AUTOMATING THE DESIGN OF DEEP LEARNING MODELS USING NEURAL ARCHITECTURE SEARCH FOR MEDICAL IMAGE CLASSIFICATION

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Ridhanya Sree Balamurugan

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Electrical and Computer Engineering

May 2024

Fargo, North Dakota

NORTH DAKOTA STATE UNIVERSITY

Graduate School

Title

AUTOMATING THE DESIGN OF DEEP LEARNING MODELS USING NEURAL ARCHITECTURE SEARCH FOR MEDICAL IMAGE CLASSIFICATION

By

Ridhanya Sree Balamurugan

The Supervisory Committee certifies that this thesis complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Umamaheswara Rao Tida
Chair

Dr. Dharmakeerthi Nawarathna

Dr. Dali Sun

Dr. Danling Wang

Approved:

05/09/2024
Date

Dr. Benjamin Braaten
Department Chair

ABSTRACT

Designing Deep Learning (DL) models for medical image classification tasks poses significant challenges, demanding substantial expertise owing to the intricate nature and critical importance of the undertaking. Creating a DL model tailored for such purposes entails iterative processes of designing, implementing, and fine-tuning algorithms to achieve optimal performance. To mitigate these difficulties, Neural Architecture Search (NAS) has risen as a key field to generate the most effective DL models automatically. However, much of the previous studies involving NAS focus on automating the design of DL models for well-established datasets such as CIFAR-10 and ImageNet. This technique should also be extended to medical image datasets where detecting crucial features accurately in medical images is essential to detect specific illnesses correctly. Therefore, in this study, we investigate NAS to autonomously design best performing DL models for skin lesion detection, thereby demonstrating its usefulness for additional medical image classification endeavours.

ACKNOWLEDGEMENTS

I sincerely thank my supervisor, Dr. Umamaheswara Rao Tida, for his invaluable guidance and support throughout this research. His provision of laboratory facilities and resources has been instrumental to my success. I am thankful for his constant encouragement and mentorship, which has significantly contributed to my academic and personal growth. My deepest appreciation goes to my mother, whose unwavering support and belief in my capabilities have been a constant source of strength and motivation. Her sacrifices and love are the foundation of my aspirations. I am very grateful to all my friends and labmates for their unwavering support, understanding and encouragement. Special thanks to my labmates for their collaborative spirit and friendship, which have created a productive environment contributing to the success of my research. I also thank the supervisory committee members for their crucial support and input. Lastly, I would like to express my gratitude to everyone who contributed to this journey, your impact has been profound, and I am deeply thankful.

DEDICATION

To the Almighty God

&

To My Dear Family and Friends

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
CNN	Convolutional Neural Network
EHR	Electronic Health Records
NAS	Neural Architecture Search
RL	Reinforcement Learning
LSTM	Long Short-Term Memory
PPO	Proximal Policy Optimization
RNN	Recurrent Neural Network
AUC	Area Under the ROC Curve
ROC	Receiver Operating Characteristics
TPR	True Positive Rate
FPR	False Positive Rate
Acc	Accuracy
R-CNN	Region-based Convolutional Neural Network
YOLO	You Look Only Once
ReLU	Rectified Linear Unit
AC	Alternating Current
miRNA	microRNA
RNA	Ribonucleic Acid
DEP	Dielectrophoresis
TIE	T-shaped Interdigitated Electrode
DNA	Deoxyribonucleic acid
DI	Distilled Water
SAM	Segment Anything Model

HR	Herbicide-Resistant
SSWM	Site-Specific Weed Management
UAS	Unmanned Aircraft System
IoU	Intersection Over Union
ss	Single-Stranded
MHz	Megahertz
R	Reward
Vpp	Peak-to-Peak Voltage
CT	Computed Tomography
MRI	Magnetic Resonance Imaging
L	Litres
M	Molarity
nm	Nanometers
ED	Elliptical Dichroism
wandb	Weights and Biases Framework
ViT	Vision Transformers
ResNet	Residual Networks
HAM10000	Human Against Machine with 10000 Training Images
EDTA	Ethylenediamine Tetraacetic Acid
TE	Tris-EDTA
SSD	Single Shot Detector
VGG	Visual Geometry Group
BN	Batch Normalization
Conv1D	One-Dimensional Convolution Layer
Conv2D	Two-Dimensional Convolution Layer

LIST OF SYMBOLS

σ	Sigmoid Function
α	Learning Rate
ϵ	Epsilon
μ	Micro
%	Percentage
#	Number of Filters in a Convolution Layer

1. INTRODUCTION

1.1. Deep Learning in Medical Research

The advent of Deep Learning (DL) within medical research signifies a paradigm shift towards data-driven insights and precision medicine. As a subset of Machine Learning (ML), DL employs artificial neural networks with multiple layers to model complex nonlinear relationships in large datasets [1]. This innovative approach has the unique capability of autonomously learning and improving, distinguishing itself from traditional computational methods through its ability to process and analyze data at an unprecedented scale and depth [2].

Integrating DL into medical research can be traced back to its foundational success in recognizing intricate patterns and anomalies within vast amounts of data, particularly well-suited to the multifaceted nature of medical diagnostics and research. DL algorithms have begun to outperform traditional statistical methods and even expert human interpretation in tasks such as image classification, natural language processing, and predictive modelling [3]. Such advancements are not merely incremental; they represent transformative progress in how medical data are analyzed, interpreted, and applied to patient care.

Central to the utility of DL in this domain is its proficiency in handling medical imaging data. Technologies such as Magnetic resonance imaging (MRI), Computed Tomography (CT) scans, and X-rays generate large volumes of high-dimensional data, presenting a challenge for traditional analysis techniques. DL models, particularly convolutional neural networks (CNNs), have demonstrated unparalleled accuracy in interpreting these images, enabling the detection of pathological features with a previously unattainable level of precision [4]. This capability extends beyond diagnostics to include monitoring disease progression and predicting treatment outcomes, thereby offering a holistic approach to patient care.

Moreover, DL has extended its impact to the realm of genomics, which aids in understanding the complex genetic underpinnings of various diseases. By analyzing the vast datasets generated by genomic sequencing, DL algorithms can identify genetic variants associated with specific conditions, facilitating the development of targeted therapies [5]. This application underscores the move towards personalized medicine, where treatments can be tailored to the individual genetic makeup of patients, optimizing efficacy and minimizing adverse effects.

Another significant contribution of DL in medical research is its ability to shift through Electronic Health Records (EHRs) to extract meaningful insights. The heterogeneous and unstructured nature of EHR data has traditionally posed a barrier to efficient analysis. However, DL techniques, through their ability to learn feature representations directly from raw data, can uncover patterns and correlations that inform clinical decision-making and policy development [6]. This includes predicting patient outcomes, identifying disease risk factors, and optimizing resource allocation within healthcare systems.

Considering these advancements, DL is not just an auxiliary tool but a foundational pillar that reshapes the medical research landscape. Its ability to decipher the complexities of biological data and translate these findings into clinical applications accelerates the transition toward a more informed, efficient, and personalized healthcare paradigm [7]. As we continue to explore and refine DL technologies, their potential to contribute to medical science is boundless, promising a future in which data-driven insights with unparalleled accuracy and depth underpin medical decisions.

1.2. Challenges in Developing DL Models for Medical Research

Incorporating DL technologies into medical research introduces a landscape replete with unprecedented opportunities and formidable challenges. The endeavour to harness the full capabilities of DL models is marked by several obstacles intricately linked to the models' inherent complexity, the unique nature of medical data, and the stringent demands for accuracy and reliability in clinical outcomes [8]. A nuanced understanding of these impediments is imperative to advance the field and realize the extensive potential of DL in medical applications.

1.2.1. Navigating Data Challenges

The bedrock of DL efficacy depends on extensive, diverse, and accurately annotated datasets. However, securing such datasets poses significant challenges in medical research. Privacy concerns, regulatory limitations, and innate heterogeneity of medical data sources contribute to the paucity and fragmentation of accessible data. Furthermore, medical datasets often exhibit imbalances with the under-representation of certain conditions or outcomes, complicating the development of models that perform uniformly across various patient demographics or disease spectra [9]. The issues of data scarcity and imbalance present substantial barriers to crafting robust and universally applicable DL models.

1.2.2. Navigating Challenges Associated with DL Model Development

Optimized neural network structures are of paramount importance in medical-related DL tasks due to the critical nature of the predictions and their direct impact on patient care and deployment. This neural network structure can be conceptualized as a graph, specifically a directed graph. This graph structure helps us understand how information flows through the network when an input is passed [10]. The structure of the neural network graph is crucial for achieving high performance and enabling the learning of complex features. A well-designed architecture can capture intricate patterns and features in the input data, leading to better performance. However, achieving an optimized neural network architecture requires a combination of expertise, experimentation, and time.

1.2.3. Addressing Model Complexity and Overfitting

DL models tailored for decoding complex medical data are distinguished by their elaborate architectures featuring a multitude of parameters and layers. Although this intricacy enables the capture of subtle data patterns, it concurrently elevates the risk of overfitting. Overfitting—a scenario in which a model memorizes rather than learns from the training data—jeopardizes the model's performance on novel, unseen datasets [11]. In the medical sphere, where prediction inaccuracies carry grave consequences, the imperative of maintaining model generalizability cannot be overstated.

1.2.4. Enhancing Interpretability and Fostering Trust

A pivotal challenge in deploying DL models for medical research is their "black-box" nature, which obscures the models' internal decision-making processes. This opacity complicates clinicians' and researchers' efforts to comprehend the rationale underlying model predictions. This is critical to engendering trust and acceptance of DL-based diagnostic and prognostic tools in clinical environments [12]. The quest for transparent, interpretable models is thus essential for integrating DL into healthcare settings, where decisions directly impact patient care.

1.2.5. Overcoming Computational and Temporal Constraints

DL models are marked by their high computational intensity, necessitating substantial hardware resources. This is particularly true for models dealing with high-dimensional data, such as those encountered in medical imaging or genomics. The considerable computational expenses associated with iterative training and refinement of DL models can render them inaccessible to many research institutions, thereby constraining the scalability and adoption of DL applications in medical research [13].

1.2.6. Navigating Ethical and Regulatory Landscapes

As DL models have become increasingly integral to clinical workflows, they use a complex array of ethical and regulatory considerations. Ensuring the ethical deployment of these models requires stringent adherence to principles that safeguard patient privacy, data security, and equitable healthcare outcomes [14]. Moreover, as regulatory frameworks evolve to align with technological progress, continuous dialogue between developers, researchers, clinicians, and policymakers is vital to guarantee that DL medical applications meet the highest standards of safety, efficacy, and fairness.

The path to leveraging the transformative potential of DL in medical research is fraught with several challenges. Addressing the intricacies of data management, model design, interpretability, computational demands, and ethical governance is paramount for successfully integrating DL in healthcare. Tackling these hurdles necessitates a collaborative, interdisciplinary approach that merges expertise from computer science, medicine, ethics, and policy, thereby unlocking the full capacity of DL to enhance patient outcomes and enrich medical knowledge.

1.3. Diving Deeper into the Hurdles in Crafting Optimized Neural Networks

In the realm of DL, particularly within medical applications, the optimization of neural network structures stands as a pivotal endeavour. The significance of this optimization stems from the profound implications associated with the predictions generated by these networks [15]. Unlike many other domains, where errors might result in inconvenience or financial loss, in the medical field, the stakes are infinitely higher. The predictions made by DL models directly influence patient care, treatment decisions, and, ultimately, health outcomes. Thus, pursuing optimized neural network architectures becomes not merely a technical challenge but a moral imperative, with the potential to significantly impact the lives of individuals seeking medical assistance. However, building an optimized, high-performing neural network for a DL task is challenging and requires considerable expertise, time, and careful consideration.

Designing an effective neural network architecture demands a deep understanding of the problem domain, the characteristics of the input data, and the desired output. This understanding allows data scientists and ML engineers to make informed decisions regarding the network's structure, including the number of layers, the types of neurons, and the connectivity patterns [16]. Furthermore, the expertise of neural network developers in architectural innovations, such as convolutional or recurrent layers for image and sequential data, respectively, and careful incorporation of such layers can significantly impact the network's ability to learn complex features relevant to the task.

Besides crafting and experimenting with different neural network architectures, DL models typically undergo refinement through a process known as Hyperparameter Tuning to optimize their performance [17]. However, optimizing a neural network's hyperparameters requires extensive experimentation and iterative tuning. Hyperparameters in the context of neural networks refer to the configuration settings that are external to the model and are not learned during the training process [18]. These settings govern the behaviour of the neural network during training and impact its performance, convergence, and generalization ability [19]. Hyperparameters such as learning rate, batch size, and activation functions are crucial in shaping the model's performance and generalization capabilities. Achieving the optimal configuration often involves conducting numerous training runs with different hyperparameter settings, evaluating their impact on the model's performance, and refining the choices based on results. This process demands a deep understanding of the underlying principles of optimization and the ability to interpret experimental outcomes to guide further adjustments.

Additionally, building an optimized neural network entails a significant investment of time, both in terms of model development and computational resources. Experimenting with different architectures and hyperparameters requires running multiple training iterations, which can be computationally intensive, particularly for large datasets and complex models. Furthermore, the iterative optimization process means that achieving the desired performance may take considerable time and effort, as data scientists iterate through various architecture configurations and fine-tune the model's parameters [20].

1.3.1. Navigating the Challenges Associated with Hyperparameter Tuning in Neural Network Optimization

In the intricate process of developing optimized neural networks for medical research, hyperparameter tuning has emerged as a critical but challenging endeavour. Hyperparameters, external configurations that dictate the model's architecture and behaviour, are instrumental in a model's learning efficacy [21]. However, identifying the optimal hyperparameter constellation presents several hurdles that can impede the creation of high-performance models.

- **The Challenge of Complexity and Dimensionality :** The task of hyperparameter tuning is daunting owing to the vast array of hyperparameters and their potential configurations. Variables such as learning rates and number of neurons per layer, along with more complex parameters such as dropout rates and convolution kernel sizes, each have a significant influence on model performance [22]. This diversity spawns a multidimensional search space, where finding the optimal setup becomes a herculean task regarding computational demand and time [20].
- **The Maze of Interdependencies :** Hyperparameters do not operate in isolation; adjustments to one parameter can necessitate changes in another, creating a tangled web of interdependencies [23]. The interplay between hyperparameter settings and model performance is typically nonlinear and counterintuitive, further complicating the optimization process [24]. Such complexity renders simple or brute-force search strategies insufficient and calls for more nuanced, iterative approaches capable of navigating the intricate hyperparameter landscape.
- **The Hurdle of Resource Intensiveness :** Evaluating potential hyperparameter sets is marked by their intense resource consumption [17]. Given the vast data volumes and complex architectural characteristics of medical research models, each evaluation can significantly strain computational resources and time [25]. This intensiveness limits the practicality of thorough searches, particularly for teams with restricted access to high-performance computing capabilities.
- **The Dilemma of Limited Generalizability :** Discovering an optimal set of hyperparameters for a given dataset or problem does not guarantee its applicability to other datasets or their variations [17]. This is particularly relevant in medical research, where dataset disparities (stemming from patient demographics and measurement techniques) are routine [26]. Consequently, the necessity to recalibrate hyperparameters for new datasets or problems markedly increases the effort and timeline required for model deployment.

- **The Influence of Expertise and Intuition :** Despite the advent of automated hyperparameter optimization technologies, the tuning process relies heavily on the modeler's experience and intuitive judgment [27]. Decisions regarding which hyperparameters to adjust and the initial search boundaries are typically more art than science grounded in personal expertise.
- **The Evolution of Solutions and Their Shortcomings :** Efforts to mitigate the challenges of hyperparameter tuning have yielded strategies such as grid search, random search, and Bayesian optimization [28]. However, each of these solutions has its own limitations: no guarantee of optimization in a random search, inefficiency in the grid and reliance on the complexity of the surrogate model for optimal results in Bayesian optimization [29].

Hyperparameter tuning is a significant bottleneck in crafting optimized neural networks, constrained by search space complexity, resource demands, and the need for specialized insight. Overcoming these constraints is pivotal for enhancing the performance and development efficiency of DL models in medical research. This underscores the critical need to develop automated approaches for neural network design.

1.4. Advent of Neural Architecture Search

The intricate process of developing optimized neural networks, coupled with the challenges inherent in hyperparameter tuning methodologies, underscores the necessity for automated algorithms capable of autonomously crafting neural network architectures. As discussed earlier, while traditional approaches to model optimization rely on manual intervention and iterative adjustments of hyperparameters, such practices often require expertise and entail substantial time investments. They may not consistently yield the most effective configurations. Consequently, there has been a growing recognition of such methods' limitations, prompting a shift towards automated solutions to streamline the network design process and enhance model performance.

In this context, Neural Architecture Search (NAS) [30] emerges as a promising paradigm that seeks to revolutionize the landscape of neural network development. NAS leverages the power of computational algorithms, often drawing inspiration from evolutionary strategies, reinforcement learning techniques, or gradient-based optimization, to navigate the vast space of possible network architectures and identify configurations that maximize performance for a given task [31]. By automating the search process, NAS transcends the constraints of manual hyperparameter tuning, offering a more efficient and scalable approach to model optimization.

One key advantage of NAS is its ability to conduct automated exploration of the architectural space, facilitating comprehensive evaluation of a wide range of network configurations. Unlike manual approaches, which may overlook promising design choices or require exhaustive experimentation, NAS algorithms systematically navigate through architectural permutations, uncovering patterns and architectures that exhibit superior performance characteristics [32]. This automated exploration not only accelerates the model development process but also enhances the likelihood of discovering novel architectures that may have been overlooked using traditional methods.

Moreover, NAS algorithms are adept at optimizing resource utilization, a critical consideration in the context of neural network design. By employing effective strategies, NAS has the potential to mitigate the computational burden associated with exhaustive exploration of the architectural space. This enables efficient allocation of computational resources and reduces the time and cost involved in model development, making NAS particularly well-suited for large-scale optimization tasks and real-world applications where computational efficiency is paramount.

Furthermore, NAS fosters transferability and generalization by identifying architectural principles and design patterns that transcend specific datasets or tasks. Through iterative refinement and knowledge transfer mechanisms, NAS algorithms leverage insights from previous searches to inform subsequent experiments, facilitating the development of architectures that generalize well across diverse domains [33]. This capacity for knowledge transfer enhances the robustness of neural network models and accelerates innovation and progress in the field of deep learning.

In summary, NAS represents a paradigm shift in the realm of neural network design, offering automated solutions to the challenges associated with manual hyperparameter tuning and model optimization. By harnessing the capabilities of computational algorithms, NAS enables efficient exploration of the architectural space, optimizes resource utilization, and fosters transferability and generalization across diverse domains. As the demand for high-performing neural network models continues to grow, NAS stands poised to revolutionize the model development process, paving the way for advancements in Artificial Intelligence (AI).

1.5. Thesis Overview

In this thesis, we delved into the forefront of NAS and the REINFORCE policy gradient method [30], aiming to significantly advance the development of DL architectures for medical image classification. This endeavour is driven by the crucial challenge of overcoming the limitations posed by the traditional manual design of neural networks, which require extensive expertise and considerable time investment, often rendering the development of high-performance DL models a daunting task. Our methodology harnesses the autonomous design capabilities of NAS, effectively reducing reliance on manual expertise, and pairs this with the strategic optimization offered by Reinforcement Learning (RL) to refine and enhance DL architectures specifically crafted for the intricate demands of medical imaging, which has not been explored before.

In order to prove the efficacy of NAS in autonomously designing neural networks for medical image analysis, our research applies this integrated approach to precisely classifying skin lesions within the DermaMNIST dataset. Through the application of NAS, augmented by the REINFORCE policy gradient method, our research seeks not only to streamline the process of architectural design and optimization for DL models but also to conduct a thorough evaluation of the performance and efficiency of these autonomously generated architectures against the current benchmarks set by state of the art human architected models.

This exploration is positioned to make significant contributions to the field of medical image analysis, showcasing the viability and efficacy of integrating NAS with cutting-edge RL techniques in developing DL models. The models emerging from our study are designed not only to meet but also to surpass existing standards of performance and efficiency in medical imaging, thereby establishing new paradigms for diagnostic accuracy and computational effectiveness. Our findings are anticipated to expand the body of knowledge on NAS and RL in medical imaging, thereby setting the stage for groundbreaking innovations that potentially transform the domain of medical diagnostics. By unveiling new avenues for developing DL models, this research underscores the transformative impact of NAS and RL, promising to revolutionize medical image classification and, by extension, enhance patient care and healthcare delivery efficiency. Additionally, this thesis extends beyond its primary focus to encompass additional projects and topics that are not central to the main research question but still relevant.

2. NEURAL ARCHITECTURE SEARCH FOR MEDICAL IMAGE CLASSIFICATION [1]

2.1. Abstract

The custom design of Deep Learning (DL) models for medical image classification represents a substantial challenge, compounded by the intricate balance between model complexity and practical constraints of point-of-care deployment. This study delves into the potential of Neural Architecture Search (NAS) with the REINFORCE algorithm as a strategy to circumvent these obstacles and streamline the development of efficient DL models tailored for the DermaMNIST dataset. Through rigorous experimentation, we unveiled the capabilities of NAS to automate the model design process and achieve an optimal balance between accuracy and computational efficiency. Our findings suggest that NAS holds significant promise in enhancing the performance and applicability of DL models in medical diagnostics. This work not only contributes to the ongoing dialogue on the feasibility of NAS in medical image analysis but also sets a precedent for future research, underscoring the necessity for innovative solutions in the era of digital medicine.

2.2. Introduction

The expanding use of DL in medical research is fueled by its capacity to offer precise, automated, and effective analysis. This advancement not only boosts diagnostic procedures but also enhances patient care, potentially leading to life-saving outcomes. The proficiency of meticulously crafted DL models in discerning intricate features in medical images has greatly propelled this trajectory [34]. However, designing a neural network that makes a DL model is

[1] This chapter is based on the manuscript "On the Feasibility of Neural Architecture Search for DermaMNIST Dataset" submitted to IEEE Research and Applications of Photonics in Defense Conference. This paper was authored by Ridhanya Sree Balamurugan, Madhava Sarma Vemuri, Dharmakeerthi Nawarathna and Umamaheswara Rao Tida. Ridhanya Sree Balamurugan was the lead author, responsible for conceptualizing the research problem, drafting the manuscript, developing the research methodology, coding the Python scripts and conducting simulations. Madhava Sarma Vemuri contributed to the result verification and proofreading of the manuscript, while Dr. Dharmakeerthi Nawarathna and Dr. Umamaheswara Rao Tida provided guidance on the research methodology, allocated resources, and contributed to the manuscript's writing.

a complex task requiring specialization. Moreover, the process itself is very time-consuming, where the developed initial neural network design is tested and fine-tuned several times to obtain an optimal DL model that has superior performance on the target task [35]. Therefore, there is a need to build algorithms that can efficiently automate this process. Recent research in this direction has given rise to NAS, an algorithm that can autonomously design neural networks efficiently for a target DL task [32].

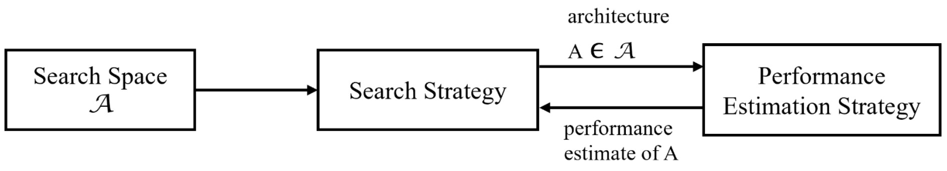
Search Space $\mathcal{A}$
Search Strategy
architecture $A \in \mathcal{A}$
Performance Estimation Strategy
performance estimate of A

Figure 2.1. Key Components of NAS

As illustrated in Figure 2.1 adapted from [36], NAS employs a sophisticated framework comprised of three fundamental components: the search space, the search strategy, and the performance estimation strategy. Central to NAS is the concept of the search space, a meticulously defined domain encompassing the fundamental building blocks of neural network architectures. Within this space, various architectural elements, such as layers, neurons, and connectivity patterns, are carefully crafted to encapsulate the diverse range of configurations that contribute to the structure of a neural network.

The search strategy is the guiding force that navigates through the vast expanse of the search space, orchestrating the assembly of these architectural components into coherent end-to-end deep learning models. Through a process of exploration and optimization, the search strategy systematically samples from the search space, iteratively synthesizing and evaluating candidate architectures with the help of a performance estimation strategy to identify those with superior performance characteristics. This iterative refinement process is essential for uncovering novel architectural designs and fine-tuning model configurations to maximize performance across various tasks and datasets. As illustrated in Figure 2.1, when the search strategy constructs Architecture A by assembling components in the search space, it relies on the performance estimation strategy to evaluate its performance. This evaluation is subsequently leveraged by the search strategy to refine the construction of superior architectures.

Central to the efficacy of NAS is the performance estimation strategy, which serves as a critical mechanism for efficiently evaluating the sampled neural network architectures. Given the computational complexity associated with training and evaluating numerous candidate models, the performance estimation strategy plays a pivotal role in enabling the search strategy to make informed decisions about the design and refinement of architectures. Moreover, the performance estimation strategy facilitates rapid and accurate assessment of architecture performance, thereby accelerating the optimization process and enhancing the effectiveness of NAS. Together, these three components form the backbone of the NAS framework, providing a systematic and automated approach to the design and optimization of neural network architectures.

Previous studies in NAS have proved the efficacy of using Reinforcement Learning (RL) as a search strategy in building architectures for DL models that outperformed state of the art architectures designed manually by experts in the Imagenet classification task [30][37]. Additionally, the neural networks designed by these RL-based NAS were lightweight, an important criterion for edge deployment, as compared to the existing state of the art DL models. Therefore, in this work, we investigate if NAS integrated with the REINFORCE policy gradient method, a class of RL algorithms, can create efficient neural network designs for medical image classification tasks. We have proved the efficacy of our approach by developing efficient DL architecture designs to detect and classify skin lesions on the DermaMNIST dataset [38].

2.2.1. Related Works

Our research is related to the previous studies in the domain of NAS with RL [30][37], where they used NAS frameworks to design Convolutional Neural Networks (CNNs) for the ImageNet Classification tasks. [30] designed NAS using Long Short-Term Memory (LSTM) and REINFORCE policy gradient to search for optimized variable-length hyperparameters for CNN layers. [37] designed a NAS framework utilizing Proximal Policy Optimization (PPO), a class of RL algorithms, and LSTM to design two cells, called the Normal and the Reduction cell. They stacked these cells to formulate an end-to-end neural network for image classification on the CIFAR-10 dataset. The optimized neural network was then scaled for its application on larger datasets. Although the above-mentioned studies inspired our research, they did not investigate generating top performing DL architectures for medical use cases, specifically for skin lesion detection and classification on the DermaMNIST dataset.

Many of the previous research [34][35][38][39][40] involving skin lesion classification using the DermaMNIST dataset utilized expert designed neural networks to report the results. Of these studies, two notable NAS-based research are [34] and [35], both of which utilized evolutionary algorithms to build a NAS framework. This is vastly different from the RL-based approach that we investigated in this study.

Additionally, [34] used AutoML techniques, which are closely related to our NAS framework in autonomously designing Machine Learning (ML) models on the DermaMNIST dataset. These techniques include auto-sklearn [41], AutoKeras [42] and Google AutoML Vision. auto-sklearn, which is based on the python package scikit-learn [43], utilizes techniques like Bayesian optimization and ensemble methods to search for optimized input processing methods and ML classifiers such as random forest along with their hyperparameters. AutoKeras, which is based on the Keras package [44], utilizes NAS framework guided by Bayesian Optimization. Google AutoML Vision, available as a service from Google Cloud (as of July 2021), is a commercial AutoML tool designed for image recognition and classification tasks. The techniques employed in these frameworks vastly differ from the approach we investigated in this study.

2.3. Methodology

The entirety of our framework's implementation is illustrated in Figure 2.2. Our NAS framework generates DL model architecture configurations autonomously using a component known as the LSTM controller. This LSTM controller sequentially samples a neural network architecture in over 30 iterations to build a DL model, referred to as the child model in our framework. This sampled architecture comprises two units, Basic and Reduction Unit, comprising three subunits each. Each subunit comprises five nodes, I1, I2, OP1, OP2 and CO, as shown in Figure 2.2. Once the child model is constructed as per the sampled architectural design, it is trained for 20 epochs to detect and classify skin lesions on the target DermaMNIST dataset. The validation accuracy obtained is used as a low-fidelity performance estimate to evaluate the effectiveness of the generated architecture designed by the LSTM controller. The REINFORCE algorithm also utilizes this performance to update the weight parameters of the LSTM controller so that it is trained to generate better performing architectures over time.

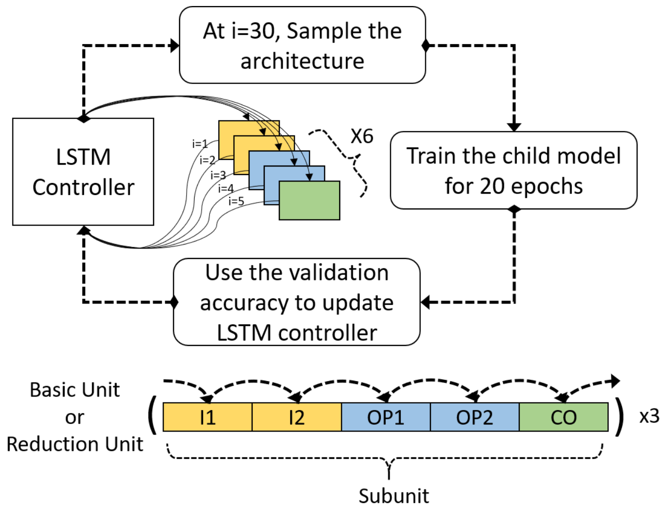
At i=30, Sample the architecture
LSTM Controller
i=1
i=2
i=3
i=4
i=5
X6
Train the child model for 20 epochs
Use the validation accuracy to update LSTM controller
Basic Unit or Reduction Unit
I1
I2
OP1
OP2
CO
x3
Subunit

Figure 2.2. NAS Framework Overview

Subsequent subsections provide a detailed explanation of the LSTM controller's implementation and operation, nuances in the child model architectural designs, and an elaboration on the REINFORCE policy gradient algorithm.

2.3.1. Child Model Design

A child model sampled by the LSTM controller consists of two units, Basic and Reduction Unit, as shown in Figure 2.3. Just like [37], the basic unit takes the input image as its input and maintains its size, while the reduction unit takes the output of the basic unit as its input and reduces the feature map size with a stride of 2 and doubles the depth size. Each of these units is composed of three subunits. Each subunit comprises five nodes: two Input nodes (I1 and I2), two Operation nodes (OP1 and OP2), and a combining operation node (CO).

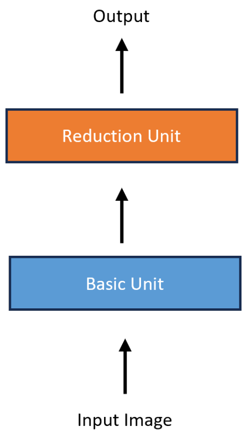

Figure 2.3. Child Model Design

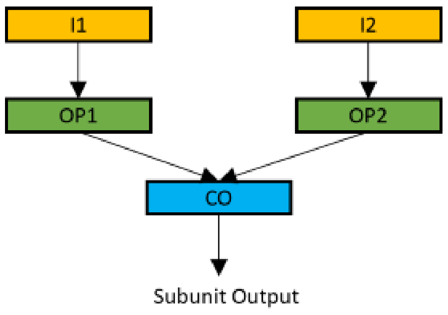

Figure 2.4. Subunit Design

As shown in Figure 2.4, to build a subunit, the two operation nodes OP1 and OP2 are applied to the input nodes, I1 and I2, respectively. The two intermediate outputs are then combined using the combining operation node, CO. The choices from which the LSTM controller samples these nodes make our search space. Similar to [37], for the basic unit, I1 and I2 can either be the input images or any previous subunit outputs within the unit. The final output of the basic unit is the output of the third subunit concatenated with any unconnected subunit outputs within that unit. Likewise, for the reduction unit, I1 and I2 can either be the final output of the basic unit or any previous subunit output within the unit, and its final output is the output of the third subunit concatenated with any unconnected subunit outputs within that unit. For both basic and reduction units, OP1 and OP2 can be from any of the 10 two-dimensional convolution operations, which include (i) 3*3 average pooling, , (ii) 5*5 max pooling, (iii) 1*1 convolution, (iv) 3*3 depthwise-separable convolution, (v) 7*7 depthwise-separable convolution, (vi) 3*3 dilated convolution, (vii) 3*3 max pooling, (viii) 7*7 max pooling, (ix) 3*3 convolution, and (x) 5*5 depthwise-separable convolution. CO node can either be add or concat operations for both the units.

In summary, the LSTM controller samples two units to generate the architecture configuration for the child model. Each unit comprises three subunits, which are, in turn, composed of five nodes. Therefore, to sample one child model's architecture, the LSTM controller has to sample 30 nodes from the search space.

2.3.2. Implementation of the LSTM Controller

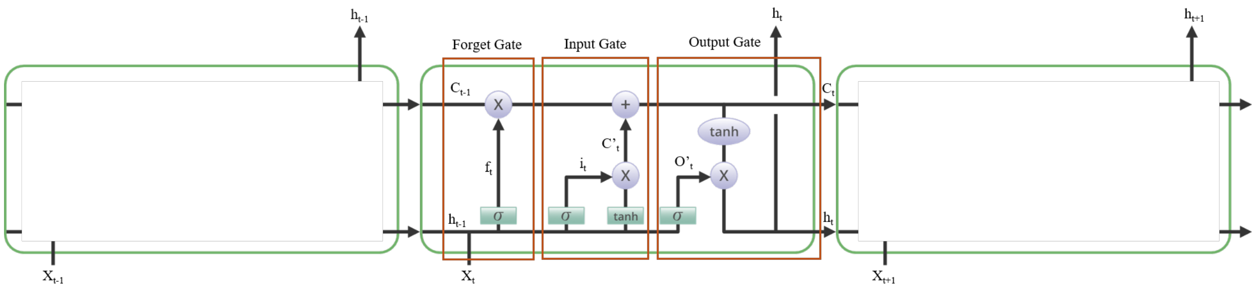

Figure 2.5. Architecture of LSTM Cell. Image edited from GeeksforGeeks

The fundamental concept guiding the sequential sampling process of the LSTM controller revolves around the utilization of LSTM layers. LSTMs are a type of Recurrent Neural Network (RNN) that are particularly well-suited for tasks involving sequential data, such as time series forecasting, natural language processing, and speech recognition. They were introduced by [45] and vastly differ from traditional neural networks because they can learn long-term dependencies. For instance, LSTM can predict the last word in a statement given the previous words as input, while the traditional neural networks struggle at this task. This is because LSTMs can practically learn to remember information over a long period of time. Since this particular task of predicting the last word in a statement depends on remembering the context using the preceding words, the traditional neural networks fail to perform well as they don't excel at capturing long-term dependencies.

An LSTM cell, shown in Figure 2.5, contains various components (or gates) that allow it to learn and forget information over time selectively. This enables LSTMs to capture long-term dependencies. The line connecting C_{t-1} and C_t is where the information flows from the previous LSTM cell to the next through the current cell. C_t, referred to as the cell state of the LSTM cell, serves as a memory that retains information over long sequences and across multiple time steps. Additionally, each LSTM cell takes a time-dependent input. Time-dependent inputs refer to data where the order or sequence of observations matters, and each observation is associated with a specific time or time step. In other words, the input data has a temporal structure, and the relationship between observations is not independent but dependent on their order in time. As illustrated in Figure 2.5, the three LSTM cells take inputs at a specific time, t-1,t and t+1.

The LSTM cells transmit information among themselves through the various gates, which determine what information to add or remove before passing it to the next cell. These gates are explained in detail below.

2.3.2.1. Forget Gate

Forget gate determines the amount of previous information or cell state, C_{t-1} to be retained using the equation below.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f)$$

where σ, referred to as a sigmoid layer, is an activation function which outputs a number between 0 and 1; h_{t-1}, referred to as a hidden state, is the output of the previous LSTM cell; x_t is the input at time t ; (W_f, b_f) are the weight parameters associated with the forget gate of the LSTM cell. f_t ranges between 0 and 1 and is multiplied with the previous cell state, C_{t-1}, as shown in Figure 2.5, to determine the amount of information to retain from the previous cell state. Multiplying f_t value of 1 with C_{t-1}, denotes that the information from the preceding cell is completely retained for further processing, and 0 denotes that this information is completely discarded.

2.3.2.2. Input Gate

The input gate is a pivotal component responsible for determining how much new information should be incorporated into the cell state, C_t. This is performed in two parts, represented mathematically using the following equations.

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$

$$C'_t = tanh(W_C.[h_{t-1}, x_t] + b_C)$$

where (W_i, b_i, W_C, b_C) are the weight parameters associated with the input gate and $tanh$, referred to as tanh layer, is an activation function which outputs values between -1 and 1. The first equation outputs values between 0 and 1, which, when multiplied with the outcome of the second equation, determines how much of the new information has to be added to the cell state, C_t.

2.3.2.3. Output Gate

The output gate determines the final cell state output, C_t, and hidden state output, h_t, of the LSTM cell. C_t, determined by the results of the input and forget gate, remains unchanged while passing through this gate. h_t is determined by the following equations.

$$O'_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$$

$$h_t = O'_t * tanh(C_t)$$

where (W_o, b_o) are the weight parameters associated with the output gate. The first equation outputs values between 0 and 1, which, when multiplied with the outcome of the second equation, determines the hidden state output h_t of the LSTM cell. The hidden state output of an LSTM cell is not only passed to the next LSTM cell in the sequence but is also produced as the final output of that cell. This hidden state basically contains the filtered version of the cell state to selectively retain and focus on relevant information while discarding less important details. As the the cell state is a long-term memory that stores information over the entire sequence of inputs at different timesteps, it can potentially contain a mix of both relevant and irrelevant information to predict outcomes for an input at that time step. Therefore, the hidden state captures the relevant short-term dependencies from the cell state to produce the output of the LSTM cell for the current time step, as determined by the output gate.

As shown in Figure 2.5, each LSTM cell, stacked horizontally, can be used to process inputs at a specific time step. Usually, this is referred to as an LSTM layer, which encapsulates the multidimensional hidden and cell states. This vector-based memory structure enhances the LSTM's capacity to learn features from sequential data, making it proficient in capturing complex patterns and dependencies.

In our implementation, we utilize an LSTM layer, akin to the depiction in Figure 2.5, where both the hidden and cell states are vectors of size 100. Input X_t at time t is passed as its input. The output from this LSTM layer at each timestep is then connected to 30 different fully connected layers of varying sizes. A fully connected layer, or Dense layer, is a fundamental component of any DL model. In a dense layer, each neuron or unit is connected to every neuron in the previous layer, forming a fully connected network topology. Mathematically, the operation performed by a dense layer is illustrated using an example as follows:

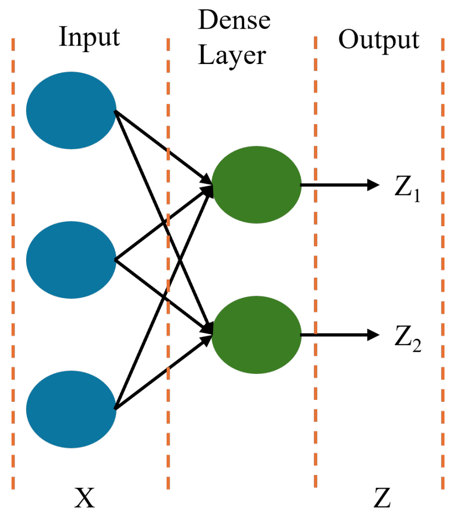

Figure 2.6. Illustrative Example of Fully Connected Layers.

Let X, a vector of size $3 * 1$, be the input to the dense layer of size 2. The output of the dense layer, denoted as Z of size $2 * 1$, can be calculated using the equation $Z = W.x + b$, where W, a matrix of size $2 * 3$ and b, a vector of size $2 * 1$, are the weight parameters of the dense layer. This is demonstrated in Figure 2.6.

The size of vector X, in our implementation, is 100 as the LSTM layer's output at each timestep is of size 100. The size of the dense layers to which this vector X is connected in the LSTM controller in our framework varies according to the number of choices available in the search space to sample a particular node. This is demonstrated in Figure 2.2. For instance, to choose a CO node, the LSTM controller uses a dense layer of size 2 as the choices for the CO node in the search space are add and concat operations. To sample a choice for a node, the LSTM controller uses probability distributions, which can be generated by applying a softmax activation function defined by $\sigma(Z_i) = e^{Z_i} / \sum_{j=1}^{K} e^{Z_j}$, where Z, of size $1 * K$, is the output of each of these dense layers. This is often referred to as softmax probability distribution.

2.3.3. Working of the LSTM Controller

To generate an architecture configuration, the LSTM controller samples choices for the 30 nodes sequentially. That is, at time t=0, it samples the choice for input node I1 for the basic unit's first subunit. Similarly, at t=29, it samples the choices for the CO node of the reduction unit's third subunit. Additionally, the five nodes in a subunit are sampled sequentially in an order - I1, I2, OP1, OP2 and CO, and the reduction unit's subunits are sampled only after the basic unit's architecture is sampled. This is demonstrated in Figure 2.2.

Initially, at time t=0, a set of 30 zeroes, denoting 30 different nodes that the controller has to sample over the course of 30 iterations, is passed as inputs to the LSTM controller. This generates 30 different softmax probability distributions. As we intend to choose only the input node I1 at t=0, we sample a choice for I1 based on the respective softmax probability distribution, ignoring the remaining 29 softmax probability distributions. The sampled choice, I1, is then added to the first position of the set of 30 zeroes, replacing the first zero. This denotes that the first node is sampled. Then, this new input formed by adding a set of 29 zeroes to the choice sampled for I1 is passed as input to the LSTM controller at time t=1 to sample I2. This process continues until all the 30 zeroes are replaced with the respective node choices. As the LSTM controller can only process numbers, each choice available for the node in the search space is assigned a numerical encoding, starting from 1 and incrementing by 1 up to the total number of choices in the search space. Thus, the numbers from this numerical encoding are utilized to replace the respective 0-s for each choice sampled by the LSTM controller.

In our implementation, as there are no other potential choices in the search space for I1 and I2 for the basic unit's first subunit other than the input image, we choose them as the two input nodes with a probability of 1. Similarly, we choose the basic unit's final output as the input nodes, I1 and I2, for the reduction unit's first subunit with a probability of 1. The strategy for choosing a particular node from a softmax probability distribution is elaborated in sections dedicated to the REINFORCE policy gradient.

Once the LSTM controller samples these 30 nodes, a child model is built and trained for 20 epochs on the target DermaMNIST dataset to get a performance estimation to assess the effectiveness of the sampled architecture. To train the LSTM controller to sample better architectures over time, this performance of the child model is also used as feedback to update its weight parameters, which includes the weights of the dense layers and the LSTM layer. This is done using the REINFORCE policy gradient method. In our implementation, we sample a batch of architectures (or a batch of 30 choices for the nodes) for the child model, train them for 20 epochs and use the cubed maximum validation accuracy in the last 5 epochs to update the LSTM controller's weight parameters using the REINFORCE policy gradient algorithm to enable it to sample better architectures over time.

2.3.4. REINFORCE Algorithm

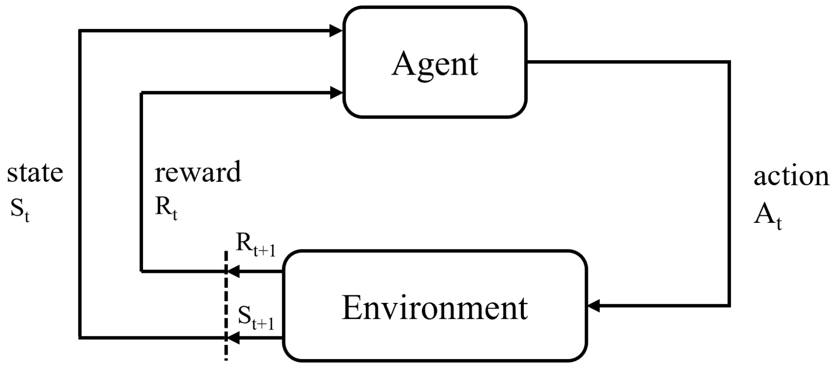

Figure 2.7. Overview of RL. Image adapted from Deeplizard

In the context of the REINFORCE algorithm, the choices that the LSTM controller samples for all 30 nodes can be considered as a list of actions $a_{1:T}$ taken by it to design the child model's architecture, where T=30 in our implementation. In RL, the component which performs the role of sampling actions is called an agent.

As illustrated in Figure 2.7, an agent observes and interacts with an environment, which is at a state S_t, at time t by taking an action A_t. This causes the environment to change its state to S_{t+1}, incurring a reward R_{t+1}. If the action, A_t, caused a favourable S_{t+1}, the reward obtained is higher, else it's lower. This reward, R_{t+1}, is then passed as feedback to the agent to sample better actions over time. This process then starts over for the next time step, t+1. When we cross the dotted line on the bottom left, the figure shows t+1 transforming into the current time step t. Usually, the agent has a set of possible actions to select for a particular state of the environment. What action to choose upon observing a particular state of the environment is decided by a policy which the agent follows to act. A policy is a function that maps a given state to probabilities of selecting each possible action. The goal of the agent in RL is to learn a policy to sample actions that maximizes the expected reward, which is simply the sum of future rewards. That is, if the agent has to sample T actions to perform a task, it has to learn an optimal policy which when followed to sample actions, each incuring an reward, gives the highest cumulative reward. In other words, $R_1 + R_2 + ... + R_T$ must be the highest value possible where each R_t is the reward incured when the agent performs action A_{t-1}.

In our case, the agent is the LSTM controller, environment is the state of the architecture; that is, the input of the LSTM at time t and reward, R, for each node selection is the maximum cubed validation accuracy obtained in the last 5 epochs when the child model is trained for 20 epochs. The policy, in this case, is the LSTM controller itself, which tunes its weight parameters to generate softmax probability distributions favoring the best actions that incur maximum expected reward, with high probabilities, given the state of the architecture as input. Usually, the choice corresponding to the highest probability in the softmax probability distribution is sampled for the node. However, to enable the LSTM controller to explore new choices and to avoid getting optimized to suboptimal choices, in our implementation, we use an epsilon greedy strategy with $\epsilon = 0.1$. In this method, we generate a random probability. If this probability is greater than ϵ, we sample the choice with the highest probability in the softmax probability distribution, and if it is lower than ϵ, we randomly choose a choice for the node.

As stated earlier, the objective of our agent, the LSTM controller, is to learn an optimal policy to sample actions $a_{1:30}$ through which it can obtain the maximum possible expected validation accuracy, R. This is represented by the objective function $J(\theta_c)$.

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

where $P(a_{1:T})$ denotes the list of softmax probabilities using which the LSTM controller makes a choice for 30 nodes using it's weight parameters, θ_c.

In order to update the weight parameters of any neural network, firstly, an objective function, also known as the loss function, like the one defined above, is partially differentiated with respect to each weight parameter in the model. This procedure is called Backpropagation. This partially differentiated value is then scaled by a learning rate α and is then subtracted from the weight parameter in order to update it. This process is called Gradient Descent. Usually, the objective function or the loss function is an error quantifying how the predictions by the neural network are far from the actual true values. Therefore, the above theory holds if the goal is to minimize the error value from this objective function. In our case, the value from the objective function, $J(\theta_c)$, needs to be maximized as the agent's goal is to maximize the validation accuracy (R). Therefore, in this scenario, the objective function, $J(\theta_c)$ defined above, must be backpropagated and then the scaled partial differentiations must be added to the LSTM weight parameters. This process is called Gradient Ascent. However, the reward R is not differentiable with respect to the weight parameters of the LSTM controller. Therefore, we use the REINFORCE policy gradient method from [46]:

$$\nabla_{\theta_c} J(\theta_c) = \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)}[\nabla_{\theta_c} \log P(a_t | a_{(t-1):1}; \theta_c) R]$$

An approximation of the above equation is:

$$\nabla_{\theta_c} J(\theta_c) = \frac{1}{m} \sum_{k=1}^{m} \sum_{t=1}^{T} \nabla_{\theta_c} \log P(a_t | a_{(t-1):1}; \theta_c) R_k$$

Where m represents the number of different architecture configurations sampled by the LSTM controller. R_k represents the reward obtained by training the k-th architecture configurations sampled for the child model. This equation contains the term $\log P(a_t | a_{(t-1):1})$, which is the log of the probability from the softmax probability distribution associated with choosing an action a_t when the previously sampled actions, $a_{(t-1):1}$ are passed as inputs of the LSTM controller. As this probability comes from the softmax probability distribution computed using the weight parameters of the LSTM controller, the above equation can be used for finding partial differentiations using backpropagation and gradient ascent can be utilized to update the weights.

Usually, the reward R_k varies drastically among different architectural configurations sampled. Therefore, this leads to drastic variations in the LSTM weight parameters update. This can potentially lead to instability in the training process of the LSTM controller and might cause it to optimize to a suboptimal architectural design for the child model. Therefore, to handle this variation in the reward signal, we use a baseline function.

$$\nabla_{\theta_c} J(\theta_c) = \frac{1}{m} \sum_{k=1}^{m} \sum_{t=1}^{T} \nabla_{\theta_c} \log P(a_t | a_{(t-1):1}; \theta_c)(R_k - b)$$

where, in our implementation, b is the exponential moving average of rewards of previous batches.

In summary, to train the LSTM controller, a batch of architecture configurations is sampled and trained for 20 epochs to generate the rewards. Then the weight parameters, θ_c, of the controller LSTM are updated using the gradient ascent rule :

$$\theta_c = \theta_c + \alpha \nabla_{\theta_c} J(\theta_c)$$

where α is the learning rate.

2.4. Results and Discussion

In the ambit of our investigation, we meticulously explored the incorporation of NAS to autonomously design DL models for the sophisticated realm of medical image classification. Central to our methodological framework was deploying a LSTM controller designated to traverse a broad spectrum of potential neural network configurations. This exploration was rigorously defined by specific implementation parameters pivotal to our experimental endeavours' success.

Our experimental framework, built using Tensorflow 2.11.1, was underpinned by a high-performance computing setup utilizing NVIDIA dual RTX A6000 GPUs and an AMD Threadripper 64-core processor, supplemented by 500GB of RAM. This infrastructure was paramount in managing the substantial computational load necessitated by our NAS framework, enabling efficient processing and evaluation of a wide array of complex model architectures.

2.4.1. LSTM Controller Sampling and Training Details

The core of our experiment hinged on the LSTM controller's capacity to sample over 50,000 architecture variations throughout the study's course. The weight parameters governing the LSTM controller were dynamically refined after evaluating every five architectural samples, leveraging the capabilities of the Adam optimizer with a default learning rate $\alpha = 0.001$. The default weight initialization provided by TensorFlow 2.11.1 was applied, except for the LSTM layer, where the weights linked to the hidden state and inputs were initialized using Glorot uniform, while biases were set to zeros. Additionally, the initial hidden and cell states of the LSTM layer were initialized as vectors of zeros, each with a size of 100.

2.4.2. Child Model Implementation and Training Details

Once the LSTM controller sampled an architecture, the node selections were decoded based on the numerical encoding performed on the search space, and the child model was built with an initial depth size (#) of 3. Due to the application of combining operation nodes in each subunit, the depth size was varied dynamically based on the addition or the concatenation operation. Additionally, 1*1 convolution layers were used when necessary during the application of the add and concat operations. The default weight initialization provided by TensorFlow 2.11.1 was applied across all the child model layers. Each generated child model was trained for 20 epochs using the Adam optimizer with a default learning rate $\alpha = 0.001$ and categorical cross-entropy loss. Data for training these models was passed in batches of size 64. Additionally, since we train the child models only for 20 epochs to get a performance estimation, the child model weights were saved at the 20th epoch to continue training them until convergence, if required, after the architecture search by the NAS framework is complete. Training until convergence refers to the process of training a child model until its performance metrics stabilize or improve to a satisfactory level.

2.4.3. Data Split and Preprocessing

Within the expansive realm of medical image datasets, the DermaMNIST dataset emerges as a specialized subset meticulously curated to propel dermatological diagnostics into the new era of ML. Central to our exploration, DermaMNIST lays the groundwork for a comprehensive evaluation of child model architectures generated by our NAS framework. This dataset, encapsulating the multifaceted challenges of dermatological imagery, serves as an ideal canvas for probing the boundaries of contemporary DL architectures. In the upcoming subsections, we will provide a detailed explanation of the data preprocessing and splitting procedures applied to the DermaMNIST dataset in preparation for training our child model.

2.4.3.1. Data Composition and split

DermaMNIST is not just a collection of images but a meticulously curated gallery showcasing dermatoscopic imagery distributed across seven distinct lesion categories. From Melanocytic nevi to melanoma and basal cell carcinoma, the dataset offers a panoramic view of dermatological conditions, enabling a holistic approach to model training and evaluation. This dataset is based on Human Against Machine with 10000 training images (HAM10000) [47][48], which contains a vast collection of dermatoscopic images depicting various common pigmented skin lesions. These images serve as valuable resources for dermatological research and analysis. In total, the DermaMNIST dataset comprises 10,015 dermatoscopic images, each categorized into one of seven different diseases. This categorization forms the basis of a multi-class classification task, where the goal is to develop a ML model, which in our case is the optimized child model, capable of accurately classifying these images into their respective disease categories.

Table 2.1. Distribution of DermaMNIST Dataset

Skin Lesion Type	Data Count
Actinic Keratoses	327
Basal cell carcinoma	514
Benign keratosis	1099
Dermatofibroma	115
Melanocytic nevi	6705
Melanoma	1113
Vascular skin lesions	142

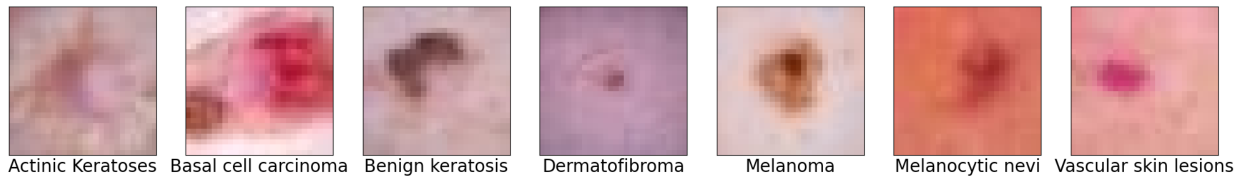

Figure 2.8. Skin Lesion Types in DermaMNIST Dataset

The distribution of the seven categories within the dataset is presented in Table 2.1. An image randomly chosen representing each category is displayed in Figure 2.8. The publishers of the dataset resized the original source images, resulting in a published dataset composed of images with dimensions 28x28 pixels, each containing three image channels (RGB). Additionally, to facilitate model training and evaluation, the publishers of the dataset also partitioned the dataset into three subsets: training, validation, and test sets. The partitioning is performed using a ratio of 7:1:2, ensuring that the majority of the data is allocated for training while also reserving portions for validation and final evaluation. Specifically, 70% of the images are assigned to the training set, 10% to the validation set, and 20% to the test set. In our experiment, we retained this default data split provided by the publishers and preserved the original size of the images without resizing them.

2.4.3.2. Data Preprocessing

Central to our preprocessing regimen is normalizing images across the dataset. It is typical to divide the pixel values by a scaling factor, often 255. This normalization step brings the pixel values into the range [0, 1]. Since pixel values in typical image datasets are represented as integers in the range [0, 255], dividing by 255 ensures that the pixel values are scaled to the range [0, 1]. Normalization helps standardize the input data and ensures all features have a similar scale, aiding faster convergence during training.

After image normalization, mean subtraction was performed. In this process, the mean of the training dataset across the three image channels is computed and subtracted across the entire dataset. Subtracting this mean value from each pixel helps center the data around zero, improving the training process's convergence and stability. To prevent data leakage, the mean of only the training dataset is computed, excluding the validation and test. When performing ML tasks, it's crucial to ensure the model does not inadvertently learn information from the validation or test sets. If the mean subtraction is applied to the images with the entire dataset's mean, the model could indirectly gain knowledge about these unseen datasets, leading to data leakage. Data leakage can significantly affect the performance and reliability of ML models. Data leakage can artificially inflate the performance metrics of a ML model. Since the model unintentionally learns information from the validation or test sets, it may appear to perform well on these datasets. However, this performance does not indicate the model's ability to generalize to unseen data, leading to overestimated performance metrics. Utilizing only the training dataset's mean to compute mean subtraction on the entire dataset ensures that the model is only trained on information in the training data.

Each of these processed images is associated with a specific category or class label that represents the type of lesion it depicts. These class labels are essential for training a classification model to predict the correct class for new, unseen images. Therefore, one hot encoding was performed to ensure that the class labels were converted into a format that the child model could understand. One-hot encoding is a technique used to represent categorical data, such as class labels, in a format suitable for ML algorithms. One-hot encoding transforms each class label into a binary vector representation. Specifically, for each image:

- A binary vector of length seven is created, where each element corresponds to one of the seven skin lesion categories.
- The element corresponding to the class label of the image is set to 1, while all other elements are set to 0. This element, represented with a 1, indicates the presence of the associated category.
- For instance, if an image belongs to category 3, its one-hot encoded representation would be [0, 0, 1, 0, 0, 0, 0].

2.4.4. Navigating Dataset Challenges

2.4.4.1. Addressing Skewed Distribution

Despite meticulous curation, DermaMNIST mirrors the real-world scenario of uneven lesion prevalence, presenting a skewed distribution that could bias learning processes. Our methodology incorporates sophisticated preprocessing measures, recalibrating this imbalance to foster equitable model learning—an endeavour critical for achieving diagnostic impartiality.

Data imbalance in medical datasets, where significantly fewer examples than others represent certain classes or conditions, can seriously affect diagnostic models and patient care. ML models trained on imbalanced datasets tend to be biased towards the majority classes. Since there are fewer examples of minority classes, the model may not learn to accurately distinguish between them. As a result, it may prioritize predicting the majority class, leading to underrepresentation or misclassification of minority classes.

In medical diagnosis, false negatives can have severe consequences, as they may result in delayed treatment or lack of necessary interventions. In the case of imbalanced datasets, models may be more likely to incorrectly predict the outcome as the majority class, leading to false negatives for minority classes. This can result in undetected medical conditions or diseases going untreated.

Data imbalance can exacerbate existing healthcare disparities by disproportionately affecting underrepresented patient populations. If minority classes are underrepresented in the training data, models may perform poorly for these groups, leading to disparities in diagnostic accuracy and potentially unequal access to healthcare services and treatments.

To mitigate this problem, in our approach, we implemented class weights within the training regimen to neutralize this imbalance. The computation of class weights is grounded in the principle of inverse frequency, ensuring that rarer categories wield a proportional influence on the learning process. In our implementation, initially, the dataset was examined to identify the category with the highest number of images. This category serves as the reference point for computing the class weights. Next, the number of images belonging to each individual skin lesion category was determined. This information is essential for understanding the distribution of data across different classes. The class weight for each category was calculated by dividing the highest image count by the number of images in the respective category. This computation results in a weight value that reflects the relative imbalance of each class compared to the category with the highest representation. For example, if the category with the highest image count has 1000 images, and a specific category contains 500 images, the class weight for that category would be $1000/500 = 2$. This indicates that the category with 500 images is represented only 50% compared to the category with 1000 images. Such a methodologically sound approach compensates for underrepresentation, facilitating an equitable learning landscape.

By integrating these class weights into the model training phase, we adjust the loss function to accurately reflect the dataset's class distribution. This adjustment guarantees that the learning process is not unduly influenced by more prevalent lesion types, promoting balanced and comprehensive learning across all categories.

2.4.4.2. Complexity in Visual Diagnosis

The dataset's visual diversity, marked by varying lesion shapes, sizes, colours, and textures, elevates the complexity of the classification challenge. It necessitates models of exceptional discernment, capable of extracting and synthesizing these intricate visual cues into accurate diagnostic conclusions.

2.4.5. Evaluating the Effectiveness of the LSTM Controller

Figure 2.9 demonstrates the effectiveness of our LSTM controller in sampling architecture configurations better than the previously sampled designs for the child models. To understand this effectiveness, the mean of maximum validation accuracy obtained in the last 5 epochs of the 20 epochs (cube root of the reward) of the top 1, 5 and 25 performing models across 50,000 sampled architectures was computed. By analyzing the trend and stability of this plot, one can gain insights into the controller's learning progress, performance, and convergence towards an optimal policy. Therefore, this is a valuable metric for assessing the LSTM controller's policy over time.

Additionally, this plot shows a consistently increasing trend, which indicates that the LSTM controller's policy is yielding favourable outcomes and maximizing rewards. This sustained increase also reflects that the LSTM controller is effectively learning from experience and making better decisions. Moreover, it provides insights into the relative effectiveness of sampled architecture configurations. That is, the LSTM controller has learnt to sample architecture configurations of superior quality over the previously sampled designs with time.

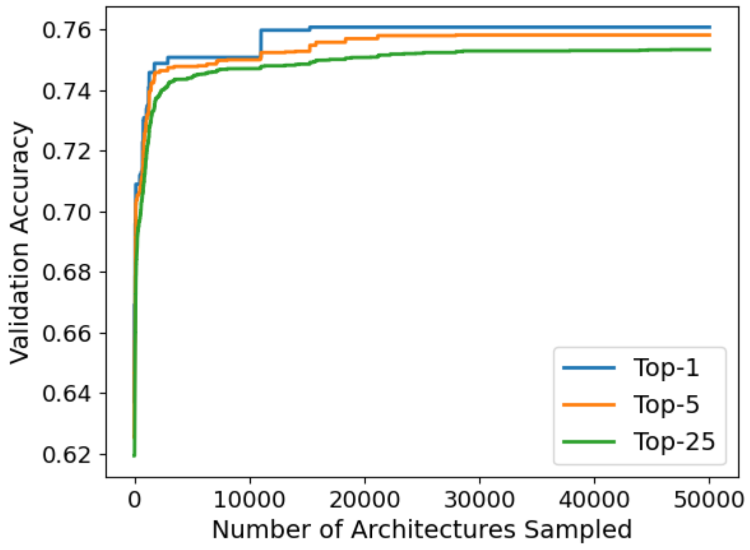

Figure 2.9. Performance of Our NAS Framework in Sampling Child Models of Superior Quality Over Time.

2.4.6. Filtering the Best Model

Of the 50,000 architectures sampled by the LSTM controller, we selected the 25 models with the best rewards during the search. These top 25 models were trained until convergence. The best performing model from this analysis was selected as the optimized child model for the skin lesion detection and classification on the DermaMNIST dataset.

2.4.7. Grid-Based Hyperparameter Search

The selected best performing child model was further optimized with a small hyperparameter tuning with grid search. A grid-based hyperparameter search is a systematic approach used in ML to optimize a model's performance by exploring a predefined set of hyperparameter combinations.

The grid search technique involves defining a grid of hyperparameter values to explore. Each dimension of the grid represents a different hyperparameter, and each point in the grid corresponds to a specific combination of hyperparameters. The algorithm then systematically evaluates the performance of the model trained with each hyperparameter combination using a predefined evaluation metric, which in our case is the validation accuracy.

In our implementation, we ran a grid-based hyperparameter search to find the optimal batch size, learning rate and number of epochs to train the child model. Here's an elaboration on each of the terms:

2.4.7.1. Batch Size

- In machine learning, particularly in training neural networks, the batch size refers to the number of training examples used in one iteration of the optimization algorithm (e.g., gradient descent) before updating the model's weight parameters.
- The choice of batch size can affect the speed and stability of training, as well as the memory requirements of the training process. Common batch sizes range from small values like 32 or 64 to larger values like 128, 256, or even higher, depending on the dataset size and the available hardware resources.

2.4.7.2. Learning Rate

- The learning rate is a hyperparameter that determines the size of the step taken during the optimization process, such as gradient descent, to update the model's weight parameters.
- It controls the rate at which the model parameters are adjusted in the direction of minimizing the loss or the objective function. A higher learning rate leads to larger parameter updates, while a lower learning rate results in smaller updates.

- Choosing an appropriate learning rate is crucial for training stable and effective models. A learning rate that is too high may cause the optimization process to diverge or oscillate, while a learning rate that is too low may result in slow convergence or getting stuck in local minima.

2.4.7.3. Number of Epochs

- An epoch refers to one complete pass through the entire training dataset during the training of a machine learning model.
- The number of epochs is a hyperparameter that specifies the number of times the entire dataset is fed to the model for training.
- Training for multiple epochs allows the model to see the training data multiple times, enabling it to learn from it and improve its performance over time.
- The choice of the number of epochs depends on factors such as the complexity of the model, the size of the dataset, and the desired level of model performance. It is typically determined through experimentation,

In summary, the batch size, learning rate, and number of epochs are essential hyperparameters in training machine learning models, particularly neural networks. Proper selection and tuning of these hyperparameters are critical for achieving optimal model performance and training stability.

In the grid search, we explored the hyperparameter space of batch size, consisting of values 4,8,16,32,64,128,256, learning rate consisting of values 0.1,0.01,0.001,0.0001,0.00001,0.000001 and number of epochs consisting of values 250,300,350,400,450,500. Therefore, we explored about 252 model configurations. Furthermore, we incorporated an early stopping strategy into our training pipeline, utilizing a callback mechanism to halt training if the validation accuracy fails to improve over consecutive epochs. Additionally, we implemented a checkpointing system to save the model parameters when the validation accuracy is at its highest. These implementations were seamlessly integrated using the Weights and Biases (wandb) framework, which provides a comprehensive suite of tools for experiment tracking, visualization, and management. Here's an elaboration on each aspect:

2.4.7.4. Early Stopping Strategy

- Early stopping is a technique used to prevent overfitting by terminating training when the model's performance on a validation set no longer improves.
- We employed a callback mechanism that monitors the validation accuracy over consecutive epochs. If the validation accuracy does not increase within a predefined number of epochs (in this case, 50 epochs), the training process is terminated.
- This strategy helps prevent the model from overfitting to the training data and ensures that it generalizes well to unseen data.

2.4.7.5. Checkpointing System

- A checkpointing system is used to save the model's parameters at specific points during training, allowing us to restore the model to its best-performing state if needed.
- We implemented a checkpoint mechanism that periodically saves the model's parameters based on the validation accuracy. Whenever the validation accuracy improves and reaches a new high, the model parameters are saved.
- This ensures that we have access to the best version of the model throughout the training process, enabling us to use it for inference or further analysis.

2.4.7.6. Integration with wandb

- We leveraged the wandb framework to seamlessly incorporate these training strategies into our workflow.
- wandb provides powerful tools for experiment tracking, visualization, and collaboration, allowing us to monitor training progress, track performance metrics, and visualize results in real-time.

Based on our hyperparameter tuning results, we found that using a batch size of 16, a learning rate of 0.00001, and 300 epochs yields the best test results. Table 2.2 shows the

Table 2.2. Performance of the Best Performing Child Model in Different Hyperparameter Configurations

Batch size	Learning Rate	Epochs	Test AUC	Test Acc
16	**0.00001**	**300**	**0.9659**	**0.7706**
4	0.00001	400	0.9665	0.7661
4	0.00001	450	0.965	0.7631
64	0.00001	350	0.9663	0.7626
8	0.00001	250	0.9647	0.7616

top 5 best performing hyperparameter configurations for the top child model chosen. The best model obtained through this methodology was then compared to the existing state of the art human-designed architectures.

2.4.8. Clarifying the Metrics

2.4.8.1. AUC

The Area Under the ROC Curve (AUC) is a widely used metric in machine learning. However, extending it to multiclass classification scenarios requires modifications to accommodate multiple classes. In our implementation, we computed AUC using the default approach followed in the Tensorflow 2.11.1 framework. The AUC metric quantifies the area under the Receiver Operating Characteristic (ROC) curve, which is a graphical representation of the trade-off between the true positive rate (TPR, also known as sensitivity) and the false positive rate (FPR, also known as 1-specificity).

In the context of multiclass classification, AUC represents the model's ability to discriminate between different classes. AUC values range from 0 to 1, where a higher value indicates better discrimination ability. An AUC of 0.5 suggests that the model's performance is no better than random, while an AUC of 1.0 indicates perfect discrimination. AUC is also robust to class imbalance, making it suitable for evaluating models in scenarios where class distributions are skewed. AUC is commonly used in medical diagnostics, fraud detection, and other applications where the cost of false positives and false negatives may vary, and a comprehensive evaluation of model performance is desired.

2.4.8.2. Acc

Accuracy (Acc) is one of the most straightforward and commonly used metrics for evaluating the performance of classification models in ML. It measures the proportion of correctly classified instances from the total number of instances in the dataset. Acc is calculated by dividing the number of correctly classified instances (true positives and true negatives) by the total number of instances in the dataset. It represents the overall correctness of the model's predictions across all classes. Higher Acc values indicate better performance, while lower values suggest poorer performance.

Acc is commonly used as a primary evaluation metric, especially when classes in the dataset are balanced (i.e., the number of instances in each class is roughly equal). It provides a simple and intuitive measure of model performance, making it easy to interpret and communicate. Acc may not be a suitable metric when classes in the dataset are imbalanced, meaning one class dominates the dataset. In such cases, a model that always predicts the majority class can achieve high accuracy while performing poorly on minority classes. Acc does not provide insights into the type of errors the model makes (e.g., false positives, false negatives), which may be crucial in certain applications. While Acc is useful, it is often used in conjunction with other metrics, especially in imbalanced datasets. Additional metrics such as AUC provide a more comprehensive understanding of the model's performance.

2.4.9. Top Model Performance Summary

Table 2.3 shows the performance of the top model on the test data compared to human designed state of the art architectures. Figure 2.10 illustrates the design of the top-performing child model configuration used to report the results. Table 2.4 also illustrates the performance of the top child model across the different data splits, thereby reflecting the model generalization. Only the metric values at the epoch where the validation accuracy reached its highest point are reported for the top model. More detailed discussions on the implications of our results are provided in the upcoming subsections.

Table 2.3. Performance of Different Architectures v.s. the Top Child Model Performance Generated Using Our NAS Framework

Architecture	Parameters	AUC	Acc
ResNet-18 ([34])	11.7M	0.917	0.735
ResNet-50 ([34])	23.5M	0.913	0.735
ViT ([39])	86.6M	0.859	0.704
AlexNet ([40])	60M	-	0.668
Top model	**0.26M**	**0.966**	**0.771**

Table 2.4. Performance of the Top Child Model Across the Train, Test and Validation Datasets

Datasplit	AUC	Acc
Train	0.9793	0.818
Validation	0.9667	0.7876
Test	0.9659	0.7706

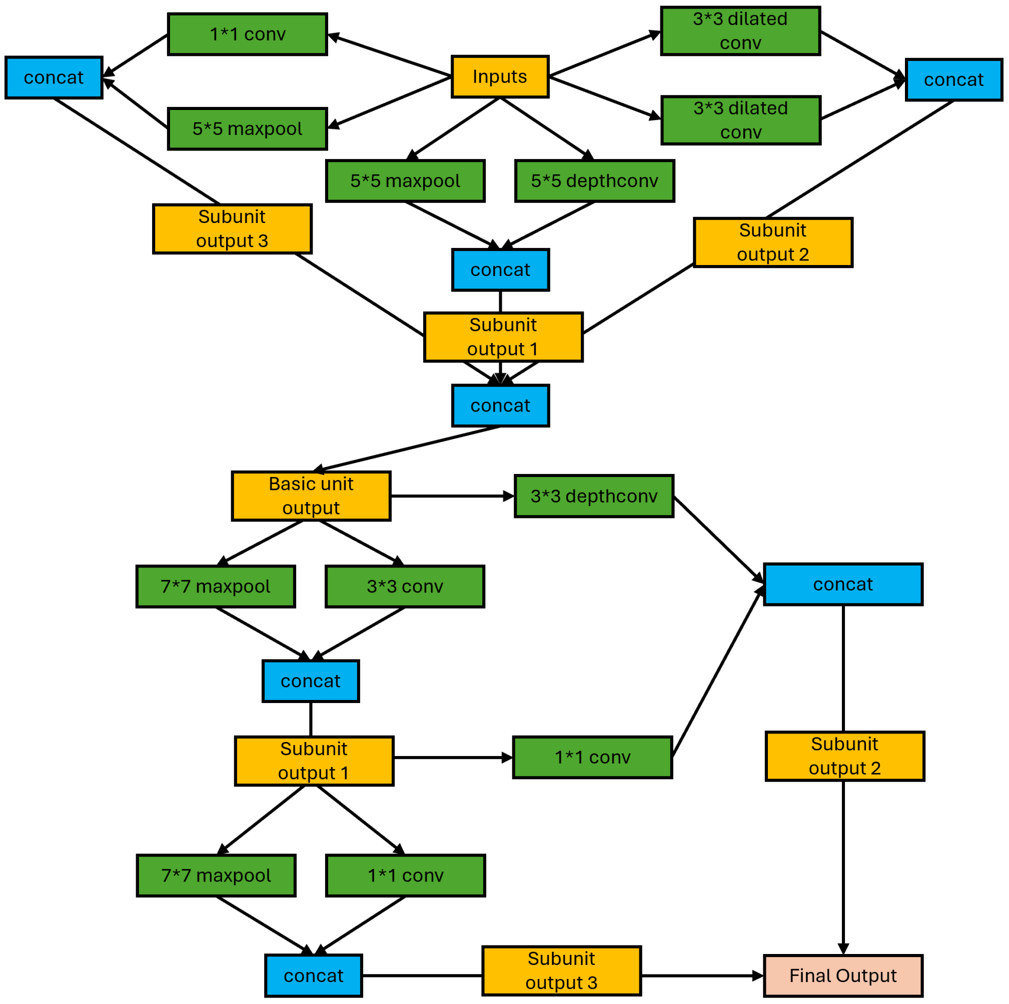

Figure 2.10. Best Performing Child Model Design

2.4.10. Implications of Our Study

Our comparative analysis, fortified by strategic hyperparameter tuning and an astute model-saving approach, unequivocally establishes the preeminence of our NAS-generated top model. The deployment of AUC and Acc as evaluative metrics furnishes a panoramic view of the model's architectural prowess, heralding its potential to redefine benchmarks in medical image classification. Through this analytical journey, we not only validate the technical virtuosity of our approach but also illuminate the pathway towards revolutionizing medical diagnostics with advanced, efficient, and precise DL solutions.

2.4.10.1. Top Model's Prowess in Outperforming State of the Art Architectures

From the results reported in Table 2.3, it is evident that the top model generated by our NAS framework outperforms the existing human designed state of the art architectures. This proves the adoption of our NAS framework has the potential to streamline the model development process, save time and effort for researchers, and accelerate progress in the field of ML. By automating the search for optimal architectures, NAS enables researchers to focus on higher-level tasks such as problem formulation, data preprocessing, and model interpretation, ultimately driving innovation and advancing the state of the art.

Instead of spending time on trial and error or relying on intuition to design architectures, researchers can delegate the architecture search process to our NAS framework, freeing up valuable time and resources for other tasks. Additionally, by leveraging computational power to efficiently explore this space, NAS can discover architectures that human designers may not have considered, leading to novel and potentially superior solutions. Moreover, by abstracting away the complexities of architecture design, NAS democratizes access to advanced ML techniques, making them more accessible to researchers with varying levels of expertise. This democratization of AI empowers a broader community of researchers to engage in model development and innovation, leading to a more diverse range of solutions and advancements.

2.4.10.2. Dissecting the Generalization Proficiency

Within the sphere of NAS, the prowess of a model is not solely gauged by its training performance but also by its capacity to generalize across unseen data. This holistic approach to model evaluation underpins our study, where we meticulously analyze the training, validation, and testing accuracies of our NAS generated model, as elucidated in the Table 2.4. Such a detailed examination serves not just as a testament to the model's learning efficiency but also as an indicator of its broader utility in real-world applications.

In ML, generalization embodies a model's aptitude to transpose learned patterns from the training dataset to novel, unseen datasets with efficacy. It is the linchpin distinguishing models of theoretical value from those with tangible, practical utility, marking their robustness across varied data landscapes. From table 2.4, it is evident that the metric values on the training dataset are the highest. While this high level of training accuracy is commendable, it necessitates a juxtaposition with commensurate validation and testing performance to preclude the pitfalls of overfitting.

The moderation observed in transitioning from training to validation AUC and Acc aligns with expectations, given the latter's assessment on unseen data. Nonetheless, the model's substantial validation accuracy intimates that it hasn't succumbed to overfitting, retaining a significant portion of its predictive prowess. This nuance underscores the model's refined ability to generalize effectively. The harmony between validation and testing Acc and AUC further cements the model's generalization capacity. The testing accuracy, drawn from a completely novel subset of the dataset, accentuates the model's resilience and adaptability, heralding its suitability for deployment in authentic diagnostic scenarios.

The synergy observed among the training, validation, and testing performance not only showcases the NAS generated model's profound ability to generalize but also highlights the NAS framework's success in traversing a complex architectural landscape to distil a high-performing model. This equilibrium of accuracies elucidates the model's readiness for real-world applications, particularly in medical diagnostics where the interpretation of unseen images is routine.

2.4.10.3. Impact of Architectural Preferences on Performance

Within the specialized niche of NAS, the quest for the quintessential DL architecture for medical image classification has been a journey of both discovery and innovation. Our NAS framework, through meticulous exploration and evaluation, has brought to light a model of unparalleled architectural design and performance. This model, distinguished by its adept employment of concatenation operations among other salient features, stands as a testament to the framework's prowess in identifying and optimizing DL architectures for complex tasks. Herein, we delve into the sampled best model's architectural landscape, dissecting its design choices and the subsequent implications on performance and application within medical diagnostics.

Concatenation Operation, serving as a conduit for merging feature maps from disparate layers, enrich the model's information representation. This mechanism enables the seamless integration of both granular, low-level features and abstract, high-level features, thereby augmenting the model's discriminative capacity across diverse skin lesion classes. Incorporating concatenation operations ensures a streamlined flow of information across the model's architecture. This attribute is invaluable in the realm of medical image classification, where the meticulous preservation of spatial relationships and textural details is paramount for achieving diagnostic accuracy.

2.4.10.4. Towards Deployment: The Model's Lightweight Nature and Edge Computing Potential

Remarkably, despite its architectural complexity, the model maintains a lightweight stature, significantly augmenting its deployment feasibility. With a minimal parameter footprint, it is ideally positioned for integration within edge computing environments characterized by computational resource constraints. This amalgamation of lightweight design, stellar performance, and architectural adaptability heralds the model as a prime candidate for real-time diagnostic applications, envisaging a future where advanced medical imaging analysis is ubiquitously accessible at the point of care.

Compared to other state of the art human-designed architectures, with a mere 0.26M parameters, the model's demand on storage space is substantially minimized, making it a perfect fit for edge devices where memory capacity is at a premium. This strategic reduction ensures the accessibility of high-calibre medical imaging analysis even in settings constrained by resources.

2.5. Conclusion and Future Work

In this research, we studied the feasibility of using NAS with the REINFORCE algorithm for medical image classification by showcasing its efficacy in building an optimized neural network for skin lesion classification. Our findings unequivocally demonstrate the superiority of our NAS framework in automatically generating architecture designs compared to hand designed DL models crafted by experts. Through rigorous experimentation, we have established that architectures produced by our NAS frameworks consistently outperform those meticulously crafted by human experts. This validation not only underscores the effectiveness of our approach but also highlights the transformative potential of automated architecture design in the realm of DL.

One of the key highlights of our research lies in attaining the best-performing model with significantly fewer parameters compared to other state of the art human designed architectures. This achievement not only validates the efficacy of our NAS frameworks but also emphasizes their capability to yield highly efficient and optimized DL models. Furthermore, our results provide compelling evidence for the applicability of our approach to real-world use cases, particularly in the domain of medical datasets and applications.

The superior performance demonstrated by architectures generated through our NAS framework suggests their potential to address the unique challenges and complexities inherent in medical datasets, where accuracy, efficiency, and interpretability are of paramount importance. By leveraging automated architecture design, researchers and practitioners in the medical field can harness the power of state of the art DL models without the need for extensive manual intervention or domain specific expertise.

Moreover, adopting NAS frameworks offers tangible benefits regarding time efficiency and knowledge acquisition for researchers. By automating the architecture design process, our frameworks empower researchers to focus their efforts on higher-level tasks, such as problem formulation, data preprocessing, and model interpretation. This streamlined workflow can potentially accelerate the pace of research and development. In addition to their practical utility, NAS frameworks hold immense promise for driving innovation and advancement in the field of DL. By democratizing access to cutting-edge architecture design techniques, our framework empower researchers across diverse domains to push the boundaries of what is possible in DL model development.

In conclusion, our research provides compelling evidence for the transformative potential of NAS framework in automated architecture design. By outperforming hand designed models and offering superior efficiency and optimization, our framework pave the way for accelerated progress and innovation in DL research and application. As we continue to refine and expand upon our methodologies, we envision a future where automated architecture design becomes the cornerstone of DL model development, unlocking new frontiers of discovery and impact across a myriad of domains and industries.

3. ADDITIONAL PROJECT 1: AUTOMATIC MICRORNA DETECTION AND CONCENTRATION PREDICTION FOR THE EARLY DETECTION OF PANCREATIC CANCER USING MACHINE LEARNING [1]

3.1. Abstract

Early detection of cancer is of paramount importance as it significantly improves treatment outcomes, reduces mortality rates, and enhances patients' quality of life. However, the lack of early symptoms in some cancers, like pancreatic, necessitates exploring noninvasive methods for early detection, such as identifying biomarkers like nucleic acids. With the recent identification of specific circulating microRNA (miRNA) panels in the blood responsible for pancreatic cancer, in this study, we explore the commercialization potential of a biomarker detection device which automatically detects and quantifies these miRNA molecules. We use Dielectrophoresis (DEP) to concentrate these biomarkers on T-shaped interdigitated electrodes (TIEs) and use a Machine Learning (ML) algorithm optimized for low-cost hardware designed by Neural Architecture Search (NAS) to automatically detect its concentration. The results we've attained so far underscore the promise of a non-invasive diagnostic device for the early detection of pancreatic cancer, which can be used for routine health checkups.

3.2. Introduction

Cancer, with a projected 11.5 million deaths globally in 2030, is one of the leading causes of death worldwide. This statistic can undergo a significant shift in the event of early-stage cancer detection [49]. However, the lack of early symptoms in some cancers, like pancre-

[1] The dielectrophoresis experimental protocols discussed in this chapter are based on the manuscript "Controlled Concentration Of Short DNA Molecules On The Electrodes Using AC Electric Fields And Capillary Flow ", submitted to the IEEE Research and Applications of Photonics in Defense Conference. This paper was authored by Akila Wijesinghe, Ridhanya Sree Balamurugan, Umamaheswara Rao Tida and Dharmakeerthi Nawarathna. Ridhanya Sree Balamurugan conducted the initial experiments and Akila Wijesinghe improved the outcomes of the experiments. Dr. Dharmakeerthi Nawarathna and Dr. Umamaheswara Rao Tida provided guidance on the research methodology and allocated the necessary resources required to conduct the research.

atic [50], necessitates exploring non-invasive methods for early detection, such as identifying biomarkers like proteins, nucleic acids, antibodies, and peptides [51].

Recently, a sensitive circulating miRNA panel (miR-22, miR-642b, miR-885-5p) was identified for early detection of pancreatic cancer [52]. Circulating miRNA targets, serving as a biomarker in the bloodstream, provide a sensitive means of identifying cancers, given their stable expression in the blood [52]. A significant feature of miRNAs is their durability: they remain stable during standard handling and storage, making them ideal candidates for developing reliable biomarker-based diagnostic tools [53]. Despite extensive research on the role of miRNAs in cancer, one of the primary challenges remains the economically feasible detection of these biomolecules at clinically relevant concentrations. Most existing detection methods are either too costly or lack the sensitivity required for effective early-stage cancer diagnosis. Therefore, our research aims to develop and validate an affordable device that accurately measures the concentration of miRNAs in blood samples, addressing the existing gap in this area. By concentrating on this objective, this study seeks to provide a practical solution for regular screenings, which could result in earlier diagnoses and improved management of pancreatic cancer. This device would not only simplify the process of miRNA quantification but also increase accessibility to early cancer detection in clinical settings.

To detect and identify miRNAs responsible for pancreatic cancer, our study employs DEP. This entails utilizing a sensor equipped with TIEs, a specific electrode configuration significantly enhancing the DEP force [54]. This force selectively attracts and traps charged biomolecules, such as miRNAs, enabling their concentration and detection. Our method's practical application was confirmed through the successful concentration of single-stranded DNA (ssDNA) molecules, which were labelled with fluorophores, on the sensor. In our experiments, fluorescence labelling served a dual purpose: it enabled the visual tracking of the ssDNA during the DEP process, and it provided a measurable concentration by evaluating the fluorescent intensity of the accumulated ssDNA molecules. Additionally, we studied advanced methods to enhance our experiment monitoring capabilities along with automation to improve the accuracy and reliability of our experiments. Specifically, we investigated ML algorithms capable of detecting and forecasting the levels of biomarkers. By integrating ML, we can now more effectively detect these critical biomarkers at clinically significant levels, thereby enabling earlier diagnosis of pancreatic cancer through non-invasive means.

3.3. Methodology

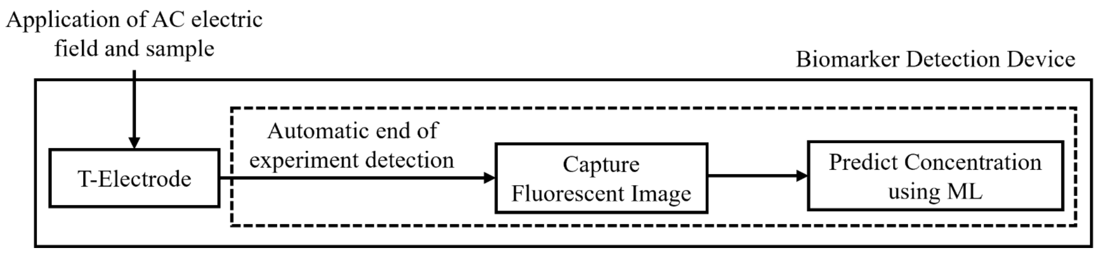
Application of AC electric field and sample
Biomarker Detection Device
T-Electrode
Automatic end of experiment detection
Capture Fluorescent Image
Predict Concentration using ML

Figure 3.1. Biomarker Detection Device Overview

Figure 3.1 illustrates the overall processes involved in the automatic circulating miRNA detection and concentration prediction for the early detection of pancreatic cancer. The first step is the application of the sample and Alternating Current (AC) electric field on the T-electrode sensor (containing TIEs). Applying an AC electric field is necessary for concentrating the biomarker in the sample on the TIEs using DEP. Ideally, the sample should contain miRNA molecules hybridized with complementary ssDNA sequences labelled with fluorophores at its 5' end. However, to demonstrate the proof of concept, we just used ssDNA molecules labelled with fluorophores for our experiments. Fluorophores are molecules that can absorb light energy at a specific wavelength and then emit light at a longer wavelength. They are commonly used in fluorescence microscopy to visualize the samples labelled with them. Labelling refers to the process of attaching or incorporating a detectable marker such as a fluorophore. This labelling of fluorophores is necessary in our experiments as we quantify the concentrated biomarkers using the fluorescence emitted by them.

Once the biomarkers concentrate on the TIEs, this is detected automatically and the fluorescence emitted by these concentrated biomarker molecules is then captured for concentration prediction. This step is essential as the concentration at which these miRNAs are circulating in the blood demonstrates the presence and stage of pancreatic cancer. The capability of ML is used to automate this step.

Experiments aimed at capturing the nucleic acid (biomarker) of interest through DEP were meticulously conducted by our collaborators. Therefore, this study provides only a brief overview of this experimental procedure while elaborating more on the experiments performed for the automations involved in the device (marked by dotted lines in Figure 3.1).

3.3.1. DEP to Concentrate ss-DNA Molecules on TIEs

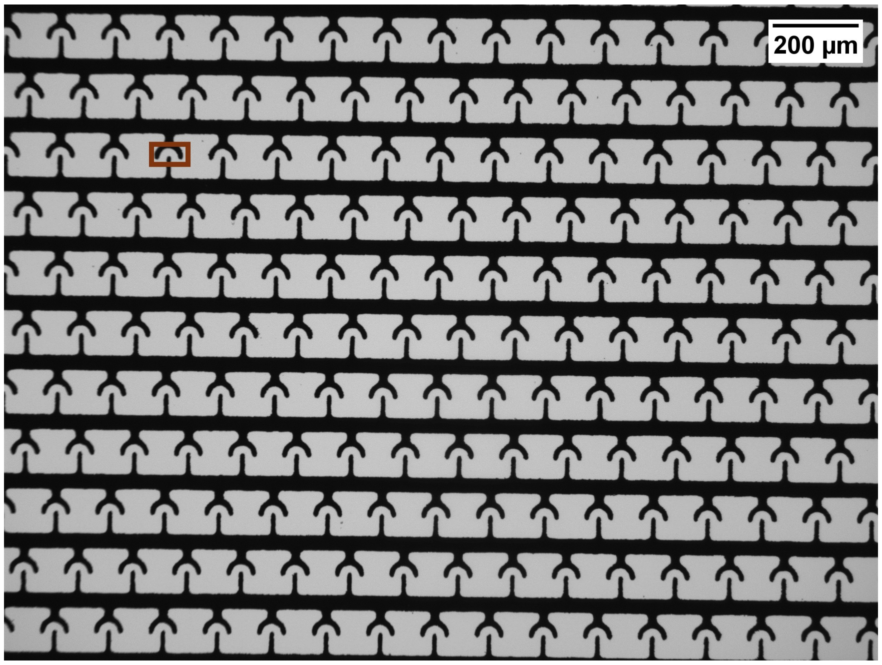

Figure 3.2. Array of TIEs with the Concentrating Region Highlighted with an Orange Box. Image captured using a Bright-Field Microscope

To concentrate the biomarker of interest, we perform DEP on TIEs, which contain an array of electrodes resembling T-shape [54] as shown in Figure 3.2. DEP is a phenomenon in which particles, typically micrometer or nanometer-sized, are manipulated in a non-uniform electric field [55]. DEP can be used to concentrate nucleic acids by applying an AC electric field. This pushes the nucleic acids towards regions of higher electric field intensity. In our case, this high electric field intensity is produced by the concentrating regions of the TIEs [54], marked by an orange box in Figure 3.2. Therefore, upon applying the AC electric field, this concentrating region is where the ssDNA molecules of interest concentrate.

Conducting DEP to concentrate ssDNA molecules on TIEs begins with sample preparation. This sample is then pipetted to the N-BK7 glass substrate with microfabricated TIEs, followed by applying an AC electric field using a function generator.

3.3.1.1. Preparation of Sample

The sample preparation involves diluting the stock solution of ssDNA with 0.01x TE buffer (15 µS/cm). This stock solution, purchased from Integrated DNA Technologies, Inc. (Coralville, Iowa, USA), contains 22 bases long ssDNA molecules of sequence AACTATACAAC-CTACTACCTCA labelled with Alexa 555 fluorophore (Quantum yield – 0.10, excitation max – 555 nm, emission max – 580 nm). In our experimentations, we prepared samples of concentration 1μM using a stock solution of 21μL volume with a concentration 100 μM.

Before the sample preparation, as this stock solution was stored at -20 degrees Celsius, we thawed it for 15 minutes at room temperature and vortexed it for about a minute before it was used. Since it contains photosensitive fluorophores, to avoid photobleaching effects, sample preparations were conducted in a dark room. Photobleaching refers to the process by which a fluorophore loses its ability to fluoresce after exposure to light. This phenomenon occurs when the fluorophore undergoes chemical reactions or structural changes that render it incapable of emitting fluorescence.

3.3.1.2. TIEs Preparation

An array of TIEs microfabricated (the process of fabrication is discussed in [54]) on an N-BK7 glass substrate was inspected for any electrode damage and was rigorously cleaned using 2-propanol, acetone and Distilled (DI) water. This cleaning process includes adding 2-

propanol, acetone and DI water in three separate beakers and exposing the TIEs to 2-propanol for 2 minutes and acetone for 1 minute. Each of these steps is followed by rinsing it with DI water. Finally, Kim wipes are used to clean the region containing the T-electrodes very carefully, avoiding any damage and ensuring that the region containing the panel of electrodes is dry.

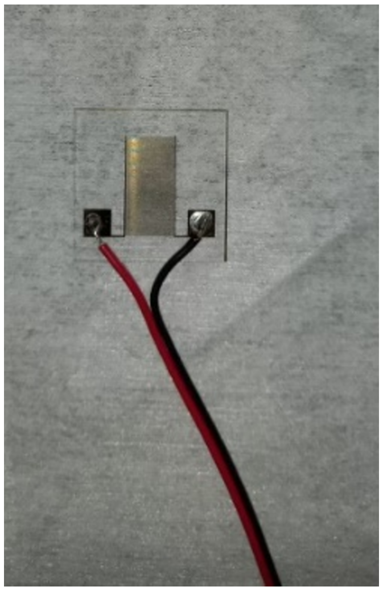
Figure 3.3. Microfabricated TIEs Setup

Two electrical wires were soldered to the pads of the electrodes, as shown in Figure 3.3. This was followed by verifying conductivity using a digital multimeter. Next, the glass substrate containing the TIEs was fixed to a clean, flat surface by attaching each edge of the glass slide using thin strips of general-purpose masking tape. This setup is of paramount importance as the adhesive glueing to a flat surface enables the electrodes to remain still to agitations, which avoids any disruptions to the process of DEP.

3.3.1.3. Function Generator Setup

A function generator is an electronic device that generates a variety of periodic waveforms. It produces precise electrical signals with specific frequencies, amplitudes, and waveforms. In our experiments, we used the AFG 3021B function generator (Beaverton, Oregon, USA) to generate a sinusoidal wave with a frequency of 3MHz and an amplitude of 10Vpp. The generated output was also validated using a digital multimeter. Then, this function generator was connected to wires soldered at the pads of the TIEs panel.

3.3.2. Performing DEP Experiment

The final step was pipetting 3μL of the sample onto the TIEs panel. This step was conducted in complete darkness, devoid of light. The end of the experiment is when the sample pipetted on the TIEs panel completely dries. To determine this end, the experiment was monitored every 10 minutes. Once the sample dried, approximately 40 minutes after its application on the TIEs, the glass substrate with TIEs was analyzed under a fluorescence microscope and fluorescence images were recorded.

3.3.3. Automatic Identification of the End of Experiment

As indicated earlier, experiments were manually monitored to determine their endpoint. Determining this endpoint is essential as it indicates proceeding to the next stage of analyzing the resulting fluorescence produced by the concentrated biomarkers and determining their concentration. However, this process of manually monitoring the experiments is tedious and time-consuming, and any significant delays in determining the endpoint might cause photobleaching effects. Therefore, there is a need to automate this process.

We experimented with two approaches to solve this challenge. Both of these approaches involve capturing sequential photographs of the TIEs on which the sample was pipetted as the experiment progresses over time. The first approach captures these images under a fluorescent microscope, and the second approach uses a bright field microscope. In both approaches, the idea is to detect the end of the experiment based on the changes in the intensities demonstrated by these time-lapse images. As the experiment progresses, there are changes in the intensity, and at the end of the experiment, the intensity stabilizes. Therefore, the idea is to detect this change point where the intensity stabilizes using image processing and curve tracking algorithms. The image processing and curving tracking algorithm used to detect this change point and results obtained from testing these approaches on different experiments are elaborately discussed in the results section.

3.3.4. Automatic Detection of miRNA Concentration Using ML

The next step after detecting the end of the experiment is automatically predicting the concentration of the biomarker. As discussed earlier, DEP causes the biomarker of interest to concentrate on the concentrating regions of the T-electrodes. Therefore, concentration must be predicted only from the biomarker's fluorescence intensities in concentrating regions, which is also known as the total fluorescence intensity. To automatically compute this total fluorescence intensity, it is essential to detect these regions of TIEs using ML.

Object detection in ML is a computer vision task that aims to identify and localize objects within an image or video frame. It involves training algorithms to recognize various objects of interest and accurately predict their bounding boxes or regions in the input data. Popular object detection algorithms are Faster R-CNN (Region-based Convolutional Neural Networks) [56], YOLO (You Only Look Once) [57] and SSD (Single Shot Multibox Detector) [58]. This concept in ML can be utilized to detect these concentration regions of the TIEs where the object of our interest in an image would be the region denoted by an orange box in Figure 3.2.

Although the existing object detection algorithms can solve the problem, deploying such models onto low-cost hardware devices can be challenging due to their typically large number of weight parameters. Often, this issue can be addressed by employing high-end hardware configurations. However, as we are investigating a low-cost biomarker detection device which potentially involves the use of low-cost hardware devices, we plan to utilize the capability of NAS that we discussed in the second chapter to build the object detection framework. We plan to investigate an approach similar to [37], where the DL classification model designed by NAS can easily replace the backbone DL model used for the object detection task.

Popular backbone DL models used in the object detection frameworks are ResNet [59], VGG [60] and EfficientNet [61]. Previous studies have investigated these popular ML models for image classification, which significantly contribute to the weight parameters of the object detection framework. Therefore, this backbone DL model can be designed through a NAS approach, similar to the methodologies in the study discussed in the second chapter 2 of this thesis.

Moreover, as reported in the second chapter 2, NAS can design highly optimized ML models. Additionally, the NAS framework explored in the second chapter can readily be adapted to become hardware-aware. This adaptation involves integrating considerations of the hardware constraints and characteristics into the search process. This hardware-aware approach aims to optimize not only the performance metrics of the neural network architectures, like Accuracy (Acc), but also their efficiency and compatibility with specific hardware platforms. By incorporating hardware awareness into the NAS framework, it becomes possible to design neural network architectures that are well-suited for deployment on diverse hardware environments, ranging from edge devices to high performance computing clusters.

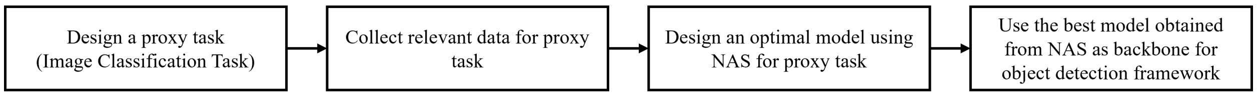
Design a proxy task
(Image Classification Task)
Collect relevant data for proxy task
Design an optimal model using NAS for proxy task
Use the best model obtained from NAS as backbone for object detection framework

Figure 3.4. Techniques Involved in Building a Hardware-Optimized Object Detection Framework for TIEs Concentrating Region Identification

Figure 3.4 illustrates the steps involved in building a hardware-optimized ML algorithm for automatically detecting the concentrating regions of TIEs. The first step involves the design of a proxy task to build a DL model using NAS. This proxy task is designed to enable the DL model to learn features of the TIEs so that when it is used as a DL backbone for the object detection framework, the accuracy of detecting these concentrating regions is close to 1. In ML, a proxy task is an auxiliary or intermediate task that facilitates learning or representing features for a primary task of interest. Since the primary task of our interest is the detection of concentrating regions of the TIEs panel, we need to design a proxy task which would enable the

NAS designed architecture to learn the features of the T-electrodes. A promising proxy task for investigation could be automatically identifying the rotation angle given a rotated image. This involves classifying the rotated fluorescent images of TIEs with the concentrated biomarker based on their rotation angles, encompassing orientations at 0, 45, and 90 degrees.

Once the concentrating regions are detected, the next step is the concentration prediction. This task can be accomplished in two ways. The first method uses ML, where we collect data for the known concentrations and use ML algorithms like regression to predict the concentration at each concentrating region of TIEs. The final concentration would be the mean of the predicted concentration at each concentrating region. The other approach uses image processing algorithms, which involve identifying an intensity threshold for each concentration range. In this case, we find the fluorescent intensity of biomarkers concentrated at each detected concentrating region of TIEs using image processing algorithms and compute the total fluorescent intensity by taking the sum of these individual intensities. Then, we compare this value with the specified threshold to categorize them into a specific concentration. Currently, we have not investigated methodologies involved in automatic concentration prediction as our collaborators are in the data collection phase to investigate these tasks.

3.4. Results and Discussion

3.4.1. DEP Experimental Results

DEP experiments were performed with samples of concentration 1μM containing ss-DNA molecules labelled with Alexa 555 fluorophore with and without the AC electric field. Figure 3.5 depicts fluorescent images showcasing concentrated biomarkers under these two experimental conditions: (a) with and (b) without the application of the AC electric field. The comparison highlights the efficacy of DEP in biomarker concentration.

The visualization of concentrated biomarkers on TIEs is possible with the help of fluorescent microscopy due to the fluorophores in the sample. Fluorescent lights are commonly used in fluorescence microscopy to illuminate samples and capture fluorescent images. Fluorescent lights emit specific wavelengths of light that can excite fluorophores within the sample. Fluorophores are molecules that absorb light energy at one wavelength (the excitation wavelength) and emit light at a longer wavelength (the emission wavelength). The excitation light from fluorescent lamps provides the energy required to excite the fluorophores in the sample. This excitation of fluorophores plays a vital role in fluorescence microscopy, enabling us to visualize.

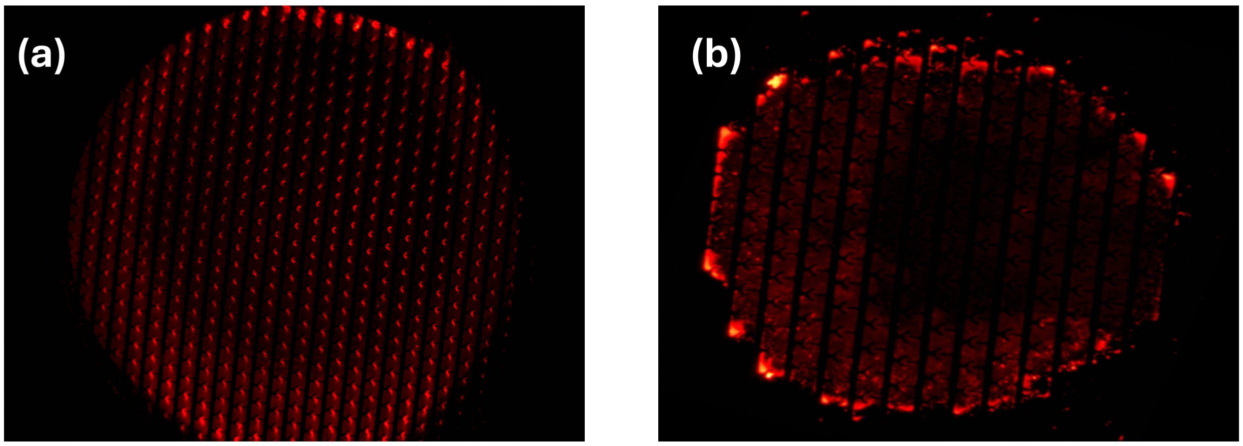

Figure 3.5. DEP Experimental Results. (a)With Electric Field (b) Without Electric Field

3.4.1.1. Recent Modification to the Experiment Protocol

One major limitation of the DEP experiments is that it is challenging to consistently avoid the coffee ring effect. The coffee ring effect is a phenomenon observed when a droplet dries on a solid surface. As the liquid evaporates, capillary forces draw the particles to the droplet's edges, forming a ring-like particle deposition pattern. Our collaborators elaborately studied reliable means to eliminate the coffee ring effect in the experiments, and the following adjustments were made to the experimental protocol.

- In the sample preparation step, ssDNA samples were suspended in a 0.0036xTris Ethylenediamine tetraacetic acid (EDTA) (TE) buffer (conductivity = 5 µS/cm), purchased from Thermo Fisher Scientific Chemical Inc. (220 Neck Road, Ward Hill, MA, 01835, USA), supplemented with .0067% (by volume) Tween solution, purchased from Fisher Scientific Company LLC (4500 Turnberry Drive, Hanover Park, IL, 60133-5491, USA).

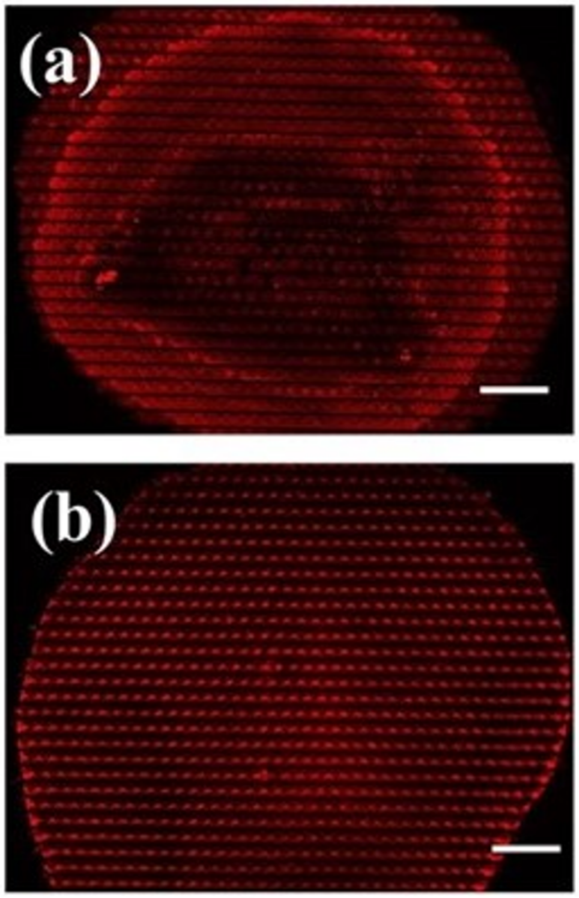

Figure 3.6. DEP Experimental Results. (a)Without Electric Field (b) With Electric Field Using the Modified Experimental Protocol to Avoid Coffee Ring Effect Consistently

- The electric field was produced by applying an electric potential of 10 Vpp at 1MHz.
- Briefly, during experiments, first, a layer of TE buffer was deposited around the electrodes to slow down the droplet evaporation. Then, the T-electrode array was covered with a non-transparent petri dish to prevent external disturbances.
- Electrodes were cleaned with Piranha solution before using them for the experiments. This solution was prepared by mixing H_2S0_4, purchased from Sigma-Aldrich Inc. (6950

Ambassador Drive, Allentown, PA 18106, USA), and H_2O_2, purchased from Sigma-Aldrich (6000 N. Teutonia Ave, Milwaukee, WI, 53209, USA), in 3:1 ratio.

Figure 3.6 shows the fluorescence images of concentrated DNA molecules in the electrodes (b) with and (a) without electric fields for an experiment performed with the modified protocol.

3.4.2. Results on the End of Experiment Detection

3.4.2.1. Results with Time-Lapse Fluorescent Microscope Images

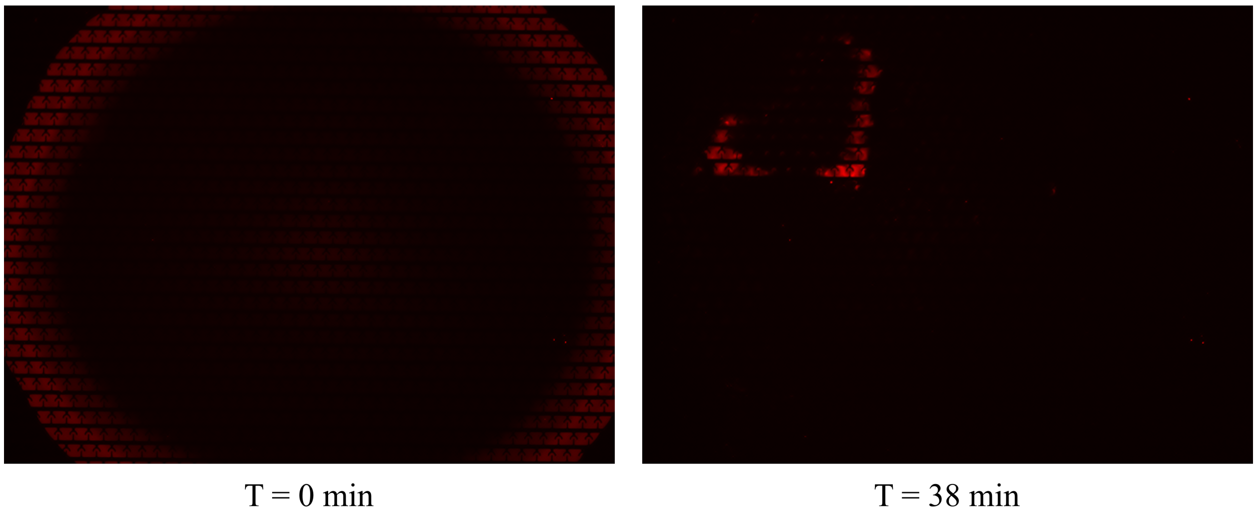

Figure 3.7. Time-Lapse Captures of Fluorescent Images Observed at Time 0 and 38 minutes

As discussed earlier, to automatically determine the end of the experiment, we captured time-lapse fluorescent images of TIEs when the experiments were progressing. However, this resulted in the disruption of the DEP effect, thereby leading to the incorrect concentration of the biomarkers on the TIEs. Additionally, we observed that due to the constant exposure to fluorescent light from the microscope, the photosensitive fluorophore eventually loses its ability to emit light for visualization. This is evident from Figure 3.7 where timelapse captures selected at time t = 0 and 38 minutes are illustrated. We captured these sequences of images every 2 minutes. Increasing the frequency of capturing these images, that is, increasing the time interval from 2 minutes to any higher duration might suppress these factors affecting the experiment. Still, at the same time, it would cause significant delays in detecting the end of the experiment.

3.4.2.2. Results with Time-Lapse Bright Field Images

Due to encountering experimental failures in the method of capturing time-lapse fluorescent images, we decided to investigate an alternative approach, bright-field time-lapse imaging.

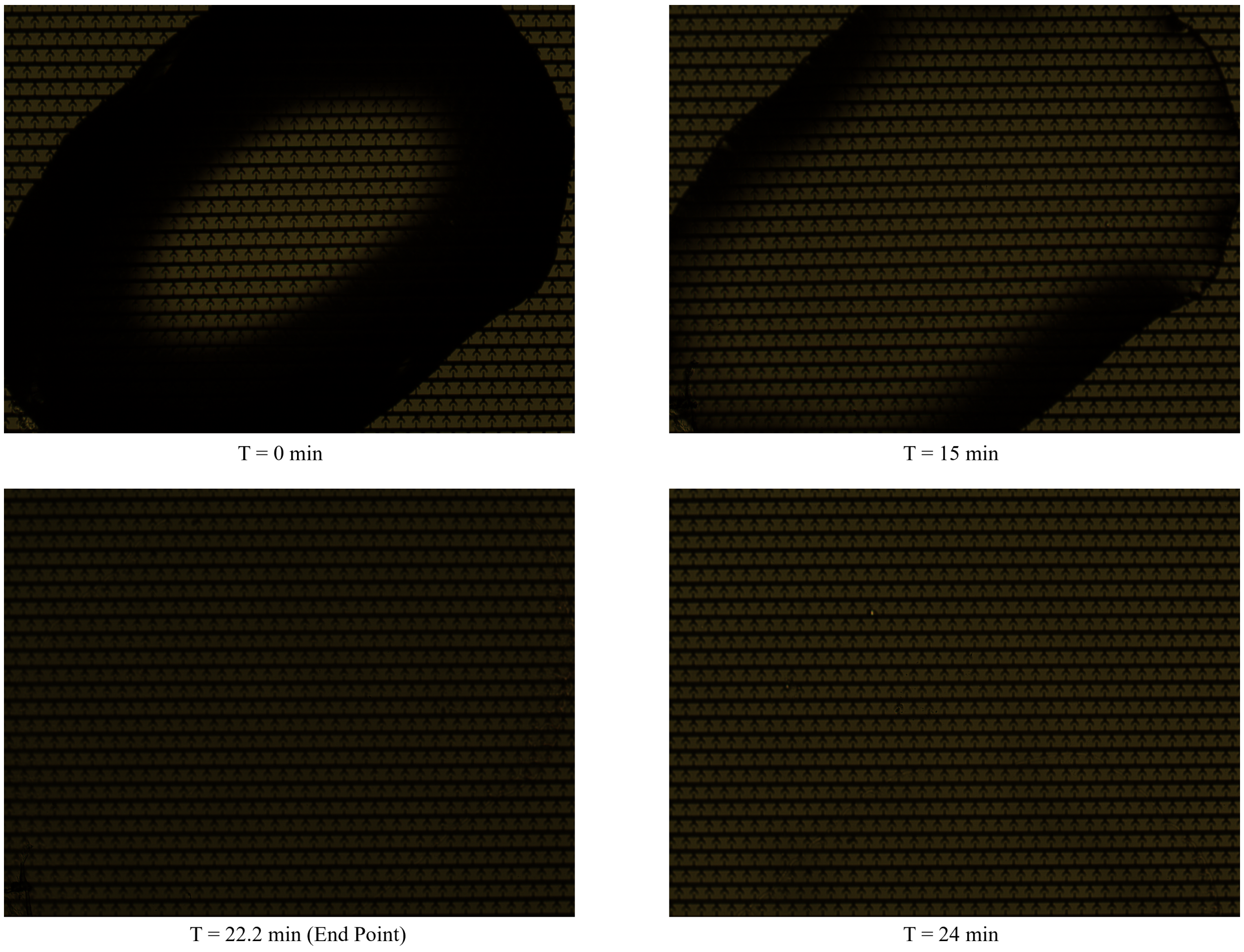

Figure 3.8. Time-Lapse Captures of Bright field Images Observed at Time 0, 15, 22.2 and 30 minutes.

In this approach, we capture time-lapse images under bright-field conditions as the experiment progresses. We observed that exposure to bright field light does not make the fluorophores lose their ability to emit light, and the concentration of biomarkers through DEP was also not disturbed. We even experimented by varying the frequency of capturing these images. We determined that capturing images every 10 seconds is optimal for avoiding disturbances with the biomarker concentration and accurately detecting the experiment's endpoint.

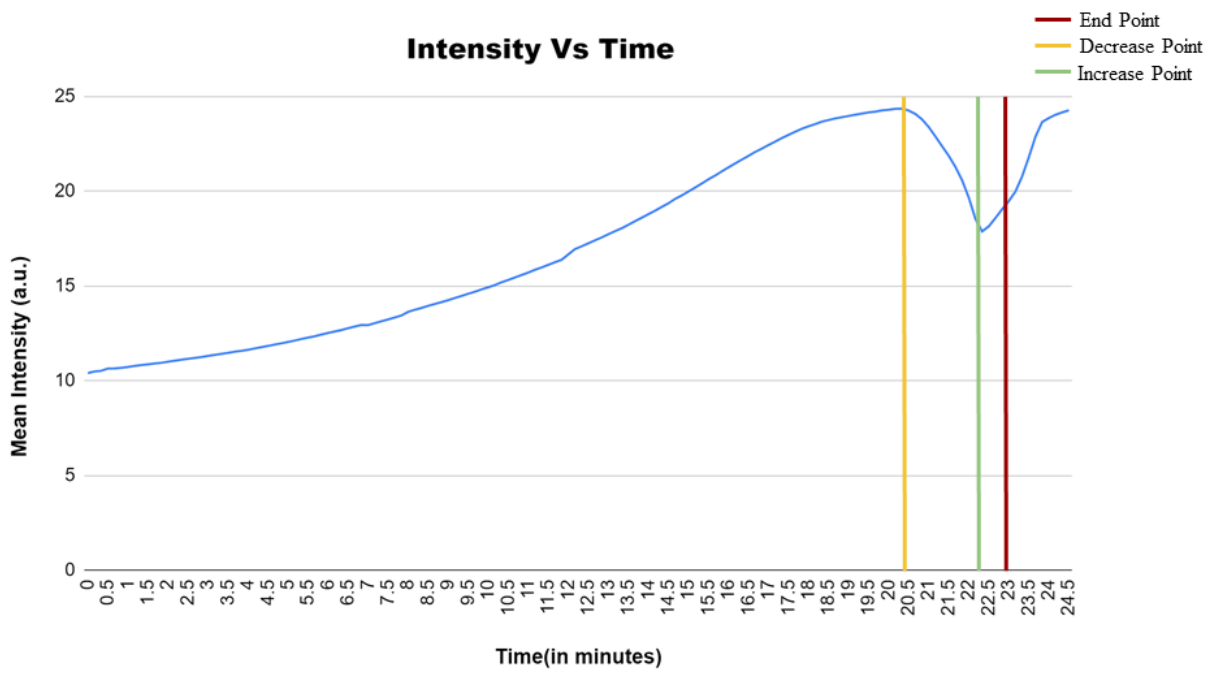

Figure 3.9. Intensity Vs Time Plot for Bright-Field Time-Lapse Captures

Figure 3.8 illustrates bright field timelapse captures at selected times. We calculated the intensity of the time-lapse images captured every 10 seconds, and the intensity vs time plot for a selected experiment is depicted in Figure 3.9. We used Python libraries such as OpenCV and Numpy to compute the intensities. The red line in the plots indicates the endpoint. It is evident from the plot that the endpoint does not fall at a specific changepoint in the plot, making their detection challenging. Therefore, we devised an approximation algorithm to detect these endpoints. Change points in a plot refer to specific locations or instances with a noticeable shift or discontinuity in the data. These shifts could represent changes in the underlying pattern, trend, or behaviour of the plotted data points. For instance, the points marked by the yellow and green lines in Figure 3.9 depict a change point.

The working of the approximation algorithm to detect the end point from the curve illustrated in the 3.9 is as follows.

- **Detecting the Change Point Where the Curve Starts Decreasing (marked by a yellow line):** The intensity value I_t of each image at a specific time t is compared to the intensity value I_{t-1} at t-1. A counter named "increase" gets incremented whenever $I_t > I_{t-1}$ and a counter named "decrease" gets incremented whenever $I_t < I_{t-1}$. Both these counters get reset to 0 after every 90 seconds. Every time the values I_t and I_{t-1} are compared, the counters are compared too. If increase > decrease, we continue the process and if increase < decrease, the process quits and the point at which it quits is marked as the change point (marked by a yellow line).
- **Detecting the Change Point Where the Curve Starts Increasing (marked by a green line):** Similar to the logic mentioned above, the intensity value I_t of each image at time t is compared to the intensity value I_{t-1} at t-1, where t-1 > time at which the first changepoint marked by the yellow line is detected. A counter named "increase" gets incremented whenever $I_t > I_{t-1}$ and a counter named "decrease" gets incremented whenever $I_t < I_{t-1}$. Both these counters get reset to 0 after every 50 seconds. Every time the values I_t and I_{t-1} are compared, the counters are compared too. If increase < decrease, we continue the process, and if increase > decrease, the process quits, and the point at which it quits is marked as the change point (marked by a green line).
- **Detecting the End Point (marked by the red line):** Based on the data from the experiments we analyzed, we observed that the endpoint was a maximum of 2 minutes away from the change point, marked by the green line.

Table 3.1. Results of the Curve Tracking Algorithm for Increase Point Detection (in minutes)

Experiment	Increase Point Actual	Increase Point Detected
1	22.33	22.83
2	19.16	19.5
3	17.33	17.83
4	23.16	23.5
5	21.83	22,16
6	18.83	19

Table 3.2. Results of the Curve Tracking Algorithm for Decrease Point Detection (in minutes)

Experiment	Decrease Point Actual	Decrease Point Detected
1	20.33	21
2	15.83	16.5
3	13.33	13.5
4	22	22.5
5	19.5	19.5
6	18	18

We tested this algorithm on 6 experimental data of concentration 1μM. Tables 3.2, 3.1, 3.3 demonstrate the results of this algorithm where the End Point Actual (marked by the red line) is the observed time at which the sample dried, and the End Point Detected is computed by the approximation algorithm discussed above. Similar notations apply to Decrease Point (marked by the yellow line) and Increase Point (marked by the green line). The results in these tables are reported in minutes.

The differences in the actual and detected change points (Decrease and Increase Point) are because the algorithm waits until it is sure that the curve indeed has a decreasing or increasing trend to avoid detecting small variations as change points. Also, the time interval employed for resetting the counters in the algorithm was carefully adjusted to reduce delays in identifying change points while simultaneously ensuring optimal sensitivity to significant variations and minimizing the detection of minor fluctuations.

Table 3.3. Results of the Curve Tracking Algorithm for End Point Detection (in minutes)

Experiment	End Point Actual	End Point Detected
1	23	24.83
2	19.43	21.5
3	19	19.83
4	23.1	25.5
5	22.2	24.16
6	19.2	21

The experimental data used to report these results were acquired by our collaborators with the modified protocol discussed in section 3.4.1.1, and they measured the time at which the sample dries as accurately as possible. We are still in the process of acquiring more data, specifically for different concentrations, to validate and fine-tune our algorithm.

3.4.3. Data Collection for the ML Approach

As mentioned before, we haven't explored the techniques proposed for automatic biomarker concentration prediction yet. We are still in the data collection phase to design object detection frameworks using NAS. For this purpose, we plan to collect fluorescent images at the end of the DEP experiment with varying concentrations of miRNA samples.

3.5. Conclusion and Future Work

In this study, we delved into techniques to build an end-to-end biomarker detection device for the identification of pancreatic cancer specific circulating miRNA in blood. We used DEP and performed experiments on TIEs where ssDNA molecules tagged with fluorophore molecules were captured in their concentrating regions to prove the utility of our approach. We also discussed methods to automate the end of the experiment detection and concentration prediction of the biomarker concentrated. Additionally, preliminary results on our experimental approach and end of experiment detection algorithm were presented.

In the future, we plan to test our end of experiment detection algorithm across various concentrations and study the implications of the intensity trend we observed with the bright field time-lapse captures. We also plan to utilize miRNA samples to execute NAS-based ML approaches for concentration prediction.

4. ADDITIONAL PROJECT 2: AUTOMATIC AMINO ACID TYPE IDENTIFICATION IN ELLIPTICAL DICHROISM SPECTROMETER USING MACHINE LEARNING

4.1. Abstract

In this study, we introduce a pioneering Machine Learning (ML) approach for automatic amino acid identification by analysing their unique absorption profiles in an Elliptical Dichroism (ED) spectrometer. This advancement is critical for pharmaceuticals, medical diagnostics, and biochemistry, offering a substantial improvement in analytical precision. The development process involved thorough preprocessing to enhance the algorithm's accuracy. Validation with a distinct dataset affirmed its exceptional performance, laying the groundwork for a new paradigm in amino acid analysis. This research not only demonstrates the potential of ML in biochemical diagnostics but also signifies a critical milestone with practical implications in clinical diagnostics and drug development.

4.2. Introduction

Identifying amino acids is pivotal across various scientific disciplines, including biochemistry, pharmaceutical research, and medical diagnostics. Traditionally, methods such as chromatography and mass spectrometry have been employed for amino acid analysis [62]. While effective, these techniques often require extensive sample preparation and are time-consuming, complex and costly [63]. This underscores the pressing need for more efficient, scalable, and accessible alternatives.

Emerging technologies in ML offer promising solutions to these challenges. Specifically, this study introduces a novel ML algorithm designed to leverage the unique absorption profiles of amino acids under elliptically polarized light, a method unexplored by traditional approaches [64]. This innovative strategy harnesses ED spectrometry to capture the distinct spectral signatures of amino acids, providing a dataset rich in discriminatory features conducive to ML analysis.

Our research aims to bridge the gap between traditional amino acid detection methods and the capabilities offered by contemporary ML techniques. By developing a ML model grounded in robust spectral data, we endeavour to achieve a level of precision and efficiency hitherto unattainable with conventional methods. The model's development involved meticulous preprocessing to enhance its predictive accuracy, followed by rigorous validation against a diverse dataset to ensure its reliability in real-world applications.

This study details our methodological framework, from data acquisition through ED spectrometry to the intricacies of model training and validation [64]. It highlights ML's potential to revolutionize amino acid analysis and further sets a foundation for its application across a spectrum of bioanalytical tasks. Our findings aim to catalyze a paradigm shift in biochemical analysis, offering a scalable, cost-effective, and highly accurate alternative to traditional amino acid detection techniques for biomedical applications in health and disease.

4.3. Methodology

4.3.1. Data Acquisition and Distribution

In the realm of ML, the initial and arguably most critical step in any project is data acquisition. Data is the foundation upon which ML models are built and trained to make predictions or decisions. The process of data acquisition involves gathering, collecting, and organizing relevant datasets that are essential for training and testing ML algorithms. In this study, as the main goal is to develop an ML algorithm to detect the type of amino acid automatically using the absorption profiles generated by the ED spectrometer, the data to train the algorithm must comprise these absorption profiles. Experiments to acquire this data were performed by our collaborators. Therefore, only a summary of the experimentation protocol is discussed below.

To collect such information, the amino acid samples were dissolved in phosphate-buffered saline at designated concentrations and introduced into a cuvette for spectrometric analysis.

Table 4.1. Data Distribution Across 17 Amino Acid Types

Amino Acid Type	Data Count
L-Alanine	132
L-Arginine	142
L-Asparagine	17
L-Cysteine	12
L-Glutamate	26
L-Glutamine	22
L-Histidine	22
L-Isoleucine	37
L-Leucine	37
L- Lysine	42
L-Methionine	17
L-Phenylalanine	37
L-Proline	22
L-Serine	17
L-Threonine	17
L-Tryptophan	12
L-Valine	17

Before measurement, the system was preheated for thirty minutes to ensure consistent readouts. Initial blank measurements of the solvent were conducted five times to mitigate the influence of optical components within the measurement pathway. Subsequently, each sample underwent quintuple measurements. Absorbance values were adjusted relative to the blank, employing specific baseline correction techniques for angle sweep and ED modes, thereby calculating differential absorbance at two distinct polarization angles. In this way, multiple absorption profiles of 17 amino acid types were acquired. Detailed data distribution is shown in Table 4.1. Each absorption profile contains 72 data points corresponding to angles ranging from 0 to 355 with intervals of 5.

4.3.2. Data Preprocessing Techniques

Data preprocessing techniques play a pivotal role in ML because they directly impact the performance, accuracy, and efficiency of ML models. The subsequent sections outline the preprocessing techniques employed in this investigation.

4.3.2.1. Baseline Correction

We initiated our analytical process by implementing a baseline correction for each amino acid dataset. For each absorption profile, we utilized absorption readings at 0 and 355 degrees, both converted to radians, to construct a linear baseline. The values from this linear line were subtracted from the corresponding absorption readings recorded at angles between 0 and 355 with an interval of 5. This step was critical for minimizing systematic biases and enhancing the reliability of the subsequent analyses.

4.3.2.2. Normalization

After the baseline correction, the absorption data underwent a normalization process. This involved adjusting the readings to a common scale, which is essential for comparative analysis across diverse experimental setups. Such normalization aids in maintaining the integrity of data variations while ensuring consistency. Here's how the normalization process was performed for our use case.

- For each experimental data, mean and standard deviation were computed across all 72 data points. The mean represents the average value of the feature, while the standard deviation measures the dispersion of values around the mean.
- Each data point in the experimental data was subtracted by the mean and divided by the standard deviation. This transformation ensures that each feature has a similar scale and variation, which are beneficial for ML algorithms to improve their stability and performance.

By normalizing the data in this manner, the range of values for each feature becomes more consistent across the dataset. This can help prevent features with larger scales from dominating those with smaller scales during the training process of ML models.

4.3.2.3. Correlation-Based Data Filtration

Correlation-based filtering is a technique used in preprocessing experimental data to identify and potentially remove outliers before applying ML algorithms. By comparing individual records against a mean sequence derived from the collective dataset under each amino acid type, we removed experimental records that were minimally correlated with the mean sequence. In our implementation, we removed three such sequences under each amino acid type.

4.3.3. Model Development and Validation Framework

4.3.3.1. Data Split

Data splitting is a crucial step in ML, where the available dataset is divided into separate subsets for training, validation, and testing. In our investigation, we performed a stratified data split, where 70% of the data is used for training, 15% for validation, and 15% for testing. Stratified data splitting is a variation of the standard data splitting technique that ensures the class distribution in the dataset is preserved across the training, validation, and testing sets. This is particularly important when dealing with imbalanced datasets, where one class may be significantly underrepresented compared to others. This imbalance in our dataset is evident from Table 4.1.

4.3.3.2. Model Development

In our study, we delved into the realm of traditional ML algorithms, namely Random Forest [65], Decision Tree [66], and XGBoost [67]. These algorithms are stalwarts in the field, renowned for their robustness and versatility across various classification tasks.

In tandem with our exploration of traditional ML algorithms, we also embarked on a deep dive into deep neural networks. These architectures represent a cutting edge approach to ML, characterized by their ability to learn intricate patterns and representations from complex data. The details of their implementation and the architecture configurations are detailed in the results section.

4.4. Results and Discussion

In our experimental setup, we leveraged deep neural networks implemented using TensorFlow 2.11.1 alongside traditional ML algorithms implemented in scikit-learn to classify between all 17 amino acid types. Since traditional ML algorithms typically do not incorporate a separate validation dataset during training, we amalgamated the validation and test data into a unified test set for each of these algorithms. This consolidation enabled us to evaluate the performance of traditional ML models on a comprehensive dataset, encompassing both validation and test samples. Also, in our study, we employed traditional ML algorithms with default settings from the scikit-learn library to ensure a standardized and fair comparison across different models. These default settings represent the baseline configuration provided by scikit-learn, offering a starting point for model training and evaluation without any manual tuning of hyperparameters.

Additionally, it's worth noting that our implementation incorporated class weights based on the data distribution within our dataset. Class weights are a crucial aspect of training ML models, particularly when class imbalance is prevalent. Class imbalance occurs when the number of instances belonging to different classes in the dataset is not evenly distributed, which can lead to biased model predictions favouring the majority class. We assigned class weights based on the inverse of the class or amino acid type frequency, with each class weight calculated as the maximum number of experimental records divided by the number of records in that class. However, as the XGBoost algorithm in scikit-learn does not provide an option for directly assigning class weights, our exploration of this particular algorithm proceeded without incorporating them.

4.4.1. Classification Performance in Distinguishing 17 Amino Acid Types

Table 4.2. Accuracy Obtained Across Different ML Algorithms in Classifying 17 Amino Acid Types

Algorithm	Train	Validation	Test
Decision Tree	1.0	-	0.19
Random Forest	1.0	-	0.33
XgBoost	1.0	-	0.26
Deep Learning Model	0.159	0.157	0.159

Table 4.2 summarizes the results we obtained in classifying between all the 17 amino acid types across different algorithms. Accuracy (Acc) obtained across the data splits is reported in this table. In ML, Acc is a metric used to evaluate the performance of a classification model. It represents the ratio of correctly predicted instances to the total number of instances in the dataset.

The lower performance observed in our study can be attributed to the imbalance in class distribution within the dataset. The limited availability of experimental data, especially for certain amino acid categories like L-Tryptophan, poses a significant challenge for ML algorithms to learn discriminative features and generalize well to unseen data effectively. Despite setting class weights to address class imbalances, the small size of the dataset severely constrains the learning capacity of the models. It undermines their ability to extract meaningful patterns and relationships from the data. Thus, our next endeavour was to perform classification between L-Alanine and L-Arginine, which had the highest count of data among all the 17 amino acid types. Additionally, since there was inadequate data in most of the amino acid categories, using different DL models with varying hyperparameters didn't improve the results. Therefore, the best DL model we obtained in classifying L-Arginine and L-Alanine was used to report the results in Table 4.2. The architectural details associated with this DL Model will be elaborately discussed in the subsequent section.

4.4.2. Classification Performance in Distinguishing L-Arginine and L-Alanine

Table 4.3 summarizes our results in classifying L-Arginine and L-Alanine across different algorithms, and Table 4.4 summarizes the individual amino acid accuracy obtained across these algorithms. Figure 4.1 illustrates the configuration of the DL model utilized in our study. The model was structured with specific hyperparameters tailored to optimize learning dynamics and model performance. Notably, a learning rate of 0.001 and a batch size of 5 were employed to regulate the training process. The Adam optimizer, a popular choice in deep learning, was utilized to optimize the model parameters during training. The categorical cross-entropy loss function was employed as the objective function for training the DL model. Furthermore, class weights were not incorporated in this experiment as L-Alanine and L-Arginine do not showcase class imbalance.

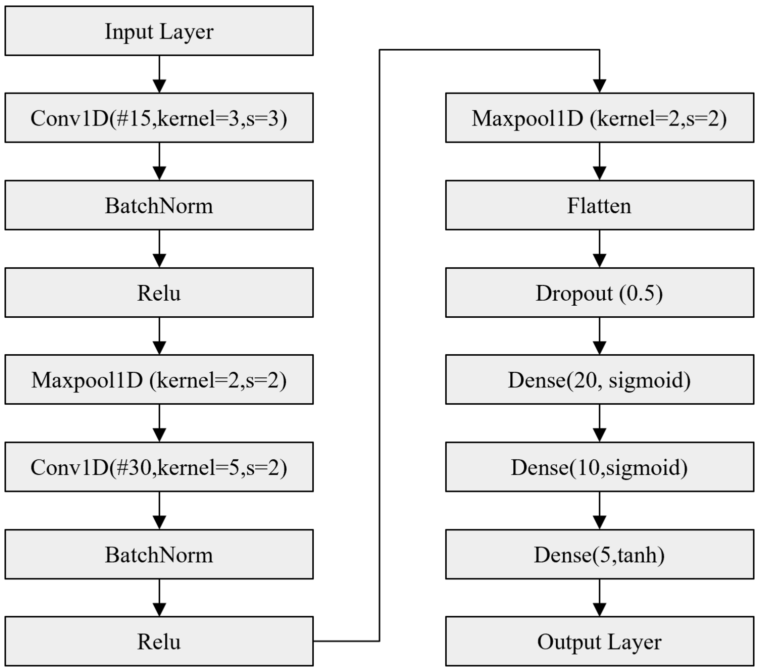

Figure 4.1. DL Model Configuration Used in Classifying L-Alanine and L-Arginine

Table 4.3. Accuracy Obtained Across Different ML Algorithms in Classifying L-Alanine and L-Arginine

Algorithm	Train	Validation	Test
Decision Tree	1.0	-	0.69
Random Forest	1.0	-	0.77
XgBoost	1.0	-	0.67
Deep Learning Model	0.83	0.86	0.8

The DL model utilized in our study incorporates several components, which are elaborately discussed below.

- Convolutional one-dimensional (Conv1D) Layers: Conv1D layers are specialized neural network layers designed to extract features from one-dimensional sequential data, such as time series or text. These layers apply a set of learnable filters to input sequences, sliding them along the sequence and computing convolutions at each position. Conv1D layers effectively capture local patterns and dependencies within the input sequences. When configuring a Conv1D layer in TensorFlow, several parameters can be specified to customize its behaviour and architecture. In our study, our primary focus was on three key parameters of the Conv1D layer in our DL model: filter size (#), kernel size, and strides. The number of filters (#) corresponds to the depth or the number of output channels of the Conv1D layer. Each filter captures different patterns or features from the input sequence. By varying the number of filters, we can control the richness and complexity of the features extracted by the Conv1D layer. The kernel size determines the length of the sliding window used for the convolution operation. It specifies the number of consecutive elements along the input sequence that each filter considers at a time. By adjusting the kernel size, we can control the scope or the size of the patterns captured by the filters. Strides determine the step size of the sliding window during the convolution operation. They specify the spacing between consecutive applications of the filters along the input sequence. By modifying the strides, we can control the spatial downsampling of the output feature maps.

- Batch Normalization (BatchNorm): Batch normalization is a technique used to improve the stability and performance of deep neural networks by normalizing the outputs of each layer. In 1D data processing, BatchNorm normalizes the outputs across the channels (features) of each sample in a batch. It computes the mean and standard deviation of the activations within each channel and then normalizes the activations using these statistics.
- Rectified Linear Unit (ReLU): ReLU is an activation function commonly used in deep neural networks to introduce non-linearity into the model. In 1D data processing, ReLU

applies an element-wise operation to the data, setting negative values to zero while leaving positive values unchanged.

- MaxPooling Layer (Maxpool 1D): MaxPooling is a downsampling operation commonly used in convolutional neural networks (CNNs) to reduce the spatial dimensions of feature maps while retaining the most salient information. When using MaxPooling with a kernel size of 2 and a stride of 2, the output size is reduced by a factor of 2.
- Flatten Layer: A Flatten layer is used to reshape a tensor to a 1D vector suitable to input to Dense Layers.
- Dense Layers: A Dense layer, also known as a fully connected layer, connects every neuron in one layer to every neuron in the subsequent layer. Dense layers are typically employed in the final stages of the neural network architecture, transforming the high-level features learned by earlier layers into predictions or classifications.
- Dropout Layer: Dropout is a regularization technique commonly used in deep neural networks to prevent overfitting and improve generalization performance. Dropout layers randomly deactivate a fraction of neurons during training, effectively "dropping out" or temporarily removing them from the network with a certain probability. By introducing noise and redundancy into the network, Dropout encourages the model to learn more robust and generalizable data representations, reducing the likelihood of overfitting. During inference (i.e., when making predictions), Dropout layers are typically deactivated, and the full network generates predictions without dropout.
- Output Layer: The output layer, also known as the output or prediction layer, is the final layer in the neural network architecture. It generates the model's predictions or classifications based on the learned features extracted from the input data.

From Tables 4.3 and 4.2, it is clear that the traditional ML algorithms demonstrate overfitting. Overfitting occurs when a model learns to memorize the training data rather than capturing underlying patterns that generalize well to unseen data. However, this is not observed

Table 4.4. Individual Amino Acid Accuracy Obtained Across Different ML Algorithms

Algorithm	L-Arginine	L-Alanine
Decision Tree	0.69	0.7
Random Forest	0.81	0.73
XgBoost	0.69	0.65
Deep Learning Model	0.76	0.85

with the performance of the DL model. This demonstrates that traditional ML algorithms, such as Random Forest, Decision Tree, and XGBoost, are susceptible to overfitting when trained on datasets with limited size or noisy features. Additionally, the good performance of the DL model in classifying L-Arginine and L-Alanine proves that with adequate data, DL can be leveraged to classify all 17 amino acid types in the future.

4.5. Conclusion and Future Work

The study successfully demonstrates the potential of an ML algorithm to precisely identify amino acids through their unique absorption profiles under elliptically polarized light. As the acquired data for the 17 amino acid types were inadequate, the potential of this innovative approach, leveraging meticulous data preprocessing techniques, has been showcased through the high accuracy across amino acid groups, L-Alanine and L-Arginine, which had highest count of data. The findings suggest a significant advancement in biochemical analysis, offering a scalable, efficient alternative to traditional methods. Despite certain limitations, like the skewed distribution of the original dataset, this research paves the way for future developments in the field, highlighting the utility of ML in enhancing diagnostic and analytical methodologies in biochemistry and medical diagnostics. In the future, we plan to acquire adequate experimental records for other amino acid types and we plan to deploy the model on the ED spectrometer device for real-time automatic amino acid identification.

5. ADDITIONAL PROJECT 3: AUTOMATING THE TASK OF DATA ANNOTATION FOR PLANT WEED IMAGE SEGMENTATION USING SEGMENT ANYTHING MODEL [1]

5.1. Abstract

Weeds are unwanted plants that compete with crops for water and nutrients and can cause yield loss in sugarbeet and other crop fields. Accurate classification of weeds and crops in the images is critically important to implement integrated weed control. Machine Learning (ML) models can be used for this purpose. However, good-quality training datasets with good annotations are essential for ML algorithm development. Therefore, in this study, we use the Segment Anything Model (SAM) to automatically annotate data for weed identification and segmentation tasks. We then develop a U-Net model by training the model with this generated dataset. Successful weed identification using this approach ensures the accuracy of a prescription weed map for effective site-specific weed control.

5.2. Introduction

Weeds are unwanted plants that grow in the field and compete with crops for water, light, nutrients, and space [68][69] and can cause the sugarbeet yield loss of up to $ 699 million per year (Beet Sugar Development Foundation, 2021). The sugarbeet industry has been using in pre- and post-emergence whole-field herbicide applications (i.e., blanket application) for weed control. The spatial density of weeds is not uniform across the field, thereby resulting in the overuse of chemicals, which causes environmental concerns and leads to the

[1] This chapter is based on the paper "Weed Identification using U-Net Machine Learning Model and SAM Segmentation" submitted to 2024 American Society of Agricultural and Biological Engineers Meeting Presentation. This paper was authored by James Y. Kim, Ridhanya Sree Balamurugan, Madhava Sarma Vemuri and Umamaheswara Rao Tida. Dr. James Y. Kim annotated and verified the required dataset for the analysis and contributed to writing the Introduction Section of the manuscript. Ridhanya Sree Balamurugan trained the U-Net segmentation framework on the dataset, optimized the parameters suitable for good results and drafted the paper's Methodology, Results and Conclusion sections. Madhava Sarma Vemuri assisted in building the U-Net framework and reviewed the paper. Dr. Umamaheswara Rao Tida provided guidance on the research methodology, allocated resources to conduct the research and reviewed the manuscript.

evolution of Herbicide-Resistant (HR) weeds [70]. There is a high demand for weed management for HR weeds that threaten sugarbeet production in the U.S. For instance, waterhemp is a formidable HR weed in west central Minnesota and North Dakota and impedes sugarbeet production with its capability of growing 1 inch per day and populating 250,000 seeds per plant (Link to the Source). Efforts to control HR weeds have been made by adding S-metolachlor pre-herbicide treatments and utilizing agronomic practices such as crop rotation, dense seeding, and inter-row cultivation, but no effective solutions are reported. Site-specific weed management (SSWM) was introduced in the early 1990s [71] and has been implemented by image-based weed detection. Vegetation segmentation followed by crop row detection is a common approach using an unmanned aircraft system (UAS) for inter-row weed classification. ML-based classifications make outstanding improvements for weed identification [72][73][74], but there is still limited performance due to the strong dependence on weed types, growth stages, and lighting conditions [75] and sensitivity to the segmentation accuracy for different crops and field conditions [76]. In practice, there are still limitations and challenges for current weed recognition research to provide weed control in large-scale crop production systems.

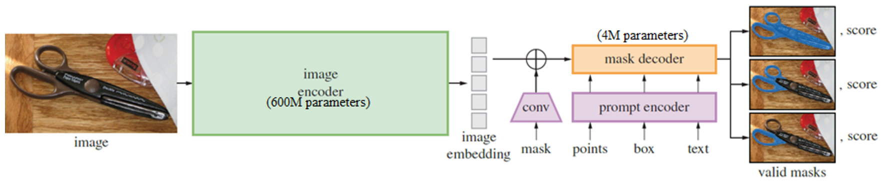

Figure 5.1. SAM Overview

A critical part of image analysis is a process known as image segmentation to identify regions of interest in an image accurately. Traditional segmentation techniques often require extensive human input and intervention for accurate results. Efforts were made to realize a more automated approach with artificial intelligence (AI) and Deep Learning (DL) methods, but still facing challenges, particularly in the effective segmentation of images with minimal human input [77]. There is a significant gap in terms of obtaining a quality dataset that has pixel-wise object class annotations for efficient weed detection. Recently, Meta AI developed an open-sourced, cutting-edge segmentation framework called Segment Anything Model (SAM) that can be applied for pixel-wise weed annotations ([78]). SAM can cut any object in any image, playing like a counterpart of ChatGPT in image segmentation. SAM consists of an encoder that embeds prompts and a lightweight mask decoder that combines the two information sources and predicts segmentation masks as shown in Figure 5.1 (adapted from [78]).

We propose a novel approach to improve the segmentation accuracy of training weed image datasets for weed detection. The goal of this study is to provide a new perspective to solve the weed annotation and identification problem. The objectives of the proposal are to 1) generate high-quality training datasets of weed images using Meta AI SAM with bounding box prompt and 2) evaluate the annotations using ML models such as U-Net [79].

5.3. Methodology

The dataset [80] utilized for our investigation comprises 3424 RGB images of weeds falling under four weed types: Horseweed (Erigeron canadensis), Kochia (Bassia scoparia), Ragweed (Ambrosia), and Redroot Pigweed (Amaranthus retroflexus). Each image in this dataset is also associated with a JSON file containing the coordinates of bounding boxes encapsulating the regions containing weeds. These images, along with their bounding box details, were passed as input to the SAM to generate ground truth masks for training the U-Net segmentation model.

Figure 5.2 illustrates the working of SAM in generating ground truth masks. When an image is passed as input, SAM segments all the objects present in the image and creates a mask using its component called SAM Automatic Mask Generator (Figure 5.2(a)). Since the ground truth masks required for training the U-Net must contain only segments of the weed regions, the bounding box data provided along with the images in the dataset helps SAM localize its segmentation to these regions. Therefore, utilizing this information, SAM specifically segments and generates masks only for the objects falling within the given bounding box coordinates using its component called SAM Predictor (Figure 5.2(b)).

Although the bounding box information enables SAM to generate ground truth masks as accurately as possible, there are some challenging scenarios where it generates incorrect masks. When the bounding box from the JSON file encompasses objects other than weeds or exhibits areas of varying brightness, with weed regions appearing dark, SAM segments the wrong objects in the image. Therefore, to ensure that the ground truth masks generated by SAM contain only masks for weeds, a visual inspection was performed to filter incorrect mask outputs from SAM. Through this process, we eliminated about 515 ground truth masks generated by SAM and their corresponding images. Therefore, 2909 images, along with their ground truth masks generated by SAM, were used for training the U-Net-based segmentation model to precisely segment weed canopy and classify the type of segmented weed.

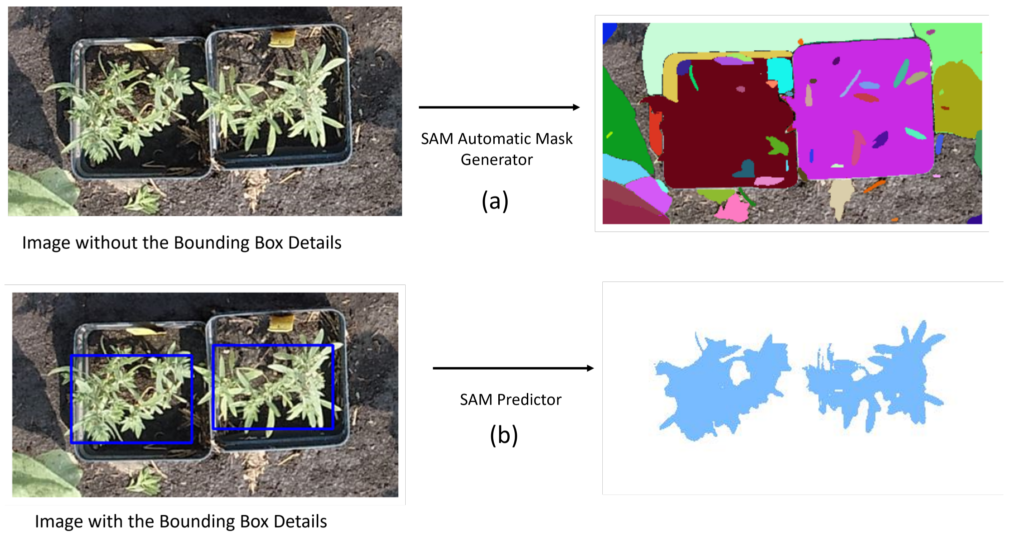
Image without the Bounding Box Details
SAM Automatic Mask Generator
(a)
Image with the Bounding Box Details
SAM Predictor
(b)

Figure 5.2. Automatic Segmentation Using SAM With and Without the Bounding Box Information; (a) Segmentation Using SAM Automatic Mask Generator, (b) Segmentation Using SAM Predictor

The architecture of UNet [81][82] investigated in this study is given in Figure 5.3. It consists of three units, including 1) Encoder, 2) Bottleneck, and 3) Decoder. An image passes through these consecutive units to generate the inference. The Encoder and Decoder unit is created using four stages. All the stages are created using a fundamental building block called the Residual Block. During the stages inside the Encoder and Bottleneck unit, the Residual Block helps to increase the dimensionality of the image by increasing the total number of features along the z-direction. Subsequently for the Decoder stages, the Residual Block helps to reduce the dimensionality of the image by decreasing the total number of features along the z-direction. The outputs of each Encoder unit are passed to the subsequent Decoder unit using skip connections, as shown in Figure 5.3. The construction of U-Net architecture is as follows:

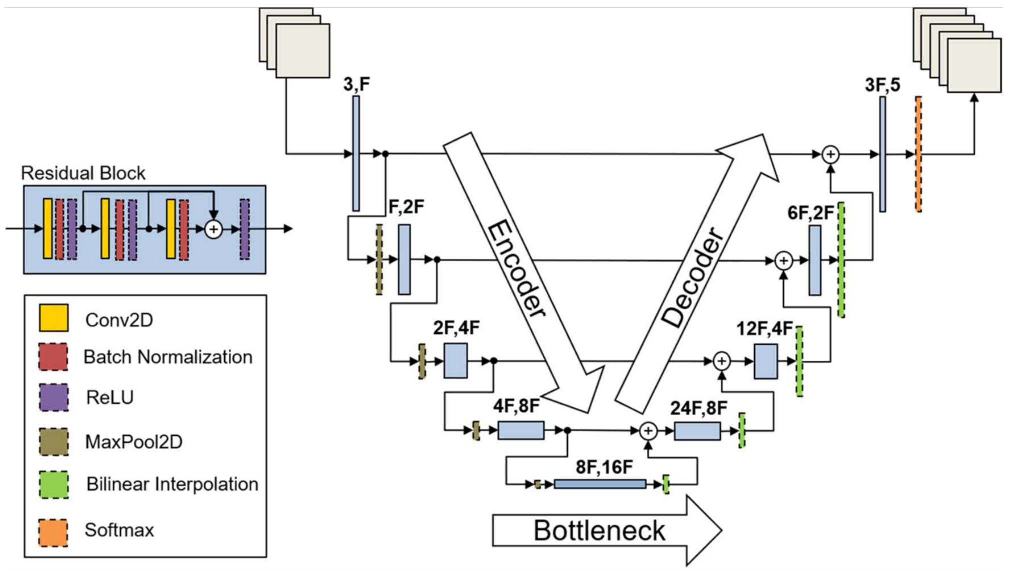

Figure 5.3. U-Net Architecture Used for the Segmentation Task

Inside the Residual Block are three sets of Convolutional layers, which are connected sequentially. The input is passed through the set of the first 2-dimensional Convolutional (Conv2D) layer with 3x3 filter size (unpadded convolutions), followed by batch normalization (BN) and rectified linear unit (ReLU) activation. The subsequent output is sent to the next set of Conv2D, BN, and ReLU operations. Finally, the output is sent to a final set of Conv2D and BN operations, where the residuals from the first and second sets of convolutions are added to the third convolution output using a skip connection, as shown in Figure 5.3. The subsequent result inside the Residual Block is passed through a ReLU activation. The output of each stage is passed to 1) the next stage and 2) the corresponding Decoder stage as a skip connection. During the subsequent stages, the size of the input images is reduced by half using a 2x2 MaxPool2D layer with a stride of 2, and the number of filters is doubled. This process continues until the Bottleneck stage, where the number of filters is increased by 16x. The output size of the Bottleneck stage is increased by 2 using Bilinear Interpolation with a scale factor of 2. The current output is concatenated to the corresponding Encoder stage output using a skip connection and passed as input to the Decoder stage. The process continues until the final decoder stage, where five filters are created using the network and a Softmax activation. These five filters contain the segmented output corresponding to the classes – Horseweed, Kochia, Ragweed, Redroot Pigweed, and Background.

5.4. Results and Discussion

The entirety of our study was implemented and executed in a workstation (Lambda, San Jose, CA) equipped with NVIDIA threefold RTX A6000 GPU and AMD Ryzen Threadripper PRO 5995WX 64-Core machine with 512GB RAM. Pytorch 2.2.1 was utilized to implement the U-Net framework (with initial filters F = 32) and to generate ground truth masks for the weed image dataset using SAM. As the images in the dataset were of varying sizes, for our experiments, we selected only the images which were under 1000 pixels in height and width and maintained the image resolution without resizing, because of the limitation of the GPU capacity. This resulted in a dataset of 2604 images, of which 80% of images falling under each weed type were randomly selected and used for training. The remaining 20% was evenly distributed for validating and testing the U-Net segmentation model.

Table 5.1. U-Net Model Performance Across Train, Test and Validation Datasets. Reported Scores are in terms of Average Dice Score.

Train	Test	Validation
0.982	0.952	0.944

For training the U-Net model, categorical cross-entropy was used as the loss function. As the data distribution of the four weed types was skewed, class weights were incorporated into this loss function to resolve the imbalance. Class weight for each weed type was set to the inverse of the total number of pixels corresponding to the weed segments in their ground truth masks. By incorporating these class weights into the training, we ensured that the U-Net segmentation model learnt features of all four weed types equally without bias towards the weed category represented by higher amounts of image data. Additionally, this U-Net model was trained for 1000 epochs on the training dataset with a batch size of 32 using the Adam Optimizer with a learning rate of 0.05. Furthermore, before training the U-Net, the pixel intensity of images in the train, test and validation datasets were normalized to range between 0 and 1 by dividing them by 255. Also, the pixel mean across the three channels (RGB) for the entire training dataset was computed and used to perform per-pixel mean subtraction on the entire train, test and validation dataset.

While training, the U-Net model was saved whenever an increase in the performance on the validation dataset was observed. This performance was measured using the popular metric called Average Dice Score, which ranges between 0 and 1. We computed the average dice score to measure the similarity between the ground truth masks generated by SAM and the output masks predicted by the U-Net on the validation dataset. Usually, a higher average dice score means that the ground truth masks and output masks predicted by U-Net almost completely overlap. The average dice score is the mean of dice scores obtained for each image in the validation dataset. The dice score between the ground truth and output mask for an image in the validation dataset can be defined as:

$$Dice = \frac{2 * \text{Area of Overlap between ground truth and predicted output masks}}{\text{Area of predicted output masks} + \text{Area of Ground truth masks}}$$

Table 5.1 summarizes the average dice score obtained across the train, test and validation datasets with the best U-Net model saved at the epoch where maximum performance was observed on the validation dataset. This result demonstrates the ability of the U-Net model to perform well on the unseen test and validation datasets compared to its performance on the training dataset. This also shows that the U-Net investigated in our study has generalized well to unseen data and did not suffer from overfitting. Overfitting is a scenario when the model performs well during training but fails to predict the correct outcomes on the unseen test dataset. Figure 5.4 also illustrates how the average dice score progressed over 1000 epochs for the training and validation dataset while training the U-Net. It is evident from the trend that with increasing epochs, the performance across the train and validation datasets improved. This demonstrates that the U-Net model used in our study has effectively learnt to segment and classify weeds from the training data.

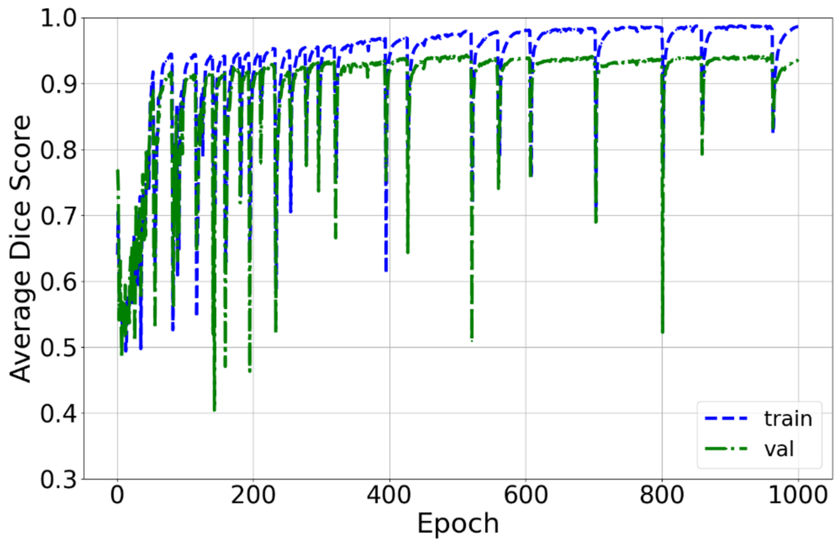

Figure 5.4. Average Dice Score vs. Epoch Observed on the Training and Validation Datasets During the Training of U-Net.

Table 5.2. U-Net Model Segmentation Results Across Different Weed Categories. Reported Scores are the Accuracy Values, ranging between 0 and 1

Horseweed	Kochia	Ragweed	Redroot pigweed	Overall Test Accuracy
0.919	0.91	0.935	0.9	0.919

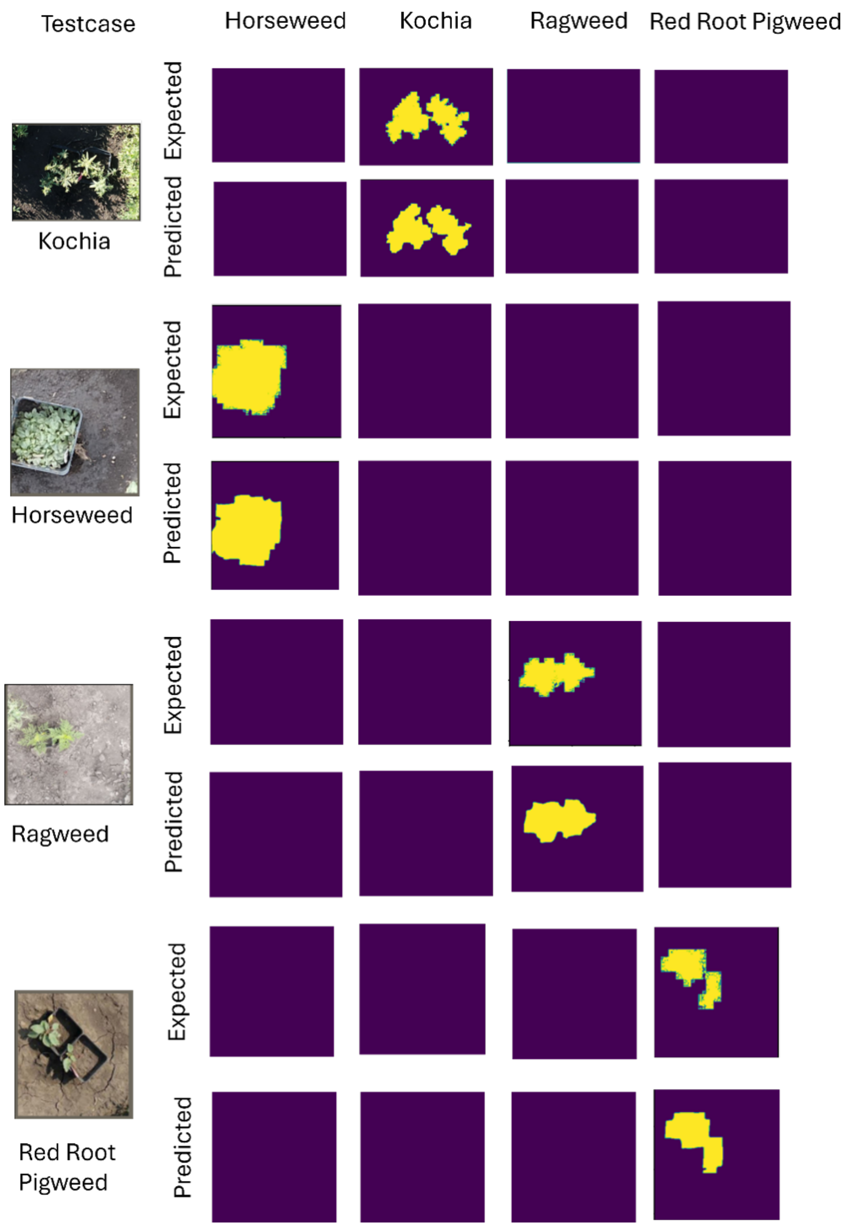

Figure 5.5. Segmentation Output Masks Predicted by U-Net in comparison to the Ground Truth Masks Across Different Weed Types in the Test Dataset. An Example Test Case from Each Weed Category was used for the Demonstration. Expected denotes the Ground Truth Masks Segmented Using the SAM, whereas Predicted denotes the Output Masks Predicted by the U-Net during Inference.

Table 5.2 summarizes U-Net's performance in terms of accuracy in classifying and segmenting four different weed types on the test dataset. This result demonstrates how accurately the U-Net model can segment the correct weed type at an Intersection over Union (IoU) greater than 0.5. With an overall test accuracy of about 0.91, we can conclude that the U-Net investigated in our study accurately distinguishes different weed types. Furthermore, the high accuracy obtained across individual weed types at an IoU threshold greater than 0.5 demonstrates that the U-Net performs reasonably well in predicting correct segmentations [83]. IoU is a popular metric used to measure the performance of ML models for object detection and segmentation tasks. Like the dice score, IoU ranges between 0 and 1. An IoU score greater than 0.5 is considered good [84]. IoU between the ground truth and output masks can be defined as:

$$IoU = \frac{\text{Area of Overlap between ground truth and predicted output masks}}{\text{Area of the ground truth and predicted output masks}}$$

We also plotted the outcomes that the U-Net predicts for test cases falling under each weed type from the test dataset. This is illustrated in Figure 5.5. Predicted are the output masks segmented by U-Net, and Expected are the ground truth masks generated by SAM. As stated earlier, images of different sizes were utilized for U-Net model training and testing, and this is illustrated in the figure. Additionally, through visual comparison, it is evident that the U-Net model investigated in our study demonstrates a higher capability to segment weeds as masks that are close to the ground truth masks generated by SAM.

5.5. Conclusion and Future Work

In this study, we investigated if data annotations associated with ML tasks involving the segmentation of weeds can be automated using the SAM. To prove the efficacy of our approach, we used a publicly available dataset consisting of images of four different weed categories and generated ground truth masks using the bounding box information provided. These annotated images were then utilized to train a U-Net segmentation model. The performance of this U-Net model demonstrated the validity of our approach.

Although in our experiments in this paper, we filtered the ground truth masks that were incorrectly segmented by SAM, in the future, we plan to use techniques like filtering masks based on pixel intensities, geometric conditions, and leaf pattern to completely avoid the use of bounding boxes to localize regions of interest. In this way, we plan to utilize the entire weed image dataset to train U-Net which can further improve the performance. Additionally, we plan to build a U-Net-based automatic data annotator for ML-based image segmentation tasks for weed image datasets as SAM does not generate masks specific to the weed images. We plan to train this U-Net-based annotator by combining publicly released weed image datasets automatically annotated using SAM. Furthermore, we plan to extend this work to the weed image datasets comprising UAV images.

6. CONCLUSION

6.1. Recapitulation of Research Premise

Our work conducted an in-depth examination of integrating Neural Architecture Search (NAS) with Deep Learning (DL) models, specifically targeted towards revolutionizing medical image classification. Conceived amidst a critical demand for advanced diagnostic tools, our investigation positioned NAS as a pivotal innovation, poised to transform the DL architecture development process. We hypothesized that NAS could markedly enhance diagnostic model precision and efficiency, heralding significant advancements in patient care and diagnostic accuracy.

The exploration was underpinned by the strategic deployment of a Long-Short-Term Memory (LSTM) controller, which navigated a comprehensive array of neural network configurations. This methodological journey, delineated by precise implementation parameters and augmented by training strategies like early stopping and checkpointing, sought to refine DL architectures capable of accurately classifying intricate medical images.

6.2. Synopsis of Key Discoveries

The zenith of our exploration revealed a model that exemplified not only superior performance metrics but also unique architectural tendencies, particularly a propensity for concatenation operations. This model's extraordinary ability to generalize across the DermaMNIST dataset significantly outstripped the existing benchmarks established by human designed architectures. These outcomes affirm the efficacy of NAS in discovering superior neural network architectures and underscore its potential to redefine medical image classification paradigms.

6.3. Study's Outcomes and Implications

Our findings illuminate the transformative impact of NAS on model development landscapes. By automating the search for optimal architectures, NAS platforms effectively minimize the necessity for extensive domain expertise among researchers, expediting the model development process and democratizing access to advanced Machine Learning (ML) techniques. This paradigm shift not only accelerates innovation but also broadens the scope for research and application within the field. Moreover, the efficiency of NAS in automatically designing optimal neural networks, exemplifies the time-saving prowess of NAS, paving the way for more inclusive and accessible advanced ML methodologies. Through automating essential aspects of the architectural design process, NAS platforms encourage wider engagement in ML, expanding research and application horizons.

6.4. Architectural Innovation via NAS

A critical aspect of NAS is its unparalleled ability to uncover accurate and novel structural designs beyond the conventional bounds of human intuition. This capability not only advances the field of DL but also introduces architectures that set new standards for performance and efficiency. NAS's role in fostering architectural innovation and excellence is indispensable for propelling the state of the art in DL forward.

6.5. NAS's Revolution in Medical Diagnostics

Centred around the DermaMNIST dataset, our empirical investigation showcases NAS's profound potential to revolutionize medical diagnostics. The automated generation of models that are both high performing and efficient for diagnostic applications highlights NAS's capacity to significantly improve patient care and diagnostic workflows. This validation of NAS's utility in medical diagnostics anticipates its wider application, poised to fundamentally alter the landscape of precision medicine.

6.6. Challenges and Perspectives

While NAS promises substantial advancements, it requires significant computational resources and complex framework construction. Nonetheless, the evolving landscape of hardware technology is progressively mitigating these challenges. NAS frameworks' scalability amplifies research efficiency, offering a versatile instrument for diverse datasets and tasks. Although expert interpretation is crucial, the NAS methodology streamlines the development process, reducing the dependency on extensive development teams and deepening our understanding of DL operations' operational implications.

In essence, our work validates the transformative potential of NAS in enhancing DL models for medical image classification, setting a foundation for future innovations in medical diagnostics. By demonstrating NAS's effectiveness with the DermaMNIST dataset, we not only underscore its capability to improve diagnostic precision across various medical conditions but also herald a new era in the democratization of advanced medical diagnostics, ensuring the broader accessibility of cutting edge Artificial Intelligence (AI) and ML innovations in healthcare.

6.7. Future Work

The completion of this thesis heralds the beginning of an expansive journey into the realms of NAS, with its successful application, using the REINFORCE algorithm for the classification of skin lesions, showcasing a seminal advancement in medical diagnostics through DL. This study underlines the efficacy of NAS frameworks not just in transcending the capabilities of manually designed models but in achieving such with significantly fewer parameters. Such progress sets the stage for an array of promising research directions.

6.7.1. Advancing Towards Hardware-Aware NAS Models

The evolution of NAS beckons a shift towards models cognizant of hardware constraints, achieved by redefining the reward function to encompass validation accuracy, inference time, and memory usage. This novel approach aims to produce models that are not only precise but also optimized for specific hardware limitations, marking a significant leap in the DL model optimization domain. Introducing a more intricate reward system could herald the development of models that embody the ideal equilibrium between performance and resource efficiency, establishing Hardware-aware NAS as a pivotal advancement in creating scalable, efficient solutions suitable for varied deployment contexts.

6.7.2. Validation Across Broader Medical Datasets

Extending the validation of our NAS framework to encompass a wider array of medical datasets is paramount. This endeavour will enable a comprehensive assessment of the architecture's adaptability and performance across different imaging challenges, affirming NAS's potential as a versatile tool in medical diagnostics. Such expansive validation is instrumental in reinforcing the applicability and effectiveness of NAS-derived models in navigating the complexities of medical image analysis.

6.7.3. Exploring Alternative Reinforcement Learning Algorithms

Avenues for enhancing NAS frameworks include the exploration and integration of alternative Reinforcement Learning (RL) algorithms. Through the assessment and comparison of various RL strategies, the architecture search process could be optimized, potentially uncovering architectures of superior efficiency and performance. This comparative analysis will shed light on the relative merits of different RL algorithms, informing the strategic selection of algorithms in future NAS endeavours.

6.7.4. Extending NAS to Other Image-Related Tasks

The extension of NAS frameworks to address other image-related tasks, such as segmentation, signifies a notable expansion of this research. Image segmentation's critical role in medical imaging underscores the potential impact of NAS in simplifying and enhancing the analysis of complex images. Developing NAS frameworks specifically for segmentation tasks could revolutionize medical imaging analyses, offering more precise, detailed insights.

6.7.5. Application in Non-Medical Fields: Agriculture

The foundational principles of NAS, as explored in this thesis, have implications far beyond medical diagnostics. In agriculture, for example, researchers often face a steep learning curve in applying ML for crop disease detection or yield prediction. NAS can significantly mitigate these challenges by automating the creation of efficient, optimized DL models, thus accelerating agricultural research and advancements without the need for in-depth ML knowledge.

6.7.6. Investigating Alternative NAS Algorithms

Lastly, exploring NAS algorithms beyond the realm of RL, such as evolutionary algorithms and one-shot NAS approaches, presents a promising research trajectory. These alternative methodologies offer fresh perspectives on navigating the architecture search space, potentially unveiling more efficient architectural solutions. Delving into the comparative effectiveness of these diverse NAS algorithms will enrich our understanding of the NAS ecosystem, fostering the development of advanced, high-impact models.

The validation of NAS through this thesis, particularly for skin lesion classification, affirms the approach's validity and catalyzes many prospects for future exploration. By broadening the application scope of NAS, refining the algorithms at its core, and venturing into new domains, we stand on the precipice of harnessing NAS's full potential. This endeavour aims to cultivate DL models that are at the forefront of innovation, offering significant contributions across a broad spectrum of fields and challenges.

REFERENCES

[1] C. Chakraborty, M. Bhattacharya, S. Pal, and S.-S. Lee, “From machine learning to deep learning: Advances of the recent data-driven paradigm shift in medicine and healthcare,” *Current Research in Biotechnology*, vol. 7, p. 100164, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2590262823000461

[2] Y. Qiu, L. Ma, and R. Priyadarshi, “Deep learning challenges and prospects in wireless sensor network deployment,” *Archives of Computational Methods in Engineering*, pp. 1–24, 2024.

[3] M. M. Taye, “Understanding of machine learning with deep learning: architectures, workflow, applications and future directions,” *Computers*, vol. 12, no. 5, p. 91, 2023.

[4] B. Sahiner, A. Pezeshk, L. M. Hadjiiski, X. Wang, K. Drukker, K. H. Cha, R. M. Summers, and M. L. Giger, “Deep learning in medical imaging and radiation therapy,” *Medical physics*, vol. 46, no. 1, pp. e1–e36, 2019.

[5] X. Shen, C. Jiang, Y. Wen, C. Li, and Q. Lu, “A brief review on deep learning applications in genomic studies,” *Frontiers in Systems Biology*, vol. 2, p. 877717, 2022.

[6] P. N. Ahmad, A. M. Shah, and K. Lee, “A review on electronic health record text-mining for biomedical name entity recognition in healthcare domain,” in *Healthcare*, vol. 11, no. 9. MDPI, 2023, p. 1268.

[7] H.-P. Chan, R. K. Samala, L. M. Hadjiiski, and C. Zhou, “Deep learning in medical image analysis,” *Deep learning in medical image analysis: challenges and applications*, pp. 3–21, 2020.

[8] S. Rasool, A. Husnain, A. Saeed, A. Y. Gill, and H. K. Hussain, “Harnessing predictive power: Exploring the crucial role of machine learning in early disease detection,” *JURIHUM: Jurnal Inovasi dan Humaniora*, vol. 1, no. 2, pp. 302–315, 2023.

[9] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, “Deep learning for healthcare: review, opportunities and challenges,” *Briefings in bioinformatics*, vol. 19, no. 6, pp. 1236–1246, 2018.

[10] I. Tobore, J. Li, L. Yuhang, Y. Al-Handarish, A. Kandwal, Z. Nie, L. Wang *et al.*, “Deep learning intervention for health care challenges: some biomedical domain considerations,” *JMIR mHealth and uHealth*, vol. 7, no. 8, p. e11966, 2019.

[11] Y. Wang, L. Liu, and C. Wang, “Trends in using deep learning algorithms in biomedical prediction systems,” *Frontiers in Neuroscience*, vol. 17, p. 1256351, 2023.

[12] C. Metta, A. Beretta, R. Guidotti, Y. Yin, P. Gallinari, S. Rinzivillo, and F. Giannotti, “Advancing dermatological diagnostics: Interpretable ai for enhanced skin lesion classification,” *Diagnostics*, vol. 14, no. 7, p. 753, 2024.

[13] M. Li, Y. Jiang, Y. Zhang, and H. Zhu, “Medical image analysis using deep learning algorithms,” *Frontiers in Public Health*, vol. 11, p. 1273253, 2023.

[14] C. Mennella, U. Maniscalco, G. De Pietro, and M. Esposito, “Ethical and regulatory challenges of ai technologies in healthcare: A narrative review,” *Heliyon*, 2024.

[15] L. Pinto-Coelho, “How artificial intelligence is shaping medical imaging technology: A survey of innovations and applications,” *Bioengineering*, vol. 10, no. 12, p. 1435, 2023.

[16] I. H. Sarker, “Deep learning: a comprehensive overview on techniques, taxonomy, applications and research directions,” *SN Computer Science*, vol. 2, no. 6, p. 420, 2021.

[17] L. Liao, H. Li, W. Shang, and L. Ma, “An empirical study of the impact of hyperparameter tuning and model optimization on the performance properties of deep neural networks,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 3, pp. 1–40, 2022.

[18] T. Yu and H. Zhu, “Hyper-parameter optimization: A review of algorithms and applications,” *arXiv preprint arXiv:2003.05689*, 2020.

[19] F. Z. El-Hassani, M. Amri, N.-E. Joudar, and K. Haddouch, “A new optimization model for mlp hyperparameter tuning: Modeling and resolution by real-coded genetic algorithm,” *Neural Processing Letters*, vol. 56, no. 2, pp. 1–31, 2024.

[20] L. Yang and A. Shami, “On hyperparameter optimization of machine learning algorithms: Theory and practice,” *Neurocomputing*, vol. 415, pp. 295–316, 2020.

[21] J. B. Awotunde, A. L. Imoize, O. B. Ayoade, M. K. Abiodun, D.-T. Do, A. Silva, and S. N. Sur, “An enhanced hyper-parameter optimization of a convolutional neural network model for leukemia cancer diagnosis in a smart healthcare system,” *Sensors*, vol. 22, no. 24, p. 9689, 2022.

[22] M. Decuyper, J. Maebe, R. Van Holen, and S. Vandenberghe, “Artificial intelligence with deep learning in nuclear medicine and radiology,” *EJNMMI physics*, vol. 8, no. 1, p. 81, 2021.

[23] L. B. Sappa, I. P. Okuwobi, M. Li, Y. Zhang, S. Xie, S. Yuan, and Q. Chen, “Retfluidnet: retinal fluid segmentation for sd-oct images using convolutional neural network,” *Journal of Digital Imaging*, vol. 34, no. 3, pp. 691–704, 2021.

[24] S. Lins, K. D. Pandl, H. Teigeler, S. Thiebes, C. Bayer, and A. Sunyaev, “Artificial intelligence as a service: classification and research directions,” *Business & Information Systems Engineering*, vol. 63, pp. 441–456, 2021.

[25] C. Barakat *et al.*, "Design and evaluation of parallel and scalable machine learning research in biomedical modelling applications," *University of Iceland, School of Engineering and Natural Sciences*, 2023.

[26] N. DeCastro-García, A. L. Munoz Castaneda, D. Escudero García, M. V. Carriegos *et al.*, "Effect of the sampling of a dataset in the hyperparameter optimization phase over the efficiency of a machine learning algorithm," *Complexity*, vol. 2019, 2019.

[27] N. Shawki, R. R. Nunez, I. Obeid, and J. Picone, "On automating hyperparameter optimization for deep learning applications," in *2021 IEEE Signal Processing in Medicine and Biology Symposium (SPMB)*. IEEE, 2021, pp. 1–7.

[28] Y. A. Ali, E. M. Awwad, M. Al-Razgan, and A. Maarouf, "Hyperparameter search for machine learning algorithms for optimizing the computational complexity," *Processes*, vol. 11, no. 2, p. 349, 2023.

[29] M. R. Hossain and D. Timmer, "Machine learning model optimization with hyper parameter tuning approach," *Global Journal of Computer Science and Technology*, vol. 21, no. D2, pp. 7–13, 2021.

[30] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[31] I. Salehin, M. S. Islam, P. Saha, S. Noman, A. Tuni, M. M. Hasan, and M. A. Baten, "Automl: A systematic review on automated machine learning with neural architecture search," *Journal of Information and Intelligence*, vol. 2, no. 1, pp. 52–81, 2024.

[32] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, X. Chen, and X. Wang, "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 4, pp. 1–34, 2021.

[33] R. Panda, M. Merler, M. S. Jaiswal, H. Wu, K. Ramakrishnan, U. Finkler, C.-F. R. Chen, M. Cho, R. Feris, D. Kung *et al.*, "Nastransfer: Analyzing architecture transferability in large scale neural architecture search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, 2021, pp. 9294–9302.

[34] Y. Wang, L. Zhen, J. Zhang, M. Li, L. Zhang, Z. Wang, Y. Feng, Y. Xue, X. Wang, Z. Chen *et al.*, "MedNAS: Multi-Scale Training-Free Neural Architecture Search for Medical Image Analysis," *IEEE Transactions on Evolutionary Computation*, 2024.

[35] M. J. Ali, L. Moalic, M. Essaid, and L. Idoumghar, "Designing Convolutional Neural Networks using Surrogate assisted Genetic Algorithm for Medical Image Classification," in *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, 2023, pp. 263–266.

[36] T. Elsken, A. Zela, J. H. Metzen, B. Staffler, T. Brox, A. Valada, and F. Hutter, "Neural architecture search for dense prediction tasks in computer vision," *arXiv preprint arXiv:2202.07242*, 2022.

[37] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[38] J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, and B. Ni, "MedMNIST v2-A large-scale lightweight benchmark for 2D and 3D biomedical image classification," *Scientific Data*, vol. 10, no. 1, p. 41, 2023.

[39] Z. Zhang, L. Zhang, L. Wang, K. Zhong, and H. Huang, "LC2R-ViT: Long-Range Cross-Residual Vision Transformer for Medical Image Classification," in *2023 International Annual Conference on Complex Systems and Intelligent Science (CSIS-IAC)*. IEEE, 2023, pp. 445–450.

[40] F. Mercaldo, L. Brunese, F. Martinelli, A. Santone, and M. Cesarelli, "Experimenting with Extreme Learning Machine for Biomedical Image Classification," *Applied Sciences*, vol. 13, no. 14, p. 8558, 2023.

[41] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," *Advances in neural information processing systems*, vol. 28, 2015.

[42] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 1946–1956.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[44] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[45] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[46] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, pp. 229–256, 1992.

[47] N. Codella, V. Rotemberg, P. Tschandl, M. E. Celebi, S. Dusza, D. Gutman, B. Helba, A. Kalloo, K. Liopyris, M. Marchetti *et al.*, "Skin lesion analysis toward melanoma detection 2018: A challenge hosted by the international skin imaging collaboration (isic)," *arXiv preprint arXiv:1902.03368*, 2019.

[48] P. Tschandl, C. Rosendahl, and H. Kittler, "The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions," *Scientific data*, vol. 5, no. 1, pp. 1–9, 2018.

[49] J. J. Ott, A. Ullrich, and A. B. Miller, "The importance of early symptom recognition in the context of early detection and cancer survival," *European Journal of Cancer*, vol. 45, no. 16, pp. 2743–2748, 2009.

[50] T. Kamisawa, L. D. Wood, T. Itoi, and K. Takaori, "Pancreatic cancer," *The Lancet*, vol. 388, no. 10039, pp. 73–85, 2016.

[51] M. Herreros-Villanueva and L. Bujanda, "Non-invasive biomarkers in pancreatic cancer diagnosis: what we need versus what we have," *Annals of translational medicine*, vol. 4, no. 7, 2016.

[52] G. A. Ganepola, J. R. Rutledge, P. Suman, A. Yiengpruksawan, and D. H. Chang, "Novel blood-based microrna biomarker panel for early diagnosis of pancreatic cancer," *World journal of gastrointestinal oncology*, vol. 6, no. 1, p. 22, 2014.

[53] B. N. Chorley, E. Atabakhsh, G. Doran, J.-C. Gautier, H. Ellinger-Ziegelbauer, D. Jackson, T. Sharapova, P. S. Yuen, R. J. Church, P. Couttet *et al.*, "Methodological considerations for measuring biofluid-based microrna biomarkers," *Critical reviews in toxicology*, vol. 51, no. 3, pp. 264–282, 2021.

[54] L. Velmanickam, V. Jayasooriya, and D. Nawarathna, "Integrated dielectrophoretic and impedimetric biosensor provides a template for universal biomarker sensing in clinical samples," *Electrophoresis*, vol. 42, no. 9-10, pp. 1060–1069, 2021.

[55] R. Pethig, "Dielectrophoresis: Status of the theory, technology, and applications," *Biomicrofluidics*, vol. 4, no. 2, 2010.

[56] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[57] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[58] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer, 2016, pp. 21–37.

[59] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[60] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[61] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*. PMLR, 2019, pp. 6105–6114.

[62] W. Xu, C. Zhong, C. Zou, B. Wang, and N. Zhang, "Analytical methods for amino acid determination in organisms," *Amino Acids*, vol. 52, no. 8, pp. 1071–1088, 2020.

[63] Y. Zhou and J. Yoon, "Recent progress in fluorescent and colorimetric chemosensors for detection of amino acids," *Chemical Society Reviews*, vol. 41, no. 1, pp. 52–67, 2012.

[64] A. Bauer, S. Elamurugan, S. A. Tolba, E. Nega, I. T. Lima Jr, W. Xia, D. Sun *et al.*, "A portable elliptical dichroism spectrometer targeting secondary structural features of tumorous protein for pancreatic cancer detection," *Biosensors and Bioelectronics*, vol. 222, p. 114934, 2023.

[65] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[66] Y.-Y. Song and L. Ying, "Decision tree methods: applications for classification and prediction," *Shanghai archives of psychiatry*, vol. 27, no. 2, p. 130, 2015.

[67] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[68] R. Llewellyn, D. Ronning, J. Ouzman, S. Walker, A. Mayfield, and M. Clarke, "Impact of weeds on australian grain production: the cost of weeds to australian grain growers and the adoption of weed management and tillage practices," *Grains Research and Development Corporation and the Commonwealth Scientific and Industrial Research Organisation*, vol. 2016, 2016.

[69] Y. Gharde, P. Singh, R. Dubey, and P. Gupta, "Assessment of yield and economic losses in agriculture due to weeds in india," *Crop Protection*, vol. 107, pp. 12–18, 2018.

[70] A. N. Veeranampalayam Sivakumar, J. Li, S. Scott, E. Psota, A. J. Jhala, J. D. Luck, and Y. Shi, "Comparison of object detection and patch-based classification deep learning models on mid-to late-season weed detection in uav imagery," *Remote Sensing*, vol. 12, no. 13, p. 2136, 2020.

[71] S. Christensen, H. T. Søgaard, P. Kudsk, M. Nørremark, I. Lund, E. S. Nadimi, and R. Jørgensen, "Site-specific weed control technologies," *Weed Research*, vol. 49, no. 3, pp. 233–241, 2009.

[72] A. I. De Castro, J. Torres-Sánchez, J. M. Peña, F. M. Jiménez-Brenes, O. Csillik, and F. López-Granados, "An automatic random forest-obia algorithm for early weed mapping between and within crop rows using uav imagery," *Remote Sensing*, vol. 10, no. 2, p. 285, 2018.

[73] R. Sapkota, J. Stenger, M. Ostlie, and P. Flores, “Towards reducing chemical usage for weed control in agriculture using uas imagery analysis and computer vision techniques,” *Scientific Reports*, vol. 13, no. 1, p. 6548, 2023.

[74] F. López-Granados, J. Torres-Sánchez, A. Serrano-Pérez, A. I. de Castro, F.-J. Mesas-Carrascosa, and J.-M. Pena, “Early season weed mapping in sunflower using uav technology: variability of herbicide treatment maps against weed thresholds,” *Precision agriculture*, vol. 17, pp. 183–199, 2016.

[75] B. Scherrer, J. Sheppard, P. Jha, and J. A. Shaw, “Hyperspectral imaging and neural networks to classify herbicide-resistant weeds,” *Journal of Applied Remote Sensing*, vol. 13, no. 4, pp. 044 516–044 516, 2019.

[76] D. Liu and F. Xia, “Assessing object-based classification: advantages and limitations,” *Remote sensing letters*, vol. 1, no. 4, pp. 187–194, 2010.

[77] L. P. Osco, Q. Wu, E. L. de Lemos, W. N. Gonçalves, A. P. M. Ramos, J. Li, and J. M. Junior, “The segment anything model (sam) for remote sensing applications: From zero to one shot,” *International Journal of Applied Earth Observation and Geoinformation*, vol. 124, p. 103540, 2023.

[78] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo *et al.*, “Segment anything,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4015–4026.

[79] M. Pugliatti and M. Maestrini, “Small-body segmentation based on morphological features with a u-shaped network architecture,” *Journal of Spacecraft and Rockets*, vol. 59, no. 6, pp. 1821–1835, 2022.

[80] N. Rai, M. V. Mahecha, A. Christensen, J. Quanbeck, Y. Zhang, K. Howatt, M. Ostlie, and X. Sun, “Multi-format open-source weed image dataset for real-time weed identification in precision agriculture,” *Data in Brief*, vol. 51, p. 109691, 2023.

[81] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer, 2015, pp. 234–241.

[82] Z. Zhang, Q. Liu, and Y. Wang, “Road extraction by deep residual u-net,” *IEEE Geoscience and Remote Sensing Letters*, vol. 15, no. 5, pp. 749–753, 2018.

[83] N. Enshaei, S. Ahmad, and F. Naderkhani, “Automated detection of textured-surface defects using unet-based semantic segmentation network,” in *2020 IEEE International Conference on Prognostics and Health Management (ICPHM)*. IEEE, 2020, pp. 1–5.

[84] A. H. Alwan, S. A. Ali, and A. T. Hashim, “Medical image segmentation using enhanced residual u-net architecture.” *Mathematical Modelling of Engineering Problems*, vol. 11, no. 2, 2024.