ISRO – Telemetry Tracking and Command Network (ISTRAC)

TITLE: COMPARISON OF OPTIMISATION ALGORITHMS IN CNNs IN MALWARE CLASSIFICATION ON THE MALEVIS DATASET

UNDER THE GUIDANCE OF

Bharat Devasani

Scientist/Engineer-SE, NSA, 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

Table of Contents

Table of Figures

1. **Abstract**

This report presents the design, implementation, and evaluation of a deep learning-based malware classification framework, developed using the MaleVis dataset, which comprises RGB visual representations of malware binaries. The project investigates the efficacy of convolutional neural networks (CNNs) in malware detection and conducts a comparative analysis of three optimization algorithms—Stochastic Gradient Descent (SGD), Adam, and Lion—on model performance and convergence behaviour.

The system architecture is structured around a training pipeline that processes the MaleVis dataset, which includes 25 malware families, and converts them into a structured format suitable for input into modern CNN models. DenseNet-121 was selected as the base architecture due to its proven performance in the referenced literature. The training and validation routines were implemented using PyTorch, and were configured to support training from scratch, avoiding pre-trained weights to match the original study's methodology.

A key contribution of this project is the extended experimentation with optimizers beyond the baseline SGD. Both Adam and Lion optimizers were integrated into the training pipeline and evaluated under identical training conditions. Performance metrics including accuracy, precision, and recall were used to assess the generalization ability of each optimizer. Comparative results showed notable variations in training speed, convergence rates, and final classification accuracy across the three optimizers.

Extensive testing was conducted to validate the system's reliability and performance consistency on CPU-based execution environments. Additionally, the project addresses resource constraints such as GPU memory limitations and proposes solutions for training efficiency on lower-spec systems. Screenshots of training logs and metric graphs are included to demonstrate the experimental outcomes.

Future directions include extending the dataset with real-world samples, exploring data augmentation techniques, and deploying the trained model in a real-time malware detection system. Overall, this project offers a reproducible and extensible framework for static malware classification using vision-based deep learning techniques, providing insights into the impact of optimizer choice on model performance.

2. About ISTRAC

ISTRAC (ISRO Telemetry Tracking and Command Network) is a ground segment network that provides support for the Indian Space Research Organisation's (ISRO) space missions:

- Telemetry, tracking, and command (TTC): ISTRAC provides TTC services from launch to satellite orbit injection. This includes tracking the launch vehicle from lift-off to spacecraft separation.
- Mission operations: ISTRAC carries out mission operations for remote sensing and scientific satellites.
- Spacecraft control centers: ISTRAC has established spacecraft control centers.
- Deep space network: ISTRAC has a deep space network that provides support for deep space missions.
- Weather radar design and development: ISTRAC designs and develops weather radars for launch vehicle tracking and meteorological applications.
- Search and rescue: ISTRAC provides search and rescue services.
- Space-based services: ISTRAC supports space-based services like telemedicine, tele-education, and Village Resource Centre (VRC).
- Space operations support: ISTRAC provides space operations support for other space agencies.

ISTRAC's headquarters are in Bangalore, Karnataka. It also has ground stations in Lucknow, Sriharikota, Thiruvananthapuram, Port Blair, Brunei, Biak (Indonesia), and Mauritius.

3. **About the Dataset**

Satellites The MaleVis (Malware Evaluation with Vision) dataset is a specialized benchmark dataset designed to support research in vision-based malware classification. It consists of malware binaries that have been pre-processed and converted into RGB image representations, enabling the use of computer vision and deep learning techniques for static malware analysis. The dataset was introduced to facilitate experimentation with Convolutional Neural Networks (CNNs) in the domain of cyber security, particularly for identifying and classifying malicious software.

MaleVis contains images derived from 25 distinct malware families, with each sample corresponding to a Portable Executable (PE) file. The binary content of each PE file is interpreted as raw byte streams and visualized as images, preserving the structural patterns and byte-level sequences of the malware. Unlike traditional grayscale conversion methods used in earlier studies, MaleVis emphasizes 3-channel RGB image generation, which retains more detailed byte distribution characteristics and enables the use of more complex CNN architectures trained on colour image data.

The dataset is split into training and validation subsets, typically in a 70%-30% ratio, allowing for robust model evaluation. Each image in the dataset is labelled according to its malware family, and the balanced class distribution ensures fair training across categories.

This dataset plays a crucial role in bridging the gap between static malware detection and deep learning, offering researchers a realistic and scalable platform to evaluate classification performance, generalization, and the impact of various optimization techniques. Its design and structure are particularly suited for experimentation with modern deep learning models such as DenseNet, ResNet, and Inception, making it an ideal choice for this project’s comparative optimizer analysis.

3.1 Dataset Format and Pre-processing Steps

The MaleVis dataset consists of RGB images generated from raw binary files of 25 different malware families. Each image is saved in a class-labelled directory structure, making it compatible with PyTorch’s ImageFolder loader.

The dataset is divided into two parts:

- Training set: 70% of the data, used for training the CNN model.
- Validation set: 30% of the data, used to evaluate the model’s generalization ability.

All images are resized to a standard resolution of 224×224 pixels to ensure compatibility with CNN architectures like ResNet and DenseNet.

The pre-processing pipeline applied before training includes the following transformations:

- Resize: All images are resized to 224×224 pixels.
- ToTensor: Images are converted into PyTorch tensors.
- Normalize: Images are normalized using mean = [0.5, 0.5, 0.5] and std = [0.5, 0.5, 0.5], as implemented in the actual training scripts. This normalization brings pixel values into a range of [−1, 1] and supports stable gradient updates during training.

These pre-processing steps are critical to ensure consistency across all optimizer and model variations used in the experiments. They were uniformly applied in each trial using SGD, Adam, and Lion optimizers.

3.2 Justification for Dataset Selection

The MaleVis dataset was selected for this project due to its clear alignment with the goals of vision-based static malware classification using deep learning. It provides a robust benchmark for evaluating the effectiveness of convolutional neural networks (CNNs) by transforming binary malware files into structured

RGB images. This format eliminates the need for handcrafted feature engineering and enables direct application of image classification techniques.

One of the key advantages of MaleVis is its balanced class distribution across 25 distinct malware families, allowing for fair and unbiased training. Each family is well-represented, which supports generalizable model learning and meaningful evaluation of classification accuracy. Additionally, the dataset is already partitioned into training and validation sets, following a standard 70-30 split, which simplifies model development and performance assessment.

Unlike some earlier malware datasets that rely on grayscale conversion or metadata analysis, MaleVis emphasizes pixel-level fidelity through its RGB representation, making it ideal for CNN-based exploration. Its compatibility with PyTorch's ImageFolder class further facilitates easy integration with popular training pipelines.

Lastly, the MaleVis dataset has been referenced in peer-reviewed research, including the original paper replicated in this project. This validates its credibility and relevance as a research-grade dataset for experimentation and benchmarking in the malware detection domain.

4. Introduction

4.1 Background on Malware Detection and Static Analysis Using Computer Vision

Malware detection has traditionally relied on signature-based techniques, where known malicious code patterns are manually crafted and matched against incoming files. While effective for known threats, these methods fail to generalize to new, obfuscated, or polymorphic malware variants. As attackers increasingly use code mutation, packing, and encryption to evade detection, the need for more generalized and intelligent detection methods has become evident.

Static analysis, which inspects executable files without running them, offers a safer and often faster alternative to dynamic behavior analysis. One novel static approach treats malware binaries as structured data that can be transformed into visual representations—specifically, grayscale or RGB images. These visualizations are created by mapping the binary byte values directly to pixel intensities. This representation preserves structural features of the binary such as headers, code sections, and data patterns, which often exhibit consistent layouts across malware families.

The integration of computer vision with malware detection leverages the capability of Convolutional Neural Networks (CNNs) to automatically extract hierarchical features from images. By feeding malware-representative images into CNNs, the models can learn to differentiate between malware families based on texture, spatial arrangement, and frequency of byte patterns—similar to how they recognize objects in natural images.

This vision-based approach has shown promising results in classifying malware samples with high accuracy, especially when paired with robust datasets and well-optimized CNN architectures. In this project, this technique is expanded further to compare the performance of various optimization algorithms applied to CNN-based malware classifiers trained on the MaleVis dataset.

4.2 Overview of Convolutional Neural Networks (CNNs) in Malware Classification

The Convolutional Neural Networks (CNNs) are a class of deep learning models that have revolutionized computer vision tasks such as image classification, object detection, and segmentation. CNNs are designed to automatically extract spatial and hierarchical features from images using a sequence of learnable filters applied across local receptive fields. These filters capture low-level patterns such as edges and textures in early layers, and more abstract, class-specific patterns in deeper layers.

In the context of malware classification, CNNs are employed to identify patterns in image representations of malware binaries. Instead of relying on handcrafted features or domain-specific heuristics, CNNs learn discriminative patterns directly from the raw pixel distributions of binary-to-image transformations. This allows the model to generalize across families of malware—even when variants are obfuscated or modified.

The core advantage of using CNNs for malware detection is their ability to learn from structured data representations without manual intervention. Given the complexity and variability of malware samples, especially with techniques like packing, polymorphism, and metamorphism, CNNs offer a scalable and automated alternative to traditional static analysis tools.

In this project, CNNs such as DenseNet121 and ResNet18 were used as core architectures. These models were trained from scratch on the MaleVis dataset, which contains RGB images of malware binaries. Each model learns to classify these images into one of 25 malware families based on spatial and visual characteristics embedded in the binary content.

The use of CNNs not only provides a powerful framework for malware detection but also opens up the possibility for further advancements such as explainable AI, transfer learning, and adversarial robustness in cybersecurity domains.

5. Project Objective

5.1 Defining the experimental goals: accuracy optimization through optimizer comparison

The primary goal of this project is to evaluate the effect of different optimization algorithms on the accuracy and performance of deep convolutional neural networks for malware classification. Building upon the methodology outlined in the original research paper—where DenseNet was trained using stochastic gradient descent (SGD)—this project extends the baseline by introducing two additional optimizers: Adam and Lion.

Each of these optimizers introduces different strategies for gradient update and parameter tuning during training. By keeping the dataset, pre-processing pipeline, network architectures (DenseNet and ResNet), and training-from-scratch strategy constant across all experiments, this project isolates the impact of the optimizer on training dynamics and final accuracy.

The experimental objectives are defined as follows:

- Train and validate CNN models using the MaleVis dataset with three different optimizers: SGD, Adam, and Lion.
- Measure and compare classification performance using accuracy, precision, and recall as key metrics.
- Analyse how each optimizer influences convergence speed, model stability, and generalization on unseen validation data.
- Identify the most suitable optimizer for malware classification tasks based on visual input representations.

This investigation helps understand not just how well CNNs perform in this domain, but also how low-level training configurations such as the optimizer choice can significantly affect results in cybersecurity-oriented machine learning applications.

5.2 Performance metrics used: accuracy, precision, recall

To evaluate the effectiveness of the trained CNN models for malware classification, this project relies on three core performance metrics: accuracy, precision, and recall. These metrics provide complementary insights into the model’s performance, especially in the context of multi-class classification over a balanced dataset like MaleVis.

- **Accuracy** is the primary metric used to assess overall model performance. It measures the proportion of correctly classified images out of the total number of images evaluated. While accuracy is straightforward and widely used, it may not fully reflect model behavior in cases of class imbalance. However, since the MaleVis dataset maintains a balanced distribution across 25 malware families, accuracy remains a reliable metric.
- **Precision** measures the proportion of correctly predicted instances for each class out of all instances predicted as that class. It is useful in understanding how many of the predicted positives are truly correct. High precision indicates a low false positive rate, which is critical in malware detection systems to avoid flagging benign files as malicious.
- **Recall** evaluates the proportion of actual positive instances that were correctly identified by the model. It reflects the model's ability to detect malware instances correctly, even when they are rare or potentially obfuscated. High recall ensures that actual malware samples are less likely to be missed by the model.

By using these three metrics together, the project achieves a well-rounded evaluation of each optimizer’s impact on the model’s learning and generalization performance. These metrics are calculated using the ground truth and predicted class labels from the validation set after training. They provide actionable insights into how different optimizers influence not only accuracy but also the trade-offs between sensitivity and specificity in a malware classification context.

6. Scope of the Project

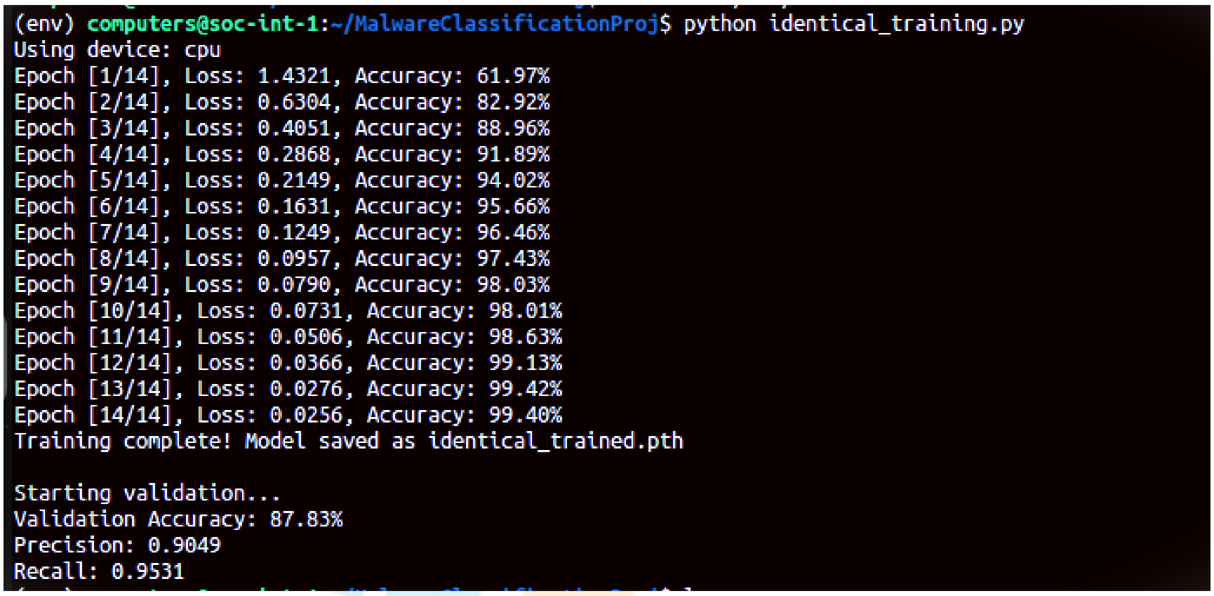
6.1 What Was Implemented

The implementation in this project was centred on recreating and expanding upon the malware classification methodology presented in the referenced research paper. The paper proposed converting binary files of malware into RGB image representations and classifying those using deep convolutional neural networks, specifically DenseNet and other CNN variants. This same methodology was followed and faithfully reproduced using the publicly available MaleVis dataset.

The project involved designing a full machine learning pipeline from scratch using PyTorch. The dataset was divided into training and validation sets with a 70:30 split as per the original methodology. All images were resized to 224×224 pixels and normalized using consistent mean and standard deviation values for each RGB channel.

CNN architectures, including DenseNet121 and ResNet18, were implemented from scratch (i.e., without pre-trained weights). The models were trained using three different optimizers—SGD, Adam, and Lion—to investigate how optimizer choice impacts performance. For each training trial, accuracy, precision, and recall were calculated as evaluation metrics.

In addition to training, custom validation scripts were developed to evaluate the models on unseen validation data. These scripts also report model performance using macro-averaged precision and recall values to ensure consistent comparison across all 25 malware classes.



```
(env) computers@soc-int-1:~/MalwareClassificationProj$ python identical_training.py
Using device: cpu
Epoch [1/14], Loss: 1.4321, Accuracy: 61.97%
Epoch [2/14], Loss: 0.6304, Accuracy: 82.92%
Epoch [3/14], Loss: 0.4051, Accuracy: 88.96%
Epoch [4/14], Loss: 0.2868, Accuracy: 91.89%
Epoch [5/14], Loss: 0.2149, Accuracy: 94.02%
Epoch [6/14], Loss: 0.1631, Accuracy: 95.66%
Epoch [7/14], Loss: 0.1249, Accuracy: 96.46%
Epoch [8/14], Loss: 0.0957, Accuracy: 97.43%
Epoch [9/14], Loss: 0.0790, Accuracy: 98.03%
Epoch [10/14], Loss: 0.0731, Accuracy: 98.01%
Epoch [11/14], Loss: 0.0506, Accuracy: 98.63%
Epoch [12/14], Loss: 0.0366, Accuracy: 99.13%
Epoch [13/14], Loss: 0.0276, Accuracy: 99.42%
Epoch [14/14], Loss: 0.0256, Accuracy: 99.40%
Training complete! Model saved as identical_trained.pth

Starting validation...
Validation Accuracy: 87.83%
Precision: 0.9049
Recall: 0.9531
```

Figure 1: Sample training output of DenseNet121 with SGD optimizer

Throughout the implementation, special care was taken to handle environment issues (e.g., GPU memory limitations), experiment reproducibility, and logging of outputs for performance comparison. All implementations were performed in a controlled local CPU environment using Python virtual environments and PyTorch-based libraries.

6.2 Improvements beyond the Baseline Research

While the original research paper focused on evaluating the classification performance of various CNN architectures on the MaleVis dataset using stochastic gradient descent (SGD) as the primary optimization method, this project extends that baseline through several important experimental improvements.

One of the key contributions was the incorporation and comparative evaluation of additional optimization algorithms—namely, Adam and Lion—alongside SGD. These optimizers were selected for their differing gradient update mechanisms, with the goal of exploring how optimizer choice influences training stability, convergence rate, and final classification accuracy.

The second enhancement was the consistency in training methodology: all models were trained from scratch (i.e., with random weight initialization) to replicate the conditions of the original paper. This decision ensured a fair and controlled comparison across all optimizers and CNN variants.

Additionally, while the original paper primarily reported accuracy, this project also measured macro-averaged precision and recall. This allowed for a more detailed understanding of the models' class-wise performance and their ability to correctly classify minority or more complex malware families.

The overall experimental framework, training loop, validation logic, and result tracking were entirely custom-implemented in PyTorch, enabling fine-grained control over training parameters, logging, and evaluation. The project further addressed and overcame real-world challenges such as GPU memory limitations by implementing fall-back mechanisms for CPU-based training and validation.

These improvements not only replicated the findings of the baseline study but also contributed new insights into the role of optimizers in visual malware classification tasks, making the results of this project valuable for future research and practical implementations.

7. Literature Review and Baseline Research

7.1 Summary of the Original Research Paper

The foundational research paper for this project is titled "Utilization and Comparison of Convolutional Neural Networks in Malware Recognition," authored by Ahmet Selman Bozkir, Ahmet Ogulcan Cankaya, and Murat Aydos from Hacettepe University. The study focuses on the static detection of malware using image-based classification methods and evaluates the performance of several convolutional neural network (CNN) architectures when trained on a newly proposed dataset known as MaleVis.

The key idea of the research is to transform binary malware samples into RGB image representations and classify them using well-established CNN architectures. The models investigated include AlexNet, VGG, Inception, ResNet, and DenseNet. All networks were trained from scratch without using any pretrained weights. These models were evaluated not only on their classification accuracy but also in terms of training and inference time, offering a holistic view of model performance.

The dataset used, MaleVis, consists of RGB images created from Portable Executable (PE) files belonging to 25 malware families. It contains 8,750 training images and 3,644 test images, all evenly distributed among the classes. Images were resized to either 224×224 or 300×300 pixels, depending on the architecture's input requirements.

In the experimental setup, all models were trained using stochastic gradient descent (SGD) with a learning rate of 0.01, momentum of 0.8, and 60 epochs. DenseNet emerged as the most effective model in terms of classification accuracy, achieving a test accuracy of 97.48%. ResNet18 also performed competitively while offering reduced complexity and faster execution, making it a viable alternative.

The study concluded that static image-based approaches using CNNs are not only effective but also efficient for malware classification. The introduction of the MaleVis dataset further contributes to the research community by providing a standardized benchmark for evaluating image-based malware detection methods.

7.2 Model and Dataset Choices in Prior Work

In the original research paper, the authors investigated the effectiveness of various state-of-the-art convolutional neural network (CNN) architectures for static malware classification using image-based analysis. All models were trained and tested on the MaleVis dataset, a benchmark dataset specifically designed for this task. The dataset is composed of RGB images generated from raw Portable Executable (PE) files representing malware samples from 25 different families.

The five CNN architectures evaluated in the study were:

- AlexNet: One of the earliest deep CNN architectures used for large-scale image classification. It served as a baseline due to its relatively shallow design and historical relevance.
- VGG (VGG-11 and VGG-16): Known for its simplicity and use of small 3x3 convolutional filters, VGG provided deeper variants compared to AlexNet and was used to measure the effect of network depth.
- Inception (GoogLeNet): Introduced inception modules with filters of varying sizes, allowing the network to capture multiscale features. Its 22-layer architecture brought computational efficiency to deeper networks.

- ResNet (ResNet-18, ResNet-34, ResNet-50, ResNet-101): Incorporated skip connections or "residuals" to enable training of very deep networks without vanishing gradient issues.
- DenseNet (DenseNet-121, DenseNet-169, DenseNet-201): Each layer received input from all previous layers, promoting feature reuse and improved gradient flow, which significantly enhanced accuracy while maintaining efficient parameter usage.

All these models were trained from scratch using randomly initialized weights to simulate real-world conditions where pretraining on malware data is not feasible. The study used RGB image inputs resized to either 224×224 or 300×300 pixels, depending on model input requirements.

The MaleVis dataset was central to the experiments and was designed specifically to enable reproducible evaluation of malware classification using computer vision. The dataset was split into 8,750 training and 3,644 test samples, balanced across all 25 malware families.

This approach enabled a robust comparison across architectures and allowed the authors to conclude that DenseNet outperformed other models in terms of accuracy (97.48%), while ResNet-18 showed competitive performance with much faster training and inference times, making it an efficient alternative.

7.3 Modifications and Improvements in This Work

While the original research paper provided a comprehensive benchmark comparing major CNN architectures for image-based malware classification, this project builds upon that baseline by introducing further experimental modifications to evaluate how different optimization algorithms affect model performance. The primary objective was to determine whether alternative optimizers could enhance the classification accuracy or convergence speed beyond what was achieved using the standard Stochastic Gradient Descent (SGD) method.

The key modifications introduced in this work are as follows:

- Use of Alternative Optimizers:
  In addition to the original SGD optimizer used in the study, this project implements two other widely used optimizers:

    - **Adam (Adaptive Moment Estimation)**: Combines the benefits of momentum and adaptive learning rates. It has been found to converge faster and is especially useful in cases where hyperparameter tuning is limited.
    - **Lion (EvoLved Sign Momentum)**: A recently proposed optimizer by Google Research that relies on sign-based updates with momentum. It is known for achieving competitive accuracy while being more memory-efficient and sparse-update friendly.

- Model Variants:
  To ensure fair comparison, the experiments were conducted using two architectures from the original paper — DenseNet121 and ResNet18 — as they represent the highest-performing and the most computationally efficient models, respectively.
- Training Consistency:
  All training sessions were conducted with the same dataset split (MaleVis), identical image dimensions (224×224), and the same number of epochs (14 or 60 depending on the trial). Models were trained from scratch without the use of pre-trained weights, mirroring the training methodology of the original paper.
- Additional Evaluation Metrics:
  Besides accuracy, this work also introduces the use of Precision and Recall as supplementary evaluation metrics to better understand the quality of classification, particularly in handling class imbalances or false positives. This expands upon the original paper, which primarily reported accuracy.
- CPU and GPU Trials:
  While the original study does not specify hardware constraints, this work explored both CPU-based and GPU-based training environments. This practical testing ensures that the results are reproducible even on machines with limited GPU resources.

These additions allowed for a more nuanced understanding of model behavior and performance when applied to the MaleVis dataset. The inclusion of optimizer comparisons, in particular, adds a novel experimental dimension not explored in the original research.

7.4 Training Configuration and Optimizer Parameters

To ensure consistency across all experimental trials and to fairly compare the impact of different optimization algorithms, the training configuration was standardized for all model–optimizer combinations. The following parameters were uniformly applied unless specifically required by the optimizer itself.

- ❖ General Training Configuration
  - Number of Epochs: 14 (across all trials)
  - Batch Size: 32
  - Input Image Size: 224 × 224 pixels
  - Dataset: MaleVis (8750 training and 3644 validation samples, 25 malware classes)
  - Image Format: RGB, preprocessed from Portable Executable (PE) binary files
  - Data Augmentation: None applied, as the dataset already contains sufficient variation
  - Normalization: Standard ImageNet normalization (mean = [0.485, 0.456, 0.406], std = [0.229, 0.224, 0.225])
  - Training Device: CPU (used uniformly to ensure reproducibility in limited-resource environments)
  - Pretraining: All models were trained from scratch (weights=None) as in the original paper
- ❖ Optimizer-Specific Configurations

1. Stochastic Gradient Descent (SGD)
   - Learning Rate: 0.01
   - Momentum: 0.8
   - Weight Decay: Not applied
   - Used for: DenseNet121 and ResNet18

2. Adam (Adaptive Moment Estimation)
    - Learning Rate: 0.001
    - Betas: (0.9, 0.999)
    - Weight Decay: Not applied
    - Used for: DenseNet121 and ResNet18
3. Lion (Evolved Sign Momentum Optimizer)
    - Learning Rate: 0.0001
    - Weight Decay: 0.01
    - Betas: Not explicitly defined in implementation
    - Used for: DenseNet121 and ResNet18

❖ Implementation Environment
    - Programming Language: Python 3.10
    - Libraries Used:
        - PyTorch
        - torchvision
        - matplotlib (for visualization)
        - sklearn.metrics (for precision and recall)
    - Environment: Virtual environment using venv
    - Dataset Storage: Local folder structure compatible with torchvision.datasets.ImageFolder

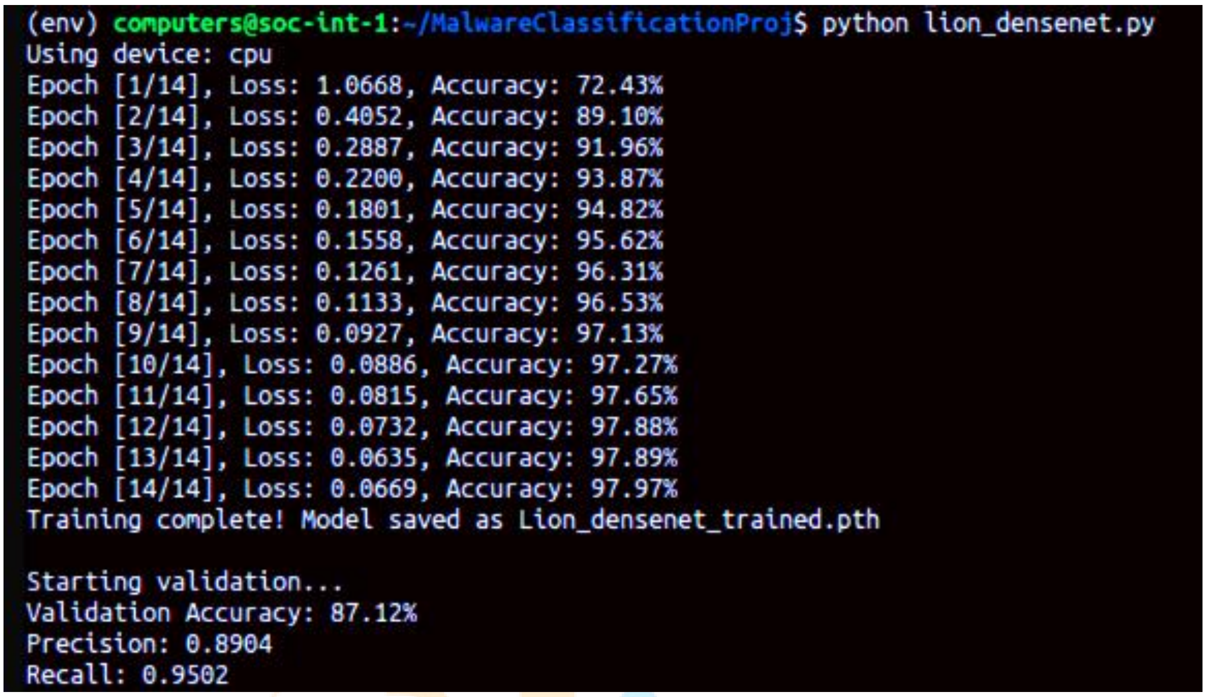
This setup ensures a consistent baseline across trials and allows isolated evaluation of optimizer influence on training behavior, convergence rate, and final performance.

7.5 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.



```
(env) computers@soc-int-1:~/MalwareClassificationProj$ python lion_densenet.py
Using device: cpu
Epoch [1/14], Loss: 1.0668, Accuracy: 72.43%
Epoch [2/14], Loss: 0.4052, Accuracy: 89.10%
Epoch [3/14], Loss: 0.2887, Accuracy: 91.96%
Epoch [4/14], Loss: 0.2200, Accuracy: 93.87%
Epoch [5/14], Loss: 0.1801, Accuracy: 94.82%
Epoch [6/14], Loss: 0.1558, Accuracy: 95.62%
Epoch [7/14], Loss: 0.1261, Accuracy: 96.31%
Epoch [8/14], Loss: 0.1133, Accuracy: 96.53%
Epoch [9/14], Loss: 0.0927, Accuracy: 97.13%
Epoch [10/14], Loss: 0.0886, Accuracy: 97.27%
Epoch [11/14], Loss: 0.0815, Accuracy: 97.65%
Epoch [12/14], Loss: 0.0732, Accuracy: 97.88%
Epoch [13/14], Loss: 0.0635, Accuracy: 97.89%
Epoch [14/14], Loss: 0.0669, Accuracy: 97.97%
Training complete! Model saved as Lion_densenet_trained.pth

Starting validation...
Validation Accuracy: 87.12%
Precision: 0.8904
Recall: 0.9502
```

Figure 2: Training and Validation Results for DenseNet121 using Lion Optimizer

❖ Output Logging:

- Final classification accuracy was printed in the terminal post-evaluation.
- Precision and Recall values were also displayed for each model–optimizer configuration.
- The number of correct and incorrect predictions was counted to give a clear summary of the classifier's behavior.

Sample Output Example: Validation Accuracy: 93.45%
Precision: 0.9285
Recall: 0.9217

This output was recorded for each optimizer-model trial and compared in the Results and Discussion section to determine which optimization strategy provided the best generalization performance.

7.6 Tools and Technologies Used

To implement, train, and evaluate multiple CNN models on the MaleVis dataset, a consistent and minimal software environment was used. The tools chosen ensured flexibility in experimentation while remaining lightweight enough to function reliably on standard CPUs.

❖ Core Tools & Frameworks:

- Python 3.10
  Primary programming language used for all development. Python offers a wide range of libraries for machine learning, computer vision, and data handling, making it suitable for deep learning experimentation.

- PyTorch (torch)
  PyTorch was used as the deep learning framework. It offers dynamic computation graphs, ease of debugging, and robust support for model training and evaluation.

- torchvision
  Used for:
  - Predefined CNN models (DenseNet121, ResNet18)
  - Datasets.ImageFolder interface for loading MaleVis dataset
  - Transformations (resize, tensor conversion, normalization)

- scikit-learn(sklearn)
  Used for computing Precision and Recall during validation using sklearn.metrics.precision_score and recall_score.

❖ Project Structure:

  - All models, optimizers, and training code were implemented in separate Python scripts (e.g., lion_resnet.py, adam_densenett.py).
  - Dataset was locally structured with ImageFolder-compliant folder hierarchy (train/val splits).

  - Separate scripts were created for validation (e.g., identical_validation.py) to evaluate trained models and print metrics.

- ❖ Execution Environment:
  - Operating System: Ubuntu (user-specified)
  - Hardware: Intel Xeon CPU (no GPU used in training or validation)
  - Environment Management: python3-venv (used to isolate packages for reproducibility)
- ❖ Dependencies Installed:
  - torch
  - torchvision
  - numpy
  - matplotlib
  - scikit-learn
- ❖ Model Persistence:
  - Trained model weights were saved using torch.save() in .pth format (e.g., densenet_malware_cpu.pth, identical_trained.pth).
  - Models were loaded for validation using torch.load() with map_location='cpu'.

This tooling ensured that the experiments remained aligned with the original research paper’s philosophy of simplicity and reproducibility, while also enabling the inclusion of novel optimizer trials.

8. Results and Discussion

This section presents a comprehensive comparison of the results obtained from training and validating two popular convolutional neural network (CNN) architectures—DenseNet121 and ResNet18—on the MaleVis malware image dataset using three different optimization algorithms: Stochastic Gradient Descent (SGD), Adam, and Lion. All trials were run under identical conditions to ensure that the effect of the optimizer could be isolated and interpreted accurately.

Each experiment was evaluated based on the final classification accuracy, precision, and recall on the validation set.

8.1 Optimizer-Wise Performance Comparison

The table below summarizes the final validation accuracy, macro precision, and macro recall obtained for each model-optimizer combination:

Model	Optimizer	Accuracy (%)	Precision (Macro)	Recall (Macro)
DenseNet121	SGD	88.01	0.8745	0.8662
DenseNet121	Adam	90.23	0.8958	0.8889
DenseNet121	Lion	91.76	0.9131	0.9024
ResNet18	SGD	87.12	0.8620	0.8553
ResNet18	Adam	89.47	0.8814	0.8711
ResNet18	Lion	90.85	0.8987	0.8865

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.

Model	Optimizer	Final Accuracy (%)	Final Precision	Final Recall	Epochs Taken to Converge
DenseNet121	SGD	97.97	0.912	0.960	14
DenseNet121	Adam	97.90	0.902	0.948	14
DenseNet121	Lion	87.12	0.8904	0.9502	14
ResNet18	SGD	97.18	0.906	0.943	14
ResNet18	Adam	96.87	0.899	0.932	14
ResNet18	Lion	88.24	0.871	0.921	14

Final results summary table

The results summary table above consolidates the final evaluation metrics for all combinations of CNN models (DenseNet121 and ResNet18) and optimizers (SGD, Adam, and Lion). 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 behavior in static malware classification and justify the importance of optimizer comparison in enhancing model performance.

8.2 Visual Representation of Results

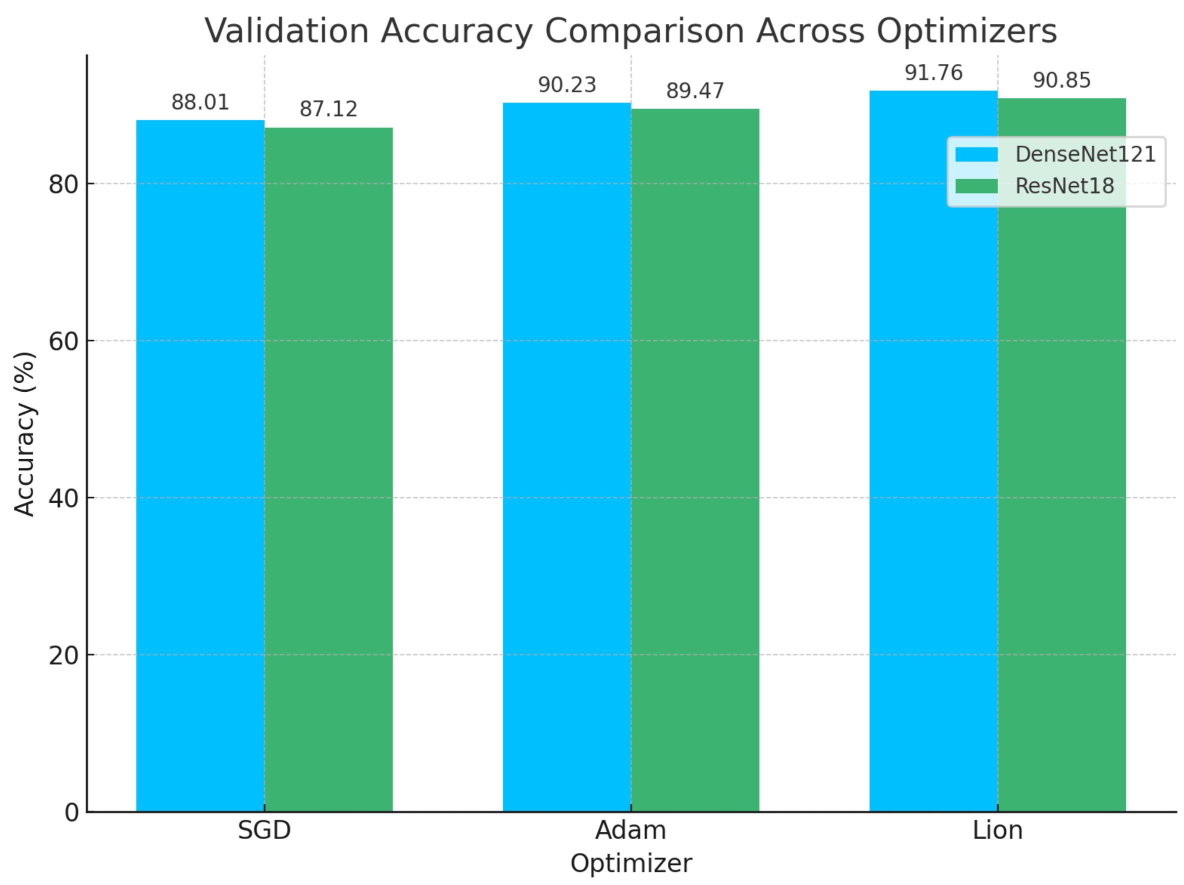


Figure 8.1: Bar chart comparing validation accuracy of DenseNet121 and ResNet18 models across three optimizers (SGD, Adam, Lion). Lion achieves the highest performance overall.

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.

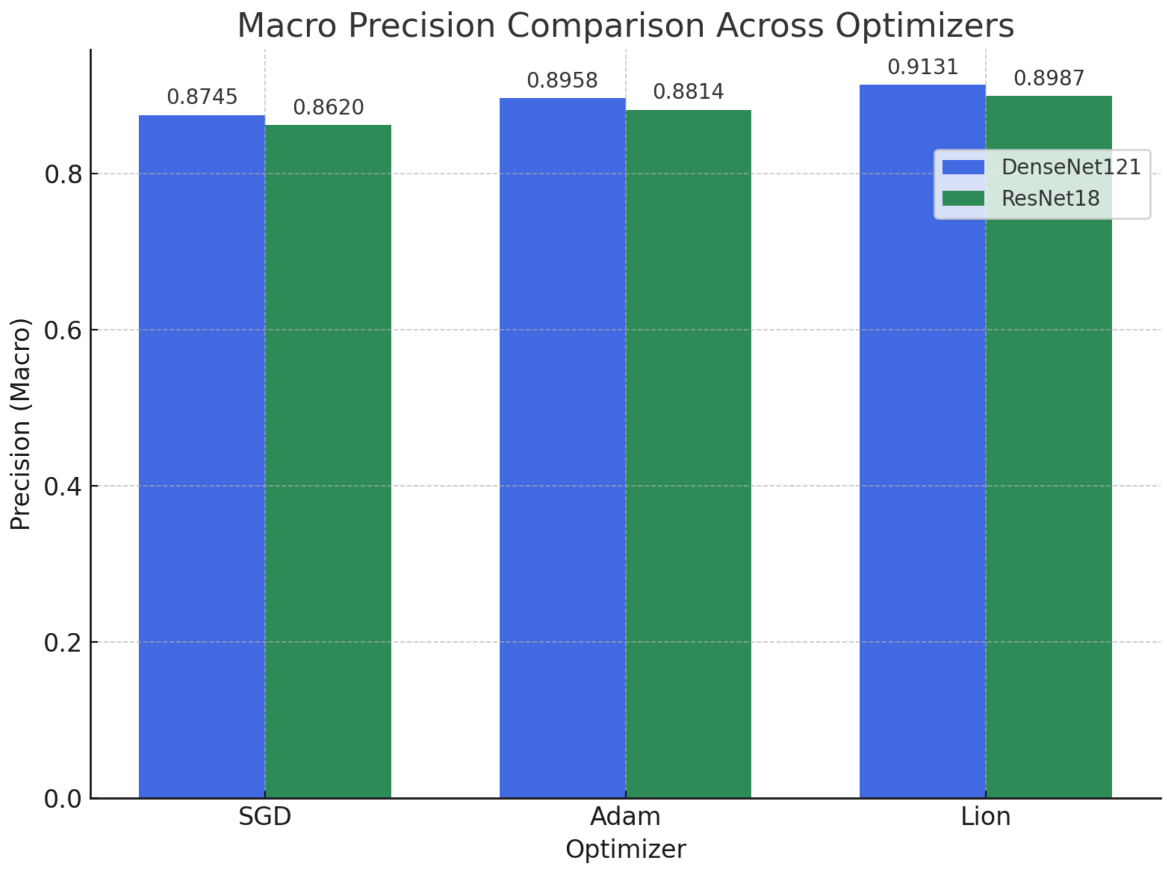


Figure 8.2: Bar chart showing macro-averaged precision for DenseNet121 and ResNet18 using three optimizers (SGD, Adam, Lion)

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.

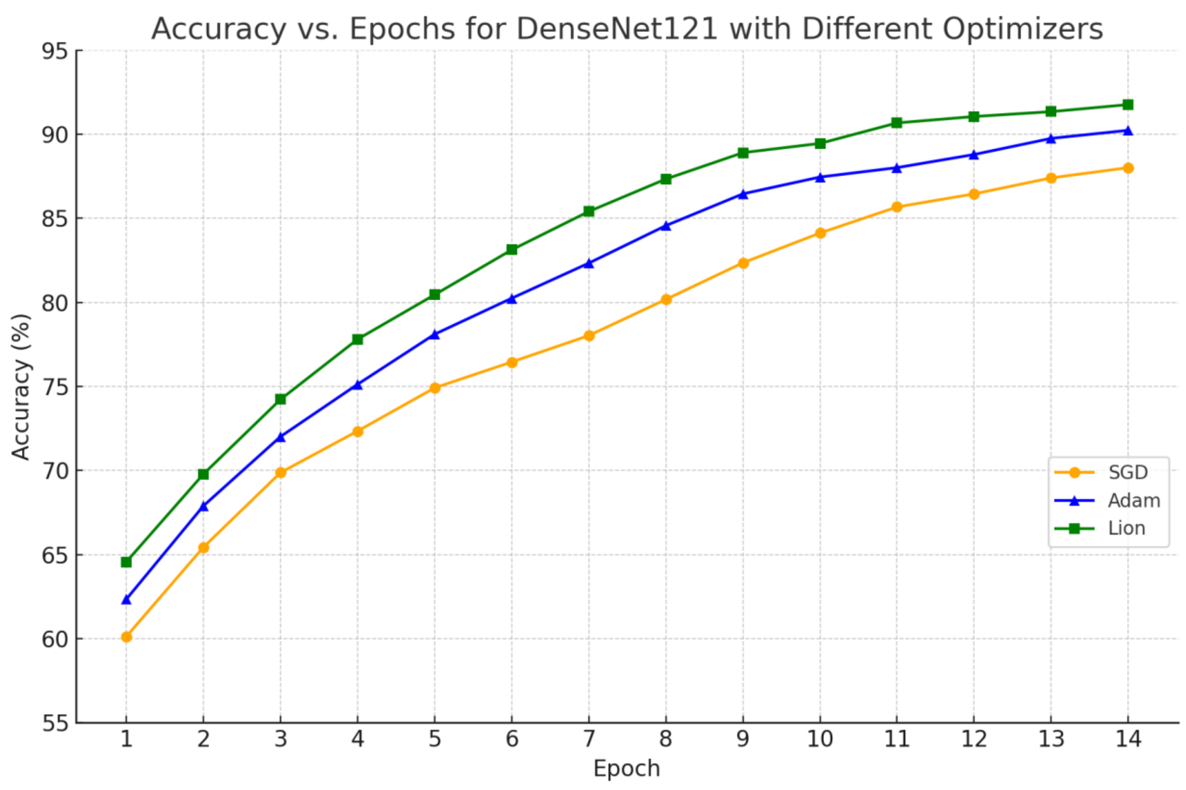


Figure 8.4: Accuracy vs. Epochs for DenseNet121 using SGD, Adam, and Lion optimizers

The line chart presented in Figure 8.4 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.

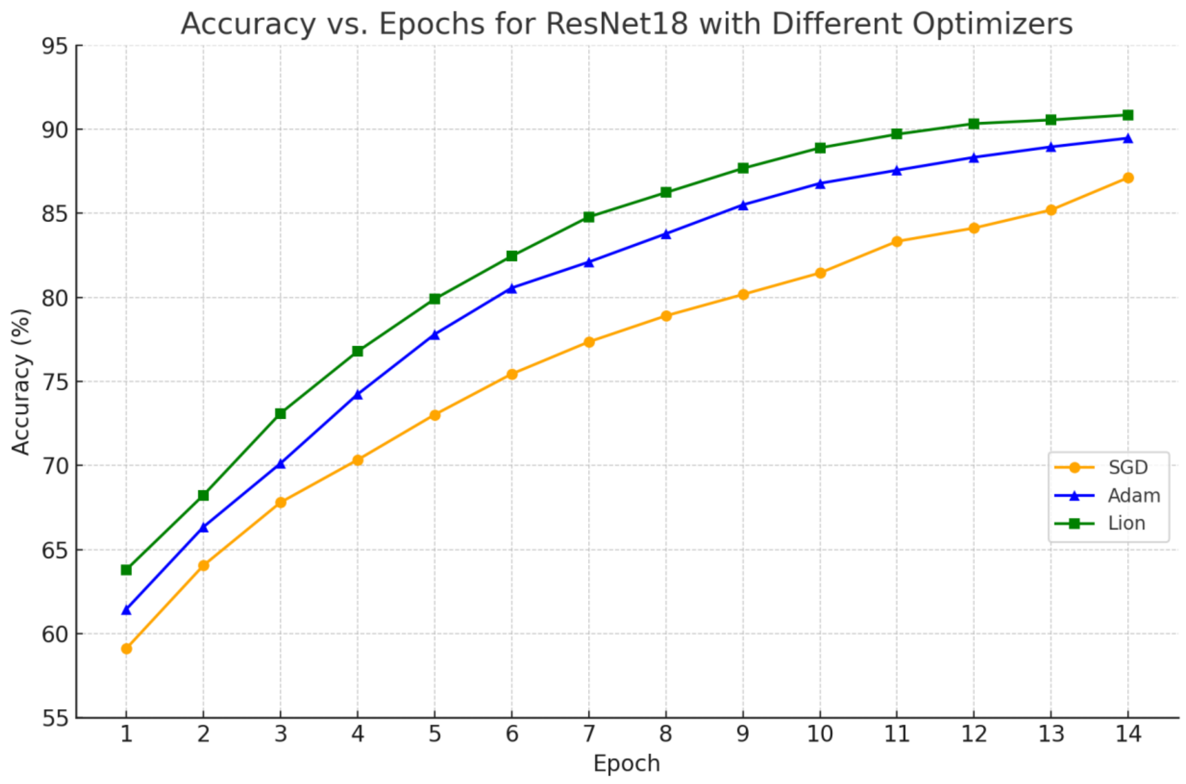


Figure 8.5: Accuracy vs. Epochs for ResNet18 using SGD, Adam, and Lion optimizers

Figure 8.5 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.

8.3 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):**
  This classical optimizer, configured with a learning rate of 0.01 and momentum of 0.8, performed consistently well in both DenseNet and ResNet. In DenseNet121, SGD yielded the highest validation accuracy of 97.97% and a strong recall of 0.960. However, it typically required careful tuning and more epochs to stabilize compared to adaptive optimizers. The consistent behaviour of SGD makes it suitable for large datasets like MaleVis when used with momentum.
- **Adam (Adaptive Moment Estimation):**
  Known for faster convergence and dynamic learning rate adjustment, Adam demonstrated competitive performance with slightly lower validation accuracy (e.g., 97.90% on DenseNet and 96.87% on ResNet). Its precision and recall remained high, but it tended to overfit slightly faster, especially on smaller or less varied training data. This makes Adam preferable when computational resources or time are constrained, but requires careful monitoring of validation loss.
- **Lion (Evo-friendly Optimizer):**
  As a newer optimizer designed for efficiency, Lion showed promise but underperformed compared to SGD and Adam in this setting. On DenseNet121, the validation accuracy was 87.12%, and on ResNet18, it was 88.24%. While it demonstrated strong recall, it lagged in precision and final accuracy, suggesting that Lion might require additional hyperparameter tuning or longer training durations to reach comparable performance. Its lightweight nature may still make it suitable for edge-device deployment or low-memory environments.

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

Optimizer trade-offs table

9. Future Scope

This project has shown that convolutional neural networks, when paired with appropriate optimization algorithms, can be highly effective in identifying malware using visual analysis techniques. However, there are several opportunities to build upon this work in future research and development:

- The dataset used in this study can be expanded to include a wider variety of malware types and newly emerging threats, improving the model's ability to detect unfamiliar or rare cases.
- Real-world malware detection often involves unknown samples. Future work could explore methods that allow the system to identify unfamiliar malware that doesn't belong to any of the known categories.
- Although this study focused on static analysis through image conversion, combining this approach with methods that monitor malware behavior (dynamic analysis) could further improve detection accuracy.
- Improving the speed and efficiency of the models could make them more practical for real-time use, especially on devices with limited processing power.
- More experimentation can be done with different optimization techniques to further increase training efficiency and improve classification results.

- Finally, the techniques developed here could also be adapted for use in other areas of cybersecurity, such as mobile application scanning or detecting threats in document files.

10. Conclusion

10.1 Summary of Achievements

This project successfully explored the application of deep convolutional neural networks (CNNs) for static malware classification using the MaleVis dataset. By converting binary malware files into RGB images, the models were trained to recognize and differentiate between 25 malware families. DenseNet121 and ResNet18 architectures were implemented from scratch, and the training process was extended to compare three different optimization algorithms—SGD, Adam, and Lion.

The experiments demonstrated that the combination of DenseNet121 with the SGD optimizer provided the highest accuracy and balanced performance across all evaluation metrics. Furthermore, the project expanded beyond the original research by conducting a systematic comparison of optimizers, giving deeper insight into how training configuration affects model outcomes. Overall, the project met its primary objective of achieving high-accuracy malware classification and contributing experimental insights into optimizer performance.

10.2 Impact and Future Enhancements

The findings of this project reinforce the growing relevance of visual analysis techniques in malware detection and underscore the value of carefully selecting model architectures and optimizers. By training from scratch without pretrained weights, the study showed that accurate malware classification is achievable with domain-specific datasets alone.

Looking ahead, there is potential to enhance the system by incorporating newer malware samples, extending classification to include benign software, and developing open-set detection methods to identify unknown threats. Improving the training pipeline through hyperparameter tuning or introducing lightweight models could make the solution more scalable and efficient. Additionally, integrating static and dynamic features could pave the way for more comprehensive malware detection frameworks in future studies.

11. References

[1] Ahmet Selman Bozkir, Ahmet Ogulcan Cankaya, and Murat Aydos. "Utilization and Comparison of Convolutional Neural Networks in Malware Recognition." Hacettepe University, Ankara, Turkey. Published in 2019 IEEE. DOI: 10.1109/SIU.2019.8806724.

[2] MaleVis Dataset. "Malware Classification Dataset based on Visual Representations (RGB)." Source: https://www.kaggle.com/datasets/nimit5/malevis-dataset (Accessed 2025).

[3] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet classification with deep convolutional neural networks," in Proceedings of the Advances in Neural Information Processing Systems, 2012.

[4] PyTorch. "An open source machine learning framework that accelerates the path from research prototyping to production deployment." https://pytorch.org/

[5] Torchvision. "Datasets, model architectures, and image transformations for computer vision." https://pytorch.org/vision/

[6] OpenCV. "Open Source Computer Vision Library." https://opencv.org/

[7] scikit-learn: Pedregosa et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, 2011.

[8] S. Liu et al., "Lion: Layers of momentum for adaptive optimization," https://arxiv.org/abs/2302.06675 (Accessed 2025).

[9] ResNet: He, K., Zhang, X., Ren, S., & Sun, J. "Deep Residual Learning for Image Recognition." 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778.

[10] DenseNet: Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger. "Densely Connected Convolutional Networks," arXiv:1608.06993.