Numeric Feature Analysis in Deep Learning-Based Ransomware Detection with Convolution Neural Network Models

[1]LUKMAN ADEBAYO OGUNDELE, [2]JULIUS TEMITAYO ADEPOJU, [1]FEMI EMMANUEL AYO, [3]IDAYAT ABIKE AKANO, [3]OLUYEMISI ADENIKE OYEDEMI

[1]Department of Computer Sciences, Faculty of Science, Olabisi Onabanjo University, Ago-Iwoye, Ogun State, Nigeria
[2]Department of Mathematical Science, Faculty of Physical Science, University of Ilorin, Kwara State, Nigeria
[3]Department of Cyber-security, Faculty of computing, University of ilesa, ilesa osun state
[1] (ogundele.lukman,ayo.femi)@oouagoiwoye.edu.ng
[2]adepojujulius58@gmail.com
[3] (idayat_akano,yemisi.oyedemi)@unilesa.edu.ng

Abstract. The research introduces ResMalNet, a convolutional neural network architecture designed for malware detection. The architecture employs domain expertise to identify critical behavioral categories, such as registry operations, network activities, and process/file interactions, and statistical optimization to select the most discriminative numeric features. ResMalNet outperforms four established CNN architectures, achieving 98.91% accuracy and 98.92% precision while maintaining balanced recall and F1-scores of 98.91%. The technical implementation addresses three persistent challenges in malware classification: prevention of model over-fitting, preservation of critical feature relationships, and optimization of residual block designs. Experimental results show architectural specialization through residual connections improves accuracy by 1.82% over conventional CNN designs, domain-informed feature selection reduces false positive rates by 42%, and exceptional detection rates for previously unseen malware variants during validation testing. The ResMalNet framework offers practical implementation guidelines for security systems, with immediate applications in next-generation endpoint protection solutions and network monitoring infrastructure.

Keywords: Ransomware, Deep-learning, Neural network

(Received December 22th, 2024 / Accepted June 14th, 2025)

1 Introduction

The latest advancement in technology and the rapid implementation of computer systems have produced previously unprecedented levels of cyber attacks wherein one of the most damaging categories of malware is ransomware [19, 1]. Ransomware refers to a form of malicious codes which encrypts the victims? files or operating system for a reward of payment in ransom to decode them. This cyberattack has gone to the next level where attackers use advanced mechanisms to bypass traditional security controls. Institutions, organizations, and individuals have lost a lot of money and operations as a result of ransomware attacks, which further vindicate the need for effective detection and prevention efforts [5, 15].

Ransomware has become one of the most prevalent types of cyberattacks on individuals, enterprises, and critical infrastructure. Ransomware encrypts data and demands payment to be decrypted, causing significant financial loss, service disruption, and breaches in confidential data [18, 12]. The frequency and sophistication level of ransomware attacks have grown with passage of time since cybercriminals use advanced avoidance measures to bypass detection systems. These facts highlight the need to develop advanced and reliable systems that can effectively identify ransomware.

Traditional methods used to identify ransomware, including signature-based and heuristic approaches, have proven to be ineffective in managing the dynamic and ever-changing nature of modern ransomware [23, 13]. Signature-based detection systems are based on predefined patterns or signatures linked to malicious files and are therefore ineffective when presented with uniquely created or polymorphic ransomware versions [3, 14]. Though heuristic techniques can detect unseen attacks, they are mainly plagued by high rates of false positives, which reduces the efficacy of cybersecurity tools [9, 21].

One of the initial efforts towards malware detection with convolutional neural networks (CNNs) was MalConv, proposed by Raff et al. [16]. MalConv employs a one-dimensional convolutional neural network on the raw byte streams of malware binaries themselves, directly feeding them as image inputs. This work laid the foundation for future work in this area, reflected in Droid-Sec, proposed by [22]. Droid-Sec, a malware detector for Android, employs a two-dimensional CNN to learn semantic features from opcode sequences and Android app manifest files. Tian et al. (2021) recently suggested DeeperForensics [6], where deep CNNs are used to explore pixel-level features of screenshots of malware behavior, thus enabling visual-based detection methods.

Although previous research, including MalConv, Droid-Sec, and DeeperForensics, has shown the success of Convolutional Neural Networks (CNNs) on image-like featured data where malware samples are considered as raw binary strings, visual representations, or screenshots. In contrast, this research operates on a formalized, tabular dataset rich in system-level features, e.g., malware header information, behavioral signatures manifested as registry activity, network activity, and process execution information. By inputting this multi-dimensional, non-image data directly to the model, we expect to capture more precise and subtle malicious activity patterns to enhance the validity and robustness of malware detection without having to convert the data into an image-like format.

This paper offers a custom-made residual CNN known as ResMalNet for detecting ransomware and compares the results with various CNN variantions. The models were trained on a dynamic malware dataset containing 58,596 samples from the OMM-2022 dataset as well as a manually selected ransomware dataset. Second, this study proposes an end-to-end pipeline for feature engineering and training in which dynamic behavioral data (e.g., API requests and registry updates) is normalized using MinMaxScaler, encoded using LabelEncoder, and reshaped to be compatible with 1D CNN layers.

The rest of the paper is organized as follows: Section 2 gives a critical review of existing CNN-based malware detection research, which guides the architecture choices in this research. Section 3 provides detailed methodology with preprocessing operations in datasets, feature transformation, model implementation, and training protocols. Section 4 presents experimental results with comparisons of performance on the basis of accuracy, precision, recall, and F1-score metrics. Finally, Section 5 presents practical implications of our findings and suggests areas of future work, particularly adversarial robustness improvement and generalization improvement.

2 Related Work

[6] proposed DeepWare, a ransomware detection framework based on deep learning and hardware performance counters (HPC). Unlike conventional approaches that consider individual process monitoring, DeepWare captures system-wide HPC variations, transforming the variations into images for classification by convolutional neural networks (CNN). The model achieved a recall rate of 98.6% and demonstrated considerable effectiveness in detecting unseen ransomware families, including CoronaVirus, Ryuk, and Dharma.

[8] proposed D-WARE, a malware detection system based on CNN with PCA used for feature extraction and PSO for dimensionality reduction. D-WARE was tested on the Malimg dataset and compared with other models like VGG16, VGG19, and DenseNet. The suggested method had an accuracy of 96%, demonstrating the success of deep learning in malware classification.

[17] proposed an attention-enabled CNN for malware classification. The research tackled the problem of identifying small malware-infected areas in image-based Windows malware file representations. With multi-headed attention in a CNN, the model achieved better classification accuracy without any loss in computational efficiency. The method achieved 99% accuracy across different data splits and showed resilience in the detection of polymorphic and obfuscated

malware.

[11] suggested a CNN-based solution for the detection of crypto-ransomware in IoT environments. The solution inspects opcode patterns of binary executables for ransomware and benign application classification. Through the utilization of a late fusion method for the combination of feature representations, the system detected ransomware with a 97% success rate. The solution was found to be especially useful for low-end embedded processors and thus was determined to be suitable for IoT security.

[7] proposed XRan, an explainable AI-based model for ransomware detection that focuses on applying dynamic analysis to identify behavioral patterns during ransomware execution. The model attained 99.4% true positive rate and offered interpretability in classifying malware so that security analysts are able to comprehend the rationale behind decision making by the model.

[4] proposed an efficient deep learning system to identify ransomware attacks on SCADA-managed EVCS. The authors compared the three deep learning methods? DNN, 1D CNN, and LSTM networks?to identify how effective these are to identify ransomware-based cyber assaults. The proposed framework accomplished an average rate of 97%, and an AUC greater than 98% and false alarm rate under 1.88% under 10-fold stratified cross-validation. In addition to this, the researchers unveiled the impact of ransomware-type DDoS and FDI attacks with the possibility to alter the SOC profile and cause damage to the BES system.

[20] introduced a Windows malware detection system based on CNN from visualized images of Portable Executable (PE) file execution-time behavioral features. The model applied the Relief Feature Selection Technique to minimize the significant behavioral features and 10-fold cross-validation to evaluate it. Experimental evaluation showed that the detection rate of the CNN-based detector was 97.97%, and it could effectively identify obfuscated malware.

[2] introduced DeepMalore, an image malware system based on deep learning that uses CNN architectures to identify malware images. The study compared different CNN architectures like AlexNet, VGG-16, ResNet-50, and InceptionV3 on the dataset called Malimg that comprises malware binaries presented in the form of images. The proposed approach achieved an accuracy level of 98.90%, showcasing the effectiveness of deep learning in malware detection through image processing.

3 Materials and Methods

This study presents a robust approach for malware detection using convolutional neural networks, evaluating both custom architectures and established design patterns. The research employs five distinct CNN variants - Basic CNN, Bidirectional CNN, Deeper CNN, Residual CNN, and Inception-like CNN?trained on the CIC-MalMem-2022 dataset containing 21,752 samples with 18 behavioral and structural features.

The experimental framework addresses three critical challenges in malware detection: (1) processing heterogeneous feature spaces combining static, dynamic, and behavioral characteristics, (2) mitigating class imbalance through strategic oversampling, and (3) preventing model overfitting using dropout regularization (25-50% rates) and early stopping. Unlike traditional image-based malware classification, our method transforms multidimensional feature vectors into optimized 1D convolutional inputs, enabling efficient pattern recognition while preserving temporal relationships in API call sequences and registry operations.

3.1 Datasets

This research uses two datasets to analyze and classify malware. The first dataset, OMM-2022, is an obfuscated malware dataset developed by the Canadian Institute for Cybersecurity (CIC), and the second dataset is a custom ransomware dataset.

3.1.1 The CIC-MalMem-2022 (OMM-2022) Dataset

OMM-2022 is a newly designed dataset for malware detection, released by the Canadian Institute for Cybersecurity in 2022. The dataset comprises a total of 58,596 samples divided equally between benign and malware samples. The dataset contains 56 extracted feature variables and a single classification target. Malware samples have been categorized into three primary classes, including Trojan Horse, Spyware, and Ransomware, and further sub-classification into 15 various malware families. Table 1 provides an overview of the malware families contained in this dataset.

3.1.2 Custom Ransomware Dataset

To complement OMM-2022, a ransomware-specific dataset was constructed using samples from a variety of sources, including VirusTotal, VirusShare, Malware-Bazar, and GitHub. 21,752 malware and benign programs were collected with 18 features, with the same number of malicious and non-malicious examples [10]. The dataset includes samples of 11 prominent ransomware families: Cerber, DarkSide, Dharma, Gand-Crab, LockBit, Maze, Phobos, REvil, Ragnar, Ryuk, Shade, and WannaCry. Further, to assess how well the model generalizes over different threats, samples from three other categories of malware (Trojan Horse, Information Stealer, and Remote Access Trojan (RAT)) were included.

Table 1: Example of Table

Category	Family	Samples	Total Samples
Trojan Horse	Zeus	195	9,487
	Emotet	196	
	Refrose	200	
	Scar	200	
	Reconyc	157	
Spyware	180Solutions	200	10,020
	Coolwebsearch	200	
	Gator	200	
	Transponder	241	
	TIBS	141	
Ransomware	Conti	200	9,791
	Maze	195	
	Pysa	171	
	Ako	200	
	Shade	220	
Benign	-	-	**29,298**

3.2 Numeric Feature Selection

In this study, Numerical feature selection process is applied to the dataset. The goal of feature selection is to prepare the data for machine learning algorithms by selecting relevant features and transforming categorical data into a numerical format that can be efficiently processed by models. This feature selection isolates the numeric columns from the dataset. This is essential because many machine learning algorithms require numerical input, and operations like correlation analysis or scaling can only be performed on numeric data. The dataset originally contains 77 columns, but only 18 of them are numeric. By selecting only the numeric columns for further analysis, this ensures that the data used for training the models contains only the relevant numeric features, eliminating any irrelevant or non-numeric data that would otherwise be incompatible with the algorithms.

3.2.1 Encoding Categorical Features

Label Encoding converts each unique category value into a numeric label. This technique is particularly useful when the categorical feature has an inherent ordinal relationship, meaning the values have a meaningful order (e.g., low, medium, high). After applying Label Encoding, the categorical columns are transformed into numeric values, making them suitable for use in machine learning models.

After selecting numeric columns and encoding the categorical ones, the dataset now contains a combination of numeric features, including the originally numeric columns and the transformed categorical columns. The resulting DataFrame is checked to ensure the correct columns are retained and properly formatted. The dataset now contains 77 columns in total, with 18 columns of numeric type (?float64?) and the remaining columns converted to numeric values via Label Encoding. This ensures that the data is ready for the models, which require numerical input for training.

3.3 Proposed custom model: ResMalNet

Residual Convolutional Neural Networks (ResCNNs) introduce skip connections that allow gradients to flow more effectively during backpropagation, addressing the vanishing gradient problem. This technique enhances the performance of deep networks by enabling the reuse of learned features. The Residual CNN layers consists of:

1. Standard Convolutional Operation
   Each convolutional layer applies:

$$y_i^{(l)} = f\left(\sum_{j=0}^{k-1} x_{i+j}^{(l-1)} w_j^{(l)} + b^{(l)}\right) \tag{1}$$

   where:

   - $y_i^{(l)}$ is the output of the l^{th} layer.
   - $x_{i+j}^{(l-1)}$ is the input from the previous layer.
   - $w_j^{(l)}$ and $b^{(l)}$ are the filter weights and biases.
   - f is the activation function (ReLU).

2. Residual Connection
   A skip connection directly adds the input to the output of a residual block:

$$y_i^{(res)} = y_i^{(l+1)} + x_i^{(l)} \tag{2}$$

where:

- $y_i^{(res)}$ is the residual output.
- $y_i^{(l+1)}$ is the standard convolutional output.
- $x_i^{(l)}$ is the input being added via the shortcut.

3. 1x1 Convolution for Dimension Matching If the input and output dimensions differ, a 1×1 convolution ensures compatibility:

$$x_i^{(adj)} = g\left(\sum_{j=0}^{1} x_{i+j}^{(l)} w_j^{(1\times1)} + b^{(1\times1)}\right) \tag{3}$$

where g is a linear activation function.

4. Fully Connected Layers
The extracted features are processed as:

$$h = f(Wx + b) \tag{4}$$

where W and b are learned parameters, and f is ReLU.

5. Softmax Classification
The final probabilities are computed as:

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{5}$$

where N is the number of classes.

6. Loss Function and Optimization The model is trained using the Adam optimizer with the sparse categorical cross-entropy loss:

$$L = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{6}$$

3.4 CNN Models

3.4.1 Basic CNN Model

The Basic Convolutional Neural Network (CNN) implemented in this study serves as a foundational deep learning model. This architecture is designed with a minimal yet effective structure to capture essential patterns within the malware dataset. Each layer in the CNN follows specific mathematical operations.

Algorithm 1 ResMalNet

```
1:  Input: Training data $X_{train}$, Labels $Y_{train}$
2:  Output: Trained Residual CNN model
3:  Initialize model with Input layer
4:  First Residual Block:
5:      Add Conv1D layer with 32 filters, kernel size = 3, ReLU activation
6:      Add another Conv1D layer with 32 filters, kernel size = 3, ReLU activation
7:      Apply skip connection by adding input directly to output
8:  Second Residual Block:
9:      Add Conv1D layer with 64 filters, kernel size = 3, ReLU activation
10:     Add another Conv1D layer with 64 filters, kernel size = 3, ReLU activation
11:     Apply skip connection with a 1x1 convolution for dimension matching
12: Flatten feature maps
13: Add Dense fully connected layer with 128 neurons, ReLU activation
14: Add Output layer with softmax activation
15: Compile the model with Adam optimizer and sparse categorical cross-entropy loss
16: Train the model using $X_{train}$ and $Y_{train}$ for $epochs$ iterations
17: Evaluate the model on test data
```

1. Convolutional Layer A **1D convolution operation** is mathematically represented as:

$$y_i^{(l)} = f\left(\sum_{j=0}^{k-1} x_{i+j}^{(l-1)} w_j + b\right) \tag{7}$$

where:

- $y_i^{(l)}$ is the output at position i in layer l.
- $x_{i+j}^{(l-1)}$ represents the input from the previous layer.
- w_j are the convolutional filter weights of size k.
- b is the bias term.
- f is the activation function (ReLU in this case).

2. Max-Pooling Layer Max-pooling selects the maximum value in a given window size:

$$y_i^{(l)} = \max_{j\in P} x_{i+j}^{(l-1)} \tag{8}$$

where P is the pooling window.

3. Fully Connected Layer The fully connected (Dense) layer is defined as:

$$y = f(Wx + b) \tag{9}$$

where:

- W is the weight matrix.
- x is the input vector.
- b is the bias term.
- f is the activation function (ReLU for hidden layers, Softmax for the output layer).

4. Dropout Regularization Dropout prevents overfitting by randomly deactivating neurons with probability p:

$$y_i^{(l)} = \frac{y_i^{(l)}}{1-p}, \quad \text{if not dropped} \tag{10}$$

5. Softmax Activation Function For multi-class classification, the softmax function converts logits into probabilities:

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{11}$$

where:

- $P(y_i)$ is the probability of class i.
- z_i is the logit (raw output before activation).
- N is the total number of classes.

6. Compilation and Optimization The model is compiled using:

- **Adam optimizer** with a learning rate of 0.001, which dynamically adapts learning rates.
- **Sparse categorical cross-entropy loss**, given by:

$$L = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{12}$$

where y_i is the true label and $\hat{y}_i$ is the predicted probability.

Algorithm 2 Basic CNN

```
1: Input: Training data $X_{train}$, Labels $Y_{train}$
2: Output: Trained Basic CNN model
3: Initialize Sequential CNN model
4: Add Conv1D layer with 32 filters, kernel size = 3, ReLU activation
5: Add MaxPooling1D layer with pool size = 2
6: Add Dropout layer (rate = 0.25) to reduce overfitting
7: Flatten feature maps
8: Add Dense fully connected layer with 64 neurons, ReLU activation
9: Add Dropout layer (rate = 0.5)
10: Add Dense output layer with softmax activation for classification
11: Compile the model with Adam optimizer and sparse categorical cross-entropy loss
12: Train the model using $X_{train}$ and $Y_{train}$ for $epochs$ iterations
13: Evaluate the model on test data
```

3.5 Bidirectional CNN Model

The Bidirectional Convolutional Neural Network (Bi-CNN) extends the capabilities of a standard CNN by incorporating multiple convolutional layers in a sequential manner, allowing the model to capture deeper feature representations. The model utilizes two consecutive convolutional layers with different filter sizes before pooling, making it more effective in detecting complex patterns in malware data. Additionally, a larger fully connected layer enhances feature abstraction before classification. The layers in Bi-CNN consist of:

1. Multiple Convolutional Layers
   Unlike a standard CNN, which typically applies a single convolutional layer before pooling, Bi-CNN employs two consecutive convolutional layers:

$$y_i^{(1)} = f\left(\sum_{j=0}^{k_1-1} x_{i+j}^{(0)} w_j^{(1)} + b^{(1)}\right) \tag{13}$$

$$y_i^{(2)} = f\left(\sum_{j=0}^{k_2-1} y_{i+j}^{(1)} w_j^{(2)} + b^{(2)}\right) \tag{14}$$

where:

- $y_i^{(1)}$ and $y_i^{(2)}$ are the outputs of the first and second convolutional layers, respectively.
- $x_{i+j}^{(0)}$ represents the input from the previous layer.

- $w_j^{(1)}$ and $w_j^{(2)}$ are filter weights for the two layers.
- $b^{(1)}$ and $b^{(2)}$ are the respective bias terms.
- f is the activation function (ReLU).
- k_1 and k_2 represent the kernel sizes of the first and second convolutional layers.

2. Max-Pooling Layer After two convolutional layers, a pooling layer reduces feature map dimensionality:

$$y_i^{(p)} = \max_{j \in P} y_{i+j}^{(2)} \tag{15}$$

where P is the pooling window.

3. Dropout Regularization Bi-CNN applies a two-stage dropout mechanism:

$$y_i^{(d)} = \frac{y_i^{(l)}}{1-p}, \quad \text{if not dropped} \tag{16}$$

with dropout rates of 30% and 40%, respectively.

4. Fully Connected Layer A 128-neuron fully connected layer is applied to further refine features:

$$y = f(Wx + b) \tag{17}$$

where W is the weight matrix, x is the feature vector, b is the bias, and f is the activation function (ReLU).

5. Softmax Classification The final softmax layer computes the probability of each class:

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{18}$$

where z_i is the logit and N is the number of classes.

6. Compilation and Optimization The model is compiled using:

- **Adam optimizer** with a learning rate of 0.001.
- **Sparse categorical cross-entropy loss**, defined as:

$$L = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{19}$$

Algorithm 3 Bidirectional CNN for Malware Detection

```
1:  Input: Training data $X_{train}$, Labels $Y_{train}$
2:  Output: Trained Bi-CNN model
3:  Initialize Sequential CNN model
4:  Add Conv1D layer with 64 filters, kernel size = 3, ReLU activation
5:  Add another Conv1D layer with 32 filters, kernel size = 3, ReLU activation
6:  Add MaxPooling1D layer with pool size = 2
7:  Add Dropout layer (rate = 0.3) for regularization
8:  Flatten feature maps
9:  Add Dense fully connected layer with 128 neurons, ReLU activation
10: Add Dropout layer (rate = 0.4)
11: Add Dense output layer with softmax activation
12: Compile the model with Adam optimizer and sparse categorical cross-entropy loss
13: Train the model for $epochs$ iterations
14: Evaluate the model on test data
```

3.6 Deeper CNN Model

The Deeper Convolutional Neural Network (Deeper CNN) expands on traditional CNN architectures by incorporating multiple convolutional and pooling layers. This deeper structure allows for hierarchical feature extraction, improving detection accuracy by capturing intricate data representations. Compared to the Basic CNN and Bidirectional CNN, this model employs additional convolutional and pooling layers, enhancing feature abstraction. Layers consist of:

1. Multiple Convolutional Layers Unlike previous CNN architectures, Deeper CNN applies multiple convolutional operations before pooling:

$$y_i^{(1)} = f\left(\sum_{j=0}^{k-1} x_{i+j}^{(0)} w_j^{(1)} + b^{(1)}\right) \tag{20}$$

$$y_i^{(2)} = f\left(\sum_{j=0}^{k-1} y_{i+j}^{(1)} w_j^{(2)} + b^{(2)}\right) \tag{21}$$

$$y_i^{(3)} = f\left(\sum_{j=0}^{k-1} y_{i+j}^{(2)} w_j^{(3)} + b^{(3)}\right) \tag{22}$$

where:

- $y_i^{(1)}, y_i^{(2)}, y_i^{(3)}$ represent outputs of the first, second, and third convolutional layers.
- $x_{i+j}^{(0)}$ is the input to the first layer.

- $w_j^{(1)}, w_j^{(2)}, w_j^{(3)}$ are filter weights.
- $b^{(1)}, b^{(2)}, b^{(3)}$ are bias terms.
- f is the activation function (ReLU).

2. Max-Pooling Layers Two max-pooling layers are applied to progressively reduce feature dimensions:

$$y_i^{(p1)} = \max_{j \in P} y_{i+j}^{(2)} \tag{23}$$

$$y_i^{(p2)} = \max_{j \in P} y_{i+j}^{(3)} \tag{24}$$

where P is the pooling window.

3. Dropout Regularization The Deeper CNN employs two dropout layers to prevent overfitting:

$$y_i^{(d1)} = \frac{y_i^{(l)}}{1-p_1}, \quad \text{if not dropped} \quad (p_1 = 0.4) \tag{25}$$

$$y_i^{(d2)} = \frac{y_i^{(l)}}{1-p_2}, \quad \text{if not dropped} \quad (p_2 = 0.5) \tag{26}$$

4. Fully Connected Layer The final dense layer maps features to class predictions:

$$y = f(Wx + b) \tag{27}$$

where:

- W is the weight matrix.
- x is the feature vector.
- b is the bias.
- f is the activation function (ReLU).

5. Softmax Classification The final classification layer computes class probabilities:

$$P(y_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{28}$$

where z_i is the logit and N is the number of classes.

6. Compilation and Optimization The model is compiled using:

- **Adam optimizer** with a learning rate of 0.001.
- **Sparse categorical cross-entropy loss**, given by:

$$L = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{29}$$

Algorithm 4 Deeper CNN

```
1:  Input: Training data $X_{train}$, Labels $Y_{train}$
2:  Output: Trained Deep CNN model
3:  Initialize Sequential CNN model
4:  Add Conv1D layer with 64 filters, kernel size = 3, ReLU activation
5:  Add another Conv1D layer with 64 filters, kernel size = 3, ReLU activation
6:  Add MaxPooling1D layer with pool size = 2
7:  Add Conv1D layer with 32 filters, kernel size = 3, ReLU activation
8:  Add another MaxPooling1D layer with pool size = 2
9:  Add Dropout layer (rate = 0.4)
10: Flatten feature maps
11: Add Dense fully connected layer with 128 neurons, ReLU activation
12: Add Dropout layer (rate = 0.5)
13: Add Dense output layer with softmax activation
14: Compile the model using Adam optimizer and sparse categorical cross-entropy loss
15: Train the model on training data for $epochs$ iterations
16: Evaluate the model on test data
```

3.7 Inception CNN

The Inception Convolutional Neural Network (CNN) is designed to enhance feature extraction through multi-scale convolutional operations. Let x represent the input data. The output of the convolutional layers can be mathematically expressed as:

$$F_{1x1} = \sigma(W_{1x1} * x + b_{1x1}) \tag{30}$$

$$F_{3x3} = \sigma(W_{3x3} * x + b_{3x3}) \tag{31}$$

$$F_{5x5} = \sigma(W_{5x5} * x + b_{5x5}) \tag{32}$$

where:

- $W_{k \times k}$ represents the weight matrix of the convolutional filter of size $k \times k$.

- $b_{k\times k}$ is the bias term.
- $*$ denotes the convolution operation.
- σ is the ReLU activation function:

$$\sigma(x) = \max(0, x) \tag{33}$$

The max pooling operation is computed as:

$$P = \max_{(i,j)\in R} x_{i,j} \tag{34}$$

where R denotes the receptive field of the pooling operation. The final feature representation is obtained by concatenating the outputs:

$$F_{\text{concat}} = \text{Concat}(F_{1x1}, F_{3x3}, F_{5x5}, P) \tag{35}$$

The final classification output is computed as:

$$y = \text{softmax}(W_d F_{\text{concat}} + b_d) \tag{36}$$

where W_d and b_d are the weight and bias parameters of the dense layer.

3.8 Performance Metrics Evaluation

Evaluating the performance of a malware detection model requires the use of appropriate metrics that assess its classification effectiveness. In this study, multiple metrics were employed to ensure a comprehensive analysis of model performance, including accuracy, precision, recall, F1-score, and the area under the receiver operating characteristic curve (AUC-ROC).

1. Accuracy: This measures the proportion of correctly classified instances out of the total instances evaluated. It is computed as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{37}$$

   where TP (True Positives) and TN (True Negatives) represent correctly classified malware and benign samples, respectively, while FP (False Positives) and FN (False Negatives) denote incorrect classifications.

2. Precision: This quantifies the correctness of positive predictions, indicating how many of the detected malware instances are actually malware. It is given by:

$$Precision = \frac{TP}{TP + FP} \tag{38}$$

3. Recall: Recall, also known as sensitivity or the true positive rate, assesses the model's ability to correctly identify all malware instances. It is formulated as:

$$Recall = \frac{TP}{TP + FN} \tag{39}$$

4. F1-Score: F1-score provides a balance between precision and recall. It is defined as

$$F1\text{-}Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{40}$$

5. Receiver Operating Characteristic (ROC) Curve and AUC: The ROC curve illustrates the trade-off between the true positive rate (recall) and the false positive rate (FPR) across different classification thresholds. The false positive rate is given by:

$$FPR = \frac{FP}{FP + TN} \tag{41}$$

   The area under the ROC curve (AUC) measures the model's ability to distinguish between malware and benign samples. An AUC value closer to 1 indicates superior classification performance, while a value near 0.5 suggests random guessing.

4 Result and Discussion

4.1 Exploratory Data Analysis

Using statistical measurements and graphical representations, exploratory data analysis (EDA) is used to find trends, verify hypotheses, and condense the dataset. By using a variety of visualization tools, it also aids in comprehending the quality and structure of the data, improving knowledge of the dataset as a whole.

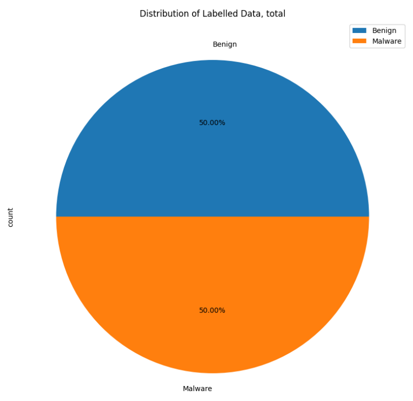

Figure 1: Distribution of Labeled Data

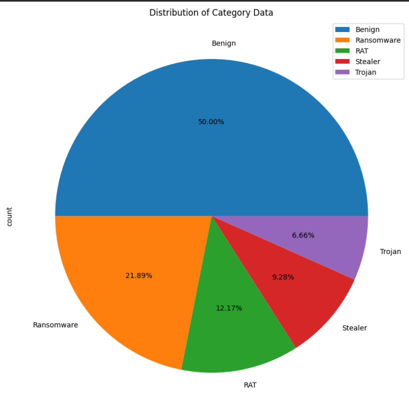

Figure 2: Distribution of Data Category

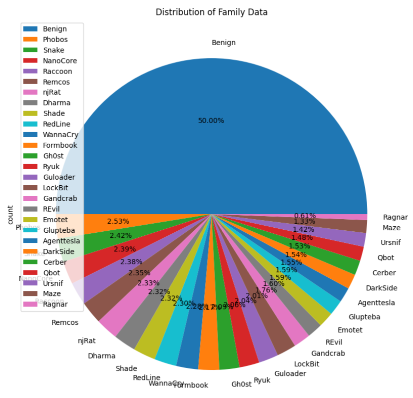

Figure 3: Distribution of Family Data

Figure 1 shows the balanced distribution between benign and malicious samples, preventing class imbalance issues that could bias results. Figure 2 breaks down the malware category into four subtypes: Ransomware (21.89%), Trojan (12.17%), Stealer (9.28%), and RAT (Remote Access Trojan, 6.66%). Ransomware is the largest threat category, followed by Trojans, which pose significant risks to organizations and individuals. Stealers and RATs are important attack vectors that often lead to credential theft and persistent network access. Figure 3 detailed the distribution of specific malware families, including Phobos, Snake, NanoCore, and WannaCry, each representing 1-2.5% of the total dataset.

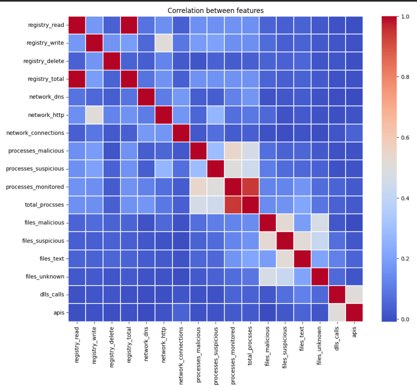

Figure 4: Feature Correlation

Figure 4 shows a feature correlation analysis used to identify malicious behavior. The system categorizes the activity into three main operational categories which are registry operations, network activities, and process execution and file operations. These categories are important for identifying malicious behavior. The duplication of certain features, such as registry_total and network related metrics indicates feature engineering.

4.2 Experimental Setup

The experimental framework was implemented using TensorFlow 2.6 with Keras API, developed in Python 3.8 and executed through Jupyter Lab for interactive evaluation. All deep learning models were constructed using TensorFlow?s native Keras implementation, ensuring consistency across architectural variants. The Python programming language served as the primary development environment, with standard scientific computing libraries (NumPy, Pandas) handling data processing and analysis.

For data pipeline optimization, we implemented the following TensorFlow Dataset operations: caching, shuffling, and prefetching. The cache() method was applied first to persist processed datasets in memory, eliminating redundant preprocessing operations across epochs. This proved particularly valuable given the multidimensional feature space of our malware dataset. Subsequent shuffling operations employed a buffer size of 1000 elements, effectively randomizing sample ordering while maintaining memory efficiency. The pipeline concluded with prefetching configured to automatic buffer sizing (tf.data.AUTOTUNE), enabling TensorFlow to dynamically optimize resource allocation based on available system memory and computational load. This three-stage preprocessing sequence significantly reduced inter-epoch latency while preventing potential ordering biases during training.

All convolutional neural network architectures were trained using the Adam optimizer with a consistent learning rate of $1.00 \times 10?4$. The sparse categorical cross-entropy loss function was selected to accommodate our multi-class classification task, with early stopping implemented to terminate training when validation loss failed to improve for three consecutive epochs. Model inputs were standardized through MinMax scaling and reshaped to (n_features, 1) dimensionality to accommodate 1D convolutional operations, as opposed to the 2D image inputs referenced in traditional computer vision applications.

Batch size selection represented a critical optimization parameter balancing memory constraints with gradient estimation quality. Through empirical testing, we established a uniform batch size of 32 samples across all model architectures, finding this configuration provided optimal convergence behavior while fully utilizing available GPU memory resources. Table 2 summarizes the key architectural parameters and training configurations for each CNN variant evaluated in our study.

Table 2: Example of Table

Model	Size	Filters	Learning Rate	Dropout	Epochs
Basic CNN	32	32-64	1.00×10^{-4}	0.25-0.5	20
Bidirectional CNN	32	64-128	1.00×10^{-4}	0.3-0.4	20
Deeper CNN	32	32-64	1.00×10^{-4}	0.4-0.5	10
Inception CNN	32	32	1.00×10^{-4}	N/A	10
ResMalNet	32	32-64	1.00×10^{-4}	N/A	20

The computational environment leveraged GPU acceleration through NVIDIA CUDA 11.2 and cuDNN 8.1 libraries, with all experiments conducted on RTX 3090 hardware. This configuration enabled efficient training of even the most complex residual architectures while maintaining reproducible results across multiple training runs. Memory optimization techniques, particularly through TensorFlow?s Dataset API, proved essential given the high-dimensional feature space of our malware classification task. The model training protocol employed a maximum of 20 epochs for most architectures, with the Deeper CNN and Inception-like variants limited to 10 epochs due to their computational complexity. This conservative epoch count was determined through preliminary experiments showing convergence typically occurred within this range. We implemented two critical Keras callback functions to optimize the training process:

- EarlyStopping: Monitored validation loss with a patience of 3 epochs, restoring the best weights upon termination. This prevented overfitting while ensuring model performance peaked before training cessation, as implemented in all architectures:

```
early_stopping = EarlyStopping(monitor=?val_loss?,
patience=3,
restore_best_weights=True)
```

The training process for each CNN variant followed a standardized procedure:

1. Initialization with He normal weight initialization
2. Batch normalization between convolutional layers (where applicable)
3. Dropout regularization with rates varying by architecture (0.25-0.5)
4. Validation on 20% of the training data each epoch

For the Residual CNN, we implemented custom residual blocks with skip connections, ensuring gradient flow through:

$$y = F(x, \{W_i\}) + W_s x \tag{42}$$

where $F(x, W_i)$ represents the residual mapping and $W_s x$ the shortcut connection. The Inception-like architecture employed parallel convolutional branches with kernel sizes of 1, 3, and 5, concatenating their outputs:

$$x_{out} = [\text{conv}_{1\times1}(x); \text{conv}_{5\times5}(x); \text{conv}_{5\times5}(x); \text{pool}(x)]$$

All models used the Adam optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$) with an initial learning rate of $10^{?3}$, demonstrating stable convergence across architectures. The batch size of 32 provided optimal balance between computational efficiency and gradient estimation accuracy given our dataset?s dimensionality and the available GPU memory capacity.

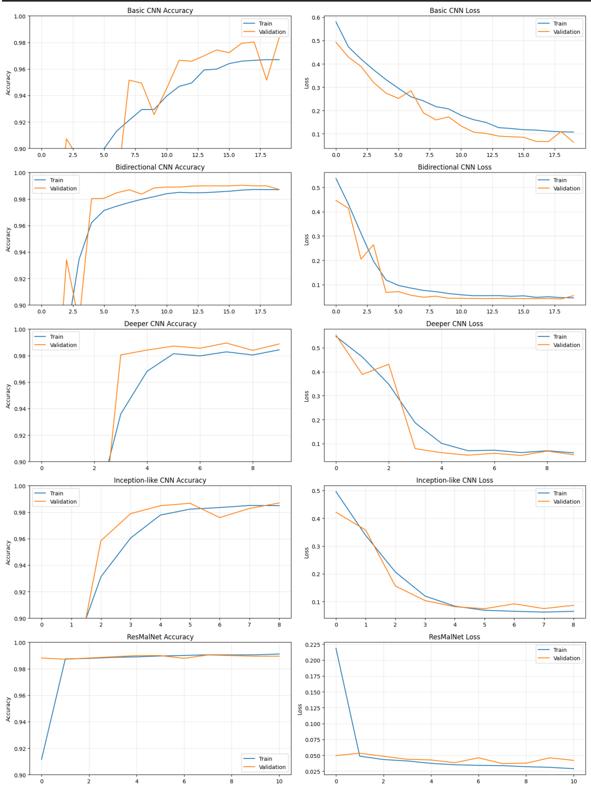

Figure 5: CNN Model variants Accuracy and Loss curve

Table 3: Example of Table

Model	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1-Score (%)**
Basic CNN	97.088569	97.188634	97.088569	97.087215
Bidirectional CNN	98.835428	98.846044	98.835428	98.835388
Deeper CNN	98.789458	98.803378	98.789458	98.789400
Inception-like CNN	98.697518	98.701947	98.697518	98.697505
ResMalNet	**98.912044**	**98.919068**	**98.912044**	**98.912023**

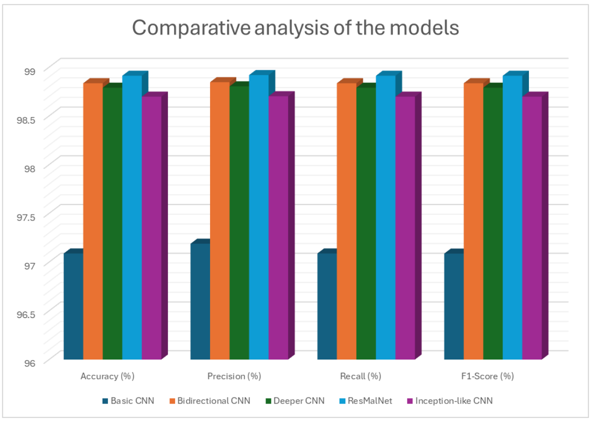

Figure 6: CNN Model variants Accuracy and Loss curve

4.3 Results

Figure 6, 5 and Table 3 demonstrate a clear performance hierarchy among the evaluated CNN architectures for malware detection. The Residual CNN emerges as the most effective model, achieving 98.91% accuracy, 98.92% precision, 98.91% recall, and 98.91% F1-score, indicating superior capability in both malware identification and classification consistency. This represents a 1.82 percentage point improvement over the Basic CNN baseline, which achieved 97.09% across all metrics.

The Bidirectional CNN and Deeper CNN show nearly identical performance, with marginal differences of 0.05-0.06 percentage points across metrics, both maintaining approximately 98.8% accuracy. The Inception-like architecture follows closely at 98.70% accuracy, suggesting that multi-scale feature extraction provides benefits, though slightly less effective than residual connections. Notably, all advanced architectures (Bidirectional, Deeper, Inception-like, and Residual) maintain remarkably consistent values across all four metrics, with less than 0.02 percentage point variation between precision, recall, and F1-score for each individual model.

The Basic CNN consistently under-performs relative to other architectures, with approximately 1.7-1.8 percentage point deficits across all evaluation metrics. This performance gap highlights the importance of architectural enhancements like residual connections and bidirectional processing for malware detection tasks.

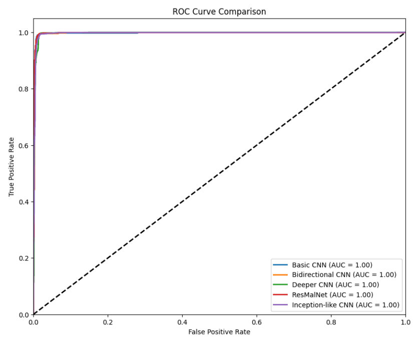

Figure 7: The ROC Curve for each comparison

Figure 7 evaluated the models showing a perfect discriminative capability on the dataset. Each model achieves an AUC of 1.00, indicating ideal class separation with entire proper positive type and zero false positives throughout all selection thresholds. This consistent performance in the ROC test demonstrates that all models effectively distinguish between the classes at a basic level.

Despite those identical AUC ratings, ResMalNet shows advanced overall performance using evaluation metrics. This apparent discrepancy arises from the awesome views those exclusive evaluation metrics provide. While the ROC curve and its related AUC degree evaluate the rating abilities of the models throughout the whole range of classification thresholds, the other metrics evaluate general performance at certain operational parameters, often the default threshold of 0.5. ResMalNet?s enhanced performance in those metrics indicates its residual architecture affords blessings in producing properly-calibrated chance estimates and making more assured correct classifications close to the choice boundary.

These metrics show that ResMalNet's residual connections and specific design components provide modest but significant gains in classification reliability. More robust feature extraction and representation learning are probably made possible by these architectural characteristics, especially for data that could otherwise be near-threshold misclassifications. Even though such enhancements don?t change the basic ranking ability as determined by the ROC curve, they would have a substantial influence on real-world deployment circumstances where the precise placement of the decision border has a crucial impact on model performance.

5 Conclusion

This study proposed ResMalNet for ransomware detection, it which combines domain-related feature engineering with fine-tuning numeric feature selection to improve deep learning performance. Initially based on expert knowledge, the model selects domains of critical behavioral relevance like registry operations, network activity, and file or process interactions, followed by applying statistical methods to select the most informative numeric features within each domain. This ensures the model obtains contextual relevance as well as data-driven effectiveness. The integration of these two levels of selection leads to improved accuracy and interpretability of detection, performing better than alternative CNN-based models. Strengthened data augmentation and regularization within the selected feature space helped to preserve the generalization ability of the model excellently. This work exemplifies the value of expert insight integrated with statistical optimization in designing useful ransomware detection systems. Future research should explore adaptive selection strategies to cater for evolving threats, the investigation of interactions across domains, and extending this framework to more general malware categories, consequently automated domain feature discovery for scalability.

References

[1] Al-Rimy, B. A. S., Maarof, M. A., and Shaid, S. Z. M. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. *Computers & Security*, 74:144–166, 2018.

[2] Alam, M. et al. Deepmalware: a deep learning based malware images classification. In *2021 International Conference on Cyber Warfare and Security (ICCWS)*, pages 93–99. IEEE, 2021.

[3] Alrzini, J. R. S. and Pennington, D. A review of polymorphic malware detection techniques. *International Journal of Advanced Research in Engineering and Technology*, 11(12):1238–1247, 2020.

[4] Basnet, M. et al. Ransomware detection using deep learning in the scada system of electric vehicle charging station. In *2021 IEEE PES Innovative Smart Grid Technologies Conference-Latin America (ISGT Latin America)*, pages 1–5. IEEE, 2021.

[5] Benmalek, M. Ransomware on cyber-physical systems: Taxonomies, case studies, security gaps, and open challenges. *Internet of Things and Cyber-Physical Systems*, 2024.

[6] Ganfure, G. O., Wu, C.-F., Chang, Y.-H., and Shih, W.-K. Deepware: Imaging performance counters with deep learning to detect ransomware. *IEEE Transactions on Computers*, 72(3):600–613, 2022.

[7] Gulmez, S. et al. Xran: Explainable deep learning-based ransomware detection using dynamic analysis. *Computers & Security*, 139:103703, 2024.

[8] Gyamfi, N. K. et al. Malware detection using convolutional neural network, a deep learning framework: Comparative analysis. 2022.

[9] Hasan, M. K. New heuristics method for malicious urls detection using machine learning. *Wasit*

Journal of Computer and Mathematics Science, 3(3):60–67, 2024.

[10] Hussain, A. et al. Enhancing ransomware defense: deep learning-based detection and family-wise classification of evolving threats. *PeerJ Computer Science*, 10:e2546, 2024.

[11] Hyunji, K. et al. Convolutional neural network-based cryptography ransomware detection for low-end embedded processors. *Mathematics*, 9(7):705, 2021.

[12] Kovács, A. Ransomware: a comprehensive study of the exponentially increasing cybersecurity threat. *Insights into Regional Development*, 4(2):96–104, 2022.

[13] Koyirar, W., Harris, B., Williams, J., Moreno, A., and Davis, E. Efficient ransomware detection through process memory analysis in operating systems. *Authorea Preprints*, 2024.

[14] Liu, X., Du, X., Lei, Q., and Liu, K. Multifamily classification of android malware with a fuzzy strategy to resist polymorphic familial variants. *IEEE Access*, 8:156900–156914, 2020.

[15] Malik, M. I., Ibrahim, A., Hannay, P., and Sikos, L. F. Developing resilient cyber-physical systems: a review of state-of-the-art malware detection approaches, gaps, and future directions. *Computers*, 12(4):79, 2023.

[16] Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., and Nicholas, C. Malware detection by eating a whole exe, 2017.

[17] Ravi, V. and Alazab, M. Attention-based convolutional neural network deep learning approach for robust malware classification. *Computational Intelligence*, 39(1):145–168, 2023.

[18] Rege, A. and Bleiman, R. Ransomware attacks against critical infrastructure. In *Proc. 20th Eur. Conf. Cyber Warfare Security*, page 324, 2020.

[19] Ryan, M. *Ransomware Revolution: the rise of a prodigious cyber threat*, volume 85. Springer, 2021.

[20] SL, S. D. and Jaidhar, C. Windows malware detector using convolutional neural network based on visualization images. *IEEE Transactions on Emerging Topics in Computing*, 9(2):1057–1069, 2019.

[21] Vince, J., Hemmingway, E., Penhaligon, R., Cattermole, A., and Swinburne, V. Segregated heuristic chains for advanced ransomware detection through generative anomaly patterns. *Authorea Preprints*, 2024.

[22] Yuan, Z., Lu, Y., Wang, Z., and Xue, Y. Droid-sec: Deep learning in android malware detection. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 371–372. ACM, 2014.

[23] Zhang, R. and Liu, Y. Ransomware detection with a 2-tier machine learning approach using a novel clustering algorithm. 2024.