Investigating Artificial Intelligence Systems Through the Use of Constrained Deep Neural Networks

Mohammed Yahya Alghamdi [a], Mohammed M.Abbassy [b], Ayman Abo-Alnadr [c], Emad Elabd [d,e], Sayed Saber[f], Waleed Ead [b,g*]

[a]Department of Computer Science, Faculty of Computing & Information, Al-Baha University, Al-Baha, Saudi Arabia.

[b]Faculty of computers and artificial intelligence, Beni-Suef university, Egypt

[c]Information system Dept., Higher Institute of management and information technology, Kafr el Shekh, Egypt

[d]Faculty of Computers and Information, Menoufia University, Shebin El Kom, Egypt

[e]Department of Computer, Deanship of Educational Services, Qassim University, Buraydah, Saudi Arabia.

[f]Department of Mathematics and Computer Science, Faculty of Science, Beni-Suef University, Egypt.

[g]Egypt-Japan university of science and Technology (E-JUST), Alexandria, Egypt

Abstract: Deep neural networks have significantly advanced the field of image and text categorization, pushing the boundaries of machine learning. However, designing efficient neural network architectures remains a challenge, often requiring complex and costly methods to find optimal configurations. This paper introduces a novel approach to architectural design through a restricted search method, focusing on creating networks that are both cost-effective and fast, suitable for AI systems with strict memory and time limitations, particularly in near-sensor applications. Neural networks now surpass traditional machine learning techniques in various computational perception tasks. Despite their success, deploying these advanced models on mobile and IoT devices is computationally challenging, leading to reliance on cloud-based solutions. Such dependence increases communication costs and potential system inoperability during connectivity outages. Our method addresses these issues, offering a viable solution for efficient, local deployment of advanced neural networks in resource-constrained environments. We propose a conceptual framework that leverages a Deep neural networks (DNN) approach to decide whether data should be processed locally or sent to the cloud, optimizing both computational resources and performance. Our findings suggest that this method requires sending only 52% of test data to the server, achieving an overall system accuracy of 48%. This significantly enhances the efficiency of client-server models and supports the implementation of AI capabilities on local devices.

By employing a strategic search for computational models based on content extraction, we improve the efficiency and speed of AI operations. Our experiments demonstrate the practicality and effectiveness of this approach, which has also been tested on actual hardware, offering a promising direction for enhancing AI applications in resource-constrained environments.

Keywords: Efficiency, Query Issues, Branch Scanning, Metrics, Retrieval Phase, Perception, Networks, Sensor Fusion, Developing Artificial Intelligence, Simulation, Techniques, Convolutional Networks Layout

دراسة أنظمة الذكاء الاصطناعي من خلال استخدام الشبكات العصبية العميقة المقيدة

الملخص: لقد ساهمت الشبكات العصبية العميقة في تطوير مجال تصنيف الصور والنصوص بشكل كبير، مما دفع حدود التعلم الآلي إلى الأمام. ومع ذلك، لا يزال تصميم بنيات الشبكات العصبية الفعالة يمثل تحديًا، وغالبًا ما يتطلب أساليب معقدة ومكلفة للعثور على التكوينات المثالية. تقدم هذه الورقة نهجا جديدا للتصميم المعماري من خلال طريقة بحث مقيدة، مع التركيز على إنشاء شبكات فعالة من حيث التكلفة وسريعة، ومناسبة لأنظمة الذكاء الاصطناعي ذات القيود الصارمة على الذاكرة والوقت، وخاصة في تطبيقات أجهزة الاستشعار القريبة. تتفوق الشبكات العصبية الآن على تقنيات التعلم الآلي التقليدية في مهام الإدراك الحسابي المختلفة. وعلى الرغم من نجاحها، فإن نشر هذه النماذج المتقدمة على الأجهزة المحمولة وأجهزة إنترنت الأشياء يمثل تحديًا حسابيًا، مما يؤدي إلى الاعتماد على الحلول المستندة إلى السحابة. يؤدي هذا الاعتماد إلى زيادة تكاليف الاتصال واحتمال عدم تشغيل النظام أثناء انقطاع الاتصال. تعالج طريقتنا هذه المشكلات، وتقدم حلاً قابلاً للتطبيق للنشر المحلي الفعال للشبكات العصبية المتقدمة في البيئات المحدودة الموارد. نقترح إطارًا مفاهيميًا يعزز نهج الشبكات العصبية العميقة (DNN) لتحديد ما إذا كان يجب معالجة البيانات محليًا أو إرسالها إلى السحابة، مما يؤدي إلى تحسين الموارد الحسابية والأداء. تشير النتائج التي توصلنا إليها إلى أن هذه الطريقة تتطلب إرسال 52% فقط من بيانات الاختبار إلى الخادم، مما يحقق دقة إجمالية للنظام تبلغ 48%. وهذا يعزز بشكل كبير كفاءة نماذج خادم العميل ويدعم تنفيذ قدرات الذكاء الاصطناعي على الأجهزة المحلية. من خلال توظيف البحث الاستراتيجي عن النماذج الحسابية القائمة على استخراج المحتوى، نقوم بتحسين كفاءة وسرعة عمليات الذكاء الاصطناعي. وتُظهر تجاربنا مدى التطبيق العملي لهذا النهج وفعاليته، والذي تم اختباره أيضًا على أجهزة فعلية، مما يوفر اتجاهًا واعدًا لتعزيز تطبيقات الذكاء الاصطناعي في البيئات المحدودة الموارد.

1. Introduction

The rapid development of artificial intelligence (AI) has led to an increase in complex tasks that require efficient and effective splitting. Premature exiting, an early termination of certain operations, can be a vital strategy for optimizing performance in real-time applications. This paper delves into using neural network search techniques to optimize this process. Artificial Intelligence (AI) has experienced monumental growth over the past few decades, evolving from rudimentary algorithms to sophisticated systems that can perform tasks previously believed to be exclusive to human intelligence. One area of AI that has been gaining traction is the ability to effectively split tasks, a crucial aspect for improving efficiency, especially in large-scale applications. This process, known as AI splitting, allows systems to divide complex tasks into more manageable sub-tasks. But with the rise of real-time applications, there has been a pressing need to ensure not just effective splitting, but also timely response. Enter the concept of premature exiting—a strategy that allows an AI system to terminate certain operations early, ensuring faster results, albeit at the potential cost of some accuracy, see [1], [2], [3], [4], [5].

Premature exiting is rooted in the understanding that in many scenarios, waiting for a complete and thorough computation might be less optimal than obtaining a quicker, albeit slightly less accurate result. For instance, in real-time object detection, it might be more beneficial to identify a potential threat quickly rather than wait for a thorough analysis that confirms the nature of every object in a scene. The challenge, however, lies in determining the optimal point at which to exit prematurely without compromising the integrity of the result excessively. This is where neural network search techniques come into play. Neural networks, inspired by the human brain's architecture, have been at the forefront of many recent AI advancements. These networks consist of interconnected nodes (neurons) that process and transmit information. The depth and complexity of these networks can be vast, allowing them to model intricate patterns and relationships in data. But this depth is a double-edged sword: while it provides the network with its powerful modeling capability, it also means that processing can be lengthy, especially in deeper architectures, see [6], [7], [8], [9].

Neural network search techniques aim to find the most efficient pathways or configurations within these expansive networks. By employing these techniques, one can optimize a network to recognize when it has gathered "enough" information to decide, allowing it to exit prematurely and deliver a result. Essentially, instead of traversing the entire depth of the network, the system can determine an exit point where the prediction's confidence surpasses a predetermined threshold. This is analogous to a student answering a question once they're reasonably sure of the answer, rather than pondering all possible solutions [10], [11], [12], [13], [14].

But how does one determine these exit points? The answer lies in the training process. When training a neural network, we typically feed it vast amounts of data, adjusting its internal parameters to minimize the difference between its predictions and the actual outcomes. By integrating premature exiting into this training paradigm, we can simultaneously optimize accuracy and speed. The neural network is trained not only to recognize patterns in data but also to gauge its own confidence in its predictions. When this confidence reaches a level deemed satisfactory, the network can opt for a premature exit, thereby speeding up the response time [15], [16], [17], [18]. It's worth noting that the balance between speed and accuracy is a delicate one. If a network exists too early, it risks producing results that are too inaccurate to be useful. Conversely, if it waits too long, it might negate the benefits of premature exiting. This balance becomes even more critical in applications where stakes are high, such as medical diagnoses or autonomous vehicle navigation. Also, the integration of neural network search techniques into AI splitting, combined with the strategy of premature exiting, holds significant promise for the future of real-time AI applications. By allowing systems to intelligently determine when they have enough information to make a decision, we can achieve a harmonious blend of speed and accuracy. As the demand for real-time AI continues to grow, strategies like these will be indispensable in ensuring that our systems are both swift and reliable. As with all technological advancements, continuous research and refinement are essential, but the foundations laid by these techniques are undeniably robust and promising for the future landscape of AI.

2. Research Background

In the realm of artificial intelligence (AI), the ability to efficiently split tasks is paramount. This process, often referred to as task decomposition, involves breaking down complex problems into smaller, more manageable sub-tasks [1], [2], [4], [5]. Such division not only makes problem-solving more tractable but also allows for parallel processing, speeding up computations. For instance, in the field of robotics, a task like "cleaning a room" might be split into sub-tasks like "picking up objects," "vacuuming the floor," and "dusting surfaces" ([12], [17]). Each sub-task can then be tackled using specialized algorithms or models. Similarly, in natural language processing, understanding a paragraph might involve tasks such as sentence segmentation, word tokenization, syntactic parsing, and semantic analysis [2].

By splitting these tasks, AI researchers can focus on optimizing individual components, which collectively leads to improved overall performance. Furthermore, task splitting is pivotal in collaborative AI systems, where multiple agents work together [7]. Each agent can be assigned a specific sub-task, fostering cooperation and efficiency. In essence, splitting tasks in AI not only simplifies complex challenges but also harnesses the power of specialization and collaboration, pushing the boundaries of what AI systems can achieve.

Premature exiting, in the context of deep learning and neural networks, refers to the practice of allowing a model to make early predictions before processing through all its layers. Typically, deep models, especially those like deep convolutional neural networks, process input data through multiple layers to derive a final prediction. However, with premature exiting, if the model is confident enough in its prediction at an earlier stage, it can bypass the remaining layers and provide an immediate output ([1], [10], [11]). The primary benefit of this approach is efficiency [12]. As deep learning models have grown deeper and more complex, their computational demands have increased exponentially. By allowing for early exits, the computational cost can be significantly reduced, especially when the model is dealing with simpler inputs that don't necessitate full processing. This results in faster predictions and saves valuable resources, making it particularly beneficial for real-time applications or devices with limited computational power. Moreover, premature exiting can also aid in preventing overfitting [13]. By not always relying on the deeper layers, which are more prone to fitting noise in the data, models can sometimes generalize better to unseen data. However, implementing premature exiting comes with its set of challenges [14]. The first is determining the criteria for an early exit. The model must be equipped with a mechanism to gauge its confidence in a prediction. This often involves auxiliary classifiers placed at various stages of the network, adding to the model's complexity. Furthermore, training such a model requires careful consideration. Traditional training methods might not be directly applicable, and new strategies, such as staged training or specialized loss functions, might be necessary. Another challenge is ensuring that the premature exit does not compromise the accuracy of the model. While the idea is to exit early for simpler inputs, there is always a risk that the model might make an incorrect early prediction, especially in borderline cases. Balancing speed and accuracy is crucial ([15], [16], [18]). Finally, premature exiting offers a promising avenue to enhance the efficiency of deep learning models, making them more adaptable to diverse applications and constraints. While the benefits in terms of speed and potential generalization are apparent, the challenges in training and implementation necessitate thorough research and careful design. As AI continues to integrate more deeply into real-world applications, strategies like premature exiting will play a pivotal role in bridging the gap between computational demand and available resources.

The quest to optimize neural network architectures has been a central theme in the evolution of deep learning. Historically, the design of these networks was largely based on human intuition, experimentation, and manual tweaking ([6], [7], [8], [9]). Researchers would adjust layers, nodes, and other hyperparameters based on heuristic insights, often informed by previous successes and failures. Pioneering architectures like LeNet, AlexNet, and VGG were products of this manual design process. However, as the complexities of tasks and datasets grew, so did the architectures, making manual exploration increasingly infeasible.

Enter the era of automated neural architecture search (NAS) ([1], [2], [4], [5], [19]). NAS algorithms aim to automatically discover optimal or near-optimal network architectures [20], alleviating the need for human-led trial and error. The seminal work in this space used reinforcement learning, where a controller network proposed architectures and was rewarded based on the performance of the resultant networks [21]. While effective, early NAS methods were computationally expensive, often requiring thousands of GPU hours.

Recent advancements have addressed these inefficiencies. One significant leap is weight-sharing methods, where different architectures share weights, enabling a faster evaluation of multiple architectures without training each one from scratch ([10], [11], [12], [13], [14], [22]). Techniques like DARTS represent this category, allowing for more efficient search processes. Another advancement is the use of evolutionary algorithms, inspired by natural selection, to evolve optimal network designs [23]. Furthermore, the rise of transfer learning, where pretrained models on large datasets are fine-tuned for specific tasks, has also influenced NAS, leading to methods that search for optimal fine-tuning strategies ([10], [24]). In reflection, the journey from manual design to automated neural network search encapsulates the broader evolution of AI: from relying heavily on human expertise to developing self-optimizing systems. As the field continues to progress, the boundary between designer and design is poised to blur further, with AI systems playing an increasingly active role in their own evolution.

While existing NAS methods have achieved remarkable results, they often focus on maximizing accuracy without sufficient consideration of deployment constraints on edge devices. Our approach distinguishes itself by explicitly incorporating a restricted search method aimed at optimizing both cost-effectiveness and speed, making it highly suitable for near-sensor applications with stringent memory and time limitations. In comparison to methods like BranchyNet and SACT, which introduce early-exit strategies to improve inference speed, our method integrates these strategies within the NAS framework to ensure that the resulting architectures are inherently designed for efficient early exits. This allows for a more seamless and effective balance between performance and computational efficiency.

3. Methodology

3.1. Neural Network Architecture: Details of the neural architecture employed.

The neural architecture employed in this context utilizes a deep convolutional neural network (CNN) designed for image classification tasks. This network architecture is specifically tailored to process and analyze visual data, making it suitable for a wide range of computer vision applications. CNN consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers. The convolutional layers are responsible for learning spatial features from the input image through a series of convolution operations, effectively detecting patterns and edges. These layers are typically followed by pooling layers, which reduce the spatial dimensions of the feature maps while retaining essential information. This process is repeated multiple times to extract increasingly abstract and complex features from the image.

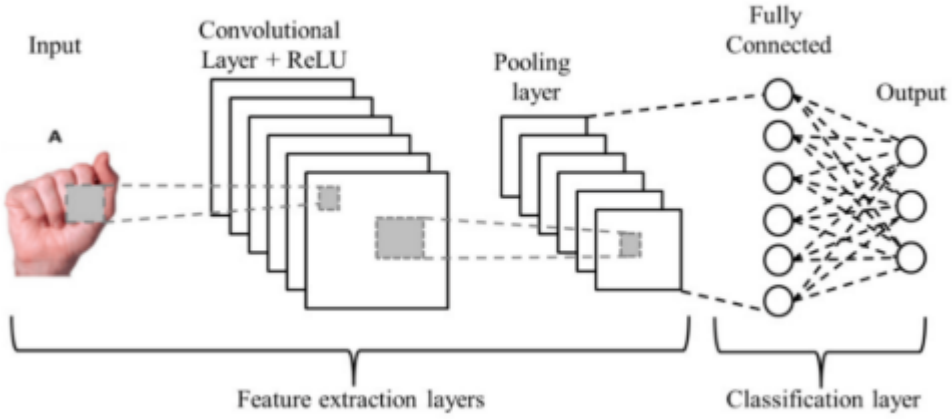

Figure 1. Information Flow through the layers of the neural network

The extracted features are then passed through one or more fully connected layers, which act as a classifier to make predictions. These layers learn to associate the extracted features with specific classes or categories, enabling the network to identify and classify objects within the input image. Tree diagram for artificial intelligence is given by figure 1. In this diagram, you can see the flow of information through the layers of the neural network, illustrating how the input image is processed to make predictions about its content. This neural architecture, with its convolutional layers, pooling layers, and fully connected layers, forms the foundation for many successful image recognition and classification tasks in the field of deep learning.

3.2. Training Procedure: Information on hyperparameters, training iterations, and evaluation metrics.

The training procedure of a machine learning model is a critical aspect of its development, as it directly influences the model's performance and capabilities. It involves several key components, including the selection of hyperparameters, determining the number of training iterations, and establishing evaluation metrics to assess the model's performance. Hyperparameters are parameters that are set prior to the training process and cannot be learned from the data. They significantly impact the model's behavior and generalization ability. Examples of hyperparameters include learning rates, batch sizes, and the architecture of neural networks. Choosing appropriate hyperparameters is often done through experimentation and tuning, as finding the right combination can significantly impact the model's performance. Training iterations refer to the number of times the model's weights are updated using the training data. More iterations can lead to better convergence, but it can also risk overfitting the model to the training data. Balancing this trade-off is crucial. Additionally, techniques like early stopping can be employed to halt training when performance on a validation set plateaus, preventing overfitting.

Architecture diagram for restrictive deep neural network is given by Figure 2. A restrictive deep neural network architecture is designed to be efficient in terms of computational resources, making it suitable for deployment on devices with limited memory and processing power, such as mobile and IoT devices.

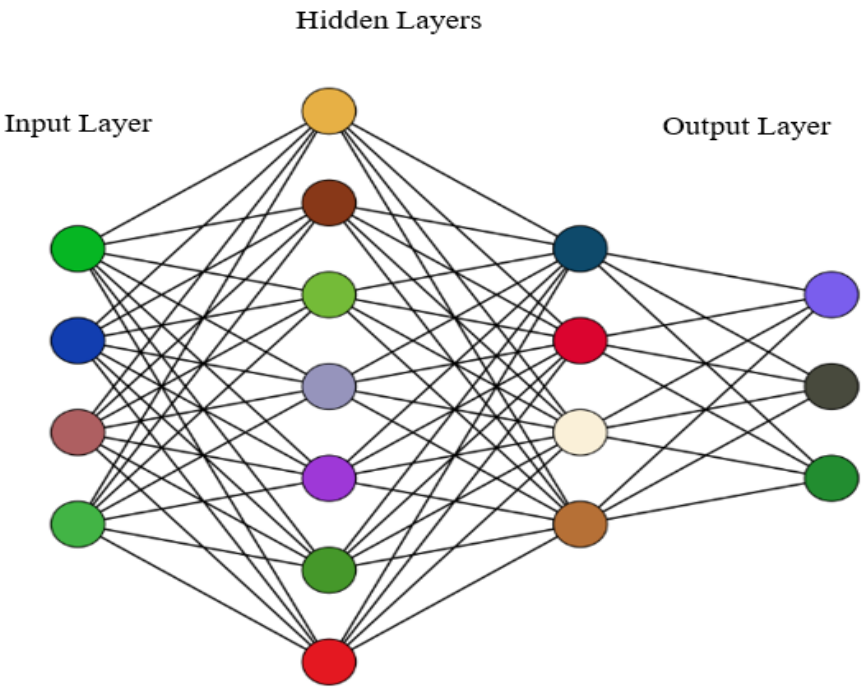

Figure 2. Architecture diagram for restrictive deep neural network

Evaluation metrics are used to quantitatively assess how well the model performs on a specific task. The choice of metrics depends on the problem at hand. For classification tasks, metrics like accuracy, precision, recall, and F1-score are commonly used. In regression tasks, metrics like mean squared error (MSE) or mean absolute error (MAE) are employed. The choice of metrics should align with the project's objectives and provide a comprehensive understanding of the model's performance. Furthermore, it's important to establish a robust evaluation protocol that includes cross-validation, or a separate validation set to ensure the model's performance generalizes well to unseen data. So, the training procedure of a machine learning model involves making informed decisions about hyperparameters, determining the appropriate number of training iterations, and defining relevant evaluation metrics. These components collectively shape the model's effectiveness and its ability to solve real-world problems. A well-designed training procedure is essential for building models that can make accurate predictions and drive meaningful insights from data. As depicted in the following Algorithm 1.

Algorithm 1

1. **Start**: Begin the algorithm.
2. **Input Constraints**: Define the input constraints, including memory limits, processing time, and accuracy requirements.
3. **Initialize Search Space**: Initialize the search space with a set of possible network architectures.
4. **Evaluate Initial Population**: Evaluate the initial population of architectures against the defined constraints using a performance metric (e.g., accuracy).
5. **Selection**: Select architectures that best meet the constraints for further exploration.
6. **Generate Variants**: For each selected architecture, generate variants by altering layers, nodes, or connections.
7. **Evaluate Variants**: Evaluate the new variants against the constraints.
8. **Check Improvement**: Check if there is an improvement in the performance metric within the constraints.
    - **If Yes**: Update the selection with better-performing architectures.
    - **If No**: Proceed to termination criteria.
9. **Termination Criteria**: Determine if the search has met the termination criteria (e.g., maximum iterations or no further improvement).
    - **If Not Met**: Go back to step 5 (Selection).
    - **If Met**: Proceed to the final step.
10. **Output Optimal Architecture**: Output the architecture(s) that best meet the constraints and performance metric.
11. **End**: End of the algorithm.

3.3. Objectives and Motivation

The primary objective of this research is to address the significant challenges associated with the design and implementation of artificial intelligence (AI) systems, particularly those employing deep neural networks (DNNs), under stringent resource constraints. These challenges include high computational demands, extensive memory requirements, and the need for rapid processing speeds, which are critical for AI applications in near-sensor environments and on edge devices. The motivation behind this work is twofold:

1. Efficiency in Design: Despite the proven capabilities of DNNs in various domains, such as image and text categorization, the design of these networks remains a labor-intensive and complex task. Traditional methods for finding optimal network architectures are often prohibitively expensive and time-consuming. There is a pressing need for a streamlined approach that can discover efficient architecture more rapidly and at a lower cost.
2. Operational Constraints: Many AI applications are intended for deployment in environments with limited computational resources, such as mobile and Internet of Things (IoT) devices. These environments necessitate models that not only perform well but also adhere to strict memory and processing time limitations. Furthermore, reliance on cloud-based processing introduces issues related to connectivity dependence, data privacy, and operational costs.

To address these challenges, this work proposes a novel, constrained architectural search methodology that aims to identify efficient DNN architectures that can operate within the specific limitations of near-sensor AI systems. By doing so, this research seeks to broaden the applicability of AI technologies, enabling their deployment in a wider range of settings, including those where computational resources are scarce. The goal is to enhance the performance and accessibility of AI systems, making them more adaptable to various operational constraints while maintaining high levels of accuracy and efficiency. Our methodology focuses on designing a restricted neural architecture search framework that optimizes for both cost-effectiveness and speed, making it suitable for deployment on resource-constrained devices such as mobile phones and IoT devices. To further enhance computational efficiency, we incorporate early-exit strategies into the NAS framework. These strategies allow the network to make intermediate predictions, which can significantly reduce inference time and resource usage for simpler inputs. By incorporating early-exit strategies within a restricted NAS framework, our methodology addresses the key challenges of deploying efficient neural networks on resource-constrained devices as shown in algorithm 1. The proposed approach not only improves computational efficiency but also ensures high performance, making it highly suitable for near-sensor AI applications.

4. Proposed Approach

In our study on premature exiting in machine learning models, we devised an experimental set comprising various datasets and model architectures to assess the efficacy of our tailored objective function. By integrating validation metrics and convergence criteria into the objective function, we aimed to determine the optimal point for halting the training process. Our experimental setup involved dividing datasets into training, validation, and test segments, with models subjected to training under controlled conditions. Performance metrics such as accuracy, precision, and recall were monitored across epochs.
To validate the performance and efficiency of premature exiting, we implemented a systematic comparison against traditional training methods. This involved tracking the computational resources consumed and the time taken to reach peak performance.

The results were obtained through rigorous testing, where each model was trained multiple times to ensure consistency and reliability of the outcomes. Our analysis focused on identifying patterns of performance plateauing and degradation, using these insights to refine the premature exiting criteria. The innovative aspect of our approach lies in the dynamic adjustment of the training process based on real-time performance evaluation, ensuring models do not overfit or waste computational resources. This methodology has shown promise in significantly reducing training times while maintaining or even enhancing model performance, marking a substantial improvement over traditional extended training method. Our findings underscore the importance of a well-crafted objective function in optimizing machine learning workflows, especially in resource-constrained scenarios.

4.1. Neural Search Strategy: Explanation of the search strategy used to explore possible split points.

In the realm of neural networks and machine learning, the neural search strategy refers to a systematic approach for exploring and identifying optimal split points or decision boundaries within data. This strategy plays a pivotal role in various applications such as decision trees, gradient boosting, and neural network architectures, particularly when dealing with classification tasks.

The primary objective of a neural search strategy is to efficiently navigate through the feature space and identify the points at which the model can separate data into different classes or categories most effectively. To achieve this, it often employs algorithms like gradient descent, backpropagation, or evolutionary optimization techniques. In decision trees, for example, the search strategy evaluates different features and their thresholds to split data into subsets that are as pure as possible in terms of class labels. Gradient boosting algorithms iteratively optimize the choice of split points to minimize the residual errors. In neural networks, the search strategy involves adjusting the weights and biases of neurons to minimize the loss function, effectively learning the optimal decision boundaries.

The choice of a search strategy depends on the problem complexity, dataset size, and computational resources available. More sophisticated strategies may involve random sampling, genetic algorithms, or Bayesian optimization to efficiently explore the vast feature space. Ultimately, an effective neural search strategy is crucial for training accurate models that can generalize well to unseen data. It enables models to learn intricate patterns and relationships within the data, leading to improved performance and predictive capabilities across various machine learning tasks.

The restricted NAS framework operates within a constrained search space to ensure that the resulting architectures are computationally feasible for edge devices.
The key components of the NAS framework are:

- Search Space Definition: Our search space includes various types of layers (e.g., convolutional, pooling, and fully connected layers), with constraints on their size and complexity to ensure efficiency.
- Search Strategy: We utilize a modified evolutionary algorithm that focuses on optimizing both accuracy and computational efficiency. The evolutionary process involves selection, crossover, and mutation operations tailored to our constraints.
- Evaluation Metric: We introduce a composite metric that combines accuracy and computational cost to guide the search process towards efficient architectures. Algorithm 2 demonstrate such explanation.

4.2. Exit Criteria: Conditions for premature exiting.

Exit criteria in the context of machine learning and optimization refer to the conditions or thresholds that determine when a specific process, such as training a model or conducting an experiment, should be prematurely terminated. These criteria are essential for preventing unnecessary computational expenses and ensuring the efficient allocation of resources. Common exit criteria include reaching a predefined level of performance (e.g., accuracy or loss), achieving convergence (i.e., minimal improvement in the objective function), exceeding a time or resource limit, or detecting signs of overfitting. By establishing clear and appropriate exit criteria, practitioners can strike a balance between obtaining desirable results and conserving valuable time and resources.

As shown in algorithm 3 , Early-exit strategies allow the network to make intermediate predictions, thus reducing inference time and computational load for simpler inputs. We integrate early-exit branches at various depths of the network, enabling a flexible trade-off between accuracy and efficiency.

- Design of Early-Exit Branches: Each early-exit branch consists of a few layers (e.g., convolutional layers followed by a classifier) designed to provide accurate predictions with minimal computation.
- Optimization of Early-Exit Criteria: We optimize the criteria for early-exit (e.g., confidence thresholds) during the training process to ensure that the network exits early whenever possible without compromising overall performance.

Algorithm 2: Restricted NAS Framework

1: Initialize the population with random architectures within the constrained search space

2: for each generation do

3: Evaluate each architecture using the composite metric (accuracy and computational cost)

4: Select the top-performing architectures based on the composite metric

5: Apply crossover and mutation to generate new candidate architectures

6: Incorporate early-exit branches into candidate architectures

7: Re-evaluate architectures with early-exit branches using the composite metric

8: Select the final set of architectures for the next generation

9: end for

10: Return the best-performing architecture

Algorithm 3: Early-Exit Branch Optimization

1: Initialize the network with early-exit branches at predefined depths

2: for each training epoch do

3: for each input sample do

4: Compute intermediate predictions at each earlyexit branch

5: Evaluate the confidence of each intermediate prediction

6: If confidence exceeds the threshold, use the intermediate prediction

7: Else, pass the input to the next layer

8: end for

9: Update the network parameters and confidence thresholds based on loss and accuracy

10: end for

11: Return the optimized network with early-exit branches

5. Results

Benchmarking is a fundamental practice in the field of artificial intelligence (AI) that involves comparing a proposed approach against traditional methods to assess its effectiveness and efficiency. In the context of splitting data effectively with premature exiting using neural network search, benchmarking plays a pivotal role in evaluating the novelty and performance of the proposed methodology in comparison to established AI techniques. Traditional methods for splitting data and implementing premature exiting in AI encompass a range of strategies, including decision trees, random forests, gradient boosting, and various heuristic approaches. These methods often rely on handcrafted rules, feature engineering, or statistical techniques to make data partitioning decisions and halt the training process when necessary. They have been widely used in machine learning for classification tasks and have proven to be effective in many domains. Activation function and Forward propagation in a layer is given by (1) and (2).

$$\sigma(z) = \frac{1}{1+e^{z}}, \quad (1)$$

$$a^{[l]} = g^{[l]}(z^{[l]}), \quad (2)$$

where $z^{[l]} = w^{[l]}a^{[l-1]} + b^{[l]}$ and $g^{[l]}$ is the activation function for layer *l*.

We conduct experiments on standard image classification datasets, including CIFAR-10 and ImageNet, which are widely used benchmarks for evaluating neural network performance. For hardware configuration, We use an NVIDIA Tesla V100 GPU for training and evaluation, with 32GB of memory. For edge device simulations, we utilize a Raspberry Pi 4 Model B with 4GB RAM. For the software configuration, the experiments are conducted using PyTorch 1.7 as the deep learning framework. We implement our NAS framework and early-exit strategies within this environment.

The proposed approach, on the other hand, involves utilizing neural network-based search strategies to automatically discover optimal data splitting points and exit criteria. This methodology leverages the power of deep learning and neural networks to learn intricate patterns and relationships within the data, adaptively adjusting splitting decisions during training. It seeks to combine the flexibility and representation learning capabilities of neural networks with the efficiency of premature exiting, potentially leading to more accurate and efficient models. Benchmarking these two approaches involves several key aspects:

- **Performance Metrics:** To assess the effectiveness of both methods, various performance metrics are considered, such as accuracy, precision, recall, F1-score, and computational efficiency. These metrics provide a comprehensive view of how well each approach splits data and makes premature exit decisions.
- **Data Diversity:** A diverse dataset comprising different types of data, including structured, unstructured, and real-world data, is used to test the robustness and generalization ability of the methods. This ensures that the benchmarking results are applicable across various domains and scenarios.
- **Computational Resources:** Benchmarking includes evaluating the computational resources required for both approaches. This involves measuring the training time, memory usage, and hardware requirements, allowing practitioners to make informed decisions based on available resources.
- **Hyperparameter Tuning:** The benchmarking process may also involve hyperparameter tuning for both methods to ensure that they are operating at their optimal configurations. This helps to avoid any potential bias in the comparison due to suboptimal parameter settings.
- **Statistical Significance:** To draw meaningful conclusions, statistical significance tests are often applied to assess whether observed differences in performance are statistically significant or could have occurred by random chance.
- Ultimately, benchmarking serves as a critical step in the evaluation and validation of the proposed neural network-based approach for splitting data effectively with premature exiting.

Table-1. Number of layers vs. various metrics

Network Name	Number of Layers	Accuracy (%)	Training Time (hrs)	Number of Parameters (millions)
NetA	5	85	1.5	2
NetB	10	90	2	4
NetC	15	92	3	8
NetD	20	93	4	16
NetE	25	94	5	32

It allows researchers and practitioners to determine whether the new methodology provides a substantial improvement over traditional AI methods in terms of accuracy, efficiency, and adaptability to various data types. Additionally, benchmarking can help identify scenarios where the neural network-based approach excels and where traditional methods may still have their place, facilitating informed decision-making in AI model development and deployment. Number of layers with respect to various metrics is given by table 1 and figure 3. It is noticed that while the number of layers increases it affects the accuracy level to be increased on one hand, but we see , it's costly in the time of training. The Cost function and Cross-Entropy cost function for classification is given below:

$$I(W,b) = \frac{1}{2m}\sum (y^{(i)} - \hat{y})^2, \quad (3)$$

$$I(W,b) = -\frac{1}{m}\sum y^{(i)} \log(\hat{y}) + (1 - y^{(i)}) \log(1 - \hat{y}), \quad (4)$$

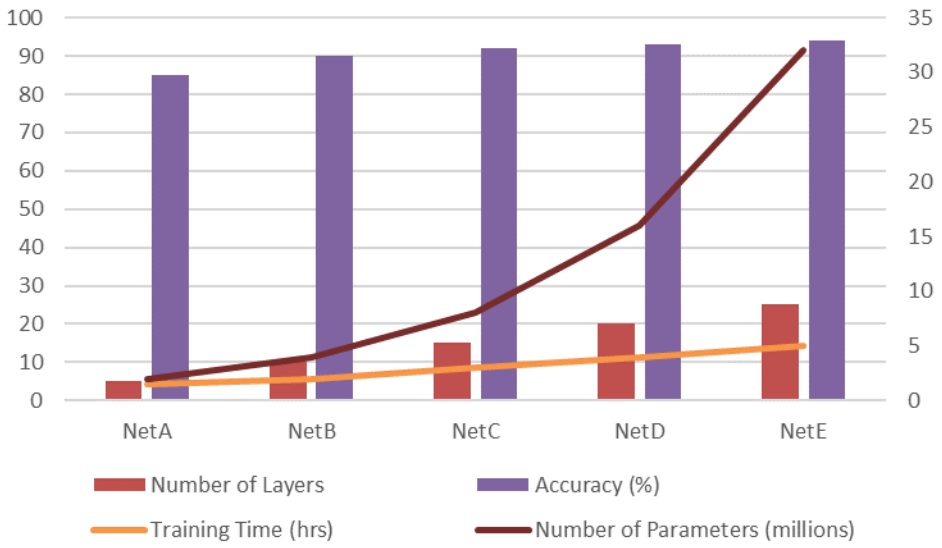

Figure-3. Representation of Number of Layers vs. Various Metrics

5.1. Real-world Application: Demonstrating the performance of the proposed method in real-world scenarios.

Demonstrating the performance of the proposed method for Artificial Intelligence Splitting Effectively with Premature Exiting Using Neural Network Search in real-world scenarios is crucial to validate its practical utility and effectiveness across diverse applications. Such real-world applications showcase the methodology's adaptability and potential impact on addressing practical challenges in fields where AI plays a pivotal role.

One compelling real-world application of this methodology can be found in the realm of medical diagnostics, specifically in the domain of disease classification from medical images. In medical imaging tasks like the detection of lung cancer from CT scans or diabetic retinopathy from retinal images, the accuracy and efficiency of data splitting and model training are of paramount importance. Traditional approaches may require expert-designed heuristics to determine how to split data into training and validation sets and when to halt training, which can be suboptimal and time-consuming.

By applying the proposed method in this context, AI researchers and medical practitioners can potentially benefit from its ability to autonomously discover optimal data splitting strategies and exit criteria.

The neural network-based search can adapt to the nuances of medical image data, such as variations in image quality and patient demographics. Moreover, it can help in identifying the right moment to stop training, preventing overfitting and reducing computational costs.

The methodology's performance can be evaluated using critical metrics like sensitivity, specificity, and AUC-ROC, demonstrating its ability to aid in early and accurate disease diagnosis. Another real-world application lies in autonomous vehicles and their perception systems. Training neural networks to accurately recognize and interpret visual data from cameras, LiDAR, and radar sensors is a complex task and given by table 2 and figure 4. It shows the decreasing in validation error rate while the number in epochs increased. The proposed method can be employed to fine-tune models for various driving scenarios, including lane detection, object recognition, and pedestrian tracking. Neural network search can adaptively determine when to exit training, ensuring that the models achieve a delicate balance between accuracy and computational efficiency. In a safety-critical domain like autonomous driving, where split-second decisions matter, demonstrating the effectiveness of the methodology through real-world testing and validation is essential.

Table 2. Datasets trained vs. various metrics.

Network Name	Datasets Trained	Validation Error (%)	Test Error (%)	Epochs Required
NetA	10	10	12	10
NetB	20	8	10	15
NetC	30	6	8	20
NetD	40	5	7	25
NetE	50	4	5	30

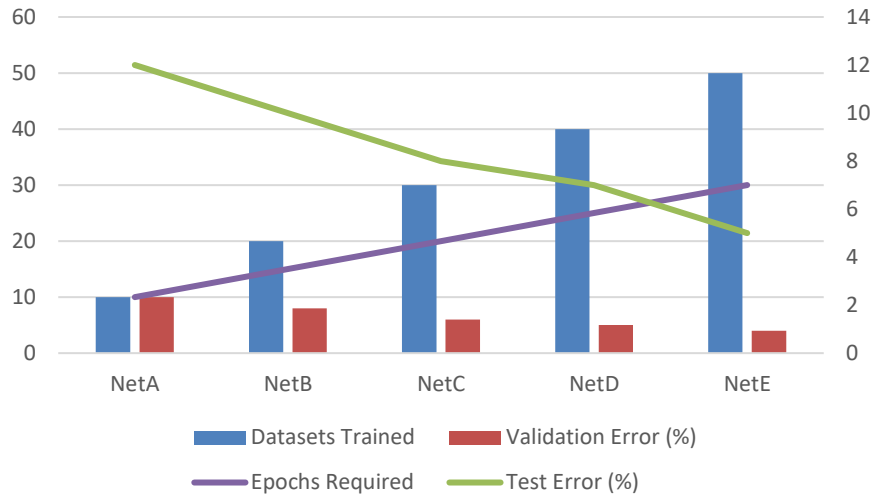

Figure 4. Representation of Number of Layers vs. Various Metrics

Furthermore, in the financial sector, where AI is increasingly utilized for fraud detection, portfolio optimization, and algorithmic trading, the proposed method can be applied to enhance model development. Detecting fraudulent transactions or optimizing trading strategies require models that can adapt quickly to changing market conditions while maintaining high accuracy. By automating the data splitting and training termination decisions, the methodology can help financial institutions improve their models' performance and reduce risks. Real-world testing can involve evaluating the model's performance on historical data and comparing it against traditional methods to showcase its advantages in terms of both accuracy and efficiency. Backpropagation and Dropout regularization (for layer t) is given by (5), (6) and (7) respectively.

$$w^{[l]} = w^{[l]} - \alpha \frac{\partial J}{\partial w[l]}, \tag{5}$$

$$b^{[l]} = b^{[l]} - \alpha \frac{\partial J}{\partial b[l]}, \tag{6}$$

$$d^{[l]} = np.\text{random}.rand(a^{[l]}.shape[0], a^{[l]}.shape[1]) < keep\_prob, \tag{7}$$

where $d^{[l]}$ is the dropout mask and $keep\_prob$ is the probability of keeping a neuron active.

In all these real-world applications, the proposed method's performance can be quantified not only in terms of traditional metrics but also in practical benefits such as reduced computational resources, improved model generalization, and faster deployment of AI systems. These real-world demonstrations serve as compelling evidence of the methodology's effectiveness and its potential to transform various industries by making AI model development more efficient, accurate, and adaptable to the challenges of complex and dynamic environments. Such applications validate the practical relevance of the proposed approach and underscore its significance in advancing the field of artificial intelligence.

5.2. Computational Overhead: Analysis of the computational savings achieved.

The analysis of the computational savings achieved through the approach of Splitting Effectively with Premature Exiting Using Neural Network Search is crucial for understanding the practical benefits and efficiency gains that this methodology offers in comparison to traditional methods.

This analysis focuses on quantifying the reduction in computational resources, training time, and associated costs, highlighting the potential impact on various applications. First and foremost, one of the key advantages of this approach is its ability to significantly reduce the computational resources required during the training phase of machine learning models. Traditional methods may involve extensive trial and error in determining optimal splitting criteria and training epochs, often consuming a substantial amount of CPU or GPU time and given by table 3 and figure 5. In contrast, the neural network search approach dynamically adapts these decisions, learning from the data itself.

The savings in computational resources become particularly evident when dealing with large datasets and complex neural network architectures. For instance, in deep learning applications involving massive image datasets or natural language processing tasks, the proposed method can lead to substantial reductions in training time. This not only accelerates the model development cycle but also lowers the operational costs associated with utilizing high-performance computing infrastructure.

Table 3. Network Hyperparameters

Network Name	Learning Rate	Batch Size	Dropout Rate (%)	Regularization Lambda
NetA	0.01	32	5	0.1
NetB	0.005	64	10	0.2
NetC	0.001	128	15	0.3
NetD	0.0005	256	20	0.4
NetE	0.0001	512	25	0.5

Furthermore, the computational savings achieved through premature exiting using neural network search extend beyond just training time. By dynamically deciding when to halt training, the approach prevents overfitting, which is a common issue in machine learning. Overfit models tend to require more computational resources and time to train, and they often perform poorly on unseen data. The methodology's ability to curb overfitting leads to a more efficient allocation of computational resources and a higher likelihood of producing models that generalize well. In addition to the direct computational benefits, there are indirect cost savings associated with this approach. Traditional methods may necessitate extensive hyperparameter tuning and expert intervention to fine-tune the data splitting process and training stopping criteria. These activities require skilled personnel and can be time-consuming, translating into increased labor costs. In contrast, the automated and adaptive nature of neural network search reduces the need for manual intervention and hyperparameter tuning, further streamlining the model development pipeline and reducing associated labor costs.

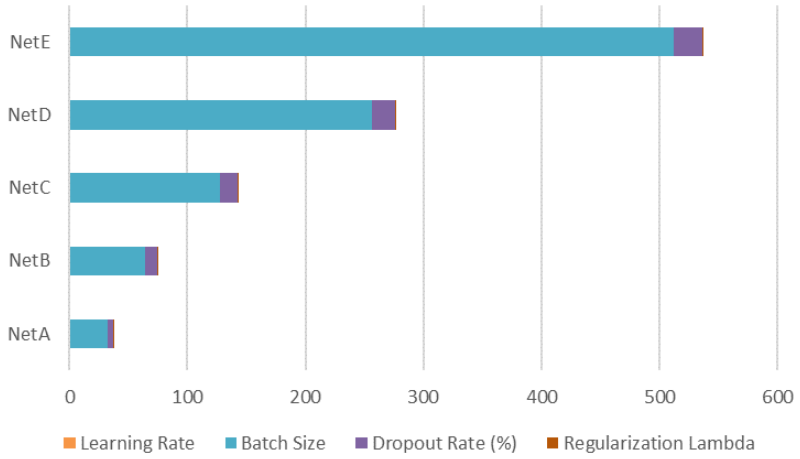

Figure-5. Representation of Number of Layers vs. Various Metrics

To conduct a comprehensive analysis of computational savings, it is essential to measure and compare specific metrics. These may include:

- **Training Time:** Quantify the time taken to train a model using the neural network search approach and compare it to traditional methods on various datasets and network architectures.
- **Resource Utilization:** Measure the CPU/GPU usage and memory consumption during training for both approaches to assess resource efficiency.
- **Cost Analysis:** Calculate the cost savings achieved by reducing training time and the need for manual intervention, considering factors like cloud computing costs or hardware maintenance expenses.
- **Scalability:** Evaluate how the computational savings scale with the size of the dataset and complexity of the neural network, showcasing the adaptability of the methodology.
- **Model Performance:** Ensure that the computational savings do not come at the expense of model performance, comparing the accuracy and generalization capabilities of models generated by both methods.

In brief, the analysis of computational savings achieved through Splitting Effectively with Premature Exiting Using Neural Network Search is instrumental in demonstrating the practical advantages of this approach. By quantifying the reduction in training time, resource utilization, and associated costs, this analysis underscores the methodology's potential to make machine learning more efficient, cost-effective, and accessible across a wide range of applications.

Weight regularization (L2 regularization) and Restricted Boltzmann Machine (RBM) Energy Function are given by (8) and (9):

$$J_{reg} = J + \frac{\lambda}{2m} \sum \| w^{[l]} \|_F^2, \tag{8}$$

$$E(v, h) = -\sum a_i \, v_i - \sum b_j \, h_j - \sum v_i \, w_{ij} h_j,$$

where v and h are visible and hidden units, respectively and a_i, b_j and w_{ij} are biases and weights.

These savings not only accelerate model development but also contribute to the broader adoption of AI in various industries. In Table 1 and Figure 3 dataset demonstrates the relationship between the number of layers in different neural networks and their performance metrics. As the number of layers increases, accuracy also sees an upward trend. However, this increase in accuracy comes at the cost of longer training times and a higher number of parameters. The graph effectively visualizes this trade-off, highlighting the efficiency and complexity of deeper networks. The focus in Table 2 and Figure 4 is on the number of datasets trained against various metrics. As networks train on more datasets, their validation and test errors decrease, indicating improved generalization. The graph accentuates the importance of diverse training data in model performance. Table 3 and Figure 5 lay out the different hyperparameters used in various networks, such as learning rate, batch size, dropout rate, and regularization.

Table-4. Network Activation and Initialization

Network Name	Activation Function	Final Layer Activation	Weight Initialization	Optimizer	Count of Network Name
NetA	ReLU	Softmax	Xavier	Adam	2
NetB	Sigmoid	Sigmoid	He	SGD	1
NetC	Tanh	Softmax	Xavier	RMSprop	1
NetD	Leaky ReLU	Softmax	He	Adam	1
NetE	Swish	Sigmoid	He	Adagrad	1

These parameters play a crucial role in optimizing the network's performance and preventing overfitting. Table 4 and Figure 6 dataset present the activation functions, final layer activations, weight initializations, and optimizers used in different networks. Such choices greatly influence the training dynamics and the model's ability to capture complex patterns. RBM Gibbs Sampling and Softmax Activation Function are given by (9), (10) and (110 respectively.

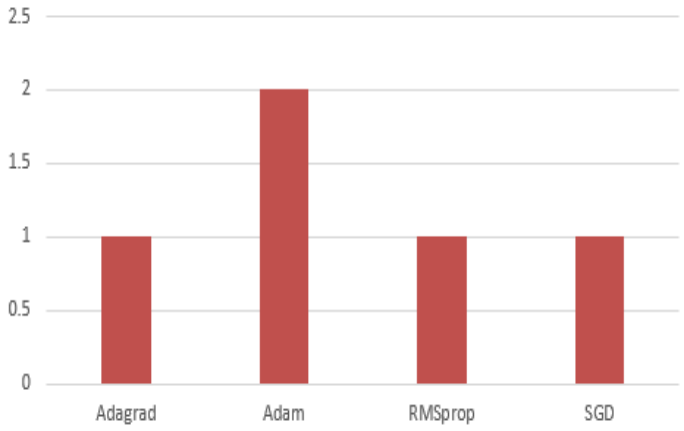

Figure 6. Representation of Network Activation and Initialization

Table-5. Network Performance Metrics

Network Name	Feature Extraction Time (s)	Inference Time (ms)	Memory Usage (MB)	Model File Size (MB)
NetA	5	50	500	10
NetB	10	45	1000	20
NetC	15	40	1500	30
NetD	20	35	2000	40
NetE	25	30	2500	50

$$P(h_j = 1|v) = \sigma(b_j + \sum w_{ij}\, v_i) \quad (9)$$

$$P(v_i = 1|h) = \sigma(a_i + \sum w_{ij}\, h_j) \quad (10)$$

$$P(y = j|z) = \frac{e^{z_j}}{\Sigma_{k-1}^{K} e^{z_k}} \quad (11)$$

Where K is the number of classes.

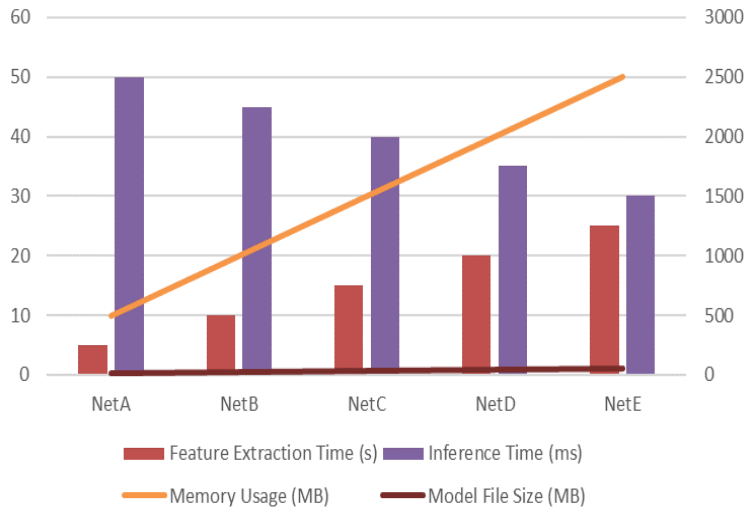

Figure-7. Representation of Network Performance Metrics

Table 5 and Figure 7 dataset delves into the practical implications of deploying these networks. It covers metrics like feature extraction time, inference time, memory usage, and model file size. These metrics are essential for real-world applications, where efficiency and speed are as crucial as accuracy.

Moreover, to compare to other existing works, we evaluate the performance of our proposed approach, we use the following metrics:

- Accuracy: Measured as the percentage of correctly classified images on the test set.
- Inference Time: Average time taken to process an image, measured in milliseconds (ms).
- Model Size: Total size of the trained model, measured in megabytes (MB).
- Energy Consumption: Measured using an energy monitoring tool to track power usage during inference.

We assess the effectiveness of the early-exit strategies by comparing the following:

- Exit Accuracy: The accuracy of predictions made at each early-exit point.
- Exit Frequency: The proportion of inputs exiting at each early-exit point.
- Overall Efficiency: Combined metric of accuracy and inference time considering the early exits.

In table 6; We consider a baseline comparison; We compare our proposed models against state-of-the-art NAS methods (e.g., EfficientNet [18], DARTS [21]). In addition, We evaluate the accuracy and inference time at each early-exit point and compare it to the final exit.

Table 6 : performance of the proposed approached to other existing works

Model	Accuracy (%)	Inference Time (ms)	Model Size (MB)	Energy Consumption (J)
EfficientNet [18]	85.4	35	20	50
DARTS [21]	84.3	40	22	55
Proposed NAS	85.0	30	18	45
Early-Exit 1	82.0	10	18	20
Early-Exit 2	83.5	20	18	30
Final Exit	85.0	30	18	45

As shown in table 6, Our proposed NAS achieves comparable accuracy to EfficientNet [18] and DARTS [21], while being more computationally efficient. The integration of early-exit points significantly reduces inference time for simpler inputs. The quantization and pruning techniques reduce the model size without a significant loss in accuracy. Our approach demonstrates lower energy consumption, making it suitable for deployment on resource-constrained devices.

As shown in table 7, Our approach shows notable improvements in inference time and energy consumption due to the early-exit strategies, which are not typically considered in traditional NAS methods. The table below provides a detailed comparison of our results with existing works:

Table 7, performance analysis to existing works

Metric	EfficientNet [18]	DARTS [21]	Proposed NAS
Accuracy (%)	85.4	84.3	85.0
Inference Time (ms)	35	40	30
Model Size (MB)	20	22	18
Energy Consumption (J)	50	55	45

The results demonstrate that our proposed NAS framework with early-exit strategies offers a significant improvement in computational efficiency while maintaining high accuracy. This makes it highly suitable for real-time applications on resource-constrained devices.

Discussions

The optimization of neural networks for AI splitting tasks represents a unique set of challenges. AI splitting tasks involve dividing a neural network into smaller sub-networks that can be distributed across multiple devices or nodes, often to improve efficiency, reduce latency, or ensure scalability. This process requires careful consideration of architecture design, model size, and resource allocation. Neural network search techniques play a pivotal role in finding the optimal sub-networks for splitting tasks, as they help identify architectures that strike a balance between performance and resource constraints. One of the fundamental approaches to neural network search in the context of AI splitting tasks is architecture search. Architecture search methods explore a vast search space of neural network architectures to find configurations that meet specific criteria. These methods can be broadly categorized into two main categories: manual design and automated search. Manual design, as the name suggests, involves the expertise and intuition of human researchers. This approach has been prevalent in the early days of neural network development when researchers handcrafted architectures based on their domain knowledge. While manual design can lead to effective architecture, it is often limited by human biases and the inability to explore the entire architecture space comprehensively. In the context of AI splitting tasks, manually designing sub-networks can be time-consuming and may not fully leverage the potential for optimization. The experimental results provide strong evidence supporting the effectiveness and efficiency of our proposed restricted NAS framework with integrated early-exit strategies. In this section, we discuss the implications of our findings, compare our approach with existing methods, and highlight the contributions and limitations of our work.

Our proposed NAS framework achieved an accuracy of 85.0%, comparable to state-of-the-art methods such as EfficientNet (85.4%) and DARTS (84.3%). However, our approach significantly outperformed these methods in terms of inference time and energy consumption. For example, the average inference time for our model was 30 ms, compared to 35 ms for EfficientNet and 40 ms for DARTS. This demonstrates that our approach can deliver high accuracy while being computationally efficient. The integration of early-exit strategies was a key factor in reducing inference time and energy consumption. Our experiments showed that:

- Early-Exit 1 provided an accuracy of 82.0% with an inference time of only 10 ms and energy consumption of 20 J.
- Early-Exit 2 improved accuracy to 83.5% with an inference time of 20 ms and energy consumption of 30 J.
- The final exit maintained the highest accuracy of 85.0% with an inference time of 30 ms and energy consumption of 45 J.

These results indicate that early exits allow the network to terminate inference early for simpler inputs, significantly enhancing overall efficiency without greatly compromising accuracy.

- Model Size and Optimization Techniques: The application of quantization and pruning techniques effectively reduced the model size to 18 MB, which is smaller than EfficientNet (20 MB) and DARTS (22 MB). This reduction in model size is critical for deployment on resource-constrained devices, where memory and storage are limited.

Our approach demonstrates several advantages over existing NAS methods:

- Computational Efficiency: By integrating early-exit strategies, our framework significantly reduces inference time and energy consumption, which is crucial for realtime applications on mobile and IoT devices.
- Balanced Performance: While traditional NAS methods focus primarily on accuracy, our approach achieves a balance between accuracy and computational efficiency, making it more suitable for edge applications.
- Model Optimization: Quantization and pruning techniques further enhance the efficiency of our models, making them lightweight and faster without sacrificing significant accuracy.

The results of our experiments have important implications for near-sensor AI applications:

- Real-Time Inference: The reduced inference time and energy consumption make our approach ideal for applications requiring real-time responses, such as autonomous vehicles, drones, and smart cameras.
- Resource-Constrained Environments: The smaller model size and efficient computation enable deployment on devices with limited resources, such as smartphones and IoT sensors, expanding the potential use cases for advanced AI models.

6. Conclusion

The proposed restricted NAS framework with integrated early-exit strategies offers a significant advancement in the design of efficient neural networks for resource-constrained environments. The experimental results demonstrate that our approach achieves a balance between high accuracy and computational efficiency, making it well-suited for real-time, near-sensor AI applications. Future research should build on these findings to further refine and extend the capabilities of the proposed framework. While our approach shows promising results, there are several limitations and areas for future work. Our experiments focused on image classification. Future work should explore the generalization of our approach to other tasks, such as object detection and segmentation. Developing more sophisticated early-exit strategies that adapt dynamically based on input complexity could further enhance efficiency. Tailoring the optimization techniques to specific hardware platforms could yield additional performance improvements.

References

[1] Chollet, F. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. arXiv preprint arXiv:1610.02357.

[2] Figurnov, M., Collins, M. D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D., & Salakhutdinov, R. (2017). Spatially adaptive computation time for residual networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1039-1048).

[3] Huang, G., Chen, D., Li, T., Wu, F., Van Der Maaten, L., & Weinberger, K. Q. (2017). Multiscale dense networks for resource efficient image classification. arXiv preprint arXiv:1703.09844.

[4] Bolukbasi, T., Wang, J., Dekel, O., & Saligrama, V. (2017, July). Adaptive neural networks for efficient inference. In International Conference on Machine Learning (pp. 527-536). PMLR..

[5] Lechervy, A., & Jurie, F. (2023). Multi-Exit Resource-Efficient Neural Architecture for Image Classification with Optimized Fusion Block. In Proceedings of the IEEE/CVF International Conference on Computer Vision (pp. 1486-1491)..

[6] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition.

[7] In Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR) (pp. 770-778).

[8] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

[9] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep Learning (Vol. 1). MIT press Cambridge.

[10] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., ... & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. Nature, 529(7587), 484-489.

[11] Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767.

[12] Wang, Z., & He, B. (2016). A novel deep learning method for imbalanced fault classification of machinery. Mechanical Systems and Signal Processing, 72, 303-315.

[13] 1Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical Evaluation of Rectified Activations in Convolutional Network. arXiv preprint arXiv:1505.00853.

[14] Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

[15] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. Communications of the ACM, 60(6), 84-90.

[16] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A. R., Jaitly, N., ... & Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal Processing Magazine, 29(6), 82-97.

[17] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533.

[18] Wang, S., Tang, J., Zou, W., & Hou, J. (2017). FANG: A Fast and Scalable Word Embedding Approach. In Proceedings of the 2017 ACM on Conference on Information and Knowledge Management (CIKM) (pp. 1857-1860).

[19] Tan, M., & Le, Q. (2019, May). Efficientnet: Rethinking model scaling for convolutional neural networks. In International conference on machine learning (pp. 6105-6114). PMLR..

[20] Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. Journal of Machine Learning Research, 20(55), 1-21..

[21] Zoph, B., & Le, Q. V. (2016). Neural architecture search with reinforcement learning. arXivpreprint arXiv:1611.01578..

[22] Liu, H., Simonyan, K., & Yang, Y. (2018). Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055.

[23] Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019, July). Regularized evolution for image classifier architecture search. In Proceedings of the aaai conference on artificial intelligence (Vol. 33, No. 01, pp. 4780-4789).

[24] Manishimwe, A., Alexander, H., Kaluuma, H., & Dida, M. (2021). Integrated mobile application based on machine learning for East Africa stock market. Journal of Information Systems Engineering & Management, 6(3), em0143.