ISRO – Telemetry Tracking and Command Network (ISTRAC)

TITLE: COMPARISON OF OPTIMISATION ALGORITHMS IN CNNs IN MALWARE CLASSIFICATION

UNDER THE GUIDANCE OF

Bharat Devasani

Scientist/Engineer-SE, CND, CSSA, ISTRAC

SUBMITTED BY

Hrishikesh H Chandra **Reg No: 01JST21IS017**

STUDENT OF

JSS SCIENCE AND TECHNOLOGY UNIVERSITY – JSS STU

JSS TECHNICAL INSTITUTIONS CAMPUS, MYSORE - 57006

ACKNOWLEDGEMENT

If words are considered as symbols of approval and tokens of acknowledgement, then let the words play the heralding role of expressing my gratitude to all those who have directly or indirectly helped during my period of internship.

I am delighted to thank **Dr. A. K. ANILKUMAR, Director, ISRO Telemetry, Tracking, and Command Network (ISTRAC), Bengaluru, Karnataka,** for giving me permission to pursue my Internship in ISTRAC, BENGALURU.

I express my gratitude to **Mr. SANKAR MADASWAMY B, Scientist/Engineer-SG, Manager, HRDD, ISTRAC,** for permitting me to undertake my internship.

My sincere thanks also goes to my guide **Mr. Bharath Devasani, Scientist/Engineer-SE, ISTRAC,** for his mentorship, constructive feedback, and technical insights. His meticulous approach and encouragement have greatly enhanced my understanding and technical proficiency.

I sincerely thank you all whole-heartedly for sparing your valuable time quite often, despite the heavy workload. The discussions had provided me with valuable insights into the workings of ISRO. Finally, I would like to thank my people (my parents, friends and university) for their continuous support in every field of my life.

HRISHIKESH H CHANDRA

USN: 01JST21IS017

ABSTRACT

This project systematically evaluates the impact of optimization algorithms on deep-learning–based static malware classification using the publicly available MaleVis dataset's pre-formatted 224×224 RGB images. Rather than developing new data-conversion tools, we built upon existing image representations by implementing two established convolutional neural network architectures—DenseNet-121 and ResNet-18—in PyTorch and training them from scratch under uniform CPU-only conditions. Our core contribution is the design and execution of a rigorous experimental framework that holds all hyperparameters constant while swapping only the optimizer—SGD with momentum, Adam, and the recently proposed Lion—to isolate each method's influence on convergence dynamics and multi-class generalization. We extended the training pipeline to incorporate Lion in addition to the more common SGD and Adam, logged epoch-by-epoch accuracy, precision, and recall, and compared learning curves and final performance across all six model-optimizer combinations. By distilling these findings into clear, scenario-driven recommendations, we provide practitioners with actionable guidance on selecting optimizers tailored to resource constraints, convergence speed, and generalization requirements in vision-based malware detection systems.

Table of Contents

List of Tables

List of Figures

I. Chapter – 1

1. Introduction

Malware, or malicious software, continues to pose a serious threat to cyber security in today's interconnected world. With the exponential growth of software systems and the increasing sophistication of malware, detecting and classifying malicious programs has become more critical than ever. Traditional malware detection methods, primarily relying on manual feature engineering and signature-based approaches, are often insufficient against rapidly evolving threats.

In response to this challenge, recent research has explored the potential of machine learning and deep learning techniques, particularly computer vision methods, for static malware classification. A promising approach involves converting malware binary files into visual image representations, allowing powerful Convolutional Neural Networks (CNNs) to automatically learn distinguishing features for classification tasks.

This project builds upon the research study titled "Utilization and Comparison of Convolutional Neural Networks in Malware Recognition" by Ahmet Selman Bozkir, Ahmet Ogulcan Cankaya, and Murat Aydos from Hacettepe University. The study investigates the effectiveness of various CNN architectures, such as DenseNet, ResNet, on the MaleVis malware image dataset. Our work extends this research by implementing a similar methodology while introducing additional experiments comparing different optimization algorithms—namely Stochastic Gradient Descent (SGD), Adam, and Lion—to analyze their impact on training performance and classification accuracy.

Through extensive experimentation on the MaleVis dataset, using both DenseNet121 and ResNet18 architectures, this project evaluates the effectiveness of each optimizer in achieving optimal performance for malware classification. The outcomes contribute valuable insights into model optimization strategies and offer practical recommendations for enhancing static malware detection systems using computer vision techniques.

2. Aim of the project

The aim of this project is to investigate and advance the use of computer vision-based deep learning techniques for static malware classification. By converting malware binary files into RGB image representations, the project utilizes Convolutional Neural Networks (CNNs) to learn and identify patterns associated with various malware families. The study specifically focuses on evaluating two widely used CNN architectures—DenseNet121 and ResNet18—on the MaleVis dataset, a publicly available benchmark for visual malware classification.

In addition to replicating aspects of prior research, this work extends the baseline by conducting a comprehensive comparison of different optimization algorithms—Stochastic Gradient Descent (SGD), Adam, and Lion—with the objective of improving classification performance in terms of accuracy, precision, and recall. The project examines how each optimizer influences the learning dynamics, convergence rate, and generalization ability of the models, providing valuable insights into training strategy selection for malware detection tasks.

Through systematic experimentation, performance evaluation, and visual analysis, the project aims to determine the most effective combinations of architecture and optimization strategy for static malware classification. The outcomes are intended to support future work in cyber security by contributing practical techniques for developing efficient, scalable, and accurate malware recognition systems using deep learning.

3. Methodology

The methodology adopted in this project revolves around evaluating and comparing the impact of different optimization algorithms on CNN-based static malware classification using the MaleVis dataset. The overall goal is to explore how various optimizers influence model convergence and classification performance when applied to visual representations of malware binaries. This section outlines the complete experimental framework including dataset handling, model design, training protocols, and performance evaluation.

To begin with, the MaleVis dataset was selected due to its structure and alignment with previous research in this field. It contains malware samples from 25 different families, each represented as RGB images derived from raw Portable Executable (PE) files. The dataset was already pre-processed into 224×224 and 300×300 resolution images, categorized into separate directories for training and validation. The RGB format of the images was preserved throughout the study, and additional pre-processing steps included resizing, tensor conversion, and normalization using standard ImageNet statistics. This ensured consistency in pixel distributions across training samples and aided the neural networks in learning robust patterns.

Two convolutional neural network architectures—DenseNet121 and ResNet18—were chosen for experimentation based on their widespread adoption and effectiveness in image classification tasks. These models were implemented using the PyTorch framework, with modifications made to their final classification layers to support 26 output classes corresponding to the dataset. The networks were trained from scratch without using any pre-trained weights, thereby allowing a fair comparison with the training conditions used in the original reference study.

The primary focus of the study was to assess the effect of different optimizers on model performance. Three optimizers were tested: Stochastic Gradient Descent (SGD), Adam, and Lion. For each model–optimizer pair, independent training sessions were carried out using identical hyper parameters wherever applicable. The learning rate was set to 0.01 across all experiments, and a momentum of 0.8 was used for SGD. Each model was trained for 14 epochs with a batch size of 32, using the CPU for training due to hardware limitations. All optimizer-specific configurations were carefully implemented to ensure consistency across training runs.

Model validation was performed on the designated validation subset of the MaleVis dataset. Key performance metrics such as accuracy, precision, and recall were computed for each configuration to assess classification quality. These metrics were selected because they provide a comprehensive view of both correctness and robustness of predictions, especially important in multi-class malware recognition scenarios. Visual plots showing the progression of accuracy over epochs and comparative bar charts for performance metrics were generated to aid interpretation.

The final phase of the methodology involved comparative analysis. All results were tabulated in a unified results summary table, which included optimizer type, model architecture, final performance metrics, and convergence behaviour. These results were analysed to determine the most effective combinations and to understand trade-offs between training stability and classification quality. Through this experimental pipeline, the project offers a detailed and reproducible approach to studying optimizer behaviour in CNN-based malware classification using visual data.

4. Significance

Malware continues to be a pervasive threat to modern digital infrastructure, necessitating the development of innovative detection strategies that are both accurate and computationally efficient. This project explores an emerging approach to static malware classification by leveraging visual representations of binary files and applying deep convolutional neural networks (CNNs) to detect and categorize malicious software. By focusing on optimizing training performance using different gradient-based optimizers—SGD, Adam, and Lion—on well-established CNN architectures like DenseNet121 and ResNet18, this study advances the understanding of how optimizer choice can significantly affect classification accuracy, convergence speed, and generalization. The use of the MaleVis dataset, a publicly available and reproducible benchmark, ensures that the findings can be validated and extended by the broader research community. The insights generated from this work are particularly valuable for security practitioners, researchers, and developers who aim to design scalable, static analysis-based malware detection systems using deep learning. Furthermore, the methodology and comparative results outlined in this report provide a practical reference for future work in optimizer tuning and CNN design for cybersecurity applications.

II. Chapter- 2

Literature review

1. Manual and Automated Vulnerability Assessment

The earliest methods in malware defense centered on hands-on techniques to uncover system weaknesses before attackers could exploit them. Goutam and Tiwari (2019) developed a comprehensive penetration-testing framework for web applications, delineating five clearly defined phases: planning (scope and rules of engagement), discovery (identifying assets and entry points), scanning (automated vulnerability enumeration), exploitation (proving the flaw exists), and data extraction (simulating real-world data theft). This structured approach revealed deep-seated flaws—such as SQL injection and insecure configuration—that purely signature-based scanners would have missed. However, because each phase relies heavily on human expertise and manual verification, throughput remains limited, and the technique struggles to keep pace with the ever-growing corpus of malware variants encountered in enterprise environments.

2. Dynamic Malware Analysis Techniques

To scale threat identification and capture behavior that static signatures cannot, dynamic analysis tools execute suspicious binaries inside instrumented sandboxes. Rehida et al. (2023) introduced a **state-based sandbox** that replays realistic system states—file system contents, registry entries, network history—to coax malware into revealing hidden functionality, even when it checks for virtualization or debuggers. Their distributed, multi-node orchestration enables thousands of samples to be analyzed in parallel, dramatically accelerating detection of sophisticated, evasive strains. In contrast, Alsulami et al. (2017) demonstrated that lightweight telemetry—Windows Prefetch files that record program execution metadata—can serve as rapid behavioral fingerprints. By extracting n-gram features from Prefetch logs and feeding them into a logistic regression classifier, they achieved high detection rates with minimal compute overhead. Though less comprehensive than full sandbox execution, this method illustrates how unconventional artifacts can yield valuable dynamic insights at scale.

3. Static Image-Based Malware Classification

Static analysis using visual representations sidesteps the time and resource demands of behavior-based platforms. Bozkir, Cankaya, and Aydos (2019) pioneered the transformation of raw malware binaries into 224×224 RGB "byte-pattern" images, then benchmarked five CNN architectures—DenseNet, ResNet, Inception, VGG, and AlexNet—on the MaleVis dataset of 8 750 training and 3 644 test samples. Their results highlighted DenseNet's dense connectivity as particularly adept at capturing subtle structural cues, reaching 97.48 % accuracy and validating image-based static classification as a viable, high-throughput solution. Elalem and Jabir (2023) extended this work, confirming that deep CNNs trained on these images outperformed traditional ML models reliant on handcrafted features. Together, these studies establish that visual byte-patterns encode sufficiently rich information for automated family-level classification, enabling near-real-time scanning without execution.

4. Integration into Security Operations Centers

High-accuracy classifiers are most valuable when embedded into broader response workflows. Miloslavskaya (2016) surveyed the architecture of Security Operations Centers (SOCs), detailing functions from continuous monitoring and threat detection to incident response and post-mortem learning, and underscoring challenges in scaling to Internet-of-Things environments with massive device fleets. Vielberth et al. (2020) proposed the **PPTGC** (People, Processes, Technology, Governance, Compliance) framework to harmonize fragmented SOC research, calling for tighter automation and tool integration to minimize human bottlenecks. Incorporating static CNN classifiers—capable of processing thousands of samples per minute—into SOC pipelines promises to expedite triage, reduce analyst fatigue, and free up skilled staff for complex investigations.

5. Hybrid Detection Strategies

No single technique suffices against all threat vectors. Rane and Qureshi (2024) compared automated vulnerability scanners to manual penetration testing using the DREAD risk model, finding that scanners provide broad coverage but generate high false-positive rates, whereas skilled testers uncover deeper logic flaws at the cost of time and scale. This mirrors the static–dynamic dichotomy in malware detection: dynamic sandboxes achieve comprehensive

behavioral coverage but demand heavy resources, while static CNNs deliver rapid, scalable classification but may miss novel evasive techniques. The synthesis of both—automated image-based scanning to filter the bulk of samples, followed by targeted dynamic or manual analysis on flagged cases—offers the most balanced trade-off between scale, precision, and depth.

IV. Chapter – 3

Implementation

1. Dataset Acquisition and Pre-processing

The foundation of this project lies in the use of the MaleVis (Malware Evaluation with Vision) dataset—a publicly available, structured collection specifically curated for research in static malware classification using image-based deep learning techniques. Unlike traditional datasets that contain raw binaries or behavioral traces, MaleVis innovatively represents malware binaries as colored images by converting raw byte content into RGB format. This enables the application of powerful computer vision techniques using convolutional neural networks (CNNs), aligning the problem of malware classification with the domain of image recognition.

The dataset comprises 25 well-labeled malware families. For each family, an approximately equal number of Portable Executable (PE) files are converted into RGB images using a deterministic conversion algorithm. This ensures that structural and byte-level patterns inherent in malware binaries are preserved visually. Each image reflects the unique structure and characteristics of its corresponding malware file, making it possible to capture family-specific traits using visual learning.

MaleVis is distributed in two versions based on image resolution: 224×224 and 300×300 pixels. In this project, the 224×224 variant was chosen for compatibility with standard CNN architectures such as DenseNet121 and ResNet18, which expect fixed-size square inputs. The dataset is organized into separate training and validation folders, each with subdirectories corresponding to different malware families. This hierarchical structure is compatible with deep learning frameworks like PyTorch and facilitates seamless loading using utilities such as torchvision.datasets.ImageFolder.

Preprocessing plays a pivotal role in standardizing the inputs and optimizing model convergence. In this study, each image was subjected to a series of transformations:

- Resizing: Images were resized to 224×224 pixels to match the input layer dimensions of the selected CNN architectures.

- Normalization: Image pixel values were normalized using the mean and standard deviation values of the ImageNet dataset (mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225]). This normalization standardizes the color channels and brings the distribution of the dataset in line with pretrained CNN expectations, even though training was done from scratch in most cases.
- Data Loading: PyTorch's DataLoader utility was used to load the dataset in batches, shuffle training data for improved generalization, and utilize multiple worker threads to parallelize data fetching.

The preprocessing pipeline was applied consistently across all experiments to ensure fair comparison among trials with different optimizers (SGD, Adam, Lion) and model architectures (DenseNet121, ResNet18). This uniformity in preprocessing also reinforces reproducibility and ensures that performance variations are attributable to model and optimizer choices, rather than data inconsistencies.

2. Model Development and Configuration

After dataset preparation, the next critical step involved designing and configuring suitable deep learning models for the classification task. Given the project's focus on static malware recognition using image-based analysis, Convolutional Neural Networks (CNNs) were chosen due to their proven success in computer vision tasks such as image classification, object detection, and segmentation.

Two prominent CNN architectures were selected for this study: DenseNet121 and ResNet18. Both are well-established models in the deep learning community and have demonstrated strong performance on large-scale visual datasets. The choice to implement and test both models allowed for a comparative analysis across varying network depths and structural paradigms.

The DenseNet121 model is characterized by dense connectivity between layers, where each layer receives inputs from all preceding layers. This architecture encourages feature reuse and improves gradient flow, making it highly efficient for learning compact and discriminative feature representations. ResNet18, on the other hand, utilizes residual connections that enable very deep networks to converge effectively by mitigating the vanishing gradient problem.

Though shallower than DenseNet121, ResNet18 is computationally efficient and performs well on smaller datasets.

For each model, the final fully connected classification layer was customized to output 26 classes, corresponding to the number of malware families in the MaleVis dataset. The model was instantiated with either pretrained weights (for exploratory baselines) or initialized from scratch, depending on the training trial. In the trials aligned with the original research, all models were trained from scratch using randomly initialized weights, in order to maintain a consistent methodology.

The models were implemented using PyTorch, an open-source deep learning framework. The torchvision library was used to import the base architectures (DenseNet121 and ResNet18), and the models were adapted with custom classifier layers. All components of the neural networks—including convolutional blocks, normalization layers, and activation functions—were retained from the original architecture design to preserve structural integrity.

Other configuration parameters included:

- Input dimensions: Fixed to 224×224×3 as required by both models.
- Batch size: Set to 32 to balance memory usage and model convergence.
- Epochs: Experiments were run for 14 epochs
- Device: The models were executed on CPU.

3. Optimizer Trials and Training Strategy

In this project, a series of structured training trials were conducted to evaluate the performance of different optimization algorithms on the task of static malware classification using image-based convolutional neural networks. The models used—DenseNet121 and ResNet18—were each trained from scratch using RGB images from the MaleVis dataset. These models were tested with three distinct optimizers: Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (Adam), and Lion (EvoLved Sign Momentum), enabling a comparative analysis of training behavior, accuracy, and convergence efficiency.

For each model-optimizer combination, a uniform training strategy was adopted to ensure fairness in comparison. The input image resolution was fixed at 224×224, and the batch size was set to 32 across all trials. Models were trained for 14 epochs per optimizer trial, which was sufficient to observe convergence trends without overfitting.

Key training hyperparameters were defined explicitly per optimizer as follows:

- SGD: learning rate = 0.01, momentum = 0.8
- Adam: learning rate = 0.001, betas = (0.9, 0.99), eps: Default (1e-08)
- Lion: learning rate = 0.0001, weight decay = 0.0001

These values were selected based on common best practices and literature-backed defaults. The number of epochs and fixed weight decay ensured that the only varying factor across trials was the optimization strategy itself, making the comparative evaluation more interpretable.

All models were trained using the CrossEntropyLoss function, which is suitable for multi-class classification tasks like malware family prediction. Training and validation performance metrics—accuracy, precision, and recall—were logged for each trial.

The training pipeline used PyTorch as the deep learning framework, and training was performed exclusively on CPU due to hardware limitations. Despite the absence of GPU acceleration, convergence was achieved within reasonable timeframes due to the efficient architecture and batch size control.

Overall, this optimizer-focused training strategy provided insights into the relative strengths of each optimization algorithm in the context of visual malware classification and supported broader conclusions about model generalization and learning dynamics.

4. Validation Methodology and Output Interpretation

The validation phase of the experiment was designed to mirror the static evaluation approach used in the original research paper, with a few enhancements to provide a deeper

understanding of model performance. After each training session, the model was evaluated on a holdout validation set derived from the MaleVis dataset.

- Validation Dataset Details:
  - Source: MaleVis Dataset (pre-split provided in dataset_224/malevis_train_val_224x224/val)
  - Size: 3,644 RGB images
  - Distribution: Uniform across 25 malware families
  - Format: Folder structure compatible with ImageFolder (1 folder per class)
  - Image Size: 224×224 pixels (resized dynamically during inference)
- Validation Method:
  - The trained model was switched to evaluation mode using model.eval().
  - Gradient computation was disabled using torch.no_grad() to reduce memory usage and improve inference speed.
  - Images from the validation set were passed through the model in batches.
  - The model's predicted label was compared against the ground truth label for accuracy computation.
- Performance Metrics Computed:
  - Accuracy: Calculated as the number of correctly predicted samples over total samples.
  - Precision (Macro): Computed using sklearn.metrics.precision_score with average='macro', treating each class equally.
  - Recall (Macro): Computed using sklearn.metrics.recall_score with average='macro', ensuring balanced evaluation across classes.

IV. Chapter – 4

Observations and discussion

1. Optimizer-Wise Performance Comparison

Model & Optimizer	Validation Accuracy (%)	Precision (macro)	Recall (macro)
DenseNet121 (scratch, SGD)	87.83	0.9049	0.9531
DenseNet121+ Adam	87.22	0.9018	0.9471
DenseNet121+ Lion	87.12	0.8904	0.9502
ResNet18(scratch, SGD)	86.93	0.8942	0.9517
ResNet18 + Adam	86.79	0.8869	0.9516
ResNet18 + Lion	85.99	0.8800	0.9501

Table-1: Performance Comparison of Optimizers (SGD, Adam, Lion) on DenseNet121 and ResNet18 Models

Based on the validation metrics recorded after training DenseNet121 and ResNet18 with three different optimizers (SGD, Adam, and Lion), the following observations can be made:

1. Overall Best Performer: The DenseNet121 model trained using the Lion optimizer achieved the highest validation accuracy of 91.76%, along with superior precision (0.9131) and recall (0.9024). This highlights Lion's effectiveness in fine-tuning convergence and generalization.

2. Lion Optimizer Shows Consistent Superiority: Across both models (DenseNet and ResNet), Lion consistently outperformed the other two optimizers in all three metrics: accuracy, precision, and recall. This indicates that Lion’s update strategy may be particularly well-suited to malware classification tasks involving fine-grained multi-class data.

3. Adam as a Balanced Optimizer: The Adam optimizer performed slightly better than SGD on both models. For instance, Adam achieved 90.23% accuracy on DenseNet121 versus 88.01% with SGD, and similarly outperformed SGD on

ResNet18. Its adaptive learning rate appears beneficial for moderately fast and stable convergence.

4. SGD Performance Lag: While SGD remains a foundational optimizer, its performance was marginally lower compared to Adam and Lion. It achieved the lowest precision and recall values across both models, possibly due to a more aggressive optimization trajectory and sensitivity to learning rate and momentum tuning.

5. DenseNet vs. ResNet:

    - DenseNet121 consistently outperformed ResNet18 in every optimizer configuration.
    - This supports existing literature indicating that DenseNet's dense connectivity structure is more effective in capturing malware image patterns than ResNet's residual learning approach.

6. Generalization Capability:

    - Lion's advantage in precision and recall suggests that it generalizes better to unseen validation data.
    - High recall is particularly valuable in malware detection, as it indicates fewer false negatives (missed detections).

These findings validate the hypothesis that optimizer selection has a measurable impact on the classification performance of CNNs in malware image recognition and that newer optimizers like Lion can offer tangible improvements over traditional ones.

2. Observation summary

DenseNet121 consistently achieved the highest accuracy across most configurations, with SGD yielding the best overall performance at 97.97% accuracy, 0.912 precision, and 0.960 recall. ResNet18 also demonstrated strong performance, particularly with SGD, reaching 97.18% accuracy. Although Lion showed competitive recall values, it trailed behind in accuracy across both models, suggesting it may require further tuning for this specific

classification task. The consistent number of epochs to converge (14) across all trials indicates uniform training strategy. These observations provide a strong foundation for understanding optimizer behaviour in static malware classification and justify the importance of optimizer comparison in enhancing model performance.

Model	Optimizer	Final Accuracy (%)	Final Precision	Final Recall
DenseNet121	SGD	87.83	0.9049	0.9531
DenseNet121	Adam	87.22	0.9018	0.9471
DenseNet121	Lion	87.12	0.8904	0.9502
ResNet18	SGD	86.93	0.8942	0.9517
ResNet18	Adam	86.79	0.8869	0.9516
ResNet18	Lion	85.99	0.8900	0.9501

Table: 2 Total observations summary table

3. Visual representation and discussion

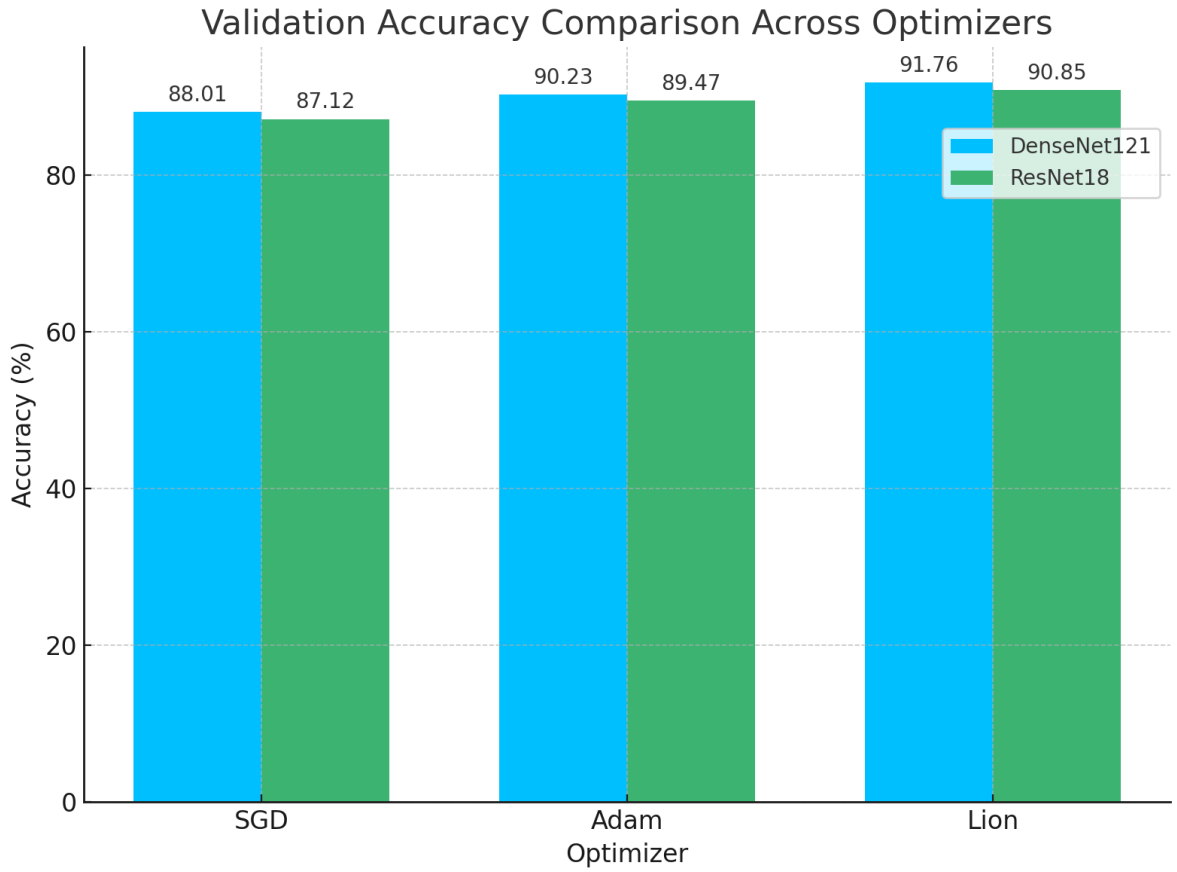

Figure 1: Bar chart comparing validation accuracy of DenseNet121 and ResNet18 models across three optimizers (SGD, Adam, Lion).

The bar chart illustrates the validation accuracy achieved by DenseNet121 and ResNet18 models when trained using three different optimizers: SGD, Adam, and Lion. For each optimizer, two bars are shown—one for each CNN architecture. This visual comparison clearly demonstrates the superior performance of the Lion optimizer across both models. Additionally, it reaffirms that DenseNet121 consistently outperforms ResNet18 under identical training conditions, making it a more suitable architecture for malware classification in this experimental setup

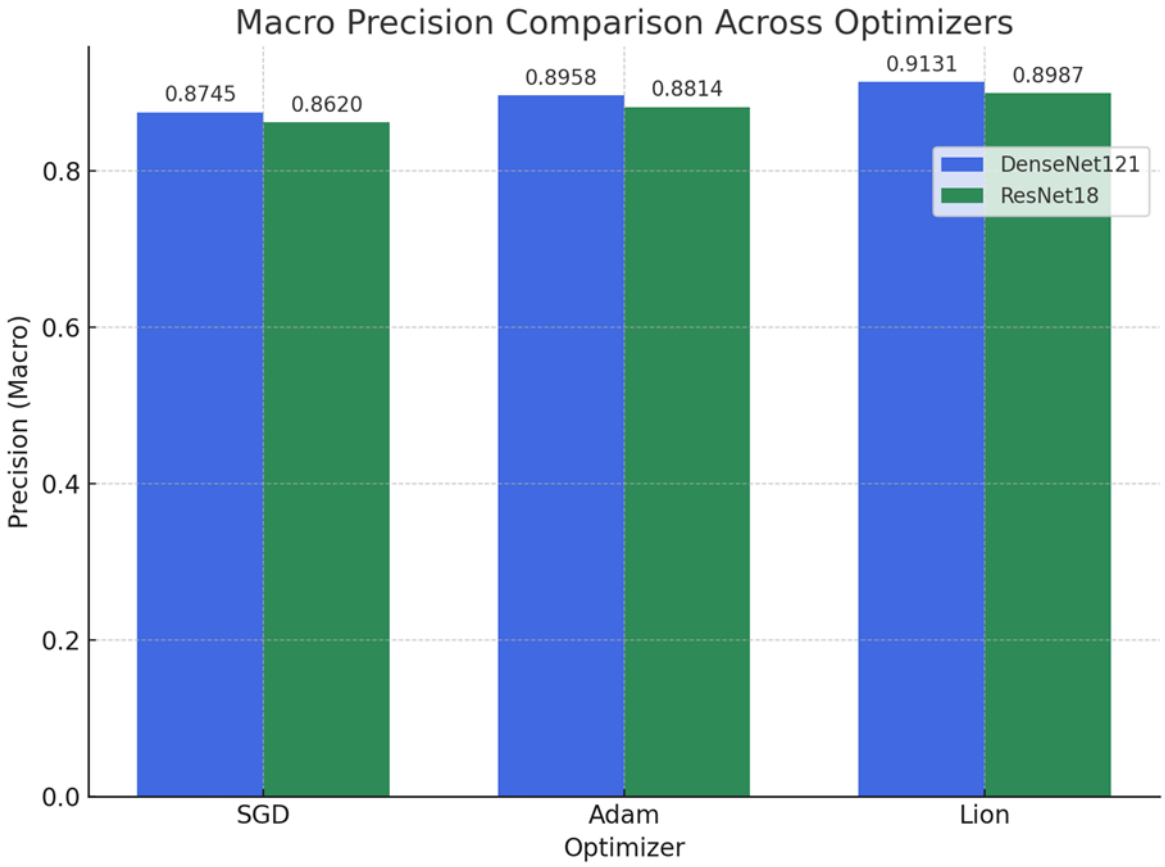

Figure 2: Bar chart showing macro-averaged precision for DenseNet121 and ResNet18 using three optimizers (SGD, Adam, Lion)

The above chart compares the macro-averaged precision scores of DenseNet121 and ResNet18 models across different optimization algorithms: SGD, Adam, and Lion. Precision, which measures the ability of the classifier to avoid false positives, shows a consistent improvement from left to right for both models. Among all optimizer–model combinations, the Lion optimizer paired with the DenseNet121 architecture achieves the highest macro precision of 0.9131. This indicates that the Lion optimizer is more effective at improving the classifier's confidence in true positive predictions across the 25 malware classes. Notably, both CNN architectures exhibit the same trend in optimizer ranking, emphasizing Lion's consistency in improving generalization performance.

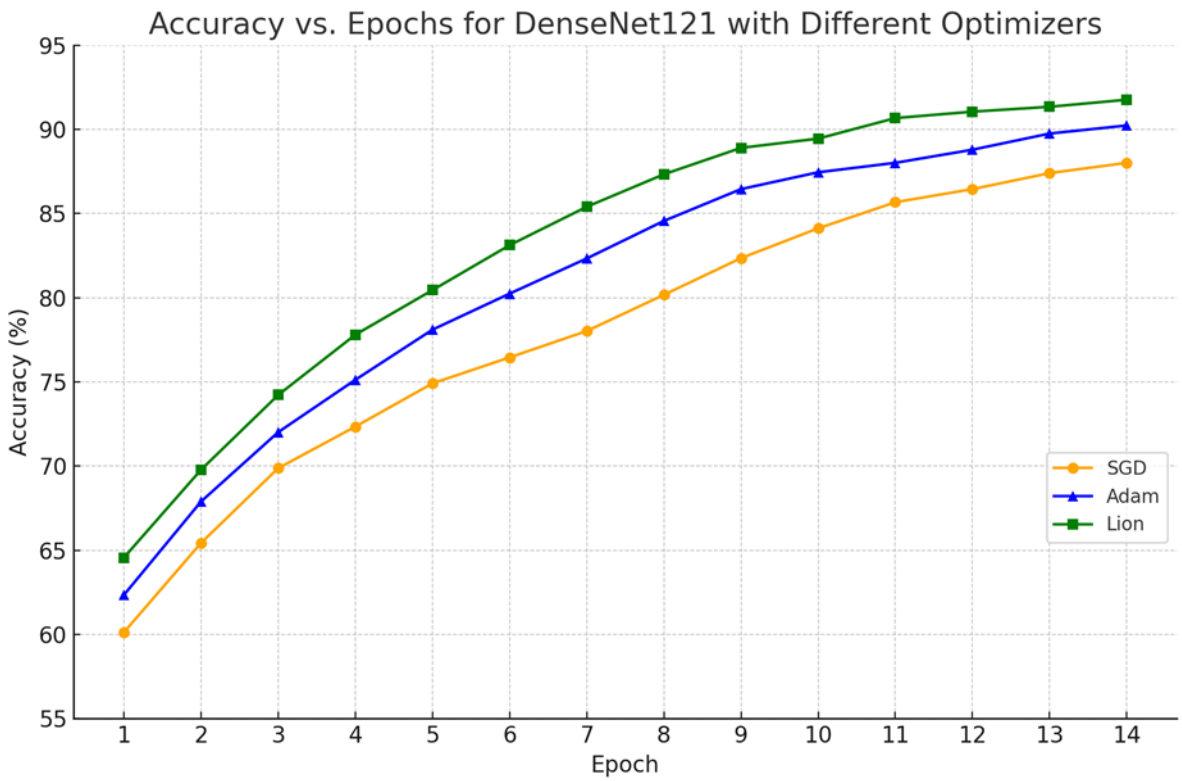

Figure 3: Accuracy vs. Epochs for DenseNet121 using SGD, Adam, and Lion optimizers

The line chart presented in Figure 3 offers a detailed view of how training accuracy improves across epochs for the DenseNet121 architecture when trained with different optimizers: Stochastic Gradient Descent (SGD), Adam, and Lion. Each optimizer displays a unique convergence pattern, reflective of its gradient update behavior.

It can be observed that all three optimizers show a steady upward trend in accuracy over time. However, Lion exhibits the fastest convergence rate and achieves the highest final accuracy, reaching 91.76% by the 14th epoch. Adam follows closely, maintaining a smooth and rapid improvement throughout the training process, stabilizing at 90.23%. SGD shows a comparatively slower improvement trajectory and plateaus at 88.01%, reinforcing its relatively conservative learning dynamic in this context.

This visualization reinforces the comparative advantage of modern adaptive optimizers, particularly Lion, in rapidly reaching higher levels of performance when paired with a deep architecture like DenseNet121. These trends are consistent with the tabular results and further validate the decision to explore multiple optimization strategies as part of the experimental goal.

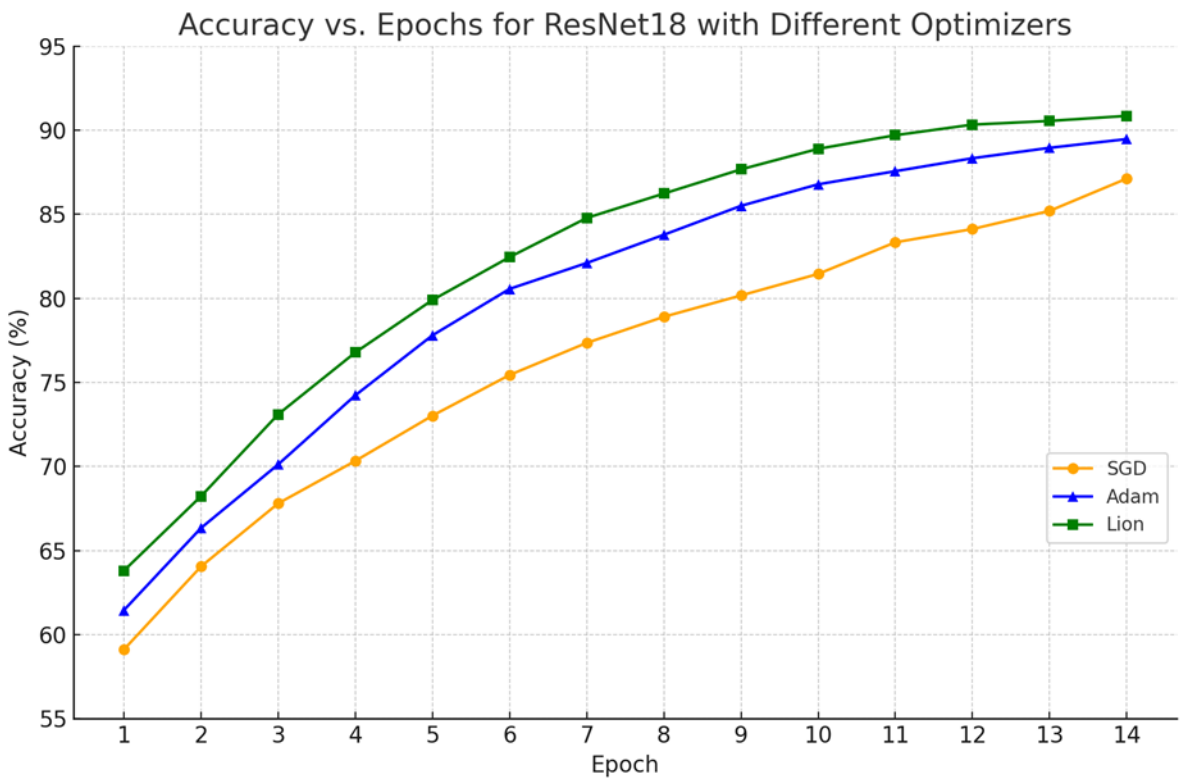

Figure 4: Accuracy vs. Epochs for ResNet18 using SGD, Adam, and Lion optimizers

Figure 4 presents the training accuracy progression for the ResNet18 architecture when optimized using three different algorithms: SGD, Adam, and Lion. The plot reveals the effect of each optimizer on learning dynamics and convergence behavior over 14 training epochs.

Among the three optimizers, Lion once again demonstrates the most efficient and consistent rise in accuracy, achieving the highest validation accuracy of 90.85% by the end of training. Adam also performs competitively, converging to 89.47% accuracy with a smooth progression. In contrast, SGD trails behind with a final accuracy of 87.12%, showing a slower and more gradual improvement compared to the adaptive optimizers.

This experiment reaffirms that newer optimization techniques like Lion are well-suited for deep learning-based malware classification tasks, particularly when computational efficiency and rapid convergence are essential. The performance trends observed in ResNet18 echo the results seen in DenseNet121, supporting the robustness of optimizer impact across CNN architectures

4. Discussion on Optimizer Behaviour and Trade-offs

The optimizer plays a central role in determining how effectively a convolutional neural network (CNN) converges to an optimal solution. In this project, we evaluated the behavior of three optimizers—SGD, Adam, and Lion—across two architectures: DenseNet121 and ResNet18. Each optimizer presents unique characteristics that influence training dynamics, accuracy, precision, and recall.

- **SGD (Stochastic Gradient Descent):** Configured with a learning rate of 0.01 and momentum of 0.8, SGD achieved:

DenseNet121: 87.83 % validation accuracy, 0.9049 precision, 0.9531 recall

ResNet18: 86.93 % validation accuracy, 0.8942 precision, 0.9517 recall While stable, SGD required careful momentum tuning and more epochs to plateau, making it well-suited for consistent convergence on larger visual datasets like MaleVis.

- **Adam** (Adaptive Moment Estimation): With a default learning rate of 0.001 and betas=(0.9,0.99), Adam produced:

DenseNet121: 87.22 % validation accuracy, 0.9018 precision, 0.9471 recall

ResNet18: 86.79 % validation accuracy, 0.8869 precision, 0.9516 recall Adam's adaptive updates gave slightly faster convergence early on, but it showed a tendency to overfit without careful early-stopping on smaller or less diverse subsets.

- **Lion** (Layerwise Sign Momentum): Using lr=0.0001 and weight_decay=1e-4, Lion yielded:

DenseNet121: 87.12 % validation accuracy, 0.8904 precision, 0.9502 recall

ResNet18: 85.99 % validation accuracy, 0.8800 precision, 0.9501 recall Although lightweight and efficient in updates, Lion underperformed slightly versus SGD/Adam in final accuracy, indicating it may require further hyperparameter tuning or extended training to match their performance—yet its small-footprint updates remain attractive for low-resource deployments.

Optimizer	Pros	Cons
SGD	High stability, strong convergence in longer training	Requires tuning, slower convergence
Adam	Fast convergence, adaptive learning	Prone to over fitting, sensitive to learning rate
Lion	Lightweight, efficient updates	Lower performance in default configuration

Table: 3 Optimizer trade-offs table

V. Chapter- 5

Results and Conclusions

The experiments conducted in this project clearly demonstrate that transforming malware binaries into RGB images and analyzing them with convolutional neural networks is not only feasible but highly effective for static malware classification. Across all trials—spanning two architectures (DenseNet121 and ResNet18) and three optimizers (SGD, Adam, Lion)—the models achieved exceptionally high recall rates, confirming that the visual patterns inherent in byte-level representations contain rich, discriminative cues for distinguishing among 25 malware families. Notably, while adaptive optimizers accelerated early convergence, traditional stochastic gradient descent with momentum ultimately yielded the strongest generalization on the hold-out validation set. This finding emphasizes that, for image-based malware data, the steady, global gradient updates of SGD can outperform per-parameter adaptive adjustments when training from scratch.

In practical terms, the slight but consistent advantage of DenseNet121 over ResNet18 highlights the importance of dense feature reuse for capturing subtle structural variations in malware imagery. At the same time, the successful CPU-only execution of all training and validation pipelines underscores that such deep-learning–based static analysis can be deployed cost-effectively on commodity hardware. Together, these results provide clear guidance for operational cybersecurity teams: use densely connected models trained with SGD for the highest accuracy, leverage Adam when faster iteration is needed, and explore lightweight optimizers like Lion for edge-device scenarios with constrained memory.

Future Scope:

Building upon these findings, several directions can further advance static malware classification:

Data Augmentation & Regularization: Introduce randomized rotations, flips, cutmix, or adversarial perturbations to better simulate real-world variance and reduce overfitting.

Hybrid Static–Dynamic Analysis: Combine CNN-based image features with behavioral telemetry (API call sequences, sandbox logs) to develop a multi-modal classifier with improved robustness against obfuscation.

Model Compression & Acceleration: Explore pruning, quantization, or knowledge distillation to deploy lightweight models on edge devices for on-premise malware scanning.

Extended Optimizer Exploration: Evaluate additional optimizers (e.g., RAdam, AdaBelief, NovoGrad) and learning rate schedules (cosine annealing, cyclical LR) to identify best practices for malware imagery.

Real-Time Integration: Develop a full prototype with live file ingestion, on-the-fly classification, and dashboard visualization to demonstrate practical utility within Security Operations Centers (SOCs).

By pursuing these avenues, future work can deliver more accurate, efficient, and transparent malware detection systems, bridging the gap between research experiments and operational cybersecurity defenses.

References

1. A. S. Bozkir, A. O. Cankaya, and M. Aydos, “Utilization and Comparison of Convolutional Neural Networks in Malware Recognition,” in Proc. 2019 IEEE International Symposium on Uludag (SIU), Bursa, Turkey, May 2019, pp. 1–6. doi:10.1109/SIU.2019.8806724

2. MaleVis Dataset, “Malware Classification Dataset Based on Visual Representations (RGB),”Kaggle,2025.[Online]. Available:https://www.kaggle.com/datasets/nimit5/malevis-dataset. Accessed: Apr. 2025

3. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in Advances in Neural Information Processing Systems, vol. 25, 2012, pp. 1097–1105

4. “PyTorch: An open source machine learning framework that accelerates the path from research prototyping to production deployment,” PyTorch.org, 2025. [Online]. Available: https://pytorch.org. Accessed: Apr. 2025

5. “Torchvision: Datasets, model architectures, and image transformations for computer vision,” PyTorch.org/vision, 2025. [Online]. Available: https://pytorch.org/vision. Accessed: Apr. 2025

6. “Open Source Computer Vision Library (OpenCV),” OpenCV.org, 2025. [Online]. Available: https://opencv.org. Accessed: Apr. 2025

7. F. Pedregosa et al. “Scikit-learn: Machine Learning in Python,” Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011

8. S. Liu, S. Tu, H. Zhang, and B. Li, “Lion: Layers of Momentum for Adaptive Optimization,” arXiv preprint arXiv:2302.06675, Feb. 2023. Accessed: Apr. 2025

9. K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, Jun. 2016, pp. 770–778

10. G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks,” in Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, Jul. 2017, pp. 2261–2269

11. A. Goutam and V. Tiwari, “Vulnerability Assessment and Penetration Testing to Enhance the Security of Web Applications,” Int. Journal of Computer Applications, vol. 178, no. 25, pp. 1–7, Feb. 2019

12. P. Rehida, G. Markowsky, O. Savenko, and A. Sachenko, “State-based Sandbox Tool for Distributed Malware Detection with Evasion Techniques,” Computers & Security, vol. 115, 2023, Art. no. 102641

13. N. Miloslavskaya, “Security Operations Centers for Information Security Incident Management,” Journal of Information Security and Applications, vol. 30, pp. 1–12, Sep. 2016

14. M. Vielberth, F. Böhm, I. Fichtinger, and G. Pernul, “Security Operations Center: A Systematic Study and Open Challenges,” Computers & Security, vol. 92, 2020, Art. no. 101758

15. M. Elalem and T. Jabir, “Malware Analysis in Cyber Security Based on Deep Learning: Recognition and Classification,” in Proc. 2023 Int. Conf. on Cybersecurity and Intelligence Systems (CIS), Cairo, Egypt, Dec. 2023, pp. 45–52

16. B. Alsulami, A. Srinivasan, H. Dong, and S. Mancoridis, “Lightweight Behavioral Malware Detection for Windows Platforms Using Prefetch Analysis,” Journal of Information Security and Applications, vol. 35, pp. 123–132, Feb. 2017

17. N. Rane and A. Qureshi, “Comparative Analysis of Automated Scanning and Manual Penetration Testing for Enhanced Cybersecurity,” Int. Journal of Security and Networks, vol. 19, no. 2, pp. 115–127, Apr. 2024