UNIVERSITY of BRADFORD

Faculty of Management, Sciences and Engineering
School of Computing and Engineering

First Semester Submission

AI Agents Deep Neural Network Architecture Search

Final Year Project – Mid-term report

Kaiane Souza Cordeiro

UoB: 25029204

A thesis submitted in part fulfilment of the degree of BEng (Hons) Computer Science

Supervisor: Dr. Amr Rashad Ahmed Abdullatif

28 December 2025

Declaration

The candidate confirms that the work submitted is her own and that appropriate credit has been given where reference has been made to the work of others. The candidate agrees that this report can be electronically checked for plagiarism.

Kaiane Souza Cordeiro

Table of Contents

1. Introduction

1.1 Motivation

As deep learning continues to expand across scientific and industrial domains, the demand for better performance, task-specific models has grown exponentially, especially when looking towards neural architectures (Atlas and Devi, 2025). However, this demand has also exposed the fragility and inefficiency of current neural network design workflows: despite the availability of powerful architectures and large-scale datasets, building optimal DNNs (Deep Neural Networks) remains a manual, slow, and often inconsistent process, as such design processes are intensive due to trial-and-error approaches and the rare expertise required in practice (Liu et al., 2023). The complexity of architectural decisions, the sensitivity of hyperparameters, and the computational cost of training have created a significant barrier for both researchers and practitioners.

In recent years, the deep learning field has seen a rise in efforts to automate aspects of deep learning development, particularly in areas like AutoML (Automated Machine Learning) and its subfield, Neural Architecture Search (NAS) (Elsken, Metzen and Hutter, 2019). These methods aim to reduce reliance on human intuition by introducing algorithms that can explore and optimize DNN configurations. However, many of these techniques are either need to many computational resources for practical use or narrowly focused on specific sub-problems, such as hyperparameter tuning, without addressing the broader architectural design.

This project is motivated by the opportunity to leverage AI agents as intelligent optimizers in the DNN design process. By integrating decision-making capabilities into the model optimization loop, agents can autonomously explore architectural variations, learn from performance feedback, and iteratively improve designs (Lopes et al., 2023). This approach promises to significantly reduce development time, minimize human bias, and make deep learning more accessible, scalable, and efficient.

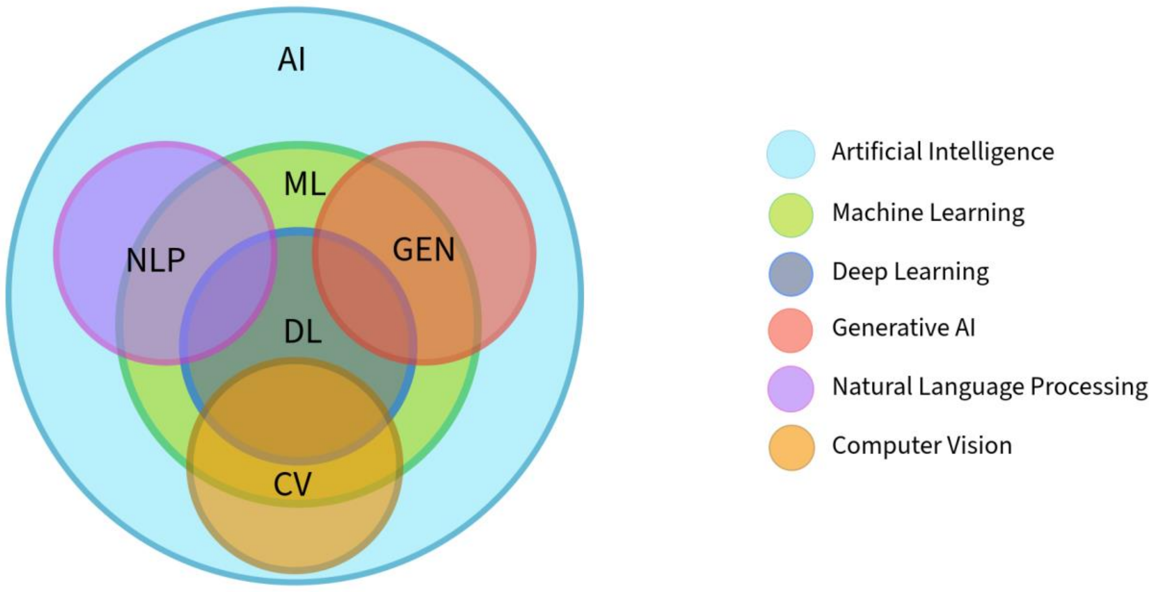
Ultimately, the motivation behind this research lies in bridging the gap between human-guided design and automated optimization, creating a system where AI supports the development of AI. By enabling more intelligent search and refinement processes, the use of agents in DNN architecture optimization can unlock a new level of adaptability in model development, addressing many of the core limitations identified in DNN's current scenario.

1.2 Deep Learning as a subfield of Artificial Intelligence

As the years progressed, Artificial Intelligence evolved into a broad and interdisciplinary field, encompassing several specialized subfields, each focused on replicating or augmenting specific aspects of human cognition. While the overarching goal of AI is to build systems capable of intelligent behaviour (Wang and Raj, 2017, Kaplan and Haenlein, 2019), the wide range of tasks involved has led to the emergence of distinct research domains. These include Machine Learning, Natural Language Processing (NLP), Computer Vision, Robotics, and more recently, Generative AI, each contributing uniquely to AI's rapid development in both academic and industrial contexts.

First of all, Machine Learning (ML) is one of the most applied subfields of AI today. It centres on designing algorithms that enable systems to identify patterns in data (Naqvi, 2020) and improve their performance over time without explicit reprogramming. Within ML and with the same "pattern recognition" behaviour, Deep Learning has emerged as a particularly powerful approach relying on multi-layered artificial neural networks, "a data processing paradigm inspired by the way biological nervous systems, such as the brain, process data" (Thorat, Pandit and Balote, 2022), to learn complex information. Analysing the subfields of AI below, it becomes increasingly evident how deep learning has become central to many of these domains, either as the underlying model or as a performance-enhancing component.

Image 01: Subfields of Artificial Intelligence



Source: Produced by the author.

This diagram illustrates how major subfields of Artificial Intelligence often overlap and reinforce one another, forming modern AI ecosystems. Deep learning is a subset of machine learning, and machine learning is a subset of AI, which is an umbrella term for any computer program that does something smart (Tiwari, Tiwari and Tiwari, 2018). Domains such as Natural Language Processing, Computer Vision, and Generative AI heavily rely on Deep Learning today, though they are not strictly confined to it. At the core of these advancements lies one of Deep Learning's most important tools: the Deep Neural Network (DNN). These networks have become the primary modelling technique for modern AI.

1.3 The evolution of Neural Networks

The development of Neural Networks is deeply intertwined with the history of artificial intelligence itself. Inspired by biological neurons, early models sought to simulate the basic processing mechanisms of the human brain. One of the first models was introduced by Frank Rosenblatt's Perceptron in 1958, "that was showed with the ability to learn in accordance with associationism" (Wang and Raj, 2017), which could enrrol weights based on labelled data and adjust them to classify inputs. Despite its potential, the perceptron was limited to solving only linearly separable problems. The inability to learn functions like XOR, as highlighted by Minsky and Papert in 1969, contributed to the first AI winter (Wang and Raj, 2017).

The limitations of single layer perceptrons were overcome with the development of Multi-Layer Perceptrons (MLPs) "by placing the perceptrons side by side" (Wang and Raj, 2017) and the backpropagation algorithm in the 1980s. These models, often referred to as Artificial Neural Networks (ANNs), allowed for the training of deeper architectures capable of modelling complex, non-linear relationships. Backpropagation became a breakthrough in training multi-layer networks. While MLPs opened new possibilities, their practical use remained constrained by limited computational resources and data availability, delaying broader adoption.

The real comeback of the ANNs came in the 2015 in the form of Deep Learning (Haenlein and Kaplan, 2019), made possible by access to large-scale datasets, advancements in GPU hardware, and architectural innovations. Models such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) enabled specialized handling of image and sequential data, respectively (Wang and Raj, 2017). These architectures, and their improvements over time, form the core of deep learning, which differs from earlier models by focusing on learning hierarchical data representations using many layers.

Table 01: Key Milestones in the Evolution of Neural Networks

Model / Concept	Description	Key Contributions	Relationship with Deep Learning
Perceptron	Single-layer model for binary classification	First learning algorithm using weights	Conceptual origin, but too shallow
XOR Problem	Logical problem that Perceptron couldn't solve	Highlighted limitations of single-layer models	Motivated development of deeper networks
Multi-Layer Perceptron (MLP)	Feedforward network with one or more hidden layers	Can learn non-linear functions	Foundation for deep learning
Artificial Neural Network (ANN)	General term for networks with neurons and layers	Includes MLPs and others	Precursor to deep architectures
Backpropagation	Gradient-based algorithm for training multi-layer networks	Enabled efficient learning in deep models	Core of deep learning training
Deep Neural Network (DNN)	Neural network with many hidden layers	Hierarchical feature learning	Backbone of modern deep learning
Convolutional Neural Network (CNN)	Uses convolutional layers for spatial feature extraction	Breakthrough in image processing	Key architecture in vision tasks
Recurrent Neural Network (RNN)	Designed to handle sequential data with feedback loops	Effective in language and time-series tasks	Fundamental to NLP and sequential learning

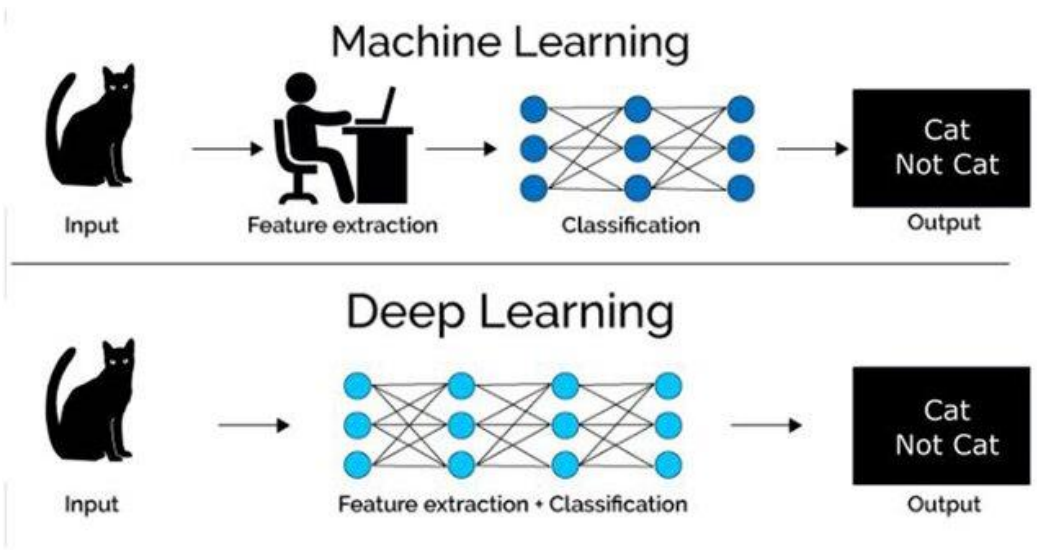
Source: Produced by the author.

The evolution from perceptrons to modern deep networks reflects the ongoing effort to build more expressive, scalable, and task-specific neural architectures.

1.4 Different types of feature extraction.

The emanation of the deep learning can be associated with its ability to perform end-to-end learning, where the model automatically extracts features (properties or characteristics extracted from raw data that helps a model recognize patterns and make predictions) and optimizes representations without manual engineering (Kalaivani, Uma and Kanimozhiselvi, 2020). For instance, in image recognition, early layers might detect edges and textures, while deeper layers identify objects and contextual relationships. This capacity for automatic feature extraction enables DNNs to outperform many hand-crafted approaches across various domains.

Image 02: Differences between Machine Learning and Deep Learning



Source: (Asher et al., 2021)

Differently from the traditional machine learning methods, the deep learning models perform automatic feature extraction, overtaking the limitations of manual feature engineering (Degadwala and Degadwala, 2024; Kalaivani, Uma and Kanimozhiselvi, 2020). “The key aspect of deep learning is that these layers of features are not designed by human engineers: they are learned from data using a general-purpose learning procedure.” (LeCun, Bengio and Hinton, 2015).

On the other hand, the underlying architecture and operational principles that enable this functionality can cause the issues of trial and error and rare expertise required in practice (Liu et al., 2023) and to solve this problem it is necessary to dive into the knowledge of the distinguished types of architectures and the functionality of DNNs.

2. Literature Review

2.1 DNNs and The Fundamentals of Deep Learning

At their core, Deep Neural Networks (DNNs) are computational models inspired by the structure and function of the human brain (Thorat, Pandit and Balote, 2022). A Deep Neural Network consists of multiple layers of interconnected neurons, with each layer transforming its input into a more abstract and informative representation. This layered architecture enables DNNs to learn com plex patterns and hierarchical features from raw data, making them particularly effective for high-dimensional tasks such as image classification, speech recognition, and language modelling (LeCun, Bengio and Hinton, 2015).

The process begins with the input layer, which receives raw data in its most basic form. For example, in image classification, this input could be a flattened vector of pixel values, while in natural language processing, it might be a sequence of embedded word vectors. Following the input, one or more hidden layers carry out the bulk of the computation introducing non-linearity, each hidden layer contains numerous neurons that perform a series of operations. As information flows through the hidden layers, the network progressively learns to extract higher-level, more abstract features. For instance, in a vision model, early layers may detect edges, intermediate layers may identify shapes or textures, and deeper layers might recognize entire objects (LeCun, Bengio and Hinton, 2015).

After generating a prediction, the network calculates the error between its prediction and the true label using a loss function. “The machine then modifies its internal adjustable parameters to reduce this error. These adjustable parameters, often called weights, are real numbers that can be seen as ‘knobs’ that define the input–output function of the machine. [...] To properly adjust the weight vector, the learning algorithm computes a gradient vector that, for each weight, indicates by what amount the error would increase or decrease if the weight were increased by a tiny amount” (LeCun, Bengio and Hinton, 2015). This process is repeated over many iterations, known as epochs, allowing the model to gradually refine its performance.

Despite their powerful capabilities, designing and training DNNs remains a complex and non-trivial task. The performance of a model is highly sensitive to a range of factors, including architectural decisions (such as the number of layers and neurons), the types of connections between layers, and hyperparameters like learning rate, batch size, and regularization strategies.

2.2 DNNs Architectures

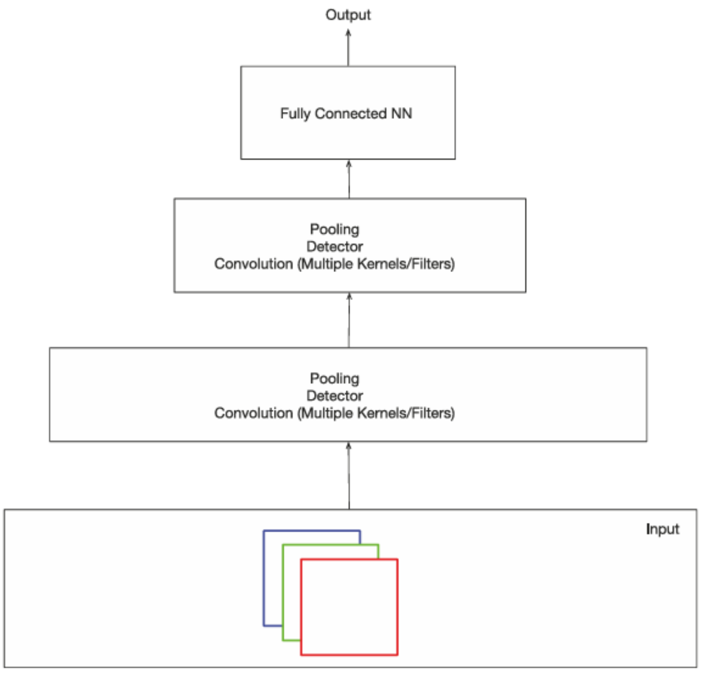
Deep Learning encompasses a diverse family of neural network architectures, each designed to model specific types of data and extract increasingly complex patterns. While all Deep Neural Networks (DNNs) share a common foundation, layered representations, non-linear transformations, and end-to-end optimization, the architecture of a network determines its strengths, limitations, and suitability for different tasks. Understanding these architectural variations is crucial not only for appreciating how modern systems achieve state-of-the-art performance, but also for motivating the need for automated architecture design methods such as NAS (Wang and Raj, 2017).

Modern DNN architectures evolved as solutions to distinct challenges that earlier models could not address. Feedforward Multi-Layer Perceptrons (MLPs), for instance, represented a major advancement over single-layer perceptrons by enabling networks to learn non-linear functions through multiple hidden layers. “For a shallow network, the representation power can only grow polynomially with respect to the number of neurons, but for deep architecture, the representation can grow exponentially with respect to the number of neurons” (Wang and Raj, 2017).

However, their fully connected structure made them inefficient for high-dimensional inputs, especially images, where spatial relationships are essential. This limitation led to the development of specialized architectures tailored to the structure of the data, such as the Convolutional Neural Network (CNN), designed “to process data that come in the form of multiple arrays, for example a colour image composed of three 2D arrays containing pixel intensities in the three colour channels” (LeCun, Bengio and Hinton, 2015).

This kind of model introduced convolutional layers, where units are “organized in feature maps” (LeCun, Bengio and Hinton, 2015), with learnable filters that operate over local spatial regions, enabling the network to capture edges, textures, and shapes in a hierarchical manner. This architectural efficiency, combined with GPU acceleration, enabled breakthroughs such as LeNet-5 and later AlexNet, which marked the beginning of the deep learning revolution in computer vision (Wang and Raj, 2017).

Image 03: CNN Architecture


Output
Fully Connected NN
Pooling
Detector
Convolution (Multiple Kernels/Filters)
Pooling
Detector
Convolution (Multiple Kernels/Filters)
Input

Source: Nihkil Ketkar and Moolayil, 2020.

For sequential data, Recurrent Neural Networks (RNNs), “a class of neural network whose connections of units form a directed cycle” (Wang and Raj, 2017), were developed to model temporal dependencies, functionality granted by this nature. RNNs have been widely used for complex tasks such as speech recognition and machine translation (LeCun, Bengio and Hinton, 2015), but despite their success, RNNs struggle with parallelization and long-range dependencies, motivating the search for more scalable architectures (Wang and Raj, 2017).

This search culminated in the emergence of attention-based models and Transformers, which replaced recurrence with attention mechanisms that learn global relationships between all elements of a sequence (Wang and Raj, 2017). Transformers enable efficient parallel training and have become the foundation of modern NLP, powering models such as BERT, GPT, and Vision Transformers (ViT). Their flexibility demonstrates how architectural innovation can fundamentally reshape entire subfields of AI (Berlina Rahmadhani, Purwono Purwono and Safar, 2024).

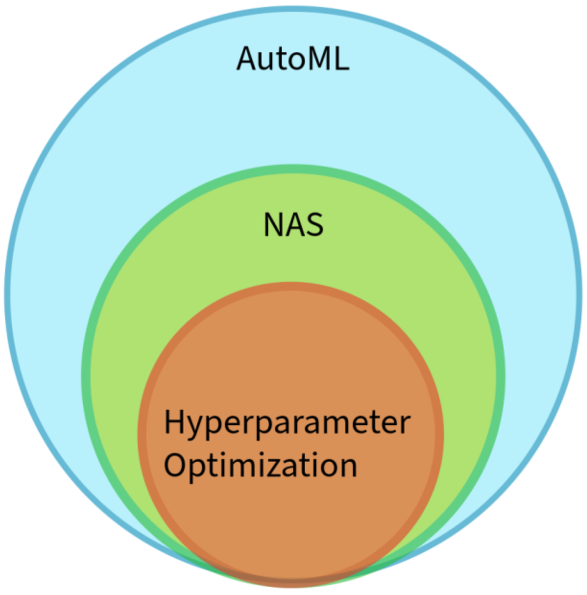
The diversity of architectures, MLPs, CNNs, RNNs, Transformers, autoencoders, graph neural networks, and hybrid models, illustrates the central role of architecture choice in deep learning performance. Each architecture embeds specific inductive biases, determines how features are extracted, and dictates computational cost and scalability. As deep learning applications continue to expand across domains, manually selecting and designing architectures becomes increasingly difficult, subjective, and inefficient (Atlas and Devi, 2025). The rapidly growing design space, combined with the sensitivity of DNNs to architectural variations, underscores the need for systematic and autonomous approaches to architecture discovery.

2.3 Neural Architecture Search (NAS)

Neural Architecture Search (NAS) has emerged as a promising research direction aimed at automating one of the most difficult and resource-intensive aspects of deep learning: the design of neural network architectures. As a subsetof AutoML, the broader field dedicated to automating components of the machine learning pipeline, NAS focuses specifically on the automated discovery of neural network structures (Atlas and Devi, 2025).

Despite the extensive trial-and-error experimentation, domain-specific intuition, and the careful tuning of numerous architectural parameters including depth, width, layer types, filter sizes, and connectivity patterns, the practitioners must also perform hyperparameter optimization (HPO). Together, these processes of adjusting numerical configurations such as learning rate and batch size decisions create an exponentially expanding design space that is extremely difficult to navigate manually (Google Cloud, 2022). As a result, developing high-performing architectures by hand becomes a slow, resource-intensive, and often prohibitively costly process (Liu et al., 2023). NAS seeks to overcome these limitations by enabling algorithms to explore, evaluate, and optimize network structures automatically.

Image 04: Hierarchical Relationship of NAS



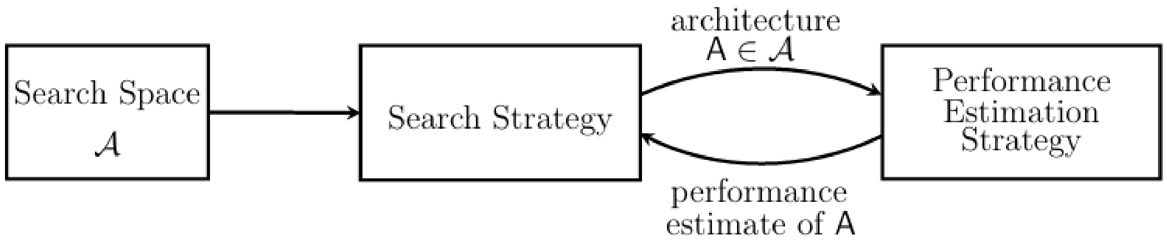
(Image inspiration source: State of the Art Neural Networks – Google Cloud)

At its core, NAS formulates neural architecture design as an optimization problem. Nowadays, given a predefined search space containing all architectural configurations that the algorithm is allowed to explore, and an objective function measuring model quality (e.g., accuracy, FLOPs, latency, or training time), the goal of NAS is to automatically identify the architectures that achieve the best performance under these criteria (Atlas and Devi, 2025).

Ultimately, most NAS frameworks are organizer around three fundamental components:

(1) **Search Space**: defines the architectural elements that can vary during the search, such as layer types, operations, connectivity patterns, network depth or width, and cell structures. The size and expressiveness of this space determine the diversity of architectures the algorithm can potentially discover;
(2) **Search Strategy**: describes how the algorithm navigates the search space. It governs the generation, selection, and refinement of candidate architectures, using methods such as reinforcement learning, evolutionary algorithms, Monte Carlo Tree Search, gradient-based optimization, or weight-sharing techniques;
(3) **Performance Estimation Strategy**: provides a mechanism for evaluating the quality of candidate architectures under computational constraints. Instead of fully training every architecture (which would be infeasible) NAS methods rely on approximations such as partial training, early stopping, shared weights, or surrogate models that predict performance more efficiently.

Image 05: Illustration of Neural Architecture Search methods



(Image source: Elsken, et al. 2019)

2.4 Evolution of Classical NAS Methods

Before having the three fundamental components, early NAS methods, such as reinforcement-learning (RL) controllers, demonstrated that neural architectures could be automatically discovered with competitive or superior performance. However, these approaches relied on training thousands of candidate networks from scratch, resulting in prohibitively high computational cost and limited scalability (Barret Zoph and Le, 2016). Although this method produced state-of-the-art architectures such as NASNet, it required approximately 1,800 GPU-days, revealing a fundamental bottleneck in early NAS systems (Liu, Zhang and Jin, 2022).

After this initial approach, alternative agent-based methods such as AlphaX were introduced to improve search efficiency. AlphaX formulates neural architecture design as a sequence of incremental decisions organized in a Monte Carlo Tree Search (MCTS), explicitly balancing exploration of unseen configurations with exploitation of promising ones, rather than relying on reinforcement-learning-based controllers. On NASBench-101, the first publicly available tabular benchmark for NAS research (Ying et al., 2019), AlphaX achieved approximately three times higher sample efficiency than random search (Wang, Zhao and Jinnai, 2019). A subsequent extension incorporated a predictive performance model (Meta-DNN) to estimate an architecture's accuracy prior to full training, resulting in an order-of-magnitude reduction in search cost compared to earlier reinforcement-learning-based NAS approaches, such as NASNet (Wang, Zhao and Jinnai, 2019).

In addition to RL and MCTS, evolutionary algorithms offered an alternative search paradigm. With the introduction of the evolutionary AutoML framework called LEAF (Learning Evolutionary AI Framework), the NAS context became able to not only optimize hyperparameters but also network architectures and the size of the network (Liang et al., 2019). Beyond improving search efficiency, the LEAF began to formalize NAS as a modular design problem that decomposes neural architecture search into "three main components: algorithm layer, system layer, and problem-domain layer" (Liang et al., 2019). Even though this evolutionary modular method naturally supports multi-objective optimization, enabling simultaneous trade-offs between accuracy, model size, latency, and energy consumption, these classical approaches remained costly because they relied on repeatedly training numerous candidate architectures.

Together, early NAS methods demonstrated the potential of automated architecture engineering but also exposed its major limitation: the evaluation bottleneck, where training each candidate dominates computational cost.

2.5 Efficiency-Oriented Advances in Neural Architecture Search

As the computational cost of classical NAS approaches became impractical, research shifted toward methods designed to significantly improve efficiency and scalability. This evolution led to differentiable NAS, one-shot and weight-sharing approaches, and hardware-aware optimization, which collectively reduced search time and enabled deployment under real-world constraints.

In response to the vast number of architecture training, the differentiable NAS method relaxes discrete architectural choices into a continuous search space optimized via gradient descent. Representative methods such as DARTS and subsequent extensions demonstrated substantial reductions in search cost compared to reinforcement-learning-based NAS, though they can be improved to bridge the gap between 'testing all options at once' during the search and 'choosing a single best winner' for the final design (Liu, Simonyan and Yang, 2019).

In parallel, one-shot and weight-sharing methods proposed replacing the training of thousands of candidate networks with a single over-parameterized supernet, from which individual architectures inherit shared weights for near-instant evaluation. While this strategy dramatically reduces the computational cost of NAS, recent studies have shown that the evaluation of candidate architectures can be significantly biased. In particular, the supernet often fails to provide a faithful ranking of architectures, leading to systematic misjudgments during the search process (Chu, Zhang and Xu, 2021).

This bias has been theoretically and empirically attributed to inherent unfairness in supernet training, where different architectural choices receive unequal optimization opportunities, resulting in distorted performance estimates (Chu, Zhang and Xu, 2021). These approaches demonstrate that efficiency alone is insufficient, as reliable performance evaluation remains a critical challenge in modern NAS.

2.6 Benchmarks and Emerging Autonomous Directions

As Neural Architecture Search evolved, the community recognized the need for standardized evaluation to ensure reproducibility and fair comparison. This led to the creation of tabular benchmarks such as NAS-Bench-101 (Ying et al., 2019), NAS-Bench (Dong et al., 2021), NAS-Bench-301 (Siems et al., 2020), and many others. Instead of researchers spending thousands of GPU hours training individual candidates on datasets like CIFAR-10 or ImageNet, they could now query a pre-computed database of hundreds of thousands of architectures. This shifted the focus from raw computational power to the design of more sophisticated search logic, allowing for rapid experimentation and the discovery of more robust search patterns.

Despite these advances, the efficiency-oriented advances revealed significant structural limitations in mainstream NAS methods. One of the chalenges remaining is the bottleneck caused by the large architecture parameter space that bumps into the problem of the computationally expensive methods existents (Lopes et al., 2023) . In response to these challenges, emerging autonomous directions are pushing the boundaries of how architectures are discovered. Multi-Agent Neural Architecture Search (MANAS) represents a significant step toward decentralization “to enable the possibility of large-scale joint optimisation of deep architectures” (Lopes et al., 2023). By treating the search process as a coordinated effort between multiple independent agents, MANAS breaks down the search space into smaller, more manageable segments. This not only significantly reduces the memory footprint but also mitigates the biases of centralized supernets, as each agent focuses on optimizing its local architectural decisions (Lopes et al., 2023).

Even though representing a significant step toward decentralised and memory-efficient Neural Architecture Search, MANAS remains primarily an optimisation-centric method rather than a fully autonomous system. In MANAS, agents are modelled as lightweight decision-makers responsible for local architectural choices, coordinated implicitly through a global loss signal. While this formulation provides strong theoretical guarantees and improved scalability, it can be improoved by explicit mechanisms for reasoning, long-term memory, or system-level orchestration.

As a result, architectural knowledge is encoded only implicitly in probability updates, limiting the framework’s ability to interpret past experiences, avoid repeated design failures, or adapt its search strategy beyond statistical feedback. In contrast, the present project proposes a agent-based NAS **framework** in which agents are endowed with specialised roles, shared memory, and reasoning capabilities. Rather than decomposing NAS solely as an optimisation problem, the proposed system treats architecture search as an autonomous, cognitive process that integrates generation, evaluation, feedback, and refinement within a coordinated workflow. This distinction positions the proposed approach not as a replacement for methods such as MANAS, but as an extensible autonomous framework capable of incorporating such algorithms while addressing broader challenges related to adaptability, sustainability, and end-to-end automation.

3. Problem Statement

3.1 Challenges in Deep Neural Network Design

Throughout the history of deep learning, progress has been driven by successive attempts to overcome the limitations of existing architectures. From early perceptron models to convolutional, recurrent, and attention-based networks, each advance introduced greater abstraction and representational power. However, as architectures became deeper and more complex, their design increasingly shifted toward processes that rely heavily on human expertise, intuition, and extensive trial and error, revealing structural limitations in how neural networks are currently developed (Atlas and Devi, 2025).

Despite their remarkable performance across diverse applications, the design and deployment of Deep Neural Networks remain complex and resource-intensive (Liu et al., 2023). Building an effective DNN requires making numerous architectural and hyperparameter decisions, each with a significant impact on performance (Liu et al., 2023). These decisions are often interdependent and context-specific, making the design process slow, costly, and difficult to generalise (Barret Zoph and Le, 2016), particularly as models scale in size and complexity.

A central challenge lies in architectural complexity and scalability (Real et al., 2018). Designers must determine network depth, layer types, connectivity patterns, and model width, all of which influence feature extraction and learning capacity. Increasing architectural complexity can introduce issues such as vanishing gradients, overfitting, or excessive model size, while architectures that perform well on one dataset may fail to generalise to others. This highlights the absence of universally optimal designs and underscores scalability as an unresolved problem in deep learning.

Besides that, the computational cost and training time further compound these challenges. Training deep models requires substantial hardware resources and long optimisation cycles, particularly for large datasets and complex architectures. The iterative nature of training amplifies energy consumption and environmental impact, restricting rapid experimentation and placing practical limitations on smaller research teams or resource-constrained deployment scenarios.

In parallel, deep learning workflows remain heavily dependent on human expertise and manual experimentation (Atlas and Devi, 2025). Selecting appropriate architectural components, optimisation strategies, and training configurations often demands specialised knowledge, while minor changes can lead to large performance variations. This reliance on intuition-driven trial and error reduces reproducibility and slows systematic progress in model development.

Finally, DNNs exhibit high sensitivity to hyperparameters such as learning rate, batch size, initialization schemes, and regularisation techniques. These parameters interact in complex, non-linear ways, making exhaustive tuning impractical. While automated hyperparameter optimisation methods offer improvements over brute-force approaches, they remain computationally demanding and insufficiently adaptable across tasks, reinforcing the need for more autonomous and intelligent optimisation strategies (Liu, Simonyan and Yang, 2019).

3.2 Need for Autonomous and Sustainable Optimization

The rapid progress in deep learning has brought to light a key challenge: as models become more complex, the practical difficulties of designing, training, and deploying them have also grown. To begin with, the sheer scale of today's AI models makes manual design nearly impossible. Modern architectures can include millions (or even billions) of parameters, where even small tweaks to depth, filter size, or activation functions can drastically impact performance. With such an enormous design space, manual exploration is no longer practical, especially as deep learning moves into new fields that bring unique design challenges. This growing complexity calls for automated, systematic methods that can identify high-performing architectures without human intervention.

Adcitionaly, the computational demands of deep learning are rising rapidly. Training large-scale models consumes vast GPU or TPU resources, which leads to high financial costs and increased energy usage. Iterative experimentation, common in traditional manual design,only amplifies this burden. This not only puts such efforts out of reach for smaller research teams but also raises serious environmental concerns. As AI continues to expand, there's increasing pressure to reduce its carbon footprint, aligning with global sustainability goals and the principles of responsible computing (Chatterjee and Rao, 2020).

These limitations point to a clear need for smarter, more efficient optimization strategies. One promising direction is the use of autonomous AI agents. Equipped with reasoning abilities, memory, feedback mechanisms, and adaptive search techniques, these agents can explore architectural possibilities more intelligently than brute-force NAS methods. They can learn from past results, focus on promising design directions, and strike a dynamic balance between exploration and exploitation. This leads to a more sustainable and cost-effective optimization process.

Ultimately, meeting the challenges of deep learning design will require moving beyond manual and resource-heavy methods toward intelligent, sustainable automation. This is where agent-based NAS comes in. By enabling the discovery of optimal architectures with less human input, lower computational demands, and a smaller environmental footprint, this approach represents a timely and necessary evolution in how we build AI systems.

4. Proposed Solution - AI Agents for Efficient Architecture Search

4.1 Conceptual Overview

The proposed solution introduces an agent-based framework for NAS, leveraging autonomous AI agents to explore, evaluate, and refine deep learning architectures with minimal human intervention. Traditionally, the architectures of DNNs play a crucial role in their performance and are typically designed manually by experts, which can be time-consuming and require significant domain knowledge (Liu et al., 2023; Elsken, et al. 2019).

At the core of the framework lies the principle that architecture design can be treated as an iterative decision-making process, where agents generate candidate architectures, assess their performance, extract insights, and refine the search trajectory (Lopes et al., 2023). Each agent operates as an intelligent component with specialised roles: generation, evaluation, optimisation, and coordination. Through structured communication and shared memory, the system collectively builds knowledge of which architectural patterns lead to strong performance and which regions of the search space should be avoided.

This paradigm offers several advantages over manual or traditional NAS workflows:

- **Efficiency**: Agents reduce redundant exploration by learning from previous attempts. They prioritise promising architecture families, minimise unnecessary training cycles, and use performance/compute trade-offs to guide the search.
- **Autonomy**: Once configured, the system can run independently, continuously generating and evaluating architectures without requiring human tuning or intervention.

- **Adaptability**: Agents dynamically adjust their behaviour based on real-time feedback, shifting between exploration and exploitation, modifying architectural parameters, and integrating new knowledge to refine the search process.

By combining coordination, reasoning, and memory mechanisms, the agent-based paradigm aims to deliver architecture search that is smarter, faster, and more sustainable than conventional approaches. This conceptual foundation sets the stage for the system architecture and operational workflow described in the following sections.

4.2 System Architecture

The proposed system is structured as a multi-agent architecture search framework, in which distinct agents collaborate to iteratively generate, evaluate, and refine neural network architectures. Each agent is designed with specialised responsibilities, and together they form a closed feedback loop that gradually improves the quality of candidate architectures. This modular organisation enables clear task separation, scalability, and structured communication among components, ensuring that the system remains adaptable and efficient throughout the search process.

The architecture consists of four primary agent types:

1. Coordinator Agent

- Serves as the central controller of the system.
- Allocates tasks to the other agents and oversees the full search pipeline.
- Ensures that exploration/exploitation strategies remain balanced.
- Monitors resource usage, search progress, and termination criteria.

2. Architecture Generator Agent

- Proposes candidate architectures based on rules, heuristics, memory, or sampled design choices.
- Operates within a defined search space (e.g., convolutional blocks, filter sizes, depth, activation types).
- Incorporates prior knowledge from past evaluations to avoid unproductive regions of the search space.
- Responsible for exploration and diversification of architecture candidates.

3. Evaluation Agent

- Instantiates the proposed architecture and trains it on the selected dataset.
- Computes key performance metrics such as accuracy, loss curves, FLOPs, parameter count, training time, and energy cost.
- Logs all results to a shared memory module for downstream reasoning.
- Acts as the empirical grounding of the entire framework.

4. Optimization/Refinement Agent

- Analyses the historical results stored in memory.
- Identifies promising design patterns and eliminates weak-performing architectural configurations.
- Adjusts the sampling distribution or mutation rules used by the Generator Agent.
- Implements adaptive strategies for exploration/exploitation based on recent performance.

In this system, a group of intelligent agents work together through a well-organized process that repeats itself until they reach a clear goal. The process follows four main steps: Generation, Evaluation, Feedback, and Refinement.

First, the Generator Agent comes up with a potential neural network design. It does this by exploring different options and using insights from past experiences, which are stored in a shared memory module. As a consequence, the Evaluation Agent tests the proposed model. It either trains it for a set number of training cycles (called "epochs") or uses a faster, approximate method to estimate how well it performs. During this step, the system tracks how accurate the model is and how efficiently it runs.

Once the evaluation is complete, the results are saved in a shared memory space. This includes performance scores, efficiency details, and characteristics of the model. The system also updates summary statistics and tracks overall progress, helping the team understand what's working and what isn't. Based on this feedback, the Optimization Agent steps in. It looks at the data in memory, adjusts the direction of the search, and advises the Generator Agent. This guidance might include which areas are worth exploring more, which patterns to avoid, or how to improve the design.

This cycle continues: generation, evaluation, feedback, refinement, until the system reaches a stopping point. That might be achieving a target level of accuracy, using up the allowed computing resources, or settling on a family of models that perform best.

4.3 Implementation Framework

The implementation of the proposed agent-based NAS system is grounded in a modular framework that enables experimentation, reproducibility, and incremental development. The design directly reflects the structure of the project repository, ensuring that each component of the system (agents, search logic, model constructors, and utilities) can evolve independently while maintaining coherence across the architecture search pipeline. This modularity is essential for testing different agent strategies, integrating new search algorithms, and scaling the system to more complex datasets and architectures.

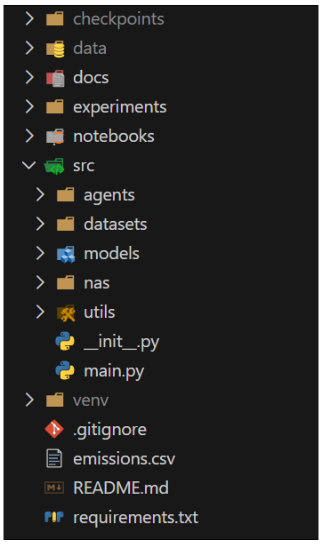
4.3.1 Tools and Software Stack

The implementation relies on a set of widely adopted machine learning and agent-orchestration tools:

- Python is used as the primary programming language due to its mature ecosystem and extensive support for machine learning research.
- PyTorch or TensorFlow provides the deep learning backend for building and training candidate architectures. Their flexibility enables dynamic model creation during the search process.
- Optuna or NNI (Neural Network Intelligence) may be integrated to support hyperparameter optimisation and provide efficient sampling strategies that complement agent-driven search.
- LangChain or similar agent-orchestration frameworks can be used to implement reasoning, memory retrieval, and multi-agent coordination for the generation–evaluation–refinement loop.
- Experiment tracking tools such as TensorBoard, Weights & Biases, or custom logging utilities record training metrics, architectural configurations, and system events to ensure complete reproducibility.

The repository's structure reflects these components, with dedicated folders for agents, search, models, utils, and experiments, enabling clear separation of responsibilities and simplifying iterative development.

Image 06: NAS Agents repository's folder structure



```
> checkpoints
> data
> docs
> experiments
> notebooks
v src
  > agents
  > datasets
  > models
  > nas
  > utils
    __init__.py
    main.py
> venv
  .gitignore
  emissions.csv
  README.md
  requirements.txt
```

Source: Produced by the author.

4.3.2 Dataset and Environment Setup

To validate the proposed system, standard computer vision benchmark datasets will be used, beginning with MNIST and CIFAR-10 as practical testbeds. These datasets provide a controlled environment for architecture search, allowing the system to explore and compare architectures efficiently before scaling to more complex, high-dimensional tasks.

The experimental environment includes:

- preprocessing pipelines (normalisation, augmentation),
- train/validation/test splits,
- configurable batch size and learning rate schedules,
- proxy evaluation options (e.g., early-stopping, reduced epochs) to accelerate search.

The environment is designed to support rapid iteration while maintaining fair comparisons across candidate architectures.

4.3.3 Computational Resources

The implementation is designed to operate under realistic computational constraints, aligning with the project's sustainability objectives. Experiments will be conducted using:

- a single GPU or limited GPU cluster when available,
- CPU-only configurations for lightweight or proxy evaluations,
- energy-aware monitoring tools when appropriate (e.g., measuring approximate compute cost).

By intentionally limiting hardware usage, the framework promotes sustainable experimentation and provides a practical demonstration that effective NAS can be achieved without large-scale compute budgets. Furthermore, the agent-based architecture allows the system to adaptively terminate unpromising candidates early, reducing unnecessary compute cycles.

4.4 Prototype Functionality

At this stage of development, the project has successfully laid the foundation for an autonomous Neural Architecture Search (NAS) framework using Python. The primary objective of this system is to minimize human involvement in the design of deep neural networks (DNNs). By enabling a fully automated pipeline, the system can generate, train, and evaluate network architectures without manual tuning. This approach not only accelerates the experimentation process but also aims to democratize AI model design by making it more accessible and less reliant on expert intervention.

A key secondary goal of the project is to reduce the environmental and computational costs of NAS, in alignment with the principles of Green AI. The framework integrates tools that track energy consumption and carbon emissions, supporting the development of sustainable and energy-efficient AI models. By focusing on low-resource consumption and smart design choices, the project aspires to contribute to a more environmentally responsible future in AI research.

The core of the system is a multi-agent architecture where each agent is responsible for a distinct part of the NAS process. The *CoordinatorAgent* manages the overall execution and flow of the experiment. The *SearchAgent*, powered by Optuna, explores various hyperparameter configurations such as kernel size, number of channels, dropout rate, and learning rate. It constructs and trains a lightweight convolutional model, SimpleCNN, for a limited number of training epochs. Once the models are evaluated, the *EvaluationAgent* collects performance data (such as accuracy, parameter count, and floating-point operations) and compiles a summary of the top-performing trials. The results are logged and saved in a structured format as a JSON report.

Technically, the implementation leverages several powerful open-source libraries. PyTorch is used as the primary deep learning framework for building and training models. TorchVision provides access to standard image datasets such as MNIST, Fashion-MNIST, and CIFAR-10. Optuna serves as the engine behind the hyperparameter search, intelligently navigating the space of possible architectures to find optimal configurations. The system also includes optional support for CodeCarbon, which monitors energy usage and CO_2 emissions, offering a sustainability assessment for each training run. Other utilities such as THOP (for measuring computational complexity), Rich (for enhanced terminal output), YAML (for managing configurations), and TQDM (for real-time progress tracking) enhance the usability and maintainability of the framework.

The project follows a clean and modular code structure, allowing for scalability and ease of maintenance. The main script (`src/main.py`) acts as the entry point and links together all the core components: the Coordinator, Search, and Evaluation agents. Agent logic is encapsulated in the *`src/agents/`* directory, while the NAS logic powered by Optuna resides in *`src/nas/optuna_search.py`*. The baseline CNN architecture is defined in *`src/models/simple_cnn.py`*, and dataset loading is handled by *`src/datasets/loader.py`*. Experiment configurations are stored in YAML format in the *`experiments/configs/`* directory, and results are saved under *`experiments/results/`*.

When the framework is executed via the command `python src/main.py --config experiments/configs/baseline.yaml`, a well-orchestrated workflow is initiated. First, the system loads the configuration file to determine the dataset, number of trials, training parameters, and other settings. It then prepares the runtime environment by setting random seeds for reproducibility and detecting the most suitable hardware (CPU or GPU). If the datasets are not already downloaded, they are automatically fetched and stored locally. The NAS process begins with the Search Agent generating model variations, training them briefly, and passing performance data to the Evaluation Agent. At the end of all trials, the system identifies the best-performing architecture and saves a detailed summary in a timestamped JSON file.

This prototype serves as a working proof-of-concept that intelligent agents can autonomously design efficient neural networks. The system acts much like an experimental scientist: generating hypotheses (model architectures), testing them, analyzing outcomes, and iteratively improving based on results. In doing so, it supports three main pillars: automation, efficiency, and sustainability. It reduces the manual workload of researchers, discovers performant models with fewer computational resources, and promotes environmentally conscious practices in AI development.

The long-term vision for the project includes expanding its capabilities in several directions. Planned improvements involve incorporating advanced NAS techniques such as Reinforcement Learning and Evolutionary Algorithms, supporting larger and more complex datasets (like CIFAR-100 and ImageNet), and enhancing agent communication for better coordination. Future updates will also aim to integrate a cloud-based version of the system and a user-friendly interface for experiment configuration and monitoring.

In conclusion, this project represents a promising step toward building intelligent, efficient, and sustainable AI systems. By combining autonomous decision-making, modular design, and environmental awareness, it not only advances the state of NAS automation but also aligns with the broader movement toward responsible AI research.

5. Bibliography

Asher, C., Puyol-Antón, E., Rizvi, M., Ruijsink, B., Chiribiri, A., Razavi, R. and Carr-White, G. (2021). The Role of AI in Characterizing the DCM Phenotype. Frontiers in Cardiovascular Medicine, 8. doi:https://doi.org/10.3389/fcvm.2021.787614.

Atlas, L.Godlin. and Devi, A.Sindhu. (2025). Neural Architecture Search (NAS): A Survey of Methods, Challenges, and Future Directions. INTERANTIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT, 09(09), pp.1–9. doi:https://doi.org/10.55041/ijsrem52609.

Barret Zoph and Le, Q.V. (2016). Neural Architecture Search with Reinforcement Learning. arXiv (Cornell University), v2. doi:https://doi.org/10.48550/arxiv.1611.01578.

Berlina Rahmadhani, Purwono Purwono and Safar, N. (2024). Understanding Transformers: A Comprehensive Review. Journal of Advanced Health Informatics Research, 2(2), pp.85–94. doi:https://doi.org/10.59247/jahir.v2i2.292.

Bolaños, X. (2021). Natural Language Processing and Machine Learning. [online] Encora. Available at: https://www.encora.com/insights/natural-language-processing-and-machine-learning [Accessed 17 Nov. 2025].

Cal Newport (2012). So Good They Can’t Ignore You: Why Skills Trump Passion in the Quest for Work You Love. New York: Grand Central Publishing.

Chatterjee, D. and Rao, S. (2020). Computational Sustainability. ACM Computing Surveys, 53(5), pp.1–29. doi:https://doi.org/10.1145/3409797.

Chu, X., Zhang, B. and Xu, R. (2021). FairNAS: Rethinking Evaluation Fairness of Weight Sharing Neural Architecture Search. 2021 IEEE/CVF International Conference on Computer Vision (ICCV). doi:https://doi.org/10.1109/iccv48922.2021.01202.

Degadwala, Dr.S.D. and Degadwala, D.V. (2024). Survey on Systematic Analysis of Deep Learning Models Compare to Machine Learning. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 10(3), pp.556–566. doi:https://doi.org/10.32628/cseit24103206.

Dong, X., Liu, L., Musial, K. and Gabrys, B. (2021). NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size. IEEE Transactions on Pattern Analysis and Machine Intelligence, pp.1–1. doi:https://doi.org/10.1109/tpami.2021.3054824.

Elsken, T., Jan Hendrik Metzen and Hutter, F. (2019). Neural Architecture Search: A Survey. arXiv (Cornell University), v3. doi:https://doi.org/10.48550/arxiv.1808.05377.

G, K., Mohan, V. and Senthilkumar, S. (2023). A BRIEF REVIEW OF THE DEVELOPMENT PATH OF ARTIFICIAL INTELLIGENCE AND ITS SUBFIELDS. International Journal of Engineering Technologies and Management Research, [online] 10(6), pp.1–12. doi:https://doi.org/10.29121/ijetmr.v10.i6.2023.1331.

Google Cloud (2022). State of the Art Neural Networks - Neural architecture search (NAS). [online] YouTube. Available at: https://www.youtube.com/watch?v=QMWQ6RG-VbI [Accessed 30 Nov. 2025].

Haenlein, M. and Kaplan, A. (2019). A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence. California Management Review, [online] 61(4), pp.5–14. doi:https://doi.org/10.1177/0008125619864925.

Kalaivani, K.S., Uma, S. and Kanimozhiselvi, C.S. (2020). A Review on Feature Extraction Techniques for Sentiment Classification. 2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC). doi:https://doi.org/10.1109/iccmc48092.2020.iccmc-000126.

Kaplan, A. and Haenlein, M. (2019). Siri, Siri, in My hand: Who's the Fairest in the land? on the interpretations, illustrations, and Implications of Artificial Intelligence. Business Horizons, 62(1), pp.01-25. doi:https://doi-org.brad.idm.oclc.org/10.1016/j.bushor.2018.08.004.

LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep Learning. Nature, [online] 521(7553), pp.436–444. doi:https://doi.org/10.1038/nature14539.

Liang, J., Meyerson, E., Hodjat, B., Fink, D., Mutch, K. and Miikkulainen, R. (2019). Evolutionary neural AutoML for deep learning. Proceedings of the Genetic and Evolutionary Computation Conference. doi:https://doi.org/10.1145/3321707.3321721.

Liu, H., Simonyan, K. and Yang, Y. (2019). DARTS: Differentiable Architecture Search. ArXiv (Cornell University), v2. doi:https://doi.org/10.48550/arxiv.1806.09055.

Liu, S., Zhang, H. and Jin, Y. (2022). A survey on computationally efficient neural architecture search. Journal of Automation and Intelligence, 1(1), p.100002. doi:https://doi.org/10.1016/j.jai.2022.100002.

Liu, Y., Sun, Y., Xue, B., Zhang, M., Yen, G.G. and Tan, K.C. (2023). A Survey on Evolutionary Neural Architecture Search. IEEE Transactions on Neural Networks and Learning Systems, [online] 34(2), pp.550–570. doi:https://doi.org/10.1109/TNNLS.2021.3100554.

Liu, Z., Li, D., Lu, K., Qin, Z., Sun, W., Xu, J. and Zhong, Y. (2022). Neural Architecture Search on Efficient Transformers and Beyond. arXiv (Cornell University). doi:https://doi.org/10.48550/arxiv.2207.13955.

Lopes, V., Carlucci, F.M., Esperança, P.M., Singh, M., Yang, A., Gabillon, V., Xu, H., Chen, Z. and Wang, J. (2023). Manas: multi-agent neural architecture search. Machine Learning, 113(1), pp.73–96. doi:https://doi.org/10.1007/s10994-023-06379-w.

Milano, M., O'Sullivan, B. and Sachenbacher, M. (2014). Guest Editors' Introduction: Special Section on Computational Sustainability: Where Computer Science meets Sustainable Development. IEEE Transactions on Computers, 63(1), pp.88–89. doi:https://doi.org/10.1109/tc.2014.4.

Naqvi, A. (2020). Rise of Machine Learning. Artificial Intelligence for Audit, Forensic Accounting, and Valuation, pp.51–67. doi:https://doi.org/10.1002/9781119601906.ch4.

Nihkil Ketkar and Moolayil, J. (2020). PRACTICAL DEEP LEARNING WITH PYTORCH : optimizing generative adversarial networks with python. Apress Berkeley, CA.

Real, E., Aggarwal, A., Huang, Y. and Le, Q.V. (2018). Regularized Evolution for Image Classifier Architecture Search. arXiv (Cornell University), v7. doi:https://doi.org/10.48550/arxiv.1802.01548.

Siems, J.N., Zimmer, L., Arber Zela, Lukasik, J., Keuper, M. and Hutter, F. (2020). NAS-Bench-301 and the Case for Surrogate Benchmarks for Neural Architecture Search. [online] arXiv.org. Available at: https://api.semanticscholar.org/CorpusID:221266218 [Accessed 9 Jan. 2026].

Thorat, M., Pandit, S. and Balote, S. (2022). Artificial Neural Network: A brief study. Asian journal of convergence in technology, 8(3). doi:https://doi.org/10.33130/ajct.2022v08i03.003.

Tiwari, T., Tiwari, T. and Tiwari, S. (2018). How Artificial Intelligence, Machine Learning and Deep Learning are Radically Different? International Journal of Advanced Research in Computer Science and Software Engineering, 8(2), p.1. doi:https://doi.org/10.23956/ijarcsse.v8i2.569.

Wang, H. and Raj, B. (2017). On the Origin of Deep Learning. [online] arXiv.org. doi:https://doi.org/10.48550/arXiv.1702.07800.

Wang, L., Zhao, Y. and Yuu Jinnai (2019). AlphaX: eXploring Neural Architectures with Deep Neural Networks and Monte Carlo Tree Search. [online] arXiv.org. Available at: https://www.semanticscholar.org/paper/AlphaX%3A-eXploring-Neural-Architectures-with-Deep-Wang-Zhao/32c3892a07e16a32604de7d2724f993f3f2ad4af [Accessed 7 Jan. 2026].

Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K. and Hutter, F. (2019). NAS-Bench-101: Towards Reproducible Neural Architecture Search. arXiv (Cornell University), pp.7105–7114. doi:https://doi.org/10.48550/arxiv.1902.09635.