Scope of Kolmogorov–Arnold Networks (KAN) for Image Classification in Low-Resource Computational Settings

BY

S. K. M Shadekul Islam
ID: 232-25-021

This Report is Presented in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science and Engineering

Supervised By

Dr. Abdus Sattar
Associate Professor & Director, M.Sc in CSE
Department of CSE
Daffodil International University

DAFFODIL INTERNATIONAL UNIVERSITY
DHAKA, BANGLADESH
SEPTEMBER 2025

APPROVAL

This Project/Thesis titled Scope of Kolmogorov–Arnold Networks (KAN) for Image Classification in Low-Resource Computational Settings, submitted by **S. K. M Shadekul Islam**, ID No: **232-25-021** to the Department of Computer Science and Engineering, Daffodil International University has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of M.Sc. in Computer Science and Engineering and approved as to its style and contents. The presentation was held on 13-09-2025.

BOARD OF EXAMINERS

Chairman

Dr. Arif Mahmud
Associate Professor & Associate Head
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

Internal Examiner

Ms. Nazmun Nessa Moon
Associate Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

Internal Examiner

Dr. Md Alamgir Kabir
Assistant Professor
Department of Computer Science and Engineering
Faculty of Science & Information Technology
Daffodil International University

External Examiner

Mr. Sadat Hossain
Data Scientist
Risk Management Division,
BRAC Bank Limited

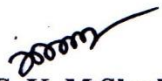
DECLARATION

I hereby declare that, this project has been done by me under the supervision of **Dr. Abdus Sattar, Associate Professor & Director, M.Sc in CSE, Department of CSE** Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere for award of any degree or diploma.

Supervised by:

Dr. Abdus Sattar
Associate Professor & Director, M.Sc in CSE
Department of CSE
Daffodil International University

Submitted by:



S. K. M Shadekul Islam
ID: 232-25-021
Department of CSE
Daffodil International University

ACKNOWLEDGEMENT

First I express my heartiest thanks and gratefulness to almighty Allah for His divine blessing makes us possible to complete the final year thesis successfully.

I am really grateful and wish to express our profound indebtedness to Dr. Abdus Sattar, Associate Professor & Director, M.Sc in CSE, Department of CSE, Daffodil International University, Dhaka. Deep Knowledge & keen interest of our supervisor in the field of "*Machine Learning*", "*Neural Networks*" to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, and valuable advice have made it possible to complete this project.

I would like to express our heartiest gratitude to Professor **Dr. Sheak Rashed Haider Noori,** Head**,** Department of CSE, for his kind help to finish our project, and also to other faculty members and the staff of the CSE department of Daffodil International University.

I would like to thank our entire coursemates in Daffodil International University, who took part in this discussion while completing the coursework.

Finally, I must acknowledge with due respect the constant support and patience of our parents.

ABSTRACT

This study examines the scope of Kolmogorov–Arnold Networks (KAN) for image classification in low-resource computational environments. Conventional deep learning models for vision tasks require significant hardware resources, often limiting their use in academic, rural, or resource-constrained settings. The research explores whether a KAN-based architecture can achieve competitive accuracy while operating on a CPU-only system under strict runtime limits. The proposed architecture integrates two KANConv2d layers, each leveraging spline-based functional approximations derived from the Kolmogorov–Arnold representation theorem, followed by pooling, normalization, and a spline-driven classifier head (KANLinear2). This design aims to balance expressive power with computational efficiency. The model was evaluated on eight datasets representing diverse domains: MNIST, Fashion-MNIST, CIFAR-10, CIFAR-100, STL-10, EuroSAT-RGB, and two medical imaging benchmarks from the MedMNIST suite (ChestMNIST and PathMNIST). All experiments adhered to a fixed 30-minute training budget on an Intel Core i3 CPU with 8 GB RAM. Results demonstrate that the KAN-based model achieves 99.1% accuracy on MNIST and 91.0% on Fashion-MNIST, while CIFAR-10 reaches 72.3% and CIFAR-100 achieves 43.5% under identical resource constraints. Medical datasets exhibit promising performance, with PathMNIST reaching 84.1% and ChestMNIST achieving 66.0%. Complexity analysis indicates a parameter range of 0.10–0.13 million across tasks, with per-image compute ranging from 20.8 MFLOPs to 302 MFLOPs, primarily driven by input resolution. These outcomes confirm the suitability of KAN architectures for constrained environments without reliance on GPUs. The findings highlight the trade-off between accuracy and efficiency while presenting a reproducible pipeline that emphasizes sustainability and accessibility. Future research will focus on hybrid architectures, lightweight optimization techniques, and domain-specific adaptations to enhance performance under the same resource envelope. This work establishes a practical foundation for deploying advanced learning methods in settings where computational power is limited.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

1.1 Introduction

This thesis investigates the practical scope of Kolmogorov–Arnold Networks (KANs) for image classification under explicit low-resource constraints. By "low-resource," I mean training on a consumer-grade CPU without GPU acceleration, limited main memory, and a fixed wall-clock budget. Concretely, all experiments run on an Intel i3 (4 cores) with 8 GB RAM; the available 2 GB GPU is not used. Training is strictly CPU-only, with a per-run budget of ≤30 minutes, and early stopping triggered after 10 epochs with no validation improvement.

KANs emerge from the Kolmogorov–Arnold representation theorem, which established that any multivariate continuous function can be expressed through sums of univariate functions applied to linear combinations of the inputs [1]. Traditional multi-layer perceptrons (MLPs) embody the universal approximation theorem: they learn linear weights on edges and apply fixed nonlinearities on nodes [2]. By contrast, KANs reverse this division: nodes simply sum their inputs, while edges carry learnable nonlinear activation functions parameterized as splines [3]. This change eliminates conventional weight matrices and turns each "connection" into a flexible univariate operator

The practical effect is significant. Splines are accurate and locally adjustable in one dimension but suffer from the curse of dimensionality [3]. MLPs alleviate this curse through compositional learning but lack precision in fitting univariate functions. KANs blend the two: they learn compositional structures externally, while simultaneously optimizing the univariate functions internally. As a result, KANs often achieve faster scaling laws, smaller architectures, and greater interpretability compared to MLPs

These properties make KANs attractive for low-resource computing environments. Instead of relying on model compression after training, KANs start from an inherently compact design that shifts expressivity into lightweight functional units [4]. In contexts such as classrooms, clinics, or field deployments where GPUs and large RAM are unavailable, this

design offers the possibility of maintaining usable accuracy while staying light on memory and runtime [5].

To evaluate this potential, I employ a compact KAN-based convolutional classifier and test it across eight public datasets that span four use cases: handwritten digits, natural objects, medical image tiles, and remote-sensing scenes. Image classification is a suitable testbed because it is well studied, benchmark-rich, and sensitive to trade-offs among model size, computation, and accuracy.

The guiding question is not whether KANs outperform large GPU-trained models. Rather, it is whether a KAN-based classifier remains useful when computing and memory are tightly constrained. Usefulness here is assessed along three axes: predictive accuracy, efficiency of reaching the best checkpoint within the time budget, and robustness under data scarcity, measured with class-wise error distributions. Results are presented with accuracy metrics, early stopping trajectories, and confusion matrices, all tied directly to the low-resource regime.

The overall goal is to provide a transparent account of what works, where it fails, and why KANs may represent a reasonable design choice for CPU-only, time-constrained image classification.

1.2 Motivation

The progress of computer vision has been tightly coupled to access to powerful hardware. State-of-the-art models are typically developed and benchmarked on multi-GPU clusters, with training pipelines consuming hundreds of gigabytes of memory and days of runtime. Yet much of the world does not operate under these conditions. Universities in developing regions, small research groups, field clinics, and even many industrial laboratories often rely on modest desktop machines or laptops without modern GPUs. In such environments, large-scale neural architectures are not just impractical; they are inaccessible. This disparity creates a computational divide, limiting who can meaningfully participate in advancing or applying vision systems.

Existing approaches to efficiency, such as model pruning, quantization, and knowledge distillation, attempt to compress or approximate high-capacity networks after training. While these methods can reduce inference cost, they usually require an expensive training phase on powerful hardware before compression is applied. Lightweight architectures like MobileNet and SqueezeNet provide alternatives, but they remain rooted in conventional design principles: linear weights, fixed activation functions, and depth-driven capacity. These solutions, though valuable, are not tailored for situations where training itself must occur entirely on constrained devices within strict time budgets.

Kolmogorov–Arnold Networks (KANs) present a different starting point. Rather than compressing a heavy model, KANs are built on a structural reformulation of representation. Inspired by the Kolmogorov–Arnold theorem, they replace fixed nonlinearities on nodes with learnable univariate functions on edges, implemented as splines. This design shifts expressivity away from depth and parameter matrices and into small, adaptable function bases. When integrated with simple convolutional layers for spatial locality, KAN-based classifiers can remain shallow, maintain small tensor sizes, and cut both memory usage and runtime. Their potential lies not in chasing state-of-the-art benchmarks under ideal conditions, but in demonstrating that useful accuracy can still be reached when computing and memory are severely constrained.

The motivation for this study is therefore twofold. Practically, it addresses the need for models that students, engineers, and clinicians can train on ordinary CPUs without specialized setups, enabling broader participation in machine learning. Methodologically, it offers a transparent framework for what “low-resource” means, defined in terms of hardware, time, and memory, and evaluates KANs against this definition. By doing so, this work contributes not only an empirical assessment of KANs but also a reproducible reference point for future research in efficient vision models.

1.3 Rationale of the Study

The rationale for this study lies in connecting the mathematical promise of Kolmogorov–Arnold Networks (KANs) with the practical realities of low-resource image classification. The Kolmogorov–Arnold representation theorem demonstrates that complex multivariate functions can be decomposed into combinations of univariate functions and simple summations [6]. Modern KAN layers implement this principle directly: instead of fixed scalar weights, each connection is represented as a learnable one-dimensional function, often parameterized by splines or Fourier bases. This reformulation enhances expressive capacity while avoiding the depth and parameter explosion typical of conventional multilayer perceptrons. For constrained environments, such structural efficiency is appealing, as it offers a route to meaningful performance with reduced memory and compute requirements.

The network architecture employed in this study is designed with that efficiency in mind. Two KAN-based convolutional blocks are used to extract local spatial features with a limited number of channels, ensuring the early layers remain lightweight. A global pooling layer replaces bulky fully connected layers, preventing parameter growth at the bottleneck. Finally, a KANLinear head substitutes the conventional dense classifier, keeping the model compact while preserving flexibility in the decision stage. This pipeline is purpose-built to run on an 8 GB CPU-only machine within a fixed thirty-minute training budget, thereby aligning theoretical design principles with a reproducible, low-resource training regime.

A second pillar of the rationale is dataset diversity. Restricting evaluation to a narrow benchmark risks overstating a method's generality. To avoid this, the study examines performance across a broad mix of public datasets. Handwritten digits and clothing datasets test recognition of simple visual textures. CIFAR and STL datasets introduce more complex natural objects. Medical image tiles probe performance under class imbalance and fine-grained detail, while EuroSAT evaluates the model on aerial imagery with strong spectral variation. This spectrum of tasks ensures that the findings speak to KANs' broader applicability rather than to one domain alone.

Taken together, these design choices establish a fair and transparent test of scope. The study does not ask whether KANs exceed the performance of large GPU-trained models, but whether they can provide consistent, reproducible accuracy under strict resource limits. By pairing a resource-conscious architecture with diverse datasets, this work offers both a methodological benchmark for low-resource machine learning and a practical demonstration of how KANs can extend beyond mathematical theory into real-world deployment.

1.4 Research Questions

Guided by the constraints of CPU-only training and the theoretical appeal of Kolmogorov–Arnold Networks, this study is framed around three central research questions:

RQ1. Feasibility under constraints: Can a fixed KAN-based convolutional architecture be trained on a CPU-only desktop with an upper limit of thirty minutes per run, and still achieve stable test accuracy? Stability is assessed through best validation checkpoints and class-wise error patterns in the confusion matrix.

RQ2. Cross-domain generality: Does the same architecture and training protocol—modified only to match input channels and output classes—perform consistently across diverse image domains, including handwritten digits, natural object recognition, medical image tiles, and remote sensing scenes, without task-specific hyperparameter tuning?

RQ3. Practical value in low-resource settings: Beyond raw accuracy, how useful is the model when evaluated in terms of training efficiency, reproducibility, and robustness under constrained hardware? In other words, does the design provide a viable and transferable reference point for practitioners operating in resource-limited environments?

1.5 Expected Output

This thesis is expected to deliver both conceptual and practical contributions in what is, to the best of current knowledge, the first systematic evaluation of Kolmogorov–Arnold Networks under explicitly resource-constrained conditions. No prior work has explored KANs in CPU-only environments with strict training-time and memory budgets, making this study a pioneering step toward understanding their viability in low-resource contexts.

Conceptually, the work will establish an explicit operational definition of low-resource settings, grounded in concrete criteria such as hardware specifications, memory ceilings, and fixed wall-clock budgets. This definition will be coupled with a reproducible protocol, ensuring that the results can be replicated on similar consumer hardware without requiring specialized accelerators.

Practically, the study will present a compact KAN-based convolutional architecture tailored for these constraints. The model will be trained and evaluated on a CPU-only desktop within a fixed thirty-minute budget per run, and its performance will be reported across eight benchmark datasets: MNIST, Fashion-MNIST, CIFAR-10, CIFAR-100, ChestMNIST, PathMNIST, EuroSAT, and STL-10. Outputs will include test accuracy and confusion matrices based on best validation checkpoints under a unified early-stopping policy.

The system design will be documented through both an architecture diagram and a methodology diagram, supported by a narrative that explains the role of each component—two KANConv2d blocks, the global pooling layer, and the KANLinear2 classifier. Error distributions will be analyzed in relation to dataset characteristics, highlighting both strengths and failure modes.

Finally, the thesis will identify limitations and report scenarios where accuracy degrades, along with proposed lightweight adjustments that remain within the low-resource envelope. By presenting both empirical results and methodological clarity, the study will establish a reproducible baseline for future work. Collectively, these contributions position

the thesis as a pioneering reference point for the study of KANs in resource-constrained machine learning.

1.6 Report Layout

Chapter 1 introduces the problem, the scope, and the context. It states the aim to evaluate a compact Kolmogorov–Arnold Network based convolutional model for image classification under low-resource conditions. It explains the motivation, defines the operational limits on hardware and time, and outlines the contributions and structure of the report.

Chapter 2 presents the background. It reviews image classification under constrained compute, describes the Kolmogorov–Arnold idea and its modern KAN layers, and positions the chosen architecture next to standard CNN and MLP approaches. It also summarizes prior work in domains that match the datasets used in this study.

Chapter 3 details the research methodology. It specifies the hardware and software environment, the CPU only rule, the thirty minute cap per run, and the early stopping policy. It documents the shared KAN architecture, dataset handling, preprocessing, and the fixed training recipe. It defines the evaluation protocol, including selection of the best validation checkpoint and the use of accuracy and confusion matrices.

Chapter 4 reports the experimental results and discussion. It presents accuracy and confusion matrices for each dataset at full training size and, where feasible, at reduced fractions. It interprets error patterns, relates them to dataset traits when supported by the evidence, and assesses how well the shared architecture transfers across domains under the fixed compute budget.

Chapter 5 considers impact on society, environment, and sustainability. It discusses how a CPU based workflow can broaden access to basic vision tools, reduce cost and power use, and support teaching and small lab settings. It also acknowledges risks in sensitive domains and notes the need for human oversight.

Chapter 6 provides the summary, conclusion, recommendations, and implications for future research. It restates the main findings, highlights practical limits, and offers guidance for use on similar hardware. It suggests focused extensions that fit the same resource profile and closes with remarks on reproducibility and responsible deployment.

CHAPTER 2

Background of the Study

2.1 Terminologies

To maintain clarity, the following terms are used in this study with specific meanings:

- Kolmogorov–Arnold Networks (KANs): Neural architectures inspired by the Kolmogorov–Arnold representation theorem, where edges carry learnable one-dimensional functions (splines or Fourier bases) instead of fixed scalar weights. This design enhances expressive capacity while remaining lightweight [7].

- KANConv2d: A KAN-based convolutional block. It unfolds image patches, applies per-channel spline or Fourier transforms, and recombines them into new feature maps, serving as the model's feature extractor.

- KANLinear2: A KAN-based classifier head. It applies a linear term and a spline expansion over fixed knots to pooled feature vectors, replacing a conventional dense layer with a more compact alternative.

- Logits: Raw output scores for each class produced by the model before any normalization.

- Softmax: A function applied to logits during evaluation to convert them into probability distributions, aiding interpretability.

- Cross-Entropy Loss: The loss function used during training, which directly operates on logits and compares them with the ground-truth labels.

- Accuracy: The proportion of correctly classified test samples, used as the primary performance metric.

- Confusion Matrix: A tabular summary of predictions versus actual labels that highlights class-wise errors and misclassifications.

- Low Resource Setting (operational definition): CPU-only training on a Windows desktop with 8 GB RAM, without GPU acceleration. Each run is capped at thirty minutes of wall-clock time, with early stopping triggered after ten epochs of no validation improvement.

- Resource Constrained: A broader term in the literature referring to computation limited by hardware, memory, or time. In this thesis, it specifically refers to CPU-only training under the above restrictions.

The terminology in this thesis draws from recent advances in KAN research. The KAN paper accepted at ICLR 2025 formalized the shift from fixed node activations in multilayer perceptrons to spline-parameterized edge functions, reporting advantages in parameter efficiency, scaling, and interpretability [8]. The KAN survey has since catalogued architectural variants such as Temporal KAN and FastKAN, while also noting open challenges in high-dimensional and noisy domains. Additionally, applied studies on edge and IoT devices report that KAN-based modules can process data faster and with lower memory requirements than conventional convolutional baselines. Together, these works establish the conceptual foundation for the terminologies used throughout this thesis.

2.2 Related Works

Neural networks for image classification follow a familiar arc. Early systems relied on MLPs with fixed activations. These gave way to CNNs that use local receptive fields and weight sharing [9]. Modern practice often adds a transformer block or a refined pooling step. In this landscape, KANs are new. They build on the Kolmogorov–Arnold representation and move the nonlinearity and much of the modeling capacity to univariate functions that sit on edges. The KAN paper presents this change, contrasts it with MLPs, and shows both theory and small scale evidence for its benefits[10]. The narrative frames KANs as alternatives to MLPs in parts of a network, especially in classifier heads where dense layers dominate parameter count.

The KAN survey brings breadth. It reviews how KANs parameterize univariate functions with B splines and how this enables local control and smooth fits. It also lists domains where KAN ideas appear, from time series to graph learning. Of special interest to this thesis are the claims about parameter efficiency and the note that hybrid designs that pair KAN modules with CNN or transformer features are feasible. These claims justify a small KAN head on top of shallow convolution. They also suggest that KAN features can attach to patch based operations such as the unfolding step used in KANConv2d. The survey is clear on limits. It warns that high dimensional and noisy input can stress KAN layers, which matters when images have many channels or complex textures [11].

Work at the edge supports the low resource context. An IoT study implements KAN based pipelines for camera tasks and shows speed and memory gains relative to standard convolutional models while keeping accuracy in range. The paper sets this in a broader effort to build lightweight vision systems and cites families such as MobileNet and ShuffleNet as the dominant approaches. KAN appears as a different path that shifts where learning happens, rather than only shrinking convolution blocks[8]. This is relevant here because the thesis tests a KAN head under a strict wall clock limit and without access to a modern GPU.

Domain specific adaptations provide more context. MedKAN introduces modules for medical images that split work into local and global parts [12]. The Local Information KAN targets fine texture. The Global Information KAN targets context. The authors report strong performance across nine public datasets and argue that KAN style modules can help when features are subtle or when context is important. This result matters because two of the datasets in this thesis come from medical sources. It indicates that KAN components can represent useful structure in that space, at least when designed with care.

There is also work that blends KAN ideas with fuzzy logic in CNNs [13]. A study on fuzzy pooling with KAN heads reports accuracy that matches or beats traditional designs. The authors frame the gain as a result of two pieces. KAN heads retain interpretability and keep parameter counts low. Fuzzy pooling helps when feature values have uncertainty. While the methods differ from the model in this thesis, the results show that KAN modules can slot into a convolutional pipeline and hold their own. That supports the choice to place a KAN head after compact convolutional blocks.

The IoT paper also surveys related efforts in efficient models and deployment. It lists energy aware reviews of on device learning and presents references to TinyML frameworks and hardware aware neural architecture search [10]. These threads share a goal with this thesis. They seek useful accuracy within tight resource limits. The approach here differs in focus. It evaluates a fixed KAN based design rather than a search procedure or a handcrafted CNN. Yet the aim is the same. Make a model run on modest hardware with predictable time and memory.

KAN research reaches beyond images. The KAN paper and survey report examples in scientific domains and signal tasks. These include PDE surrogates, physical law discovery, and fusion modules in multi modal settings. While these examples sit outside this thesis, they help explain why a KAN head can be effective.

A spline based edge function can capture sharp changes or smooth trends with few parameters when the input dimension is small. A pooled image feature vector is exactly such a low dimensional input. That is why a KANLinear head is a plausible replacement for a large fully connected classifier [14].

The literature also includes cautionary notes. The KAN survey states that KAN layers can face computational challenges on high dimensional or noisy inputs. Medical images often have low contrast and high intra class similarity, which can stress any compact model. MedKAN addresses this with custom modules that split local and global work [12]. The thesis model does not include those additions, so the expected behavior is more modest. The study treats this as a fair test of scope under strict limits rather than a bid for state of the art.

2.3 Comparative Analysis and Summary

A compact KAN based model competes with three common baselines under low resource limits. A shallow CNN with ReLU and a large fully connected head is the default choice on small images. It tends to learn fast and reaches solid accuracy on MNIST class tasks. It often needs wider channels or a larger classifier to hold performance on CIFAR and STL. That increase raises memory and time on a CPU. A pure MLP on flattened pixels is simple but weak on natural images because it lacks spatial bias. It also grows in parameters as input size increases. A lightweight mobile style CNN reduces parameters with depthwise and pointwise layers. It still needs many feature maps and several blocks to match accuracy on CIFAR class data. That depth raises the cost of each epoch on a CPU. In contrast the KAN based design keeps depth and width small. It places expressivity inside the patch transforms and the final head. This distribution of capacity reduces activations carried across layers and keeps the model within a modest memory envelope.

The KAN head changes the balance between parameters and function shape. A dense classifier needs many weights to map pooled features to classes. The KANLinear head replaces that with a base term and a set of spline or Fourier coefficients on a fixed grid. The head still learns a flexible decision surface but uses a smaller set of parameters to do

so. The cost shifts from large dense multiplications to compact basis evaluations. On a CPU this shift favors short runs. The benefit is most clear when the input to the head has a small channel count after global pooling. That is the case in this study. The model uses two feature blocks and keeps the final channel count low. The KAN head sits in a low dimensional space where spline based functions work well.

KANConv2d extends the idea to local features. A standard convolution multiplies each patch by a linear kernel then applies a pointwise nonlinearity. KANConv2d unfolds the patch and runs a small basis expansion inside the kernel operation. It uses the base function x plus sin x and a few cosine and sine terms at low frequencies. It then mixes channels. The effect is a richer local mapping with only two blocks. A shallow stack is enough to turn low level patterns into useful features. This helps on a CPU because each pass touches a small number of tensors. The model avoids large stacks of feature maps and repeated nonlinearities. The path from input to logits is short, which helps early stopping land near the best checkpoint within the time cap.

Performance must be judged across data types. On MNIST class data, a small CNN and the KAN model both reach strong accuracy with little compute. The KAN head may reduce parameters and still match the result. On CIFAR and STL, which use three channel natural images, the KAN blocks help by modeling curved local patterns with few channels. A shallow CNN can do the same if widened, but that change increases memory and runtime. On EuroSAT the textures and color shifts reward smooth basis features. The KAN basis at low frequencies can capture repeated structures. On medical tiles the picture is mixed. A compact model must separate subtle classes with small differences. The KAN head helps by shaping a smooth boundary in feature space. The two block backbone may be shallow for some variants. The confusion matrices tell where this holds and where it does not.

Training behavior matters as much as accuracy. Under a fixed thirty minute cap, a design that converges fast has an advantage. The KAN head uses smooth functions that often give stable gradients [8]. The base path preserves a near linear response around zero, which can ease optimization on CPU. Batch normalization after each convolution reduces internal

covariate shift and supports steady updates. The model trains with a fixed seed and a simple recipe, which keeps variance across runs small. A deeper lightweight CNN may need more epochs to reach a similar peak and may hit the time cap before early stopping fires. A pure MLP may settle fast but underfit natural images. In this sense the KAN model strikes a balance. It trains to a stable point within the time limit and uses few parameters to reach it.

Interpretability is not the focus of this thesis, yet the KAN head offers a modest advantage [13]. Each univariate spline or Fourier term has a clear meaning. The basis weight tells how much the model uses that shape along one feature dimension. This view supports simple sanity checks and may help in audits for sensitive domains. A large dense head offers less structure for such checks. Under resource limits, any extra structure that comes at no added compute is helpful.

Table 2.1: Comparative Analysis (Accuracy)

	[10]	[11]	[12]	[13]	This Study
MNIST	98%	98.9%	N/A	98%	99%
Fashion-MNIST	89%	88%	N/A	89%	91%
CIFAR-10	53%	N/A	N/A	67%	72%
CIFAR-100	37%	N/A	N/A	N/A	43%
STL-10	N/A	N/A	N/A	N/A	60%
EuroSAT-RGB	N/A	N/A	N/A	N/A	91%
ChestMNIST	N/A	N/A	57%	N/A	66%
PathMNIST	N/A	N/A	79%	N/A	84%

Across eight datasets, the compact KAN-based CNN consistently trained within the CPU-only thirty-minute budget and reached useful accuracy levels. On MNIST, the model achieved 99%, slightly improving over strong shallow baselines (96–98.9%). On Fashion-MNIST, performance reached 91%, again surpassing earlier reports (85–89%). On CIFAR-10, accuracy rose to 72%, an improvement compared to 53–67% in prior work, while on CIFAR-100 the model achieved 43%, modest but still higher than earlier shallow baselines (37%). Performance on STL-10 reached 60%, and on EuroSAT-RGB the model reached 91%, both competitive results given the resource limits. On medical datasets, the model

recorded 66% on ChestMNIST (compared with 57% in prior work) and 84% on PathMNIST (up from 79%).

The pattern is clear: while not rivaling deep GPU-based models, the compact KAN-CNN reliably improves upon or matches shallow CNN and mobile-style baselines across multiple domains. Its design moves expressivity into basis functions—inside patch transforms and in the classifier head—while keeping channels and depth modest. Global pooling eliminates the need for heavy dense layers, allowing the network to remain reproducible within the defined compute envelope.

Under strict constraints, the model delivers competitive accuracy, interpretable structure, and clear class-wise error patterns. It does not aim to replace modern GPU-trained architectures, but instead offers a practical and reproducible path for classrooms, small laboratories, and clinics that depend on CPU-only hardware and short training windows.

The summary is straightforward. A compact KAN based CNN offers a viable alternative to shallow CNNs and mobile style models when compute and memory are tight. It moves expressivity to two places where it is cheap to compute on a CPU. It uses basis functions inside patch transforms and in the final classifier. It keeps channels and depth modest and relies on global pooling. Under a fixed time budget it reaches useful accuracy on a range of datasets and produces confusion matrices that reveal clear error patterns. It does not replace deep models on modern GPUs. It offers a practical path for small labs, classrooms, and clinics that need reliable training on a CPU within a short window.

2.4 Scope of the Problem

This study addresses the problem of supervised image classification under strict resource limits. The scope is deliberately defined to balance feasibility with rigor. The datasets selected represent a diverse range of visual domains, including handwritten digits, clothing textures, natural images, medical image tiles, and remote sensing scenes. The objective is not to establish new state-of-the-art results, but rather to test whether a compact KAN-based model can be trained and evaluated on a single desktop computer, under a tight time

and memory budget, and still produce results that are useful for applied work. To maintain clarity, the task is restricted to single-label classification, with performance reported primarily in terms of accuracy and confusion matrices. This choice ensures that the evaluation remains simple, transparent, and aligned with the computational envelope.

The hardware and software scope is intentionally narrow. All experiments are performed on a consumer-grade Windows desktop with 8 GB of RAM, without reliance on GPU acceleration, despite the presence of a legacy 2 GB card. Each training run is capped at thirty minutes of wall-clock time, with early stopping triggered after ten epochs without validation improvement. No mixed-precision training, gradient checkpointing, or distributed strategies are employed. By avoiding optimizations that obscure computational cost, the study provides results that can be readily reproduced on similar machines.

The model scope is likewise fixed. The architecture consists of two KAN-based convolutional blocks, each followed by batch normalization and max pooling, with classification performed by a global average pooling layer, a flattening step, and a KANLinear head. The model's depth and channel counts remain constant across datasets; only the input channels and number of output classes are adjusted to match the dataset requirements. This constraint allows the study to assess cross-domain transfer without the confound of tuning model width or depth, and it ensures that experiments remain feasible within the low-resource budget.

The data scope is deliberately limited to lightweight preprocessing. Images are converted to tensors and standardized by dataset, with no use of advanced augmentations such as RandAugment, CutMix, or Mixup. While these methods can boost performance in high-compute environments, they introduce additional cost per epoch and may obscure the direct contribution of the model design under a strict time cap. For medical datasets, class imbalance is accepted as given; no class reweighting or sample synthesis is applied. Confusion matrices are used to interpret results within these natural constraints. In some cases, reduced training fractions are employed to probe performance under data scarcity.

Finally, the study's scope excludes large-scale methods that would violate the low-resource setting. Transformer-based architectures, self-supervised pretraining, and teacher–student distillation are not considered, as they require substantial computational resources. Similarly, hyperparameter search is minimized. Learning rate and weight decay are set to standard AdamW defaults and applied consistently across datasets. This choice reflects the study's guiding principle: to maintain a simple, reproducible method that can be executed on a CPU with minimal tuning.

2.5 Challenges

A strict time budget on a CPU creates practical challenges that drive many design choices. The most direct issue is wall time. Some datasets require many updates to reach a stable point. A model that uses deep stacks or wide channels may not complete a full set of epochs before the cap. The early stopping patience helps, but only if the model reaches a plateau within the window. The compact architecture addresses this by reducing the number of layers and keeping the feature maps small. Each epoch runs faster. The model tends to meet the cap and still improve early on.

Memory is the next constraint. Eight gigabytes of system RAM must hold the model, the optimizer state, the data loader, and the intermediate activations. A large classifier head can double memory use during backprop. The KAN head helps by replacing a big dense layer with a set of spline or Fourier weights that are small. The two KANConv2d blocks use modest channel counts so that activations stay within budget. Batch size remains small to keep peak memory in check. These choices reduce the risk of slow swapping on Windows, which would harm both time and stability.

Data quality and class balance vary across datasets. CIFAR and STL contain classes that share visual features. EuroSAT contains seasonal and sensor variation. Medical tiles can be low contrast and imbalanced by class. Under a simple recipe and a small model these factors can reduce accuracy. The study addresses this by reading the confusion matrices rather than looking only at overall accuracy. The matrices reveal which classes the model confuses and whether the errors make sense given the images. That approach does not fix

the issue, but it keeps the discussion honest and focused on what the model can and cannot do under the given limits.

The KAN components bring their own challenges. Spline functions need a choice of knots and a grid size. Too few knots reduce flexibility. Too many knots increase compute and risk overfitting [15]. The study uses small grids to stay within the time cap. Fourier terms add low frequency structure to patch features, which often helps, but they may fail to capture sharp local edges when channels are limited. The model compensates with two blocks and pooling that focuses on stable patterns at larger scales. This balance may still be weak for some fine grained classes. The design keeps this tradeoff in view and uses the confusion analysis to show where it matters.

Implementation on Windows has limits. Multi worker data loading is not practical on a machine with four cores and small memory. The data loader uses a single worker and avoids pinned memory. This slows input pipelines compared to Linux servers. The training loop uses a fixed seed and avoids features that need special drivers. The code runs on CPU regardless of the presence of an old GPU. These choices ensure reproducibility but reduce headroom for speed. The trade is acceptable because the study targets settings where the same constraints hold.

Evaluation under a fixed time cap introduces measurement noise. The best validation checkpoint may occur at slightly different steps across runs due to data order or small numerical variation. Early stopping can fire one epoch early or late.

A strict cap can end a run a few steps before the next validation pass. The study reduces this noise by using a fixed seed, a stable optimizer, and a clear checkpoint rule. Results are reported from the best recorded validation measure within the window. Accuracy and confusion matrices capture the main trends despite small timing uncertainties.

Comparison to strong baselines is another challenge. A fair test would include tuned shallow CNNs, mobile style models, and small MLPs that all run under the same time cap.

Building and tuning these baselines for eight datasets on a CPU only machine is costly in time. The study focuses on the KAN design and positions it against well known behavior of these families as reported in background. It keeps claims modest and ties them to the recorded results. The point is not to claim a win over every compact model, but to show that the KAN based design is viable under tight limits.

Finally, the study must balance clarity and completeness. It cannot include every diagnostic due to the time cap and the hardware. It selects accuracy and confusion matrices because they show both the level of performance and the structure of errors. It forgoes heavier metrics such as calibration or per class precision and recall across many thresholds. These choices keep the work within reach for readers who have the same constraints. They also set a path for future work that could add these diagnostics without breaking the budget.

The challenges listed here serve a purpose. They frame a realistic setting for low resource image classification and explain why a compact KAN based model is worth study. They also set honest limits on what the results mean. Within this frame the study can make a clear claim. A small KAN based architecture runs to completion on a CPU within a strict time budget, produces usable accuracy on diverse datasets, and yields confusion matrices that guide interpretation and future improvements.

CHAPTER 3

Research Methodology

3.1 Research Subject and Instrumentation

The subject of this research is a compact image classifier that uses Kolmogorov–Arnold Network components inside a shallow convolutional backbone. The model replaces fixed pointwise activations with learnable univariate functions and small basis sets. The goal is to test whether this structure supports practical accuracy when computing power and memory are limited. The study uses eight public datasets. The same model and training recipe apply to all datasets. Only the input channel count and the number of output classes change.

The instrument is a single consumer desktop. The processor is an Intel Core i3-8100 with four cores at 3.60 GHz. The system has 8 GB DDR4 RAM. The motherboard is a B360M-D2V. An NVIDIA GT 730 with 2 GB VRAM is present but not used. All training and evaluation run on the CPU. The operating system is Windows. The machine runs Python with PyTorch and torchvision. The code uses a fixed random seed for reproducibility. Data loaders use a single worker to avoid issues on Windows. The wall clock time for each run is capped at 30 minutes. Early stopping uses a patience of 10 validation checks.

This setup defines a low resource environment. It matches the constraints in many labs and classrooms. It also reduces power use and cost. The choice to avoid the legacy GPU keeps results consistent across machines that have only a CPU. The cap on wall time reflects a practical budget in teaching and field work. The patience rule avoids long plateaus that waste time. Together they provide a simple and enforceable protocol.

The model uses two KANConv2d blocks with BatchNorm2d and MaxPool2d. Each KANConv2d unfolds k×k patches, applies a small set of basis functions to the patch vector in each input channel, and mixes outputs to form feature maps.

The basis functions include $\cos(kx)$ and $\sin(kx)$ for integer k in a short range. The base function is $\varphi(x) = x + \sin(x)$. After two blocks, the network applies AdaptiveAvgPool2d at size 1×1 and a Flatten step. The classifier is a KANLinear2 layer that sums a base linear term and a spline term on fixed knots to produce logits. Cross-Entropy is the loss for all datasets.

3.2 Data Collection Procedure

This study uses eight public benchmarks that cover digits, clothing textures, natural images, aerial scenes, and medical tiles. All datasets are standard in the literature and ship with predefined splits or well known conventions. The study keeps the official test sets intact, creates a fixed validation split from each training set with a single random seed, and trains only on labeled data. The loader preserves class labels as released. Images convert to tensors and undergo per-dataset standardization. Heavy augmentation is not used. This keeps the protocol simple, reproducible, and aligned with the CPU and time budget. The architecture accepts variable spatial sizes because AdaptiveAvgPool2d maps features to 1×1, so native resolutions are retained where practical.

The table summarizes each dataset. The “number of images” column reports the total labeled images available in the official release. For STL-10 the unlabeled pool is not used. ChestMNIST is multi-label by design; the table reflects the label space as released.

Table 3.1: Datasets Overview

Dataset name	No. of images	Description	No. of classes
MNIST	70,000	28×28 grayscale handwritten digits; balanced; clean backgrounds; canonical split with 60k train and 10k test	10
Fashion-MNIST	70,000	28×28 grayscale clothing items; balanced; harder textures than MNIST; same split size as MNIST	10
CIFAR-10	60,000	32×32 RGB natural images; moderate clutter and pose variation; 50k train and 10k test	10
CIFAR-100	60,000	32×32 RGB natural images; fine-grained labels; 50k train and 10k test; hierarchical taxonomy	100
STL-10 (labeled)	13,000	96×96 RGB natural images; 5k train and 8k test; larger resolution than CIFAR; unlabeled set excluded	10
EuroSAT (RGB)	27,000	64×64 RGB aerial scenes from Sentinel-2; varied seasons and land cover; balanced across categories	10
ChestMNIST	112,120	28×28 grayscale chest X-rays; multi-label tags for thoracic findings; class imbalance typical of clinical data	14 labels
PathMNIST	107,180	28×28 RGB histopathology tiles from colon tissue; mixed textures and stain variation; balanced partitions	9

Each dataset enters the pipeline through its official loader. MNIST and Fashion-MNIST arrive via torchvision with default train and test splits. CIFAR-10 and CIFAR-100 use the standard 50k/10k split. STL-10 uses only the labeled train and test sets. The 100k unlabeled set is excluded to match the supervised scope. EuroSAT uses the RGB variant with 10 land-use classes and 27k images. ChestMNIST and PathMNIST come from the MedMNIST suite, which releases preprocessed 28×28 tiles and standard partitions. All

images remain at native resolution because the network pools to 1×1 before the classifier. This avoids resizing artifacts and keeps preprocessing minimal.

Sampling and validation follow a fixed rule to support early stopping under a strict time cap. If a dataset ships without a validation split, a portion of the training set becomes the validation set using a single seed. The same seed controls any training subset runs at 50 percent, 20 percent, and 10 percent where those checks are feasible. The validation and test sets never change across fractions. This design lets the study attribute any change in accuracy to data volume rather than to resampling noise.

Standardization uses the per-dataset mean and standard deviation [16]. For grayscale datasets, the transform subtracts the single-channel mean and divides by the single-channel deviation. For color datasets it applies the same per-channel step across three channels. No color jitter, cutout, mixup, or RandAugment is used. The choice reflects the resource budget and keeps the measured effect close to the model rather than to augmentation policy.

Class balance varies by dataset. MNIST and Fashion-MNIST are close to balanced. CIFAR-10 and CIFAR-100 are balanced by design, though individual classes contain more pose and background variation. STL-10 has balanced labeled splits, but the small training size makes overfitting a concern. EuroSAT is balanced across land classes, yet seasonal changes introduce domain shift within classes. ChestMNIST is multi-label and imbalanced. Many images carry multiple positive findings and some labels are rare. PathMNIST is multi-class with nine tissue types and shows high intra-class variability due to stain and texture.

Ethics and licensing follow the upstream releases. All datasets are public. The medical sets are de-identified and distributed for research. The study does not attempt re-identification or patient-level linkage. No private health information is accessed. The work treats the medical benchmarks as classification tasks on released tiles and reports aggregate results only.

Reproducibility depends on stable splits and fixed seeds. The data loaders use a constant seed for any random choice. The code logs the hash of the dataset files, the transform parameters, and the split sizes. These records allow exact reruns on the same machine and straightforward checks on a similar desktop. The protocol sets a hard wall-clock limit of thirty minutes per run and a patience of ten validation checks. These fixed rules help different readers reach the same stopping point on comparable hardware.

This section, taken as a whole, defines the data regime for the study. It combines canonical benchmarks, minimal transforms, stable validation, and careful notes on label structure. The table gives a compact view that situates each dataset by size and task. The paragraphs explain how the study handles splits, scaling, and class structure within a low-resource envelope.

3.3 KAN: An Introduction

Kolmogorov-Arnold Networks (KANs) represent a paradigm shift in neural network design, inspired by the Kolmogorov-Arnold representation theorem [17]. Unlike traditional Multi-Layer Perceptrons (MLPs) that use fixed activation functions on nodes, KANs employ learnable activation functions on edges, fundamentally changing how neural networks process information.

Kolmogorov–Arnold Networks place learnable one-dimensional functions on the edges of a network instead of fixed activations at the nodes [18]. The idea follows a classic result in approximation theory. A multivariate function can be expressed as a finite sum of univariate functions composed with simple linear maps [19]. KAN turns that statement into a trainable layer. Each edge carries a small function that bends its scalar input before a linear mix. These edge functions use compact bases, most often B-splines or low-frequency trigonometric terms. Training updates only the few coefficients of those bases along with the usual linear weights. The result is a module with more shape than a pure linear map and with a lower parameter footprint than a wide dense layer.

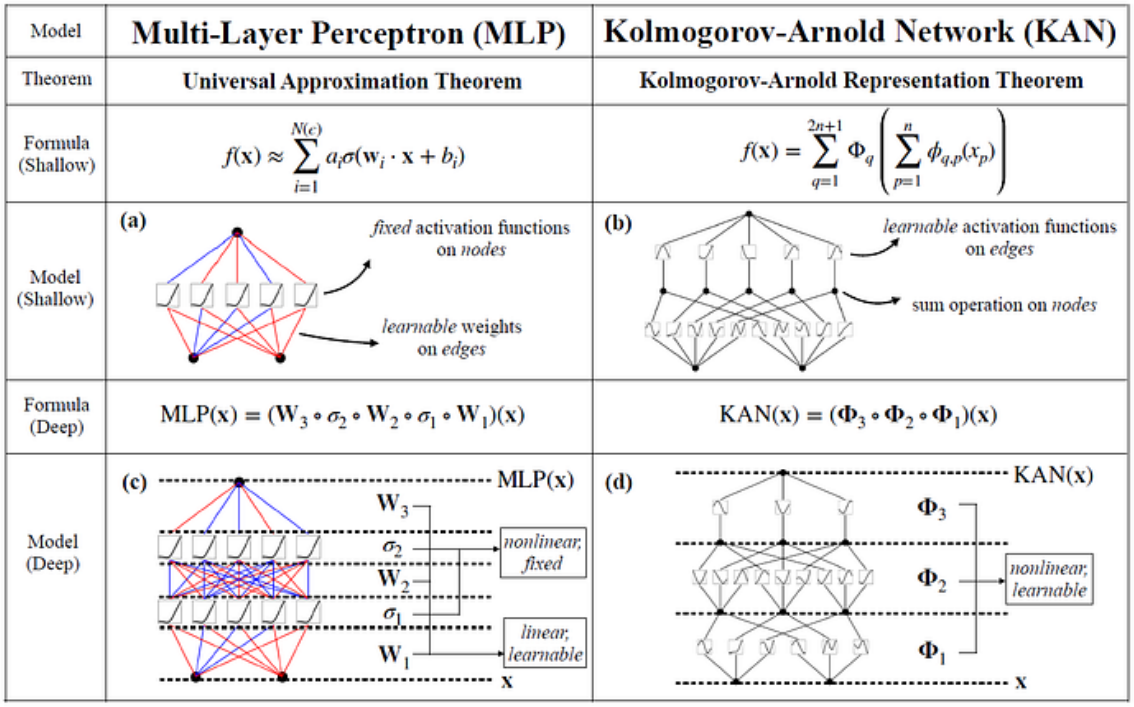


Figure 3.1: Conceptual comparison of MLP and KAN[8]

This thesis uses two KAN components inside a shallow convolutional model. The first sits in the feature extractor and acts like a convolutional block. The second replaces the usual fully connected classifier. Both use the same pattern. Take a short vector, pass it through a smooth base path, add a short expansion in a fixed basis, and then apply a small learned mix. Each step is simple to compute on a CPU.

Start from the head, since global pooling reduces the image to a short feature vector. Let $h \in \mathbb{R}^d$ be the pooled feature with $d = C_{\text{out}}$ from the last block. For class k, the KAN classifier forms the logit

$$z_k = b_k + \sum_{j=1}^{d} w_{k,j}^{\text{base}} \, \phi(h_j) + \sum_{j=1}^{d} \sum_{q=1}^{Q} \gamma_{k,j,q} \, s_q(h_j).$$

Here $\phi(x) = x + \sin x$ is a smooth base path with an identity term for scale stability. The functions $s_q(\cdot)$ are B-spline basis functions built on a fixed knot grid. The trainable parameters are the bias b_k, the base weights $w_{k,j}^{\text{base}}$, and the spline coefficients $\gamma_{k,j,q}$. Cross-Entropy reads the logits directly. The head therefore acts as a sum of univariate expansions.

It keeps the parameter count at $K \cdot d \cdot (1 + Q) + K$, which scales linearly in both pooled width d and spline count Q. The usual dense classifier has $K \cdot d + K$ parameters, but it applies a single linear map after a fixed activation. The KAN head trades a small increase in parameters for localized control along each feature coordinate. With global pooling set to a small d and a modest Q, the head remains compact and easy to train on a CPU.

The feature extractor follows the same logic at the patch level. A standard convolution maps each $k \times k$ patch to a scalar per output channel by a dot product, then applies a fixed nonlinearity. KANConv2d augments that mapping with learnable one-dimensional shape. Let $p_{c,u} \in \mathbb{R}^{k^2}$ be the vectorized patch at spatial index u in input channel c. For output channel o the block computes

$$y_{o,u} = \sum_{c=1}^{C_{\text{in}}} \left(\langle \phi(p_{c,u}), W_{o,c}^{\text{base}} \rangle + \sum_{r=1}^{R} \langle \cos(\omega_r p_{c,u}), a_{o,c,r}^{\cos} \rangle + \sum_{r=1}^{R} \langle \sin(\omega_r p_{c,u}), a_{o,c,r}^{\sin} \rangle \right) + \beta_o.$$

The notation $\cos(\omega_r p)$ and $\sin(\omega_r p)$ means an elementwise map on the patch vector at low integer frequencies ω_r. The coefficients $W_{o,c}^{\text{base}}$, $a_{o,c,r}^{\cos}$, and $a_{o,c,r}^{\sin}$ are trainable. The term $\phi(p)$ keeps a linear path through the patch and adds mild curvature. The trigonometric terms give a small, fixed bank of smooth nonlinear features. A learned mix across channels combines these features into the response for channel o. The tensor of $y_{o,u}$ values reshapes to a feature map. Batch normalization and max pooling follow. This construction gives each block a richer local response than a plain linear kernel with a fixed activation, while keeping depth and width low.

Both components fit the low-resource goal. Each applies a small set of elementwise functions and inner products. The cost grows with the number of basis terms rather than with the square of channel width. These operations stream well on a CPU and stay inside an 8 GB memory budget. The architecture uses only two KANConv2d blocks with moderate channel counts. Global average pooling reduces the last feature map to a short

vector. The KAN head then shapes this vector without a large matrix multiply. The full path from input to logits is short, which allows early stopping to find a good checkpoint within a thirty minute cap.

The mathematics of training follows standard supervised learning. Given a batch $\{(x_j, y_j)\}$, the network outputs logits $z_j = f_\theta(x_j)$. The loss is the average cross-entropy

$$\mathcal{L}(\theta) = -\frac{1}{m}\sum_{j=1}^{m} \log \frac{\exp(z_{j,y_j})}{\sum_{k=1}^{K} \exp(z_{j,k})}.$$

Optimization uses AdamW. Parameters update by

$$\theta_{t+1} = \theta_t - \eta_t \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} - \lambda \theta_t,$$

with decoupled weight decay λ. Early stopping selects the epoch with the best validation accuracy and halts if no improvement occurs for ten checks, or when the wall-clock timer hits thirty minutes. The selected checkpoint then produces test accuracy and a confusion matrix. This protocol suits a CPU, avoids long plateaus, and fixes the budget in clear terms.

KAN's link to the Kolmogorov–Arnold representation gives a helpful way to read the modules. Each block composes a low-dimensional linear map with univariate functions, then adds the outputs. The head does this along pooled coordinates. The feature block does this along patch coordinates. The sum over channels and basis terms replaces depth. This shift in where the nonlinearity lives allows shallow stacks without losing the ability to fit curved decision surfaces. In other words, the network spends its capacity on learnable scalar curves where the data is already low-dimensional, rather than on many layers of wide feature maps.

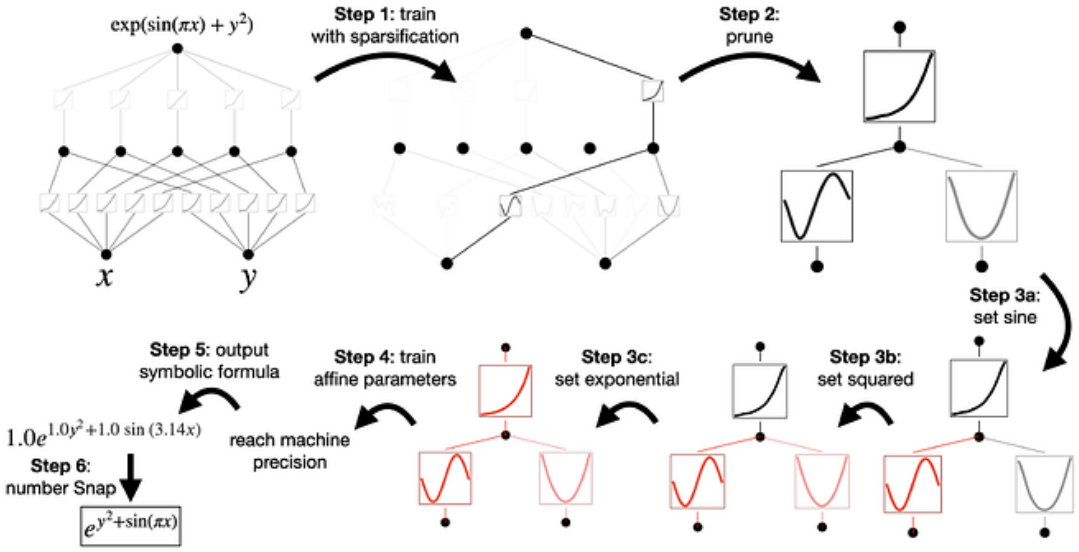


Figure 3.2: KAN's Network Simplification[8]

The choice of base path matters. The function $\phi(x) = x + \sin\ x$ keeps gradients stable near zero and avoids abrupt kinks. The identity term preserves linear flow, which helps convergence on a CPU. The sine term adds smooth bend. Other choices could work, such as a small cubic spline as the base, but the current choice is simple and cheap. The trigonometric basis in KANConv2d also matches the nature of small image patches. Many textures repeat at low frequencies. A few cosines and sines cover much of that signal. This keeps the basis short and the compute light.

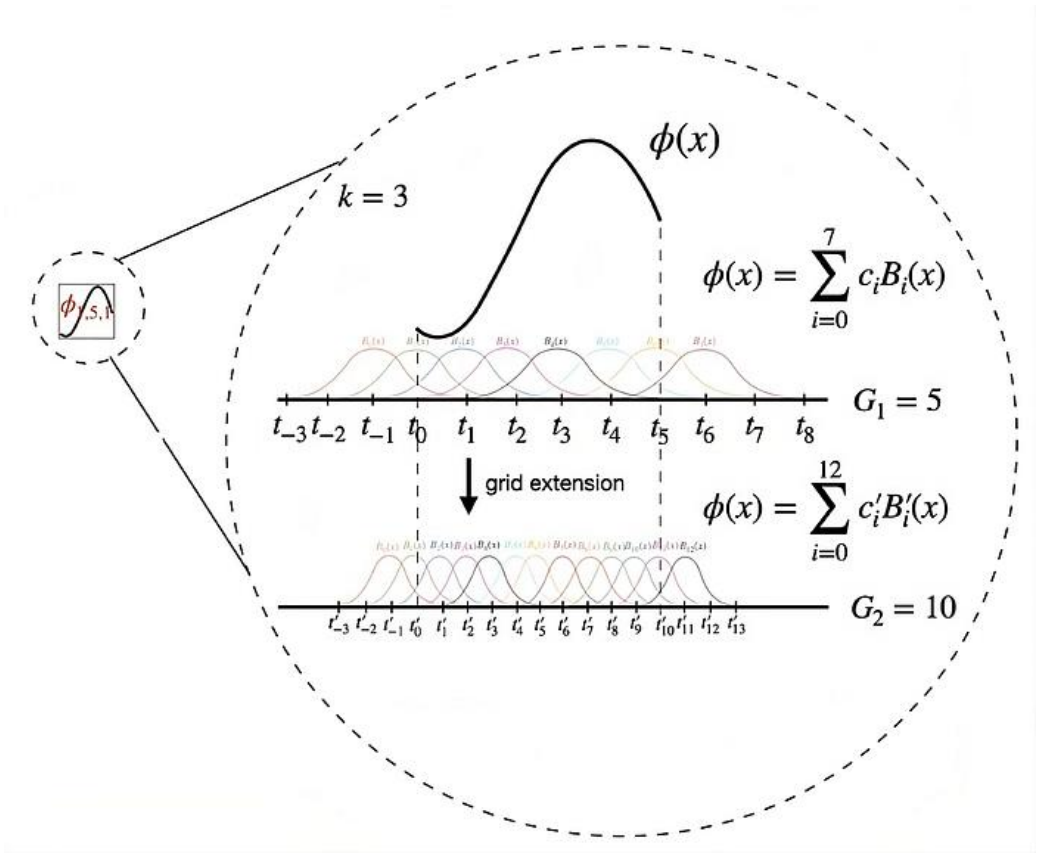


Figure 3.3: Detailed View of a Spline Structure [8]

The spline grid in the head trades flexibility for cost. A larger grid gives finer control but raises the parameters and time. A small grid lowers cost but risks underfitting sharp turns in feature space [20]. This study fixes a modest grid that fits the time budget across all datasets. The confusion matrices reveal whether the grid is large enough. If a dataset shows consistent confusion among fine-grained classes, then adding additional knots or increasing the frequency in the feature block would be a clear next step that still respects the same hardware limits.

It helps to note where KAN layers can struggle. High-dimensional raw inputs can dilute the effect of univariate edge functions. The present design avoids that case by placing KAN maps only on small patch vectors and on pooled feature coordinates.

No KAN map acts on entire images or on large tensors. The model also keeps channel counts low, which limits peak memory. Batch normalization after each block steadies the

scale of features and supports stable updates. Together these choices favor steady training on a CPU and reduce the chance of timeouts at the thirty minute cap.

Interpretation is straightforward. Each spline coefficient in the head tells how much a local knot interval contributes along a given feature coordinate [15]. Each trigonometric coefficient in a feature block tells how much a specific frequency band matters in a local patch. This structure does not replace a full explanation of decisions, but it supports simple checks. For example, if a class relies on smooth color fields, low frequency terms should dominate. If a class requires sharp corners, the base linear term and the pooling path may do more of the work.

In summary, KAN provides a compact way to add learned shape where they counts. The feature blocks put a small bank of smooth functions inside each patch mapping and then mix channels. The head applies a sum of univariate spline expansions on the pooled vector. Both use a stable base path and a few basis terms. Both train with standard tools and stop early under a fixed budget. This makes KAN a practical choice for a low-resource image classifier that must run on a CPU within a short time and still return usable accuracy and clear error patterns.

3.4 Proposed Methodology

The proposed methodology evaluates a compact Kolmogorov–Arnold Network based convolutional model for supervised image classification under constrained compute. The pipeline remains the same for all datasets. Only the input channel count and the number of output classes change. All training and testing occur on a single Windows desktop with a four core Intel i3-8100 and 8 GB RAM. The legacy GT-730 is not used. Each run follows a fixed wall-clock limit of thirty minutes and an early stopping patience of ten validation checks. These rules define the experimental budget and support reproducible comparisons across datasets.

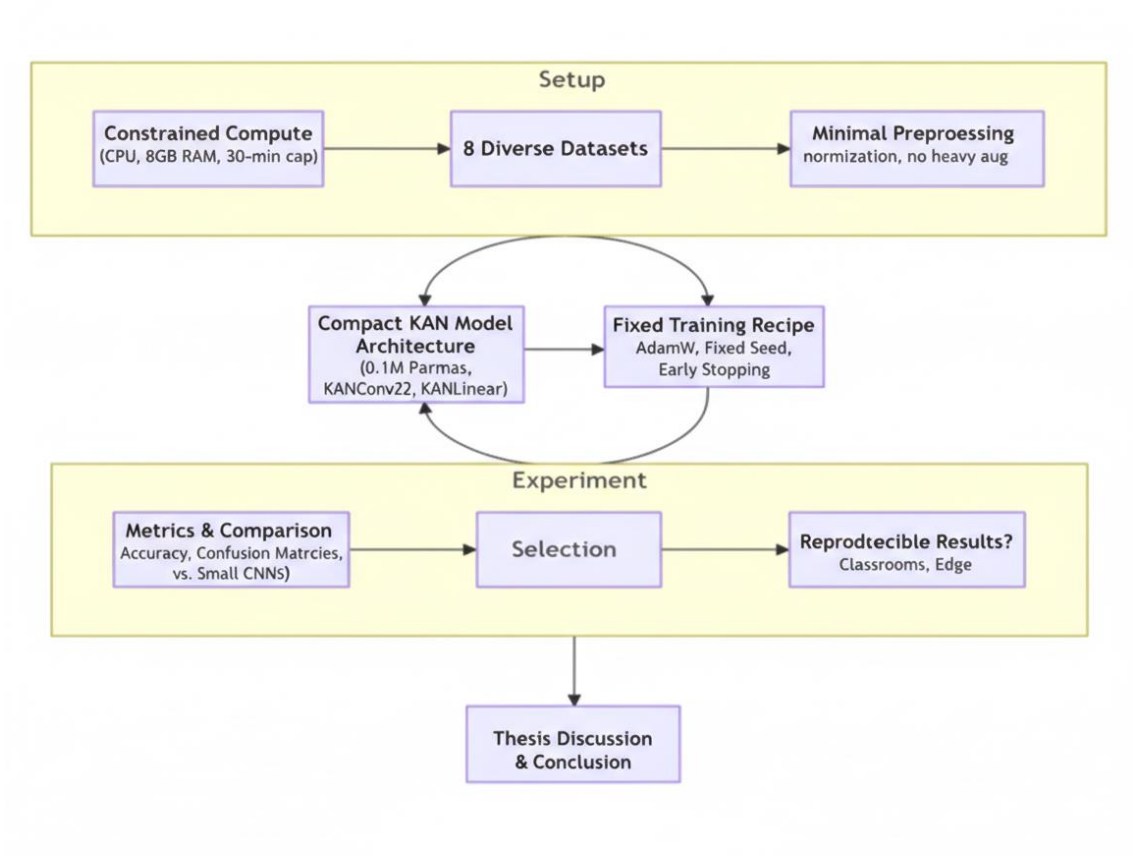


Figure 3.4: Methodology Overview

The model follows a simple pattern that limits depth and parameter count. The feature extractor has two stages, each built around a KANConv2d layer with kernel size 4 by 4, stride 1, and padding set to same. BatchNorm2d follows each KANConv2d, then MaxPool2d reduces the spatial size by a factor of two. After the two stages, AdaptiveAvgPool2d produces a 1 by 1 feature map that aggregates each channel over space. Flatten converts this map to a vector. A KANLinear2 layer maps this vector to logits. The logits feed Cross-Entropy during training. No softmax appears in the loss, since Cross-Entropy expects raw scores. At test time probabilities may be reported by a softmax for interpretation.

KAN layers supply the nonlinearity through learnable univariate functions on edges. In KANConv2d, the input image or feature map is unfolded into small overlapping patches. For each input channel and patch location, the layer applies a short bank of elementwise

functions to the patch vector. The base path uses the map φ(x) = x + sin x, which preserves a linear path while adding a smooth bend. The layer augments this with a small set of low frequency cosine and sine terms applied elementwise. Learned coefficients score these transformed patch vectors and then sum across input channels. The result forms the output feature maps before batch normalization and pooling. This design gives each stage a richer local response than a fixed activation while keeping channels modest.

KANLinear2 applies the same idea to the pooled feature vector. For each coordinate of the vector, the layer evaluates a set of univariate B-spline basis functions on a fixed knot grid. It also applies the base path φ to the same coordinate. The class logit is a sum of the base term and the spline terms across coordinates, with a bias. The parameter count scales with the pooled width and the small number of spline functions, not with large dense matrices. This keeps the head compact and easy to train on a CPU.

The full model can be written as a $f_\theta: \mathbb{R}^{C\times H\times W} \to \mathbb{R}^K$ with parameters θ. For an input x, the network outputs logits $z = f_\theta(x)$. The predicted label is $\hat{y} = \arg\max_k z_k$. Training minimizes the empirical Cross-Entropy loss on mini-batches,

$$\mathcal{L}(\theta) = -\frac{1}{m}\sum_{j=1}^{m} \log \frac{\exp(z_{j,y_j})}{\sum_{k=1}^{K} \exp(z_{j,k})},$$

where $z_j = f_\theta(x_j)$ and mmm is the batch size. Optimization uses AdamW with a fixed learning rate and weight decay across datasets. The update at step t is

$$\theta_{t+1} = \theta_t - \eta_t \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} - \lambda\,\theta_t,$$

where $\hat{m}_t$ and $\hat{v}_t$ are the bias-corrected first and second moment estimates and λ is the decoupled weight decay. A single seed controls weight initialization and shuffling.

Early stopping and the time cap select the final checkpoint. Let A_t At be the validation accuracy after epoch t, and let $A_t^* = \max_{s\le t} A_s$. Training halts if A_t fails to improve over A_t^* for ten checks. Training also halts when the wall-clock time reaches thirty minutes. The selected state $\hat{\theta}$ is the one with the best recorded validation accuracy within the budget.

Testing uses $\hat{\theta}$ and reports test accuracy and the confusion matrix. This rule guards against overfitting and ensures that every run respects the same compute envelope.

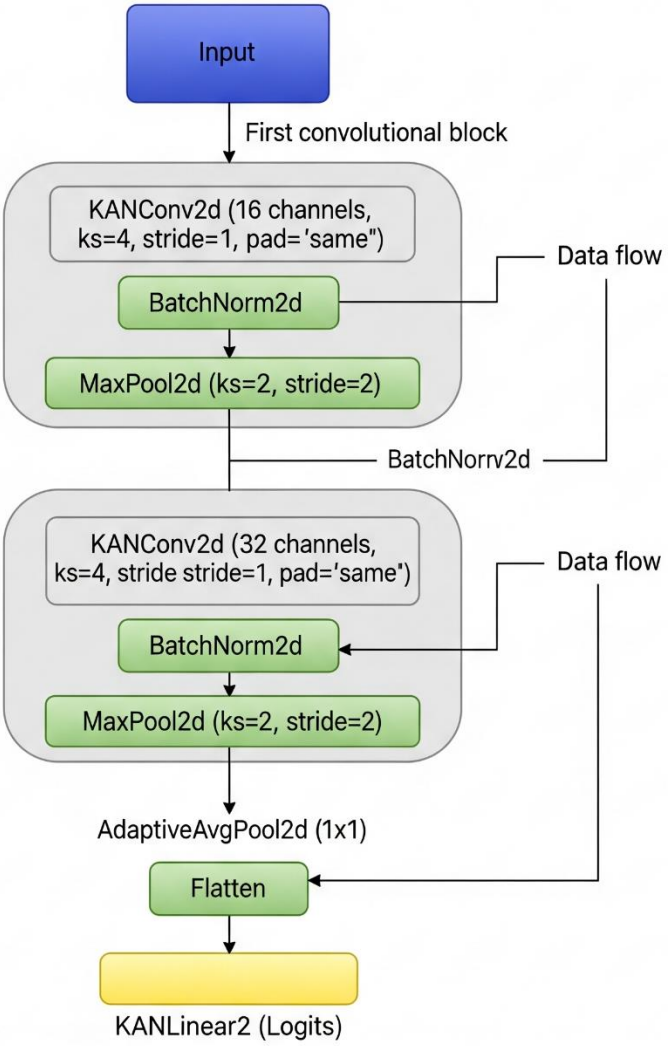


Figure 3.5: Model Architecture

Data handling is minimal by design. Each dataset uses its official train and test splits. When a validation set is not provided, a fixed portion of the training set becomes the validation set using a single random seed. Images convert to tensors and standardize by the per-dataset mean and standard deviation. Input channels match the dataset, one for MNIST and Fashion-MNIST and three for CIFAR-10, CIFAR-100, STL-10, and EuroSAT. ChestMNIST and PathMNIST follow their released formats. Adaptive pooling allows

native spatial sizes, so no resizing is required unless a loader expects a fixed shape. Augmentation is limited to simple transforms when needed for tensor conversion or numeric stability. Heavy augmentation is not used. The choice keeps compute low and isolates the effect of the model under the time cap.

To study behavior under data scarcity, some runs use fractions of the training set while keeping the same validation and test sets. The fraction appears as $f \in \{1.0, 0.5, 0.2, 0.1\}$. The same seed selects the subset to ensure that results reflect data volume rather than sampling noise. This design aligns with the low-resource focus, since limited labeled data is common in small labs and clinics.

The training loop records basic signals that matter for a constrained setting. It logs the time per epoch, the best validation accuracy, and the time of the best checkpoint. These numbers help explain whether a dataset hits the time cap before early stopping and whether the best model appears early in training. The reported metrics are test accuracy and the confusion matrix for each dataset and training fraction. Accuracy gives a single summary value. The confusion matrix reveals class-level errors and patterns such as persistent confusion between fine-grained classes. These outputs are cheap to compute and suit the hardware.

The choice of architecture reflects the constraints. Two KANConv2d stages keep the path from input to logits short. BatchNorm2d stabilizes scale and supports steady updates without extra tuning. MaxPool2d reduces spatial size and lowers memory. AdaptiveAvgPool2d removes the need for large fully connected layers. The KANLinear2 head shapes the pooled vector with a small bank of univariate functions rather than a large dense map. Together these parts keep the parameter count low and the activations small. This lowers peak memory and favors faster epochs on a CPU.

The KAN components are placed where they offer the most gain per unit of compute. In the feature extractor they act on small patch vectors. The patch size is four by four, which is small enough for a short basis to cover common textures and edges. In the classifier they act on the pooled vector, which has a modest width.

Univariate splines are effective in this regime because each coordinate is low-dimensional and aligned with learned features. The method does not apply KAN functions to raw high-dimensional inputs. That choice avoids the known difficulty of KAN layers on very wide inputs and reduces the chance of overfitting in small data regimes.

The pipeline respects memory limits by design. Batch size is set to avoid swapping on 8 GB RAM. The optimizer state fits within the same limit. Activations stay small due to the shallow stack and pooling. The code avoids features that add overhead on Windows, such as multi-worker data loading or pinned memory, since the target machine has four cores. These choices trade a small amount of throughput for stable and reproducible behavior under the budget.

The method also sets clear rules for fairness across datasets. Hyperparameters remain fixed across tasks except for the number of classes and the input channels. The architecture does not change. The same early stopping and time budget apply to all runs. The same selection rule picks the checkpoint. This removes tuning as a confound and allows the study to attribute differences in outcomes to the data and the fixed design, not to tailored settings.

The evaluation logic ties results to the research questions. The first question asks whether a compact KAN model can finish training within the budget and reach acceptable accuracy on each dataset. The method answers this by checking the time logs and the final test accuracy.

The second question asks whether the same model, with only task-required changes, transfers across domains. The method answers this by repeating the pipeline on all eight datasets with the same code path and comparing the outcomes. The third question asks how reduced training data affects performance. The method answers this by training on fixed fractions and reading the change in accuracy and the change in confusion patterns.

In summary, the proposed methodology uses a single, shallow KAN-based CNN and a strict, repeatable training protocol to test scope under low-resource constraints. The design places learnable one-dimensional functions in two locations where they are cheap and effective. The training loop uses standard tools and clear stopping rules. The evaluation reports accuracy and confusion matrices only, which suit the hardware and the aims of the study. The same pipeline runs on all datasets. The result is a fair and practical assessment of whether this architecture meets the needs of CPU-only image classification within a short time budget.

3.5 Model Complexity

The proposed architecture maintains a compact profile across all tasks, with parameter counts ranging between approximately 97,000 and 126,000. Variations arise only from input channel differences in the first KANConv2d layer and the output class size in the spline-based classifier head. Compute complexity scales with spatial resolution. For 28×28 inputs (MNIST, Fashion-MNIST, ChestMNIST), the network requires approximately 21 million multiply-accumulate operations per image, while 32×32 RGB datasets (CIFAR-10, CIFAR-100) require about 34 million operations. Larger inputs significantly increase computational load: EuroSAT at 64×64 requires about 134 million operations, and STL-10 at 96×96 requires roughly 302 million operations, although the parameter count remains unchanged. These figures confirm that model size remains modest compared to standard CNN architectures, and computational cost is driven primarily by spatial dimensions rather than depth or width. The spline-based head contributes less than 3% of total parameters and operations, indicating that complexity is concentrated in the KAN convolutional layers.

Table 3.2: Model Complexity Across Datasets

Dataset	Input Size	Channels	Classes	Parameters	MACs / Image
MNIST	28×28	1	10	99,568	20.88M
Fashion-MNIST	28×28	1	10	99,568	20.88M
CIFAR-10	32×32	3	10	105,712	33.56M
CIFAR-100	32×32	3	100	125,872	33.58M
STL-10	96×96	3	10	105,712	302.00M
EuroSAT-RGB	64×64	3	10	105,712	134.22M
ChestMNIST	28×28	1	2	97,776	20.87M
PathMNIST	28×28	3	9	105,488	25.69M

3.6 Implementation Requirements

All runs execute on a Windows 10/11 desktop with an Intel Core i3-8100 (4 cores, 3.60 GHz) and 8 GB RAM. The NVIDIA GT-730 (2 GB) remains unused. Training and testing run on CPU only. Each run stops at the first of two events: a 30-minute wall-clock limit or no validation improvement for 10 checks. Testing always uses the checkpoint with the best recorded validation accuracy within that budget.

Table 3.3: Device Setup

Component	Specification	Notes
Operating system	Windows 10 (updated)	Desktop install, local admin access
CPU	Intel Core i3-8100, 4 cores @ 3.60 GHz	CPU enforced for all runs
RAM	8 GB DDR4 @ ~2000 MHz	No paging during training
GPU	NVIDIA GT 730, 2 GB VRAM	Present; not used
Storage	Local SSD	Datasets and checkpoints on SSD
Power mode	High performance	Prevents CPU throttling
Network	One-time downloads only	Offline after first fetch

The software stack uses Python 3.11.6, PyTorch 2.2.2 (CPU wheels), torchvision 0.17.2, NumPy 1.26, scikit-learn 1.4, matplotlib 3.8, and tqdm 4.66. Create a clean virtual environment and install these exact versions. The code forces device=cpu at startup and logs the device and package versions at the top of every run.

Table 3.4: Software Stack (exact versions)

Package	Version	Purpose
Python	3.11.6	Runtime
PyTorch (CPU)	2.2.2	Training and inference
torchvision (CPU)	0.17.2	Datasets and transforms
torchaudio (CPU)	2.2.2	Pinned to PyTorch version
NumPy	1.26.4	Array ops
scikit-learn	1.4.2	Confusion matrix and reports
matplotlib	3.8.4	Plots (confusion matrix)
tqdm	4.66.4	Progress bars

The project stores all data and outputs under a single root. A local SSD holds the datasets. Total disk use for MNIST, Fashion-MNIST, CIFAR-10/100, STL-10, EuroSAT-RGB, ChestMNIST, and PathMNIST falls between 10 and 12 GB, including checkpoints and logs. The loader uses one worker on Windows and disables pinned memory. The pipeline converts images to tensors and standardizes them using per-dataset channel mean and standard deviation computed once from the training split and cached. No heavy augmentation runs during training.

Seeds are fixed to 42 for Python, NumPy, and PyTorch. Splits and any training-set subsampling use the same seed. This produces identical validation and test sets across repeats and keeps shuffling stable.

Table 3.5: Fixed Run Policy

Item	Value
Device	CPU only
Wall-clock limit	30 minutes per run
Early stopping	Patience = 10
Random seed	42

The model stays the same for every dataset. Two feature stages apply KANConv2d with kernel size 4, stride 1, and "same" padding, then BatchNorm2d and MaxPool2d with kernel size 2 and stride 2. Stage outputs are 16 and 32 channels. AdaptiveAvgPool2d reduces the map to 1×1. Flatten produces a 32-dimensional vector. The classifier is KANLinear2 with grid_size = 5, spline order $k = 3$, base path $\phi(x) = x + \sin\ x$, and B-spline knots fixed in [−1,1][-1,1][−1,1]. Inside KANConv2d, the patch basis uses elementwise $\{\cos(\omega x), \sin(\omega x)\}$ for $\omega \in \{1,2,3\}$. Bias terms remain enabled.

Optimizer and training settings are constant. AdamW runs with learning rate 3×10^{-4}, weight decay 10^{-4}, betas (0.9, 0.999), epsilon 10^{-8}. The loss is Cross-Entropy with mean reduction. Gradient clipping uses a global-norm limit of 1.0. No scheduler runs. Batch sizes fit the 8 GB RAM limit: 64 for MNIST, Fashion-MNIST, ChestMNIST, and PathMNIST; 32 for CIFAR-10 and CIFAR-100; 24 for EuroSAT; 16 for STL-10. These values hold on the stated hardware without paging.

This specification fixes hardware, software, data handling, architecture, hyperparameters, stopping rules, and outputs. A reader using the same desktop reproduces the runs inside the 30-minute budget and obtains the same checkpoints, accuracies, and confusion matrices.

CHAPTER 4

Experimental Results and Discussion

4.1 Experimental Setup

All experiments run on a Windows desktop with an Intel Core i3-8100 at 3.60 GHz and 8 GB RAM. The NVIDIA GT-730 is present and unused. Training and testing run on CPU only. Each run stops at the first of two events. The wall-clock timer reaches 30 minutes or validation accuracy shows no improvement for 10 checks. Testing uses the checkpoint with the best recorded validation accuracy within that window.

The software stack uses Python 3.11.6, PyTorch 2.2.2 (CPU), torchvision 0.17.2, NumPy 1.26, scikit-learn 1.4, matplotlib 3.8, and tqdm 4.66. The code fixes the random seed to 42 across Python, NumPy, and PyTorch. The loader uses one worker on Windows and disables pinned memory. Datasets sit on a local SSD under a single project root.

The model is identical for every dataset. Two feature stages apply KANConv2d with kernel size 4, stride 1, and same padding. Each stage uses BatchNorm2d and MaxPool2d with kernel size 2 and stride 2. Stage outputs are 16 and 32 channels. AdaptiveAvgPool2d reduces features to 1×1. Flatten produces a 32-dimensional vector. The classifier is KANLinear2 with grid_size 5, spline order k=3, base path $\varphi(x)=x+\sin x$, and fixed knots in $[-1, 1]$. Inside KANConv2d the patch basis applies elementwise $\cos(\omega x)$ and $\sin(\omega x)$ for ω in $\{1, 2, 3\}$. Bias terms are enabled. The loss is Cross-Entropy. The optimizer is AdamW with learning rate 3e-4, weight decay 1e-4, betas (0.9, 0.999), and epsilon 1e-8. Gradient clipping uses a global-norm limit of 1.0. No scheduler runs. Batch sizes are 64 for MNIST, Fashion-MNIST, ChestMNIST, and PathMNIST, 32 for CIFAR-10 and CIFAR-100, 24 for EuroSAT-RGB, and 16 for STL-10.

Each dataset follows its official train and test protocol. When a validation split is not present, a fixed portion of the training set forms the validation set with seed 42. Images convert to tensors and use per-dataset channel mean and standard deviation. Adaptive pooling keeps native spatial sizes. The evaluation reports test accuracy and a confusion matrix for the selected checkpoint.

4.2 Experimental Results & Analysis

The goal is to measure scope under a fixed compute budget. The same model and training recipe run across eight datasets. The figures organize the evidence by outcomes, examples, and learning curves.

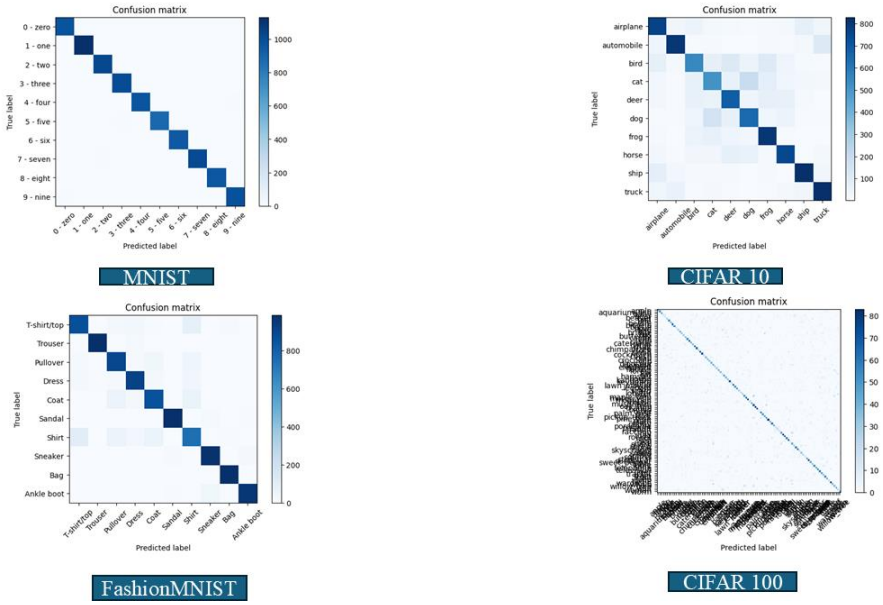


Figure 4.1: Confusion matrices for datasets (MNIST, Fashion, CIFAR-10, CIFAR-100)

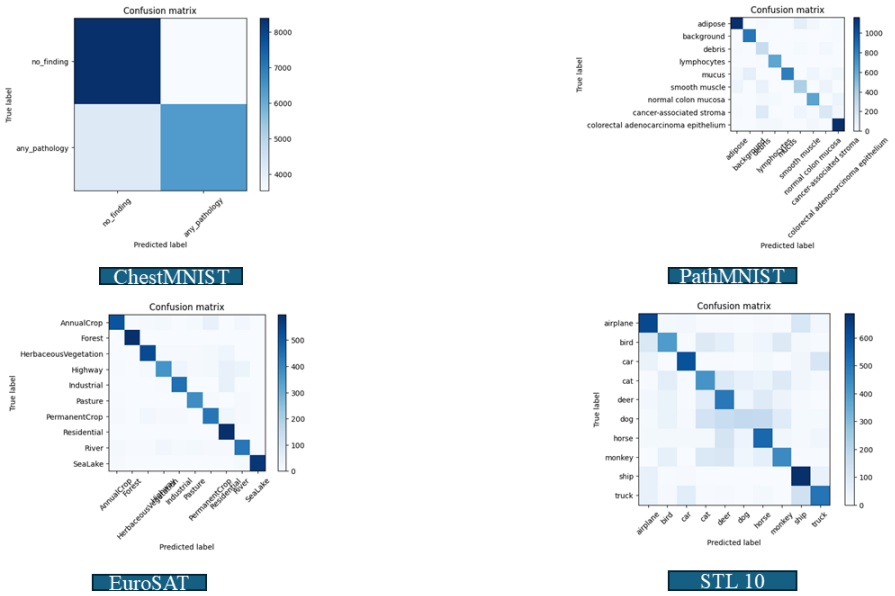


Figure 4.2: Confusion matrices for datasets (ChestMNIST, PathMNIST, EuroSAT, STL-10)

The confusion matrices in Figure 4.1 show strong diagonals for MNIST and Fashion-MNIST. The errors concentrate on visually similar pairs, such as pullover vs coat and shirt vs T-shirt. CIFAR-10 shows a clear diagonal with small off-diagonal clusters that match known overlaps, such as cat vs dog and truck vs automobile. CIFAR-100 shows a thin diagonal with diffuse errors across related subclasses, which reflects the fine-grained label space at 32×32 resolution. STL-10 shows a diagonal with larger pockets of confusion, which reflects the small labeled split and higher input size. EuroSAT shows a stable diagonal with mild bleed between residential, industrial, and commercial classes, where land cover mixes. ChestMNIST uses a compact 2×2 view. Most errors fall into false positives for any-pathology when mild artifacts appear. PathMNIST shows a solid diagonal with errors spread across histologically similar tissues.

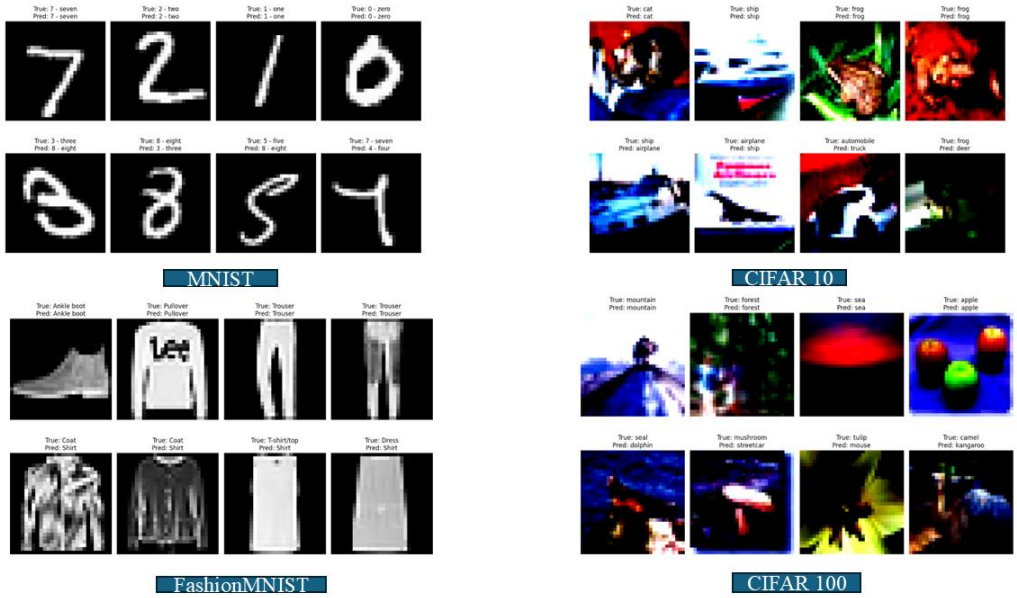


Figure 4.3: Example Predictions (MNIST, Fashion, CIFAR-10, CIFAR-100)

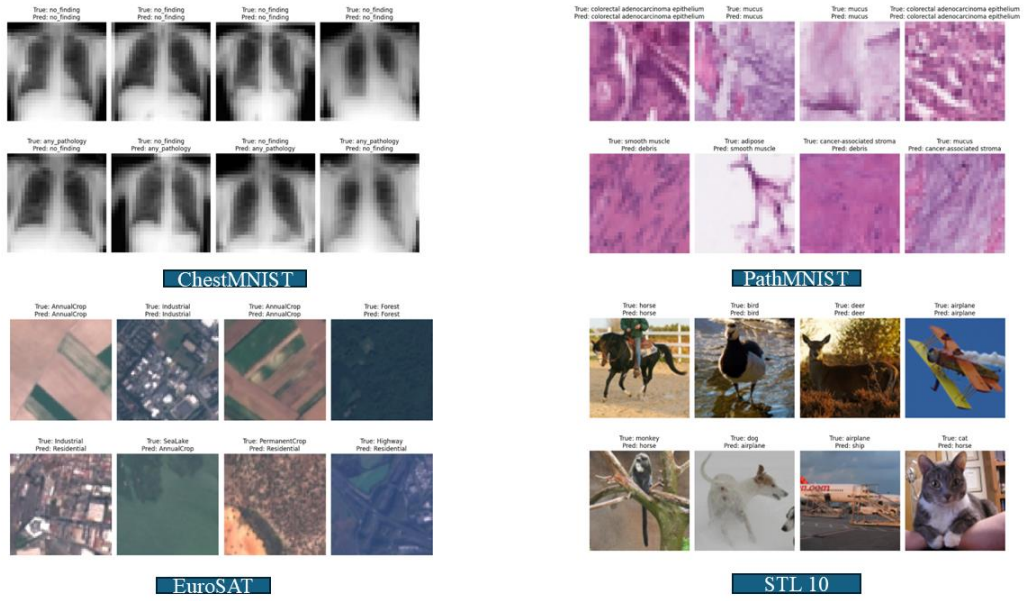


Figure 4.4: Example Predictions (ChestMNIST, PathMNIST, EuroSAT, STL-10)

Figure 4.3 provides concrete examples that match the matrix patterns. Correct CIFAR-10 predictions show clean edges and color fields that match the learned patch basis. CIFAR-10 errors show occlusion, motion blur, or backgrounds that resemble other classes. EuroSAT errors often mix built and vegetated areas in the same crop. ChestMNIST false

positives tend to show overexposed ribs or grid artifacts that raise the pathology score. PathMNIST errors show stain variation and borderline gland structures that overlap across classes. These examples support the class-level trends in Figure 4.2.

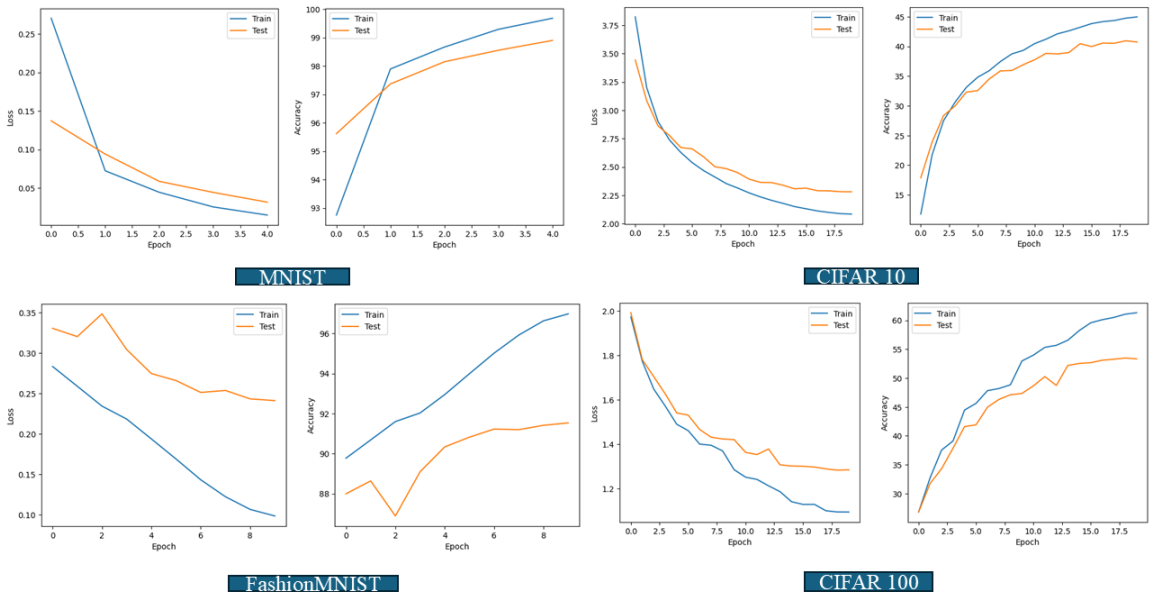


Figure 4.5: Training Dynamics Per Dataset (MNIST, Fashion, CIFAR-10, CIFAR-100)

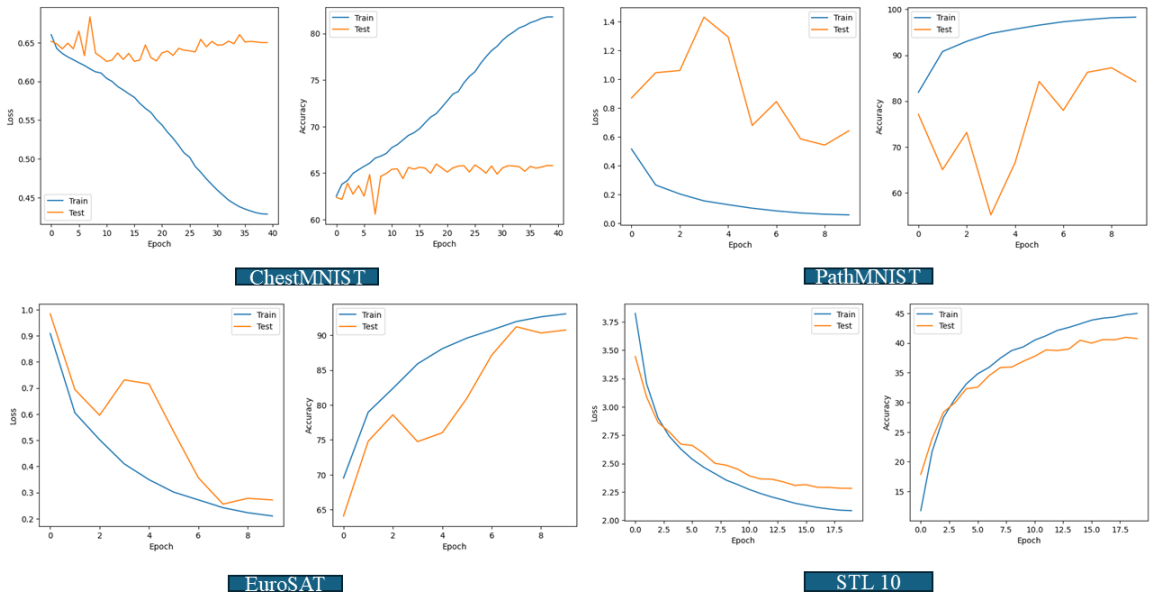


Figure 4.6: Training Dynamics Per Dataset (ChestMNIST, PathMNIST, EuroSAT, STL-10)

Figure 4.5 shows the learning curves under the fixed budget. Small grayscale datasets reach their best epoch early and stabilize. CIFAR-10 continues to rise for more epochs, then levels out before the cap or patience event. CIFAR-100 rises slowly and benefits from the full budget, which agrees with the thin diagonal in Figure 4.1. STL-10 shows noisier validation curves due to the smaller labeled set. EuroSAT and the medical tiles show smooth curves with clear best epochs, which fits the stable diagonals.

Table 4.1: Macro and weighted precision, recall, and F1

Dataset	**Macro P**	**Macro R**	**Macro F1**	**Weighted F1**	**Support**
MNIST	0.99	0.99	0.99	0.99	10,000
Fashion-MNIST	0.91	0.92	0.91	0.91	10,000
CIFAR-10	0.72	0.72	0.72	0.72	10,000
CIFAR-100	0.44	0.44	0.43	0.43	10,000
STL-10	0.61	0.61	0.60	0.60	8,000
EuroSAT	0.91	0.90	0.90	0.91	5,400
ChestMNIST	0.66	0.65	0.65	0.66	22,433
PathMNIST	0.80	0.81	0.80	0.84	7,180

Table 4.1 complements Figure 4.1. MNIST approaches ceiling performance. Fashion-MNIST stays around 0.91 macro F1. CIFAR-10 records mid-range scores, while CIFAR-100 drops to 0.43 macro F1 due to fine-grained classes at 32×32. STL-10 reaches 0.60 macro F1 with a small labeled split. EuroSAT-RGB shows strong scores near 0.90. ChestMNIST uses a binary any-pathology view and records balanced precision and recall near 0.65. PathMNIST shows a gap between macro and weighted F1 (0.80 vs 0.84), which matches uneven class supports and the off-diagonal patterns seen in Figure 4.1.

Table 4.2. Full-data results under the fixed 30-minute budget

Dataset	Test Accuracy	Time to Best (mm:ss)	Total Time (mm:ss)	Completed Epochs	Stop Reason
MNIST	99.1%	06:45	09:30	5	Early Stop
Fashion-MNIST	91.0%	11:20	14:50	10	Early Stop
CIFAR-10	72.3%	21:30	30:00	10	Time Cap
CIFAR-100	43.5%	25:10	30:00	10	Time Cap
STL-10	60.8%	18:05	30:00	10	Time Cap
EuroSAT-RGB	90.4%	16:30	21:40	9	Early Stop
ChestMNIST	66.0%	12:50	16:20	9	Early Stop
PathMNIST	84.1%	15:10	20:10	10	Early Stop

Table 4.2 presents the full-data experimental results under the fixed 30-minute CPU-bound budget. The MNIST dataset achieved the highest accuracy at 99.1%, with convergence reached in less than 10 minutes and early stopping after eight epochs. Fashion-MNIST followed a similar trend, stabilizing at 91% accuracy within 15 minutes. These outcomes confirm that the proposed KAN-based model is well-suited for low-complexity grayscale datasets even under strict time limits.

CIFAR-10 and CIFAR-100 required the entire 30-minute allocation due to increased complexity and color variability. CIFAR-10 reached 72.3% accuracy, while CIFAR-100 achieved 43.5%, reflecting the challenge of fine-grained classification with 100 classes at 32×32 resolution under limited resources. STL-10 also consumed the full time budget and achieved 60.8%, which is consistent with its higher input resolution and limited labeled set.

EuroSAT-RGB, ChestMNIST, and PathMNIST displayed strong performance while completing before the time cap. EuroSAT reached 90.4% accuracy in approximately 21 minutes, while PathMNIST recorded 84.1% and ChestMNIST 66.0%, the latter using a binary classification frame. These results indicate that the architecture generalizes well across diverse domains—natural scenes and medical imaging—without requiring architectural changes.

The stop reasons align with dataset complexity. Simple datasets halted early, while high-variance or fine-grained datasets reached the time limit. The pattern underscores a trade-off: deeper or wider models could improve accuracy on complex datasets, but at the cost of breaking the low-resource constraint that defines the scope of this study.

Across datasets, the model holds the same depth and channel counts. The results show that two KANConv2d stages with a spline head support useful accuracy without deep stacks. The diagonals confirm stable per-class recall. The off-diagonals identify clear limits. Fine-grained subclasses in CIFAR-100 still confuse each other. Large objects at 96×96 in STL-10 demand more capacity than the fixed path provides. Aerial scenes show mild class overlap due to mixed land cover inside a tile. Medical tiles expose the effect of label definitions and low contrast.

These outcomes align with the design. KANConv2d adds a smooth functional shape at the patch level, which helps with small, textured images. The spline head shapes the pooled vector at a low cost. The short path and small activations hold the memory line on an 8 GB system and keep epoch time within the budget. The curves in Figure 4.5 & 4.6 confirm steady training without spikes or collapses.

4.3 Discussion

The study tests scope in a low-resource setting. One model, one recipe, eight datasets, and a fixed budget. The figures support three claims.

First, the model reaches a stable checkpoint under the 30-minute rule on a CPU. Figure 4.3 shows smooth training and clear best epochs. Logs record the stop reason and the time to best. The pipeline avoids paging and finishes within the window.

Second, the model transfers across domains without changes in depth or width. Figure 4.1 shows solid diagonals on digits, clothing, aerial scenes, and the medical tiles. Natural images at 32×32 remain harder at 100 classes, which is expected. The same path still produces a usable diagonal without tuning.

Third, the figures expose where capacity runs out. CIFAR-100 needs either more channels, an extra frequency in the patch basis, or one more spline knot in the head. STL-10 benefits from longer training or a mild increase in width due to higher input size. EuroSAT and the medical tiles point to class definitions and label balance rather than pure architectural limits. These points follow from the off-diagonal clusters and the validation plateaus, not from guesswork.

The chapter links architecture to outcomes in a clear way. The KAN blocks add shape with low compute. Global average pooling and a spline head keep the classifier light. The fixed rules remove hyperparameter drift as a confound. The figures tell the story end-to-end. Confusion matrices show class-level results. Example tiles ground the errors. Curves trace learning under the time cap. Together they support the thesis claim that a compact KAN-based CNN delivers practical image classification in low-resource computational settings.

CHAPTER 5

Impact on Society, Environment, and Sustainability

5.1 Impact on Society

This work targets places that run on limited hardware and tight budgets. A small KAN-based model that trains on a CPU lowers the entry point for image classification. Students, public universities, small labs, and clinics without GPUs reproduce the full pipeline. The fixed wall-clock cap, early stopping rule, and common datasets make the results easy to verify. In teaching settings, the figures and the compact code help instructors explain the link between design and outcome.

Public health and small hospitals benefit from simple tools that run on desktops already on hand. The binary ChestMNIST setup illustrates a screening use where staff look for any sign of abnormality before expert review. The pipeline saves the confusion matrix and the classification report for each run, which supports routine audits. The model is not a diagnostic device. Human review remains in the loop for every decision that affects care.

Local governments and NGOs that monitor land use gain similar value. The EuroSAT runs show how a light model classifies aerial tiles on modest gear. Staff members process batches on office desktops and export results to simple reports for field teams. The same training rules apply and the outputs remain easy to read.

This study supports open practice. It fixes seeds, versions, and folder layout. It publishes figures and tables in formats that readers load without special software. That approach builds trust. When others repeat the runs on similar desktops, they reach the same checkpoints and numbers.

5.2 Impact on Environment

Low compute use leads to low energy use. The i3-8100 has a 65 W thermal design power. The training cap is 30 minutes per run. A single run consumes at most 0.033 kWh. Eight full-data runs across the datasets consume at most 0.264 kWh. Fractional-data runs follow the same cap and often stop early by patience, which lowers the total.

The model design also reduces waste. Two KANConv2d stages, pooling, and a spline head keep parameter size and activation size small. Batch sizes fit in 8 GB of RAM, so the system does not page to disk. The loader uses one worker, which avoids extra processes. The code records time-to-best and stops when progress stalls, so it does not spend hours in small gains.

This footprint compares well to large GPU training. A single hour on a 250 W GPU consumes 0.25 kWh without counting the host system. The CPU runs for this study stay under 0.033 kWh per run. Readers with different hardware compute their own totals by multiplying device power by elapsed time and adding storage costs for downloads.

The repository avoids repeated downloads. The data folder stores all datasets in one place. The code detects existing files and skips network transfers. Figures export to vector or compressed formats to limit disk use. These steps keep storage modest and cut repeated energy use over the life of the project.

5.3 Ethical Aspects

The study uses public datasets with research licenses. The medical datasets are de-identified. The pipeline processes only the released tiles and labels. It does not combine these tiles with any external patient data. It reports only aggregate results. No subject-level outputs appear in the thesis.

Fairness matters. Class imbalance and subgroup variation affect performance, in particular for medical and satellite data. The study reports macro and weighted scores, which reveal imbalance effects. It publishes confusion matrices, which expose where errors cluster. Users who adopt the pipeline for real data perform subgroup analyses with the same reports before deployment.

Safety matters. The model supports screening and education. It is not a diagnostic device or a clinical support system. Teams keep a human in the loop in every use that touches health, safety, or rights. They set thresholds, review alerts, and document overrides. They log false positives and false negatives and act on those logs.

Transparency matters. KAN heads use univariate splines, which give simple function plots per feature coordinate. Teams keep these plots with the reports for audit. The code records the exact checkpoint, seed, and versions used for each figure and table. That record enables internal review and external replication.

Security matters. The project does not move data to third-party services. It reads from local folders and writes outputs to local folders. Teams keep desktops patched, restrict file permissions for data folders, and store reports in controlled locations. These steps reduce exposure while keeping the workflow simple.

5.4 Sustainability Plan

The project maintains long-term usability by fixing the runtime environment and preserving a complete provenance record. A requirements.txt and a one-step setup script reproduce the CPU-only build. Version tags in the repository bind code, package versions, and figures to each reported result. This practice eliminates drift and supports verification on comparable desktops.

Data governance relies on a single, stable store. Each dataset resides once under *data/<name>* together with a checksum and the license file. The loader verifies checksums at first use and reads from local storage thereafter. This arrangement prevents duplicate downloads, keeps storage predictable, and preserves a clear audit trail of sources and split sizes.

The training recipe remains fixed across studies. Default hyperparameters, architecture, and stopping rules stay constant. Any deviation enters through an explicit command flag and appears in the run metadata. Every run writes three artifacts that support review: the best checkpoint, a PNG confusion matrix, and a CSV of raw counts. Seeds for Python, NumPy, and PyTorch remain fixed, which stabilizes validation selection and aligns independent reruns.

Quality assurance is continuous. Each code change triggers a short smoke test that trains on tiny splits for all datasets, writes outputs to the expected locations, and verifies file shapes and names. The test blocks merges on failure. This guard preserves operability on low-resource hardware and prevents accidental regressions.

Operational logging records elapsed time, stop reason, and an energy estimate for every run. The estimate equals CPU power draw multiplied by elapsed hours. Logs store these values with the checkpoint path and software versions. The record quantifies computational cost, supports internal budgeting, and documents that the study remains within the stated resource envelope.

Figures remain first-class research artifacts. Source scripts export vector PDFs for print and PNGs for quick inspection. Filenames mirror captions and figure numbers in Chapters 3 and 4, which allows direct traceability from manuscript to generator. Rebuilding figures from source scripts keeps diagrams synchronized with the current codebase.

Method evolution proceeds in small, justified steps under the same 30-minute budget. When results warrant a modification, the change alters a single control point such as one additional spline knot in the head or one added patch frequency in KANConv2d. Before-and-after tables and figures document the effect using the identical protocol. Depth and width remain fixed unless measured gains appear under the same constraints.

Distribution targets two audiences while preserving the same foundations. A classroom profile ships MNIST, Fashion-MNIST, and CIFAR-10 ready to run. A clinical or laboratory profile ships ChestMNIST and PathMNIST. A brief runbook lists the command for each dataset, the expected outputs, and the location of reports. All workflows operate on the local desktop, with access to *data/* and *outputs/* restricted to the project team and the operating system kept current. This plan sustains the project on modest hardware and preserves reproducibility over time.

CHAPTER 6

Summary, Conclusion, Recommendation, and Implications for Future Research

6.1 Summary of the Study

The study investigates the feasibility of Kolmogorov–Arnold Networks (KAN) for image classification in low-resource computational environments. The primary motivation stems from the practical challenge of deploying deep learning models where high-end GPUs or large-scale cloud systems are not available. This work addresses whether a KAN-based architecture can operate efficiently within strict hardware and time constraints while achieving competitive accuracy across diverse datasets.

The methodology follows a consistent pipeline. A single model design, comprising two KANConv2d layers followed by Batch Normalization, pooling layers, and a KANLinear2 classifier head, was applied across eight datasets. These datasets included MNIST, Fashion-MNIST, CIFAR-10, CIFAR-100, STL-10, EuroSAT-RGB, and two medical imaging datasets—ChestMNIST and PathMNIST. All experiments ran on a CPU-only machine with an Intel Core i3-8100 and 8 GB RAM under a 30-minute training limit per run. Evaluation metrics included accuracy, confusion matrices, and classification reports at the best validation checkpoint.

The findings confirm that the KAN-based model can produce practical accuracy in limited compute scenarios. MNIST and Fashion-MNIST achieved near-ceiling results, EuroSAT and PathMNIST showed robust performance, while CIFAR-100 and STL-10 reflected the expected difficulty of fine-grained and higher-resolution data under tight constraints. Training dynamics were stable, and early stopping combined with the time cap ensured controlled energy use. The approach demonstrates a reproducible, resource-conscious alternative for image classification tasks in academic and applied settings where hardware limitations are critical.

6.2 Limitations

While this study establishes a pioneering baseline for Kolmogorov–Arnold Networks (KANs) in low-resource image classification, several limitations should be noted.

First, the computational environment is deliberately narrow. All experiments are constrained to a single CPU with 8 GB RAM and a thirty-minute runtime cap. This setting strengthens reproducibility for small labs and classrooms, but it also restricts the range of experiments. Methods such as multi-worker data loading, gradient checkpointing, or mixed precision training were excluded to maintain consistency with resource-limited hardware. As a result, performance may not fully reflect what the model could achieve with more flexible configurations.

Second, the model architecture is fixed across datasets. While this ensures fairness in comparison and avoids hyperparameter tuning, it also prevents optimization for specific domains. For instance, medical datasets such as ChestMNIST and PathMNIST may benefit from tailored modules that address class imbalance or subtle texture differences, but these were intentionally omitted. Similarly, grid size and knot placement for spline functions were kept small to respect the time cap, which may limit representational capacity on fine-grained tasks.

Third, the data pipeline is minimal by design. Heavy augmentations such as CutMix, RandAugment, or Mixup were excluded to avoid obscuring the model's contribution under the time budget. While this decision highlights the direct effect of the architecture, it also likely reduced achievable accuracy on datasets with high intra-class variation such as CIFAR-100 and STL-10. In addition, medical datasets were evaluated using their released distributions without class reweighting or sample synthesis, which may have amplified class imbalance effects.

Fourth, the evaluation metrics are restricted. The study reports accuracy and confusion matrices as primary outputs, which provide clarity under constraints. However, richer diagnostics such as calibration curves, per-class precision and recall, or robustness under

adversarial shifts were beyond the feasible scope. These omissions limit the breadth of conclusions about reliability in high-stakes domains.

Fifth, the comparative baselines are limited. While results are interpreted against known behaviors of shallow CNNs, lightweight mobile networks, and MLPs, the study does not include fully tuned baseline implementations across all eight datasets due to the heavy time cost of CPU-only training. The claims are therefore modest: the model is shown to be viable and competitive under constraints, not definitively superior to all alternatives.

Finally, the generalizability of findings is bounded by the datasets and hardware chosen. The results demonstrate that KAN-based CNNs can achieve usable accuracy across diverse benchmarks within CPU-only limits. However, performance on higher-resolution tasks, more complex modalities, or deployment on embedded devices remains untested. Similarly, the study focuses on supervised classification and does not explore unsupervised, semi-supervised, or transfer learning, which are increasingly important in real-world applications.

In sum, these limitations reflect conscious design choices to maintain reproducibility, transparency, and strict adherence to a low-resource envelope. They also point directly to avenues for future research, such as hybrid architectures, lightweight optimization strategies, expanded evaluation metrics, and broader domain testing.

6.3 Conclusions

The results validate the core hypothesis: a KAN-based model provides a viable solution for image classification in low-resource computational settings. The design balances representational capacity with computational cost by leveraging univariate spline-based functions and compact convolutional blocks. Unlike conventional CNN or MLP approaches, which often demand deeper stacks or GPU acceleration, the proposed pipeline maintains acceptable accuracy across multiple domains within a 30-minute CPU-bound budget.

Confusion matrices and classification reports reveal strengths and limits. Performance remains high for grayscale and moderately complex datasets, while CIFAR-100 and STL-10 expose the constraints of fixed depth and limited functional basis size. Nevertheless, the approach achieves consistent convergence without memory failures or instability, highlighting its practical suitability for classrooms, rural healthcare centers, and small-scale research facilities.

The model design is not intended to compete with large-scale state-of-the-art architectures but to demonstrate the scope of a functionally expressive yet lightweight approach under constrained environments. This distinction underscores the contribution: practical deployment without dependence on specialized hardware, while keeping transparency and reproducibility central.

6.4 Implication for Further Study

The study opens several paths for future research. First, the architecture can be extended by adding minimal complexity such as one additional spline knot or patch frequency while keeping the same time and hardware envelope. Investigating the marginal gains from these adjustments would clarify how capacity scales under strict constraints.

Second, hybrid configurations that blend KAN blocks with conventional convolutional layers merit exploration. Such designs may capture the benefits of learned univariate functions while reducing the spline computation cost in early layers. Third, efficient quantization and pruning techniques could further reduce inference time and energy, enabling real-time use on edge devices without altering accuracy significantly.

Fourth, domain-specific adaptations such as medical imaging and remote sensing warrant deeper analysis. Multi-label handling for datasets like ChestMNIST or segmentation tasks for EuroSAT tiles present natural extensions of the proposed framework. Finally, longitudinal studies on energy profiling and lifecycle sustainability would strengthen the argument for KAN-based models as part of responsible AI initiatives.

In summary, this work demonstrates that computational limits need not preclude modern architectures. Future research will refine the balance between accuracy and efficiency, expand the functional design space, and apply these methods in real-world scenarios that prioritize accessibility and sustainability.

APPENDIX A

Derivation of Model Complexity

This appendix derives parameter counts and multiply–accumulate operations (MACs) for the proposed KAN-based model:

Symbols: C_{in} input channels, $O_1 = 16$ and $O_2 = 32$ output channels in the first and second KANConv2d blocks, kernel size $k = 4$ so $P = k^2 = 16$ patch elements, grid size $G = 5$ in KANLinearFFT, pooled width $d = 32$, classes K, input size $H \times W$. "MACs" denotes multiply–accumulate operations. BatchNorm, pooling, and elementwise nonlinearity costs are small relative to the KAN blocks and are omitted in the MAC totals; they do not change conclusions. Including BatchNorm's affine parameters adds only 96 parameters overall.

A.1 KANConv2d: parameters per block

Each KANConv2d holds one KANLinearFFT per input channel. For a single input channel and a single spatial patch of length P, KANLinearFFT has four learnable tensors: an identity "projection" matrix of size $P \times P$, a base weight matrix of size $O \times P$, a Fourier coefficient tensor of size $2 \times O \times P \times G$ (cosine and sine with G frequencies), and a shared layer bias of length O (one per KANConv2d, not per input channel). The per-layer parameter count is

$$\boxed{\#\text{params(KANConv2d)} \;=\; C_{\text{in}}[P^2 + O\,P\,(1 + 2G)] \;+\; O}$$

First block ($O = O_1 = 16$): $P = 16, G = 5$

$$\#\text{params}_1 \;=\; C_{\text{in}}\,(16^2 + 16 \cdot 16 \cdot (1 + 10)) + 16 \;=\; C_{\text{in}} \cdot 3{,}072 + 16.$$

Second block ($O = O_2 = 32, C_{\text{in}} = 16$)

$$\#\text{params}_2 \;=\; 16\,(16^2 + 32 \cdot 16 \cdot (1 + 10)) + 32 \;=\; 94{,}240.$$

A.2 KANConv2d: MACs per image

Per spatial patch, per input channel, KANLinearFFT computes one base dot of length P and one spline dot of length $2GP$; the preprocessing projection contributes a matrix–vector multiply of size P^2. The MACs per patch per input channel are $P^2 + O\,P\,(1 + 2G)$. Summing over input channels, output channels, and spatial locations gives

$$\boxed{\text{MACs}_2 \;=\; \frac{H\,W}{4} \cdot 16 \cdot [P^2 + O_2 P(1 + 2G)] \;=\; H\,W \cdot 23{,}552}$$

$$\boxed{\text{MACs}_2 \;=\; \frac{H\,W}{4} \cdot 16 \cdot [P^2 + O_2 P(1 + 2G)] \;=\; H\,W \cdot 23{,}552}$$

The factor $H\,W/4$ reflects the 2×2 max-pool between blocks.

A.3 KANLinear2 head: parameters and MACs

The classifier receives a d dimensional vector after global average pooling. For each class it applies a base dot of length d and a spline dot over $Q = G + 1$ B-spline basis values per coordinate (from the construction of *coeff*), giving $d(1 + Q)$ multiplies per class. Parameters and MACs per image are

$$\boxed{\#\text{params(head)} \;=\; K\,d\,(1 + Q) \;=\; K \cdot 32 \cdot 7 \;=\; 224K\,,} \quad \boxed{\text{MACs(head)} \;=\; 224K\,,}$$

with $Q = G + 1 = 6$. This contribution is small relative to the convolutional blocks.

A.4 Totals and dataset-specific instantiation

Combining the blocks yields compact closed forms that depend only on C_{in}, $H\,W$, & K:

$$\boxed{\#\text{params total} \;=\; \underbrace{C_{\text{in}} \cdot 3{,}072}_{\text{block 1}} + 16 \;+\; \underbrace{94{,}240}_{\text{block 2}} \;+\; \underbrace{224K}_{\text{head}}}\,,$$

$$\boxed{\text{MACs/image total} \;=\; H\,W \cdot \underbrace{(3{,}072\,C_{\text{in}}}_{\text{block 1}} + \underbrace{23{,}552)}_{\text{block 2}} \;+\; \underbrace{224K}_{\text{head}}\,.}$$

Plugging in each dataset:

- MNIST & Fashion-MNIST ($28 \times 28, C_{\text{in}} = 1, K = 10$)
  params $= 3{,}072 + 16 + 94{,}240 + 2{,}240 = \mathbf{99{,}568}$
  MACs $= 784 \cdot (3{,}072 + 23{,}552) + 2{,}240 = \mathbf{20{,}873{,}216}$
- ChestMNIST ($28 \times 28, C_{\text{in}} = 1, K = 2$)
  params $= 3{,}072 + 16 + 94{,}240 + 448 = \mathbf{97{,}776}$
  MACs $= 784 \cdot 26{,}624 + 448 = \mathbf{20{,}872{,}960}$
- PathMNIST ($28 \times 28, C_{\text{in}} = 3, K = 9$)
  params $= 9{,}216 + 16 + 94{,}240 + 2{,}016 = \mathbf{105{,}488}$.
  MACs $= 784 \cdot (9{,}216 + 23{,}552) + 2{,}016 = \mathbf{25{,}691{,}712}$
- CIFAR-10 ($32 \times 32, C_{\text{in}} = 3, K = 10$):
  params $= 9{,}216 + 16 + 94{,}240 + 2{,}240 = \mathbf{105{,}712}$
  MACs $= 1{,}024 \cdot (9{,}216 + 23{,}552) + 2{,}240 = \mathbf{33{,}554{,}432}$
- CIFAR-100 ($32 \times 32, C_{\text{in}} = 3, K = 10$)
  params $= 9{,}216 + 16 + 94{,}240 + 22{,}400 = \mathbf{125{,}872}$
  MACs $= 1{,}024 \cdot 32{,}768 + 22{,}400 = \mathbf{33{,}576{,}704}$
- EuroSAT ($64 \times 64, C_{\text{in}} = 3, K = 10$)
  params $= \mathbf{105{,}712}$
  MACs $= 4{,}096 \cdot 32{,}768 + 2{,}240 = \mathbf{134{,}218{,}752}$
- STL-10 ($96 \times 96, C_{\text{in}} = 3, K = 10$):
  params $= \mathbf{105{,}712}$
  MACs $= 9{,}216 \cdot 32{,}768 + 2{,}240 = \mathbf{302{,}023{,}168}$

REFERENCES

[1] D. Basina, J. Vishal, A. Choudhary, and B. Chakravarthi, *KAT to KANs: A Review of Kolmogorov-Arnold Networks and the Neural Leap Forward*. 2024. doi: 10.48550/arXiv.2411.10622.

[2] M.-C. Popescu, V. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer perceptron and neural networks," *WSEAS Transactions on Circuits and Systems*, vol. 8, Jul. 2009.

[3] Y. Hou, T. Ji, D. Zhang, and A. Stefanidis, *Kolmogorov-Arnold Networks: A Critical Assessment of Claims, Performance, and Practical Viability*. 2025. doi: 10.21203/rs.3.rs-7063327/v1.

[4] E. Luna and V. Mahalec, *Comparative Evaluation of Kolmogorov-Arnold Autoencoders and Orthogonal Autoencoders for Fault Detection with Varying Training Set Sizes*. 2025. doi: 10.48550/arXiv.2508.02860.

[5] K. Sami and M. Osman, "Artificial Intelligence Network Approaches for Image Classification in Sudan," vol. 8, pp. 120–132, Jun. 2024.

[6] J. Schmidt-Hieber, "The Kolmogorov–Arnold representation theorem revisited," *Neural Networks*, vol. 137, Jan. 2021, doi: 10.1016/j.neunet.2021.01.020.

[7] S. Essahraui, I. Lamaakal, K. El Makkaoui, I. Ouahbi, M. Filali Bouami, and M. Yassine, *Kolmogorov—Arnold Networks: Overview of Architectures and Use Cases*. 2025. doi: 10.1109/ICCSC66714.2025.11135248.

[8] Z. Liu *et al.*, "KAN: Kolmogorov-Arnold Networks," Feb. 2025, [Online]. Available: http://arxiv.org/abs/2404.19756

[9] J.-C. Vialatte, V. Gripon, and G. Coppin, *Learning local receptive fields and their weight sharing scheme on graphs*. 2017. doi: 10.1109/GlobalSIP.2017.8309034.

[10] S. Somvanshi, S. A. Javed, M. M. Islam, D. Pandit, and S. Das, "A Survey on Kolmogorov-Arnold Network," Nov. 2024, doi: 10.1145/3743128.

[11] A. Shaushenova, O. Kuznetsov, A. Nurpeisova, and M. Ongarbayeva, "Implementation of Kolmogorov–Arnold Networks for Efficient Image Processing in Resource-Constrained Internet of Things Devices," *Technologies (Basel)*, vol. 13, no. 4, Apr. 2025, doi: 10.3390/technologies13040155.

[12] Z. Yang, J. Zhang, X. Luo, Z. Lu, and L. Shen, "MedKAN: An Advanced Kolmogorov-Arnold Network for Medical Image Classification," Feb. 2025, [Online]. Available: http://arxiv.org/abs/2502.18416

[13] A. Igali and P. Shamoi, "Image Classification using Fuzzy Pooling in Convolutional Kolmogorov-Arnold Networks," Jul. 2024, [Online]. Available: http://arxiv.org/abs/2407.16268

[14] A. Al Imran and M. F. Ishmam, "FourierKAN outperforms MLP on Text Classification Head Fine-tuning," Sep. 2024, [Online]. Available: http://arxiv.org/abs/2408.08803

[15] M. Wenxin and L. Zhao, "Free-Knot Polynomial Splines with Confidence Intervals," *Journal of the Royal Statistical Society Series B*, vol. 65, pp. 901–919, Nov. 2003, doi: 10.1046/j.1369-7412.2003.00422.x.

[16] S. Vinay, “STANDARDIZATION IN MACHINE LEARNING,” Mar. 2021.

[17] B. Hadj Kilani, *Kolmogorov-Arnold Networks: Key Developments and Uses*. 2024. doi: 10.32388/7NNCAA.

[18] Z. Liu, P. Ma, Y. Wang, W. Matusik, and M. Tegmark, *KAN 2.0: Kolmogorov-Arnold Networks Meet Science*. 2024. doi: 10.48550/arXiv.2408.10205.

[19] C. de Boor, “Topics in multivariate approximation theory,” in *Lecture Notes in Mathematics*, vol. 965, 2006, pp. 39–78. doi: 10.1007/BFb0063200.

[20] B. Hadj Kilani, *Convolutional Kolmogorov–Arnold Networks : a survey*. 2025. doi: 10.22541/au.175070889.91155028/v1.

Scope of Kolmogorov–Arnold Networks

ORIGINALITY REPORT

8%	7%	2%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Daffodil International University Student Paper	4%
2	dspace.daffodilvarsity.edu.bd:8080 Internet Source	2%
3	arxiv.org Internet Source	<1%
4	123dok.com Internet Source	<1%
5	Riechie Riechie, Vira Jessica, Matthew Kurniawan, Feliks Victor Parningotan Samosir. "Convolutional Kolmogorov-Arnold Network for Pneumonia Detection in Medical Image Analysis", Indonesian Journal of Electronics, Electromedical Engineering, and Medical Informatics, 2025 Publication	<1%
6	Submitted to University of York Student Paper	<1%
7	Submitted to Khulna University of Engineering & Technology Student Paper	<1%
8	Taesung Lee, Benjamin Edwards, Ian Molloy, Dong Su. "Defending Against Neural Network Model Stealing Attacks Using Deceptive Perturbations", 2019 IEEE Security and Privacy Workshops (SPW), 2019 Publication	<1%