significantly lowering associated deployment costs. Experiments on dynamic marine datasets demonstrate that a multi-step pruning iteration combined with iterative fine-tuning produces PC-DNNs that outperform their FC-DNN counterparts in prediction accuracy, storage efficiency, and computation time. By leveraging this adaptive pruning strategy, StradNet can be

extended to larger and deeper architectures with billions of parameters, such as LLMs [46], enabling them to potentially operate on resource-limited edge devices. Our experimental results also indicate that there is no universal pattern for initializing efficient DNNs; performance depends strongly on the specific application domain and dataset characteristics.

To further illustrate StradNet's efficiency and scalability, we compared it with a conventional pruning method and observed that it consistently identified effective network structures substantially faster across datasets of increasing size and complexity. Together, these findings provide compelling evidence that adaptive structural pruning is not merely an optimization technique but a robust design principle capable of guiding the development of streamlined neural architectures. These enhancements highlight the potential of StradNet as a practical framework for reducing model complexity while preserving predictive fidelity, supporting efficient retraining, and improving scalability in data-intensive or resource-constrained environments.

Future work in this area could focus on identifying effective values for additional hyperparameters or model parameters. In this paper, we primarily focus on compressing the number of hidden neurons while keeping other hyperparameters fixed to evaluate the benefits of hidden neuron reduction. Advanced hyperparameter tuning could involve finding optimal structures via dynamic learning rate adjustments or reducing the number of training epochs as accuracy improves. At a later stage, combining multiple automated structure-search algorithms could work interchangeably to generate the most effective network hyperparameter set. Other promising directions for structurally adaptive neural networks that could enhance StradNet include dynamic expansion, intelligent pruning schemes, and adaptive loop scheduling [47]-[49]. Dynamic expansion involves adding neurons to enable bidirectional adaptation; although StradNet already implements this in the fine-tuning step, multiple iterations may offer additional benefits. Currently, StradNet uses a magnitude-based pruning scheme, but a more sophisticated criterion could preserve important connections more reliably [37]. In addition, adaptive loop scheduling could accelerate convergence by varying the pruning ratio, enabling the network to prune more aggressively in certain iterations, achieve an effective structure more quickly, and gain greater flexibility. Finally, we plan to extend the StradNet framework to more complex DNNs, including CNN architectures [50] and transformer-based models [51], and to evaluate it on non-marine datasets from dynamic environments to further assess robustness and demonstrate broader generalization across domains.

Conflict of Interest

The authors declare no conflict of interest.

Acknowledgment

We thank the editors and the anonymous referees for their careful review of this paper and the many valuable suggestions they provided. This material is based on work supported by the Office of Naval Research (ONR) under the MUST IV Grant at the University of Massachusetts Dartmouth.

References

[1] S. Natarajan, S. A. R. Al-Haddad, F. A. Ahmad, R. Kamil, M. K. Hassan, S. Azrad, J. F. Macleans, S. H. Abdulhussain, B. M. Mahmmod, N. Saparkhojayev, et al., "Deep neural networks for speech enhancement and speech recognition: A systematic review," Ain Shams Engineering Journal, **16**(7), 103405, 2025, doi:10.1016/j.asej.2025.103405.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," Communications of the ACM, **60**(6), 84-90, 2017, doi:10.1145/3065386.

[3] S. Bhat, H. Xu, and J. Carberry, "Automated medical coding using a hybrid decision tree with deep learning nodes," in Proceedings of the 11th IEEE International Conference on Big Data Computing Service and Machine Learning Applications (IEEE BigDataService 2025), 81-88, Tucson, Arizona, USA, 2025, doi:10.1109/BigDataService65758.2025.00017.

[4] M. Stinchcombe and H. White, "Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions," International 1989 Joint Conference on Neural Networks, **1**, 613-617, Washington, DC, USA, 1989, doi:10.1109/IJCNN.1989.118640.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," Communications of the ACM, **60** (6), 84-90, May 2017, doi:10.1145/3065386.

[6] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), **2**, 3104-3112, December 2014, doi:10.5555/2969033.2969173.

[7] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., "Language models are few-shot learners," in Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS), Article No.: 159, 1877-1901, December 2020, doi:10.5555/3495724.3495883.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS), 6000-6010, December 2017, doi:10.5555/3295222.3295349.

[9] D. Chenna, "Why the latest AI model isn't always best for edge AI," IEEE Spectrum, July 20, 2025. Retrieved on September 15, 2025 from: https://spectrum.ieee.org/edge-ai

[10] X. Hu, L. Chu, J. Pei, W. Liu, and J. Bian, "Model complexity of deep learning: a survey," Knowledge and Information Systems, **63**, 2585-2619, 2021, doi:10.1007/s10115-021-01605-0.

[11] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," Journal of Machine Learning Research, **13**, 281-305, Feb. 2012, doi:10.5555/2188385.2188395.

[12] C. Guo, Y. Qiu, J. Leng, X. Gao, C. Zhang, Y. Liu, F. Yang, Y. Zhu, and M. Guo, "SQuant: on-the-fly data-free quantization via diagonal hessian approximation," in Proceedings of the 10th International Conference on Learning Representations (ICLR 2022), 2269-2286, 2023.

[13] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," in Proceedings of the 29th International Conference on Neural Information Processing Systems (NIPS' 15), **1**, 1135–1143, 2015, doi:10.5555/2969239.2969366.

[14] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in Proceedings of the 5th International Conference on Learning Representations (ICLR 2017), 2017, doi:10.48550/arXiv.1611.01578.

[15] W. Roth, G. Schindler, B. Klein, R. Peharz, S. Tschiatschek, H. Fröning, F. Pernkopf, and Z. Ghahramani, "Resource-efficient neural networks for embedded systems: a survey," The Journal of Machine Learning Research, **25** (1), 2506-2556, 2024, doi:10.5555/3722577.3722627.

[16] M. A. K. Raiaan, S. Sakib, N. M. Fahad, A. Al Mamun, Md. A. Rahman, S. Shatabda, and Md. S. H. Mukta, "A systematic review of hyperparameter optimization techniques in convolutional neural networks," Decision Analytics Journal, **11**, 100470, 2024, doi:10.1016/j.dajour.2024.100470.

[17] D. H. Shin, D. H. Ko, J. W. Han and T. E. Kam, "Evolutionary reinforcement learning for automated hyperparameter optimization in EEG classification," in Proceedings of the 2022 10th International Winter Conference on Brain-Computer Interface (BCI), 1-5, Gangwon-do, Korea, Republic of, 2022, doi:10.1109/BCI53720.2022.9734935.

[18] K. Khadka, J. Chandrasekaran, Y. Lei, R. N. Kacker, and D. R. Kuhn, "A combinatorial approach to hyperparameter optimization," in Proceedings of

the 2024 IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI (CAIN), 140-149, Lisbon, Portugal, 2024.

[19] P. Neary, "Automatic hyperparameter tuning in deep convolutional neural networks using asynchronous reinforcement learning," in Proceedings of the 2018 IEEE International Conference on Cognitive Computing (ICCC), 73-77, San Francisco, CA, USA, 2018, doi:10.1109/ICCC.2018.00017.

[20] J. Qiu, Y. Zhao, and W. Li, "Neural architecture search method based on improved Monte Carlo tree search," in Proceedings of the 2023 China Automation Congress (CAC), 9033-9037, Chongqing, China, 2023, doi:10.1109/CAC59555.2023.10451718.

[21] T. Kim, Y. Na, and S. Park, "Multi-head convolutional neural network compression based on high-order principal component analysis," in Proceedings of the 2023 International Conference on Electronics, Information, and Communication (ICEIC), 1-4, Singapore, 2023, doi:10.1109/ICEIC57457.2023.10049909.

[22] H. Yang, Y. Liang, X. Guo, L. Wu, and Z. Wang, "Random pruning over-parameterized neural networks can improve generalization: a training dynamics analysis," Journal of Machine Learning Research, **26**(84), 1-51, April 2025.

[23] I. Salehin, Md. S. Islam, P. Saha, S.M. Noman, A. Tuni, Md. M. Hasan, and Md. Abu Baten, "AutoML: a systematic review on automated machine learning with neural architecture search," Journal of Information and Intelligence, **2**(1), 52-81, 2024, doi:10.1016/j.jiixd.2023.10.002

[24] J. Hu, P. Lin, H. Zhang, Z. Lan, W. Chen, and K. Xie, "A dynamic pruning method on multiple sparse structures in deep neural networks," IEEE Access, **11**, 38448-38457, 2023, doi:10.1109/ACCESS.2023.3267469.

[25] X. Huang, J. Zhou, X. Yan, W. Ye, and X. He, "A pruning method for echo state network based on neuron importance and iterative fine-tuning," in Proceedings of the 2023 China Automation Congress (CAC), 1034-1039, Chongqing, China, 2023, doi:10.1109/CAC59555.2023.10450422.

[26] L. K. Kalyanam and S. Katkoori, "Unstructured pruning for multi-layer perceptrons with tanh activation," in Proceedings of the 2023 IEEE International Symposium on Smart Electronic Systems (iSES), 69-74, Ahmedabad, India, 2023, doi:10.1109/iSES58672.2023.00025.

[27] W. Na, K. Liu, W. Zhang, F. Feng, H. Xie, and D. Jin, "Automated multilayer neural network structure adaptation method with L1 regularization for microwave modeling," IEEE Microwave and Wireless Components Letters, **32**(7), 815-818, July 2022, doi:10.1109/LMWC. 2022.3153058.

[28] L. Yang and D. Fan, "Dynamic neural network to enable run-time trade-off between accuracy and latency," in Proceeding of the 2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC), 587-592, Tokyo, Japan, January 2021.

[29] R. Ming, H. Xu, S. E. Gibbs, D. Yan, and M. Shao, "A deep neural network based approach to building budget-constrained models for big data analysis," in Proceedings of the 17th International Conference on Data Science (ICDATA'21), 1-8, Las Vegas, Nevada, USA, July 26-29, 2021, doi:arxiv.org/pdf/2302.11707.

[30] P. Wu, "Research on improved BP algorithm by computer simulation based on adaptive learning rate," in Proceedings of the 2023 3rd Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS), 9-12, Shenyang, China, 2023, doi:10.1109/ACCTCS58815. 2023.00138.

[31] B. Noordijk, M. L. G. Gomez, K. H. Tusscher, D. de Ridder, A. D. J. van Dijk, and R. W. Smith, "The rise of scientific machine learning: a perspective on combining mechanistic modelling with machine learning for systems biology," Frontiers in Systems Biology, **4**, August 2024, doi:10.3389/fsysb.2024.1407994.

[32] G. Minai, D. Bobeldyk, and J. P. Leidig, "Evaluating the impact of diverse marine environments on image classification performance," in Proceedings of the OCEANS 2024 - Halifax, 1-4, Halifax, NS, Canada, September 2024, doi: 10.1109/OCEANS55160.2024.10753966.

[33] A. A. Patil and K. Kulkarni, "A hybrid machine learning - numerical weather prediction approach for rainfall prediction," in Proceedings of the 2023 IEEE India Geoscience and Remote Sensing Symposium (InGARSS), 1-4, Bangalore, India, 2023, doi:10.1109/InGARSS59135.2023.10490397.

[34] W. Lin, T. Li, and X. Li, "Deep learning-based object detection for environmental monitoring using big data," Frontiers in Environmental Science, **13**, June 2025, doi:10.3389/fenvs.2025.1566224.

[35] H. Xu and A. Gade, "Smart real estate assessments using structured deep neural networks," in Proceedings of the 2017 IEEE International Conference on Smart City Innovations (IEEE SCI 2017), 1126-1132, San Francisco, CA, USA, August 4-8, 2017, doi:10.1109/UIC-ATC.2017.8397560.

[36] G. J. Sawale and M. K. Rawat, "Stock market prediction using sentiment analysis and machine learning approach," in Proceedings of the 2022 4th International Conference on Smart Systems and Inventive Technology (ICSSIT), 1-6, Tirunelveli, India, 2022, doi:10.1109/ICSSIT53264. 2022.9716326.

[37] J. Frankle and M. Carbin, "The lottery ticket hypothesis: finding sparse, trainable neural networks," in Proceeding of the Seventh International Conference on Learning Representations (ICLR 2019), New Orleans, USA, May 2019, doi:10.48550/arXiv.1803.03635.

[38] S. Liu, I. Ni'mah, V. Menkovski, D. Mocanu and M. Pechenizkiy, "Efficient and effective training of sparse recurrent neural networks," Neural Computing and Applications, **33**, 9625-9636, 2021, doi:10.1007/s00521-021-05727-y.

[39] GFW, "Datasets and code," Global Fishing Watch (GFW), 2024. Retrieved on December 1, 2024 from https://globalfishingwatch.org/data-download/datasets/public-training-data-v1

[40] D. Elreedy, A. Atiya, and Firuz Kamalov, "A theoretical distribution analysis of synthetic minority oversampling technique (SMOTE) for imbalanced learning," Machine Learning, **113**, 4903-4923, 2024, doi:10.1007/s10994-022-06296-4.

[41] T. Masters, Practical Neural Network Recipes in C++. San Diego, CA, USA: Academic Press, 1993.

[42] A. Khan, "Oil spill - imbalanced classification," Kaggle, 2024. Retrieved on December 1, 2024 from https://www.kaggle.com/code/ashrafkhan94/oil-spill-imbalanced-classification

[43] A. Ilyas, S. Santurkar, D. Tsipras, L. Engstrom, B. Tran, and A. Mądry, "Adversarial examples are not bugs, they are features," in Proceedings of the 33rd International Conference on Neural Information Processing Systems, 125-136, Red Hook, NY, USA, 2019, doi:10.5555/3454287.3454299.

[44] G. Patrini, A. Rozza, A. K. Menon, R. Nock, and L. Qu, "Making deep neural networks robust to label noise: a loss correction approach," in Proceeding of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2233-2241, Honolulu, HI, USA, July 21-26, 2017, doi:10.1109/CVPR.2017.240.

[45] UC Irvine, "Connectionist bench (sonar, mines vs. rocks)," UCI Machine Learning Repository, Retrieved on December 1, 2024 from https://archive.ics.uci.edu/dataset/151/connectionist+bench+sonar+mines+vs+rocks

[46] M. Arya and Y. Simmhan, "Understanding the performance and power of LLM inferencing on edge accelerators," in Proceedings of the 2025 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 1108-1111, Milano, Italy, 2025, doi:10.1109/IPDPSW66978. 2025.00173.

[47] J. Guo, C. L. P. Chen, Z. Liu and X. Yang, "Dynamic neural network structure: a review for its theories and applications," IEEE Transactions on Neural Networks and Learning Systems, **36**(3), 4246-4266, March 2025, doi:10.1109/TNNLS.2024.3377194.

[48] A. Heyman and J. Zylberberg, "Fine granularity is critical for intelligent neural network pruning," Neural Computation, **36**(12), 2677-2709, November 2024, doi:10.1162/neco_a_01717.

[49] F. Kasielke, R. Tschüter, C. Iwainsky, M. Velten, F. M. Ciorba and I. Banicescu, "Exploring loop scheduling enhancements in OpenMP: an LLVM case study," in Proceedings of the 2019 18th International Symposium on Parallel and Distributed Computing (ISPDC), 131-138, Amsterdam, Netherlands, 2019, doi:10.1109/ISPDC.2019.00026.

[50] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," Artificial Intelligence Review, **53**(8), 5455-5516, December 2020, doi:10.1007/s10462-020-09825-6.

[51] P. Xu, X. Zhu, and D. A. Clifton, "Multimodal learning with transformers: a survey," IEEE Transactions on Pattern Analysis and Machine Intelligence, **45**(10), 12113-12132, October 2023, doi:10.1109/TPAMI.2023.3275156.