Article

Multimodal Deep Learning for Android Malware Classification

James Arrowsmith *, **Teo Susnjak**, **Julian Jang-Jaccard**

School of Mathematical and Computational Sciences, Massey University, Auckland 0632, New Zealand; t.susnjak@massey.ac.nz (T.S.); julian.jang-jaccard@ar.admin.ch (J.J.-J.)
* Corresponding author: james.arrowsmith.1@uni.massey.ac.nz

Abstract: This study investigates the integration of diverse data modalities within deep learning ensembles for Android malware classification. Android applications can be represented as binary images and function call graphs, each offering complementary perspectives on the executable. We synthesise these modalities by combining predictions from convolutional and graph neural networks with a multilayer perceptron. Empirical results demonstrate that multimodal models outperform their unimodal counterparts while remaining highly efficient. For instance, integrating a plain CNN with 83.1% accuracy and a GCN with 80.6% accuracy boosts overall accuracy to 88.3%. DenseNet-GIN achieves 90.6% accuracy, with no further improvement obtained by expanding this ensemble to four models. Based on our findings, we advocate for the flexible development of modalities to capture distinct aspects of applications and for the design of algorithms that effectively integrate this information.

Keywords: multimodal deep learning for Android malware detection; enhanced malware analysis; graph neural networks; function call graphs (FCG); efficient multimodal late fusion; CNN GNN Ensemble; bytecode image analysis; Android APK analysis; data fusion

check for updates

Academic Editor: Francesco Buccafurri

Received: 19 January 2025
Revised: 10 February 2025
Accepted: 24 February 2025
Published: 28 February 2025

Citation: Arrowsmith, J.; Susnjak, T.; Jang-Jaccard, J. Multimodal Deep Learning for Android Malware Classification. *Mach. Learn. Knowl. Extr.* **2025**, *7*, 23. https://doi.org/10.3390/make7010023

1. Introduction

Multimodal machine learning integrates diverse data sources, or *modalities*, to deliver richer representations and more robust predictive capabilities [1,2]. In malware detection, research often focuses on a single modality, such as byte-level signatures or high-level control flow structures. Yet, malicious software can manifest in complex ways that demand broader perspectives. Integrating complementary modalities—specifically, binary images encoding Dalvik Executable (DEX) bytecode and function call graphs (FCGs)—can thus offer a more comprehensive characterisation of malicious behaviours.

Malware detection and classification remain pivotal challenges given the rapid proliferation of harmful Android applications. Three-quarters of the global market share is dominated by Android (as of December 2024), making it a highly attractive target for cyberattacks [3–5]. Since traditional unimodal detection methods risk overlooking critical patterns, particularly when apps employ encryption or obfuscation tactics [6,7], harnessing multimodal data fusion increases the probability of detection by capturing more informative features from both byte-level images and structural call graphs [8,9].

Several fusion strategies have been explored in multimodal learning, which are commonly classified into early, intermediate, and late fusion [2]. Early approaches concatenate modalities at the input level, intermediate approaches fuse extracted features in a shared layer, while late fusion integrates decision-level outputs. Studies suggest that late fusion reduces model complexity and the risk of overfitting by allowing each base classifier to specialise in its own domain [10–12]. Within this framework, convolutional neural networks

(CNNs) handle image-based texture patterns, and graph neural networks (GNNs) capture topological interactions in code. Such diversity in base classifiers can mitigate bias and variance by leveraging different algorithmic strengths [13–15].

This study addresses a critical gap in existing scholarship by integrating deep learning approaches to Android malware detection and classification that have traditionally been limited to unimodal analyses. We propose a late fusion approach that integrates CNNs trained on binary images and GNNs trained on FCGs via a multilayer perceptron (MLP) meta-classifier. Our results show that this strategy often surpasses unimodal deep learning across key classification metrics, underscoring the value of modality fusion for complex malware analysis.

Contributions

- We propose a novel multimodal approach for Android malware classification, combining CNNs on bytecode images and GNNs on FCGs. Predictions from these networks are concatenated as input to an MLP meta-classifier, effectively fusing low-level visual data and high-level control-flow structures.
- We assemble a dataset by matching FCGs from MalNet-Tiny [8] with the corresponding binary images in [9], both drawn from the AndroZoo repository [16].
- We evaluate four fusion strategies—two intermediate and two late—and identify late fusion as superior for detection and classification tasks.
- We benchmark four CNN architectures (ResNet18 [17], DenseNet [18], MobileNet-V2 [19], and a plain network) and two GNNs (GCN [20] and GIN [21]). Our findings reveal that all multimodal ensembles exceed the performance of their unimodal baselines, and that this improvement is fully realised by incorporating a single algorithm per modality.
- We extend the experiments of Freitas et al. [9] by introducing transfer learning via unfrozen ImageNet [22] weights, demonstrating improved CNN performance. We also show that encoding semantic information into colour channels can enhance classification accuracy, contrasting earlier observations [9].

The rest of the paper is organised as follows. Section 2 reviews relevant literature on malware detection and multimodal learning. Section 3 provides background on the methods used. Section 4 details our proposed model and experimental setup and presents a comprehensive performance analysis on MalNet-Tiny. Section 5 presents our results. Finally, Section 6 discusses our key findings and outlines directions for future research.

2. Related Works

Efforts to enhance malware detection and classification have ranged from static and dynamic to hybrid analyses [23]. Static methods examine programs without execution, yielding high coverage [24] but limited resilience against obfuscation and runtime-specific behaviour [25]. Dynamic approaches offer deeper behavioural insights by executing programs in sandboxes [26], though they often incur higher overhead and lower code coverage. Hybrid solutions combine static and dynamic features [27], yet mismatches between these analyses can complicate implementation [28].

Within static analysis, traditional signature-based techniques match attack signatures in a known database, detecting only previously observed variants. These methods can be easily evaded through polymorphism, where small source code changes significantly alter the compiled code [29]. Machine learning (ML), by contrast, provides more robust and automated methods [6,7]. Shallow ML algorithms, however, demand laborious feature engineering [30]. Modern deep learning circumvents this by learning directly from data, a property advantageous in malware detection given persistent evasion tactics and the lack of up-to-date labelled samples [31]. Acknowledging these challenges, the MalNet datasets [8,9] collectively supply over 1.2 million samples spanning 47 types and 696 families. While our work focuses on a subset for proof-of-concept, these large-scale resources address a central limitation identified in the literature: difficulty in capturing diverse, evolving threats. Interpretability of classifiers and explanations of their outputs, although critical [31], remains an open issue, alongside measures of efficiency to assess real-world feasibility of deployment and operationalisation of ML in this domain, are further issues raised [31].

2.1. Unimodal Methods: Images and Graphs

2.1.1. Image-Based Representation

Early research [32,33] transformed malware bytes into grayscale images and used K-nearest neighbours to attain strong results against polymorphic obfuscation. Subsequent work leveraged CNNs to automate feature extraction: Gibert et al. [34] outperformed hand-engineered features [32,35,36], ref. Rezende et al. [37] applied transfer learning, and Yadav et al. [38] fused EfficientNetB0 with ensemble learning to achieve high accuracy. DexRay [39] further demonstrated that one-dimensional CNNs excel in classifying large bytecode sets. Extending beyond grayscale, Gennissen et al. [40] and Freitas et al. [9] introduced semantic colour coding, heightening detection robustness.

2.1.2. Graph-Based Representation

Another line of research employs function call graphs (FCGs) to exploit high-level code structures. Early work by Kinable and Kostakis [41] used graph similarity metrics to cluster malware. Recognising limitations with traditional graph matching (e.g., inefficiency and limited scalability), research shifted to embedding-based methods: Hassen and Chan [42] turned FCGs into feature vectors via Minhash signatures. Pektaş and Acarman [43] combined FCG embeddings with deep learning for malware similarity detection. Other work includes Gascon et al. [44] and Xu et al. [45], who applied graph kernels and NLP-inspired embeddings, respectively. Gao et al. [46] integrated GNNs to detect malicious nodes, underlining the effectiveness of graph representation. Recently, attention-based architectures have emerged as a promising alternative to message passing for graph learning, including FCG malware detection [47–49]. Meanwhile, Freitas et al. [8] contributed MalNet-Graph, a large-scale FCG dataset, underscoring the efficacy of structured information in identifying malicious behaviours.

2.2. Beyond Unimodality: Multimodal Integration

While deep learning approaches—whether using images or graphs—are effective, unimodal approaches nevertheless risk overlooking critical elements of malware. Techniques such as DroidCat [50] and TFDroid [51] incorporate multiple features within a single static or dynamic modality, respectively, yet still do not unify distinct data modalities. Multimodal frameworks, however, offer a broader perspective. For instance, using Windows PE files, Ahmadi et al. [36] combined statistical and content-based features via early and late fusion with XGBoost, while Gibert et al. [52] proposed HYDRA, operating on multiple modalities such as API calls and opcode sequences. In Android detection, Kim et al. [53] merged seven features to train a deep neural network ensemble, and de Oliveira and Sassi [54] fused CNN, DNN, and Transformer outputs but relied partly on dynamic features, raising efficiency concerns.

2.3. Malware Detection with FCGs and Images

Studies with a closer resemblance to our work, notably Song et al. [55] and Li et al. [56], incorporated FCGs and DEX bytecode images for Android malware detection. Song et al. [55]

employed GraphSAGE [57] with SAGPool [58] alongside ResNet18 [17] (with CBAM [59]), using a soft attention-based fusion to assign modality-specific weight coefficients. On a subset of CICMalDroid2020 [60] and Androzoo [61] samples, they reported an F1-score of 98.6%. Li et al. [56] leveraged Androguard [62] to extract suspicious code snippets corresponding to sensitive API-adjacent nodes in an FCG. They vectorised these strings using UniXcoder [63] and deployed a fine-tuned Vision Transformer [64] on the bytecode images. Their fine-grained fusion used a two-layer transformer encoder [47] with attention across both modalities.

In contrast, our approach extends prior work by leveraging FCGs' structural properties directly and avoids complex alignment procedures. We employ distinct CNN and GNN pipelines for image and graph analysis and fuse high-level predictions via a meta-classifier. This modular strategy clarifies each modality's contribution and allows experimentation with diverse algorithms—such as ResNet [17], DenseNet [18], MobileNetV2 [19], GCN [20], and GIN [21]—in variably-sized ensembles, capturing broad algorithmic diversity [10–12]. Notably, prior research has not explored the impact of incorporating multiple, competing unimodal algorithms in multimodal malware detection and represents a gap that this study has sought to fill. Moreover, we focus of multi-category classification as it is a more challenging task than malware detection with a finer granularity [65]. Our evaluation on a curated subset of MalNet [8,9] offers a comprehensive test bed, with plans to extend to full-scale datasets in future work. Therefore, within the context of existing gaps in the literature, we formulate the following three research questions to guide our study:

- RQ1: Can simple multimodal fusion improve Android malware classification compared with unimodal models, justifying the additional computation?
- RQ2: Which simple fusion strategy is most effective?
- RQ3: Does adding further base models enhance multimodal malware classification?

RQ1 explores the effectiveness of simple feature fusion as evaluated in ablation studies across a range of algorithms. We incorporate a time metric to assess whether any performance improvements justify the additional computation. RQ2 compares the performance of four simple fusion strategies. RQ3 investigates the impact of additional unimodal models to determine if this improves multimodal performance.

3. Preliminaries

This section introduces key concepts for classifying Android malware, starting with the structure of Android APK packages to provide context before explaining how binary images and function call graphs are derived from APKs. Key machine learning models utilised in the study are also covered, aiming to establish the motivations and the design of our approach.

3.1. Android Packages

Android programs are usually written in Java and compiled into bytecode. APKs package compiled Android applications in an archive comprised of elements shown in Table 1.

Table 1. Components of an Android APK package.

File/Directory	Description
`AndroidManifest.xml`	Contains the application's metadata, such as name, version, and permissions.
`assets`	Directory of application asset files.
`META-INF`	Directory containing verification data, including signatures and certificates.

Table 1. *Cont.*

File/Directory	Description
`lib`	Directory of compiled native libraries with subdirectories based on the platform.
`res`	Directory containing resource values, such as colours, styles, and dimensions.
`resources.arsc`	Compiled resources file.
`classes.dex`	Compiled classes converted from JVM-compatible class files to Dalvik Executable (DEX) files, optimised for mobile devices with limited memory and processing power.

3.2. Binary Images

The DEX bytecode contains the primary execution logic, exposing the application's behaviour. Thus, malware developers commonly use packing and other obfuscation techniques to evade detection. However, by representing executables' bytecode as images, the texture may be analysed with computer vision [9]. This is often preserved in packed malware, as monotonic transformations rarely obscure these visual patterns. The process of statically extracting binary images from APKs is described by Freitas et al. [9] and summarised briefly as follows. The extracted DEX bytecode is converted into a one-dimensional array of 8-bit unsigned integers representing pixel activations (in the range of $[0, 255]$, where 0 is black and 255 is white). These 1D arrays are converted into 2D greyscale images using standard linear plotting with a fixed width and height determined by the file size, then scaled to 256×256 using Lanczos filtering. The contextual usage of each byte (e.g., pointer address, opcode, or ASCII character) can be optionally encoded into RGB channels based on its position within the DEX file structure with red for header bytes, blue for identifiers/class definitions, and green for data. In Section 4.2.1, we benchmark the inclusion of this semantic layer. Figure 1 illustrates the structure of an image representing an Android DEX file.

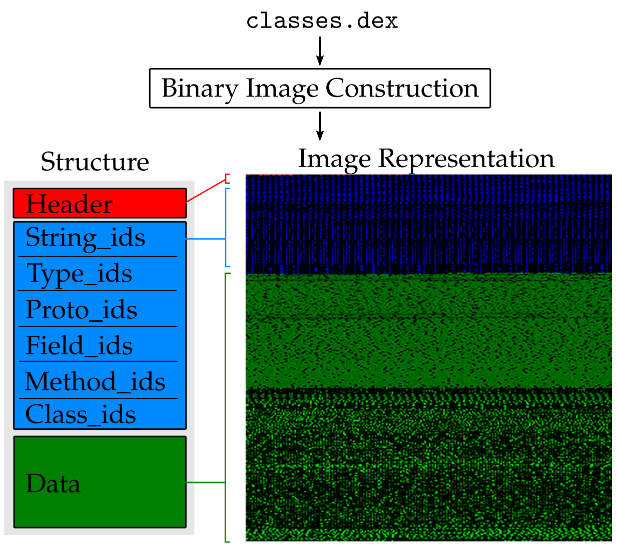

Figure 1. Binary image structure.

3.3. Function Call Graphs

Malicious actions (e.g., gathering sensitive information and transmitting it to an attacker) are often associated with a series of API calls. This behaviour can be modelled with an FCG, which represents all possible runtime execution paths. FCGs are highly valuable for malware classification, as they capture the caller–callee relationships between methods. An FCG extracted from an APK represents its executable control flow as a directed graph, $G(V, E)$, where V is the set of methods, and E represents inter-procedural calls. An edge $(v, w) \in E$ exists between $v, w \in V$ if v contains an `invoke` instruction referring to w. V may be partitioned into V_{internal} and V_{external}, such that $V_{\text{internal}} \cup V_{\text{external}} = V$ and $V_{\text{internal}} \cap V_{\text{external}} = \emptyset$, where V_{internal} contains methods that are defined and implemented within the application, and V_{external} comprises methods for which only definitions are included in the DEX file. Methods in V_{external} are imported from libraries and provide insight into the program's behaviour, as API packages provide interfaces for specific Android functionalities.

Freitas et al. [8] use Androguard [62] to produce the graphs by statically analysing APK files sampled from the AndroZoo repository [16]. Androguard parses the DEX code and extracts its methods and constructs the FCG by traversing the parsed code, following `invoke` functions. Directionality, disconnected components, and node isolates are preserved, while nodes are numerically relabelled to omit associated attribute information, as including this could increase the potential for reverse engineering. Instead, Local Degree Profile (LDP) [66] assigns node features $X \in \mathbb{R}^{|V| \times 5}$ composed of five-degree statistics that summarise the neighbourhood of v as shown in Figure 2. Since G is directed, there exists $v \in V$ for which $\deg^{-}(v) \neq \deg^{+}(v)$. Thus, $\deg(v)$ is given by the average indegree and outdegree, noting that for every $v \in V_{\text{external}}$, $\deg^{+}(v) = 0$, since these methods are not included in the bytecode. $\mathrm{DN}(v)$ denotes the multiset of the degree of all neighbouring nodes: $\mathrm{DN}(v) = \{\deg(w) | (v, w) \in E\}$. Figure 3 compares FCG and binary image representations of an APK sample.

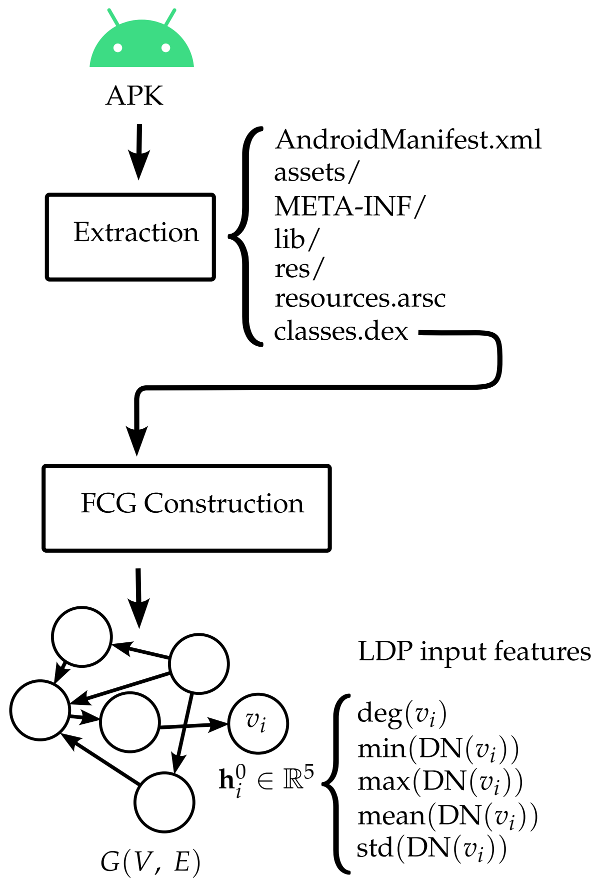

Figure 2. FCG extractor.

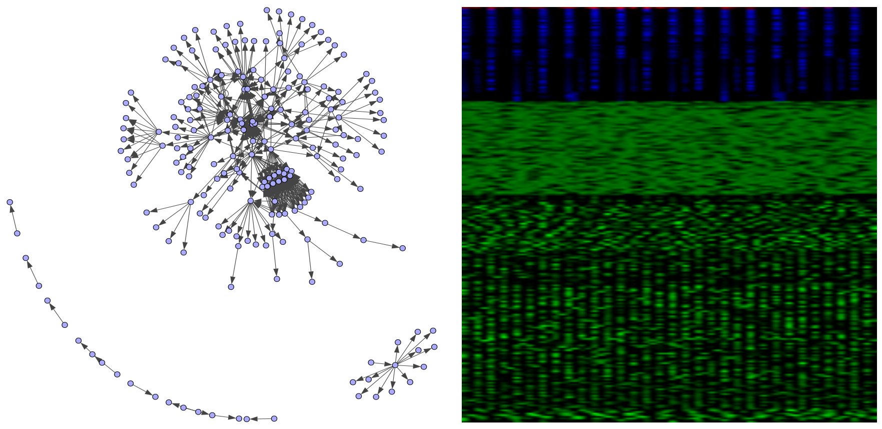
Figure 3. An example processed `adware` APK. (**Left**): FCG, (**right**): binary image.

3.4. Convolutional Neural Networks

Convolutional neural networks (CNNs) are pivotal for image classification and are thus relevant for this study. CNNs employ convolutional and pooling layers to extract features, followed by fully connected layers for classification. Convolutional layers apply learnable, spatially shared kernels across image regions, reducing model complexity and improving efficiency [67], with activations such as ReLU [68] introducing non-linearities to generate feature maps. As illustrated in Figure 4, multiple kernels produce distinct feature maps, each neurone connected to a local receptive field. Formally, in the kth feature map of layer l, the activation at (i, j) is

$$a^l_{i,j,k} = \sigma\left(\mathbf{w}^{l\,T}_k \mathbf{x}^l_{i,j} + b^l_k\right), \tag{1}$$

where $\mathbf{w}^l_k$ and b^l_k are the shared weights and bias, and σ is the activation function. By exploiting local connectivity, CNNs achieve translation invariance and locality [69]. For multi-channel inputs like RGB, kernels adjust in depth while remaining spatially compact.

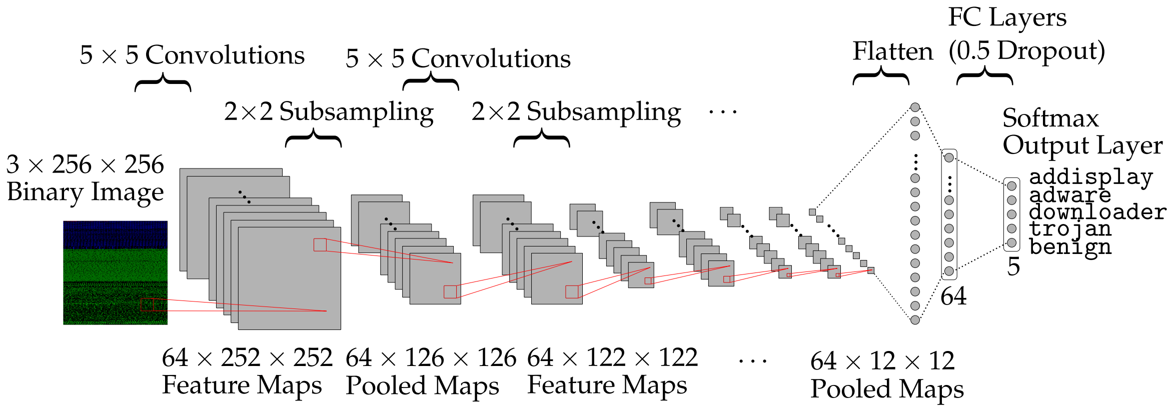

Figure 4. Three-layer CNN architecture.

Pooling layers downsample feature maps by computing summary statistics (e.g., max pooling [70]) over local regions:

$$p^l_{i,j,k} = \rho\left(a^l_{m,n,k}\right), \forall (m, n) \in \mathcal{R}_{ij}, \tag{2}$$

where $\mathcal{R}_{ij}$ denotes a local neighborhood. Stacking multiple convolutional and pooling layers (see Figure 4) enables detection of increasingly abstract features and compositionality [69]. The output is flattened, optionally regularised with dropout, and passed through fully connected layers, concluding with a (log) softmax for classification. Parameters are optimised by minimising a loss function, typically via stochastic gradient descent. Beyond the application of a plain CNN on our dataset as illustrated in Figure 4, we also leverage three advanced CNN architectures.

3.4.1. ResNet18

Deep networks improve efficiency for complex tasks [13], but plain architectures suffer from vanishing gradients. ResNet [17] mitigates this by using residual blocks with skip connections, which facilitate gradient flow and allow effective deep learning. Each residual block computes

$$F^k_{m+1} = \sigma\Big(g_c(F^k_{l\to m}, k_{l\to m}) + F^k_l\Big), \tag{3}$$

where g_c transforms the input F^k_l and σ is an activation function like ReLU. These identity shortcuts require no additional parameters, accelerating convergence and preventing vanishing gradients. ResNet18, a lightweight variant with 18 layers organised into four residual blocks of two 3×3 convolutional layers each, balances performance and computational cost. As reported in [8], among six evaluated models on the full MalNet-Image dataset, ResNet18 achieved the best compromise between accuracy and efficiency.

3.4.2. DenseNet121

DenseNet [18] combats vanishing gradients with cross-layer connectivity, linking each layer to all preceding layers. Instead of additive shortcuts, DenseNet concatenates feature maps:

$$F^k_l = g_k(F^k_1, \ldots, F^k_{l-1}), \tag{4}$$

resulting in $\dfrac{l(l+1)}{2}$ connections that improve gradient flow and regularisation. DenseNet's architecture allows layers to distinguish new information from that of previous layers, mitigating overfitting, though its narrow structure can become parameter-expensive with additional feature maps. DenseNet121, a lightweight variant featuring four dense blocks with 6, 12, 24, and 16 layers, respectively, connected by transition layers, maintains the benefits of deeper DenseNet variants while balancing performance and computational cost. It matches the performance of DenseNet169 on the full MalNet-Image dataset [8].

3.4.3. MobileNetV2

MobileNetV2 [19] is a lightweight CNN architecture designed for mobile and embedded applications, enhancing its predecessor MobileNetV1 [71] with more efficient feature extraction and representation learning. It retains depth-wise separable convolutions [72], which reduce parameters by decomposing convolutions into depth-wise and point-wise operations, applying kernels to each channel separately before combining outputs.

Key innovations include inverted residual blocks with linear bottlenecks, which expand channels using 1×1 convolutions, process them via 3×3 depth-wise separable convolutions, and then compress channels back with another 1×1 convolution. While ReLU activation is applied after the expansion and depth-wise steps, it is omitted before the final projection layer, preserving low-dimensional information and enabling more efficient feature learning. This design efficiently balances expressiveness and computational cost by maintaining critical information in low-dimensional bottleneck layers. MobileNetV2 begins with a fully convolutional layer containing 32 filters, followed by 19 residual bottleneck layers.

3.5. Graph Neural Networks

Many types of data are not applicable to a Euclidean domain and better suit a graph structure. These include social networks, molecular models, and the control flow of an executable. As shown in Figure 5, graph neural networks (GNNs) generalise CNNs to this data structure [69].

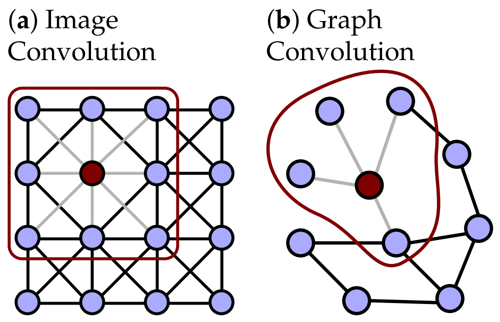

Figure 5. Image versus graph convolution. (**a**) Image convolution aggregates a (red) pixel's receptive field (red line), and (**b**) graph convolution aggregates a (red) node's neighbourhood (red line).

GNNs generally entail some form of recursive neighbourhood aggregation scheme whereby each node in the graph derives its new feature vector by aggregating the feature vectors of its neighbours [73]. The number of iterations determines the size of the neighbourhood whose structural information is captured in the transformed feature vector. The final node representations can be used for node classification and link prediction. For graph classification, a readout function aggregates these node embeddings into a graph-level representation. The readout function can be a basic permutation-invariant function, such as summing the node vectors, or a more complex graph-level pooling mechanism [74]. This way, GNNs may be applied to Android malware classification by way of the graph-level classification of extracted FCGs (Figure 6).

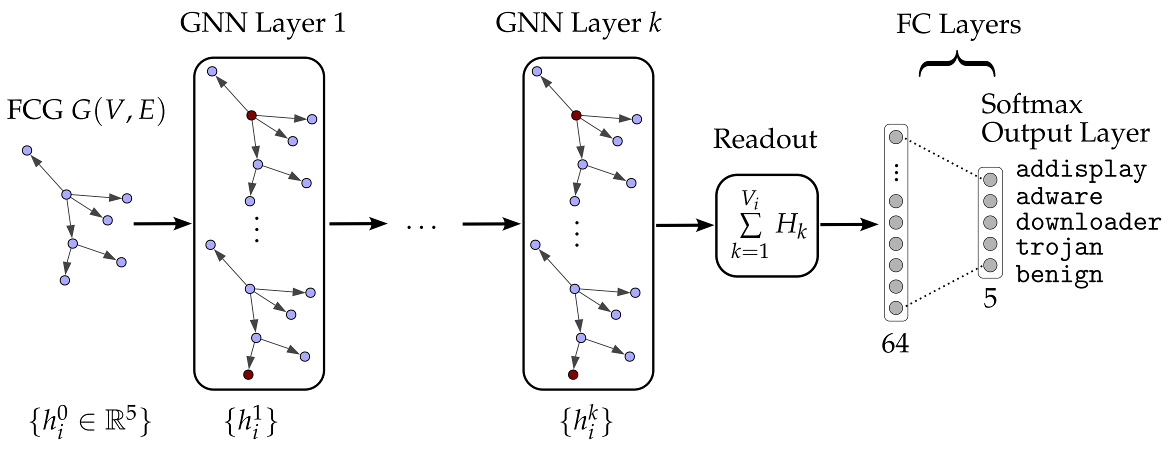

Figure 6. A Convolutional GNN for FCG classification.

The neighbourhood aggregation and graph-level pooling scheme depend on the GNN variant used. Moreover, GNNs can be classified into one of two types based on their neighbourhood aggregation strategy [15]. Spectral GNNs, motivated by graph spectral decomposition, seek to approximate spectral filters in the aggregating layers [20]. In contrast, spatial GNNs implement neighbourhood aggregation according to each node's spatial relations, rather than explicitly learning the graph's spectral features [21]. This paper evaluates one example of each category within the ensemble structure for Android malware classification: the GCN [20], a first-order approximation of a spectral GNN, and the GIN [21], a spatial GNN tailored for graph classification. To improve efficiency, graphs are fed in a block-diagonal adjacency matrix where each block represents one graph. A batch vector $\mathbf{b} \in \{0, \ldots, b-1\}^V$ assigns vertices to their graphs, where b is the batch size.

Feature matrices are concatenated into $X \in \mathbb{R}^{(\sum_1^b |V_i|) \times |F|}$. A simple pooling matrix then collects features from each respective graph. Thus, parallelisation over mini-batches is achieved with sparse block-diagonal adjacency matrices and concatenating feature and target matrices in the node dimension.

3.5.1. Graph Convolutional Network

The graph convolutional network (GCN) [20] extends convolutional operations to irregular graph structures by aggregating neighbouring node features to generate node embeddings. For a graph $G = (V, E)$ with feature matrix $X \in \mathbb{R}^{|V| \times k}$, self-loops are added to include each node's features in the aggregation: $\tilde{A} = A + I$, where A is the adjacency matrix and I is the identity matrix. The resulting matrix $\tilde{A}$ is then symmetrically normalised:

$$\hat{A} = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}, \tag{5}$$

where $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ is the diagonal degree matrix.

The GCN propagation rule for an n-layer network is

$$H^{l+1} = \sigma\left(\hat{A} H^l W^l\right), \tag{6}$$

where $H^0 = X$, $H^n = Z$, the final node embeddings. Here, σ is a non-linearity (e.g., ReLU), and W^l is the learnable weight matrix for layer l. This operation aggregates neighbourhood features in a permutation-invariant manner, analogous to convolutional filters in CNNs but adapted for unordered graphs. Figure 7 illustrates the GCN layer.

$$\begin{tikzcd}
h_i^l \arrow[r] & {} & & & \\
& & H^l \arrow[r] & \mathrm{ReLU} \arrow[r] & h_i^{l+1} \\
\{h_j^l\} \arrow[r] & \mathrm{Mean}_j \arrow[ur] & & &
\end{tikzcd}$$

Figure 7. GCN layer.

For graph classification, GCNs require a pooling operation to generate fixed-size graph embeddings. Global sum pooling, defined as

$$\mathbf{h}_{G_i} = \sum_{k=1}^{|V_i|} Z_k, \tag{7}$$

aggregates node embeddings Z for G_i into a graph-level representation, which is passed through a dense layer with a log-softmax output for classification. As in [21], the GCN is trained with the Adam optimiser [75]. We extracted node features using LDP, adding self-loops, as detailed in Section 3.3.

3.5.2. Graph Isomorphism Network

The graph isomorphism network (GIN) [21] is a spatial-based GNN designed to match the expressive power of the Weisfeiler–Lehman (WL) graph isomorphism test [76]. Unlike earlier GNNs that rely on heuristic aggregation schemes and can fail to distinguish non-isomorphic graphs, GIN employs an injective aggregation mechanism to enhance expressivity. Its node update rule is given by

$$h_v^k = \mathrm{MLP}^k\left(\left(1 + \epsilon^k\right) h_v^{k-1} + \sum_{u \in \mathcal{N}(v)} h_u^{k-1}\right), \tag{8}$$

where h_v^k is the feature of node v at layer k, $\mathcal{N}(v)$ denotes the set of its neighbours, and ϵ^k is a learnable or fixed scalar parameter. Figure 8 illustrates the GIN layer architecture.

In this study, we employ the GIN-0 configuration with $\epsilon = 0$, based on the effectiveness reported in [21]. For graph-level classification, node embeddings are aggregated by summing over the nodes in each layer and concatenating to form the final graph representation:

$$h_G = \text{concatenate}\Big(\text{sum}\{h_v^{(k)} \mid v \in G\}\Big)_{k=0}^{K}, \tag{9}$$

where K is the number of GIN layers. The final graph embedding h_G is passed to a classifier for prediction. Training is optimised using the Adam optimiser [75], with node features extracted using LDP [66] with added self-loops to improve the network's learning ability [77].

$$\begin{tikzcd}
h_i^l \arrow[r] & 1+\epsilon \arrow[dr] & & & & \\
& & \Sigma \arrow[r] & \text{MLP} \arrow[r] & \text{ReLU} \arrow[r] & h_i^{l+1} \\
\{h_j^l\} \arrow[r] & \Sigma_j \arrow[ur] & & & &
\end{tikzcd}$$

Figure 8. GIN layer.

3.5.3. Final Meta-Classifier

We employ a multilayer perceptron (MLP) as a meta-classifier to combine predictions from the CNN and GNN models. For each input, class probabilities from both models are concatenated into a $2n$-dimensional feature vector, which the MLP maps to an n-dimensional output representing final class predictions. The MLP is trained using stochastic gradient descent to minimise cross-entropy loss, effectively learning to fuse the complementary outputs of the base models for enhanced classification accuracy.

3.6. Data Fusion

Features from different modalities can interact in complementary, redundant, or cooperative ways [78]. Complementary fusion captures aspects of a phenomenon inaccessible to other modalities, while redundant fusion enables cross-validation by providing similar information. Cooperative fusion combines inputs to generate more complex insights, such as in audio–video data. Modality fusion strategies are typically categorised into early, intermediate, and late approaches based on the stage of data combination [2]. These strategies strike different balances between marginal representations—latent factors within individual modalities—and joint representations, which encode complementary, redundant, or cooperative information across modalities [1].

Early fusion concatenates input data into a single vector for model training, enabling joint representation learning while ignoring the origin of features. While straightforward, it does not identify meaningful marginal representations or higher-level cross-modal relationships and struggles with heterogeneous modalities and differences in dimensionality.

Intermediate fusion leverages knowledge of each feature's modality by first learning marginal representations. These representations are then fused, either for direct classification or to learn joint representations through additional layers. This approach preserves intra-modality information while also modelling cross-modal interactions, enabling the identification of further multimodal disentangled factors.

Late fusion combines predictions from unimodal models, allowing base models to learn marginal representations [1,2]. This benefits from imperfectly correlated errors of base models, which enhance complementarity without explicitly learning multimodal interactions [2]. Common aggregation strategies include simple averaging, assigning weights based on uncertainty, or optimising weights as hyperparameters. Meta-learning is an advanced approach where a meta-classifier learns to combine base model outputs, capturing complex relationships to improve generalisation.

4. Methodology

In this section, we introduce our proposed late-fusion model, which concatenates CNN and GNN predictions for use as input features for an MLP classifier (Figure 9). We first provide an overview of the data and evaluation methods used, which is followed by descriptions of the various experiments conducted with respect to training and testing phases.

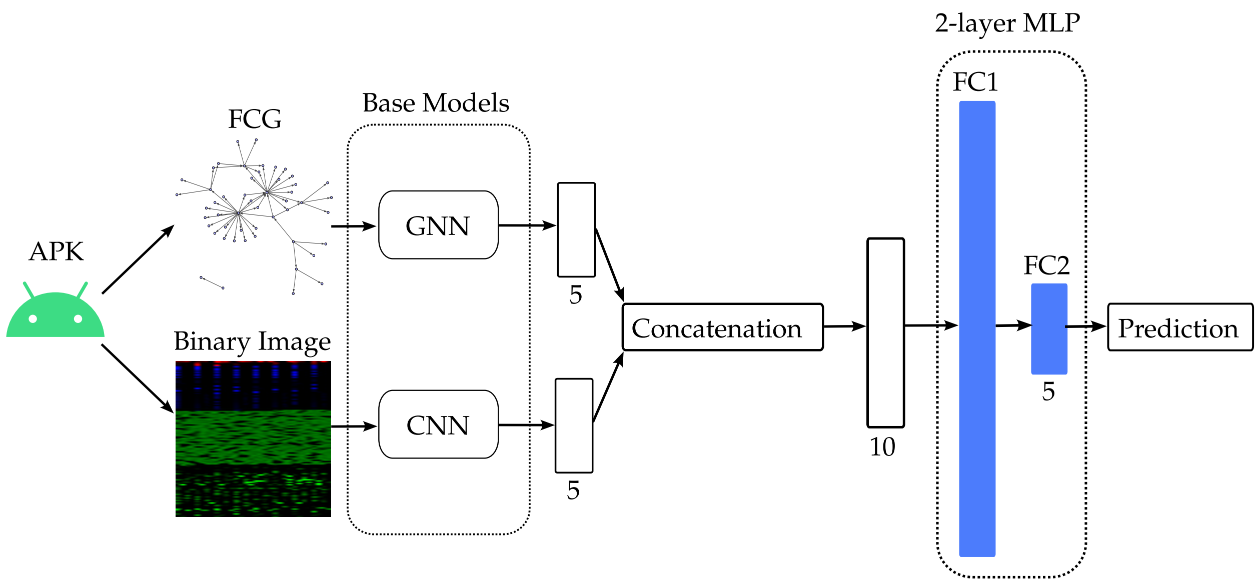

Figure 9. Late fusion approach for binary images and FCGs.

The CNN and GNN are pretrained on binary images and FCGs, respectively, which are derived from the same Android package samples via a process outlined in subsequent sections. Once trained, the CNN and GNN are frozen and form base classifiers within the ensemble structure. This structure is then trained. For each sample, the base-classifier predictions take the form of a probability vector for the five class labels: addisplay, adware, downloader, trojan, and benign. The probability distributions are concatenated, forming a 10-column vector. These become feature vectors for the meta-classifier, which outputs the final predicted label.

4.1. Data

The datasets used in this study are summarised in Table 2. The MalNet images dataset comprises 1.2 million 256 × 256 RGB binary images across 47 types and 696 families, while the MalNet-Tiny graphs dataset contains 5000 function call graphs representing five balanced malware classes. Both datasets share a common SHA-256 nomenclature, enabling the construction of an ensemble dataset of 5000 APK samples, each represented by both a binary image and an FCG, split into training, validation, and test sets in a 7:1:2 ratio.

Table 2. Summary of datasets used in this study.

Attribute	MalNet Images	MalNet-Tiny Graphs
Data Type	256 × 256 RGB binary images	Function Call Graphs (FCGs)
Scale	1.2 million images	5000 graphs
Features	Extracted from the APKs' DEX file (see Section 3.2)	Extracted from the APKs' DEX file (see Section 3.3)
Graph Details	N/A	Up to 5000 nodes per graph
Classes	47 types, 696 families	Five balanced types—addisplay, adware, downloader, trojan, benign
Nomenclature	SHA-256 hash	SHA-256 hash
Ensemble Dataset	5000 APK samples represented by both binary images and FCGs	
Data Split	7:1:2 training-validation-test split	

4.2. Training and Evaluation Metrics

While the full MalNet datasets are imbalanced, the constructed ensemble dataset is balanced. To that end, we evaluate our model using standard classification metrics—accuracy, precision, recall, F_1 score, and area under the receiver operating characteristic (ROC) curve. Precision and recall quantify the trade-off between false positives and false negatives for each class, with the F_1 score providing a harmonic mean of these two measures. Although accuracy is reliable on our balanced dataset, reporting precision, recall, and F_1 score ensures a more rigorous understanding of class-specific performance. To further assess discrimination capability independent of threshold selection, we analyse the ROC curve and compute the area beneath it (AUC). We reserve 10% of the data for validation, selecting models by validation accuracy. All experiments are run on an NVIDIA Tesla P100.

4.2.1. CNNs on Binary Images

We experiment with four CNN architectures on the 5000 MalNet image samples and adopt early stopping (patience of 10 epochs). Beyond a plain CNN, we evaluate ResNet18 [17], DenseNet121 [18], and MobileNetV2 [19].

Semantic information encoding. Following [9], we compare greyscale and RGB images that embed bytecode context into colour channels. An initial two-layer CNN shows that RGB encoding boosts validation (test) accuracy from 81.8% (80.1%) to 84.8% (82.2%). ResNet18 further confirms this trend, with accuracy rising from 84.8% (84.1%) to 87.6% (84.3%), so we proceed with RGB images. See Section 5 for details regarding the testing data.

Transfer learning with ImageNet. Transfer learning was evaluated by fine-tuning a ResNet18 model pretrained on ImageNet [22] for malware classification using our 5000 binary images from the MalNet dataset [8]. Freitas et al. [9] previously observed that freezing pretrained weights led to underperformance on the larger MalNet dataset, which the authors attributed to MalNet's size. In our experiments, freezing the pretrained weights reduced the validation (testing) accuracy from 86.4% (83.4%) to 78.6% (78.1%), suggesting that this approach is unsuitable even for a smaller subset. However, unfreezing the pretrained weights improved the accuracy to 88.2% (83.6%), enabling the model to better adapt to the specific characteristics of our images. This indicates that fine-tuning pretrained features bridges the domain gap more effectively than freezing, as it retains generalised knowledge while allowing task-specific learning.

Optimising a plain CNN. A hyperparameter grid search over layers {2, 3, 5}, hidden dimensions {32, 64, 128}, kernel sizes {3 × 3, 5 × 5}, dropout (applied before the output layer) {0, 0.5}, and learning rate {0.0001, 0.001} is conducted using ReLU [68] and Adam [75]. This approach is similar to the search used in [8] to optimise GNN architectures. The best model achieves 86.2% validation accuracy on epoch 49 with three convolutional layers, 64 hidden units, 0.5 dropouts, and a 0.001 learning rate (Figure 4).

Advanced CNN architectures. We evaluated three widely used CNN architectures—ResNet18 [17], DenseNet121 [18], and MobileNetV2 [19]—using RGB images with unfrozen ImageNet weights [22]. Each model was trained for 100 epochs with a learning rate of 0.001, and the epoch with the highest validation accuracy was selected. ResNet18 achieved 88.2% accuracy (epoch 67), DenseNet121 achieved 89.2% (epoch 60), and MobileNetV2 achieved 86.8% (epoch 66). Since these models demonstrated acceptable performance, we did not pursue further optimisation.

4.2.2. GNNs on FCGs

We followed the hyperparameters found in [8] for GNN architectures on the MalNet-Graph Tiny dataset—derived from a grid search similar to that used to optimise our plain CNN. Both GCN and GIN employ five layers with 64 hidden units without dropout and a learning rate of 0.0001. Each was trained for 1000 epochs without early stopping.

4.2.3. Fusion Strategies

We focus on intermediate and late fusion to accommodate the distinct data distributions of binary images and FCGs, as early fusion can degrade performance when feature distributions differ substantially [2]. For each method, the CNN and GNN models were pretrained and frozen before being integrated.

In intermediate fusion, we concatenate penultimate-layer embeddings from each unimodal model and optionally balance embedding sizes by adding two dense layers to the larger model. This approach captures cross-modal interactions at a higher abstraction level and potentially in a cooperative manner, for example, linking obfuscation-related artefacts in binary images with suspicious control flows in FCGs. For late fusion, we concatenate the final prediction probabilities of each model, reducing complexity and mitigating overfitting risks common in multimodal systems [79]. We train an MLP meta-classifier to integrate these and compare its performance to simple averaging—a tuning-free, interpretable approach.

We implemented both approaches using DenseNet121 (CNN) and GIN (GNN), training each ensemble for 100 epochs. ReLU activations [68] were used in the hidden layers for fusion, while the final layer employed a log-softmax function for classification across five malware classes. For balanced embeddings, we set the intermediate dimension of the added pre-concatenation layers to the mean of the base models' embedding sizes. The late-fusion variants with concatenated predictions achieved the highest validation accuracies (92.0% and 92.6%), surpassing intermediate-fusion methods (89.8% and 89.0%).

4.2.4. Optimising Ensembles

Subsequently, we extended experiments by optimising the MLP meta-classifier for learning rate, layer count, hidden dimensions, and dropout prior to the final layer. We employed a grid search to optimise CNN-GCN and CNN-GIN. CNN-GCN attained 90.4% validation accuracy on epoch 36 with a 0.001 learning rate, 0.5 dropouts, one hidden layer, and 32 hidden units. Its convergence is shown besides those of its base models in Figure 10. Next, CNN-GIN obtained 93.2%.

For subsequent pairings with larger CNN architectures, we used Optuna's Bayesian search [80] (30 trials, 20 epochs) to expedite this process. A MobileNetV2-GCN ensemble yielded 89.6% accuracy. DenseNet121-GCN obtained a validation accuracy of 91.2%, while DenseNet121-GIN achieved 93.2% on epoch 5, tuned with a 3.92×10^{-5} learning rate, 0.1 dropout rate, 3 hidden layers, and 89 hidden units.

Finally, we constructed a quad ensemble combining GIN, GCN, DenseNet121, and MobileNetV2 to assess the impact of ensemble size. Optuna found that 4 dense layers, 66 hidden units, 0.2 dropout, and a 4.15×10^{-4} learning rate matched DenseNet121-GIN with a best validation accuracy of 93.2%.

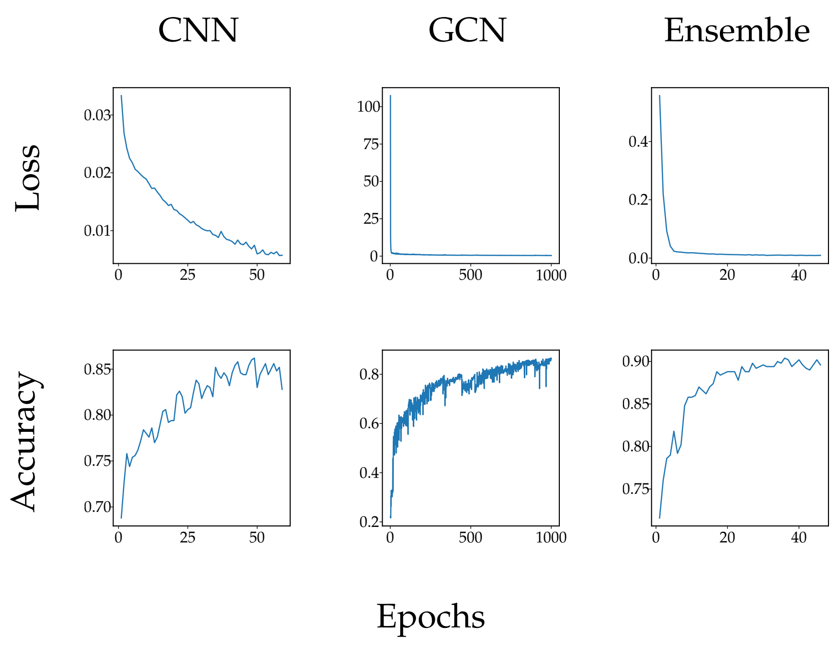

Figure 10. Training loss and validation accuracy over epochs for CNN, GCN, and CNN-GCN.

5. Results

The models described in Section 4 are evaluated on a dedicated test set comprising 20% of the data, reserved to verify the performance of the ensembles against each other and their constituent base models. We use the split in [8]—where MalNet graphs were divided into training, validation, and test sets with a 7:1:2 ratio, stratified across MalNet-Tiny labels. We source corresponding images for testing from the full MalNet image dataset [9] following the process described in Section 4.1.

5.1. Evaluating Fusion Strategies

The fusion strategy significantly impacts the performance of the DenseNet121-GIN ensemble. As shown in Table 3, the ensemble achieves its best testing performance with the late-fusion strategies across all metrics except time (see Section 5.2.4). Predictions, though compact, are highly informative. Embeddings provide richer input but may introduce unnecessary complexity, while balancing these with additional dense layers may result in information loss. The better overall performance of late fusion could suggest that the primary interaction between binary images and FCG features is complementary (non-overlapping). Within late fusion, simple averaging achieved performance comparable to meta-learning with our initial hyperparameters. Although optimising the MLP yielded the best results, simple averaging offers a highly interpretable alternative and avoids extra tuning.

Table 3. Test performance of DenseNet121-GIN across fusion approaches with macro-averaged metrics.

Model	Accuracy (%)	Precision (%)	Recall (%)	F_1 (%)	Time (s)
DenseNet121	86.3	86.7	86.3	86.3	3.61 ± 0.02
GIN	89.1	89.2	89.1	89.0	**1.28 ± 0.01**
Embeddings [1]	86.6	87.3	86.6	86.7	5.27 ± 0.05
Balanced Embeddings [1]	85.8	86.6	85.8	85.8	5.10 ± 0.15
Simple Averaging [2]	90.3	90.6	90.3	90.2	5.20 ± 0.09
Meta-Classifier (Initial) [2]	90.3	90.6	90.3	90.3	5.16 ± 0.12
Meta-Classifier (Optimised) [2]	**90.6**	**91.1**	**90.6**	**90.6**	4.85 ± 0.04

[1] Intermediate fusion. [2] Late fusion.

5.2. Model Evaluation

Table 4 compares the base and optimised late-fusion models on the test dataset across key performance metrics.

Table 4. Test performance of base and multimodal late-fusion models with macro-averaged metrics.

Model	Accuracy (%)	Precision (%)	Recall (%)	F_1 (%)	Time (s)
Plain CNN	83.1	83.5	83.1	83.1	1.70 ± 0.02
ResNet18	83.6	84.4	83.6	83.8	2.16 ± 0.04
DenseNet121 [1]	86.3	86.7	86.3	86.3	3.61 ± 0.02
MobileNetV2	84.7	85.0	84.7	84.6	2.21 ± 0.03
GCN	80.6	80.9	80.6	80.3	1.40 ± 0.08
GIN [2]	89.1	89.2	89.1	89.0	1.28 ± 0.01
Plain CNN + GCN [3]	88.3	88.8	88.3	88.4	2.49 ± 0.03
Plain CNN + GIN	90.3	90.6	90.3	90.3	2.42 ± 0.03
DenseNet121 + GCN	88.9	89.2	88.9	88.8	5.05 ± 0.10
DenseNet121 + GIN [4]	90.6	91.1	90.6	90.6	4.85 ± 0.04
MobileNetV2 + GCN	85.9	86.0	85.9	85.9	3.73 ± 0.07
DenseNet121 + MobileNetV2 + GCN + GIN	90.5	90.6	90.5	90.6	8.31 ± 0.17

[1] Best-performing CNN. [2] Best-performing GNN. [3] Largest improvement over base models. [4] Best-performing model.

5.2.1. Unimodal Algorithms

Advanced CNN architectures with unfrozen ImageNet weights outperform the plain CNN, with DenseNet121 leading. DenseNet121's improved gradient flow, owing to its dense connectivity and feature reuse, enables it to extract features from the binary images more effectively than the more efficient MobileNetV2, which notably outperforms the residual learning of ResNet18. For the GNNs, GIN outperforms GCN across each metric. This suggests that GIN's highly expressive injective aggregation function better captures the topological patterns of FCGs than the averaging mechanism of GCN. Comparing across modalities, while GIN surpassed each CNN, these, in turn, outperformed the GCN, highlighting the validity of both modalities and underscoring the significance of model choice.

5.2.2. Ablation Experiments

In all five ablation experiments involving two base models, integrating modalities with our late-fusion strategy improves performance across each metric, demonstrating the advantages of our approach for malware classification. The largest improvement is to the weakest unimodal models—the plain CNN-GCN ensemble achieves a test accuracy score of 88.3%, outperforming its best base model (CNN) by 5.2%. Following this are DenseNet121-GCN (2.6%), DenseNet121-GIN (1.5%), Plain CNN-GIN (1.2%), and MobileNetV2-GCN (1.2%).

The choice of GNN significantly affected the ensemble performance. For example, the plain CNN-GIN and DenseNet121-GIN ensembles boost accuracy to similar scores of 90.3% and 90.6%. In contrast, plain CNN-GCN and DenseNet121-GCN yielded lower accuracy scores of 88.3% and 88.9%. However, with the exception of MobileNetV2, stronger base models led to stronger ensembles with diminishing returns.

5.2.3. Quad Ensemble

Regarding the quad ensemble, our results reveal that increasing classifier diversity in a larger multimodal ensemble does not necessarily translate to better performance. The quad ensemble performed similarly to its strongest dual constituent (DenseNet121-GIN). Considering its protracted classification time, this argues against incorporating multiple base models per modality, as comparable performance gains may be achievable by extracting marginal representations with a single algorithm.

5.2.4. Inference Time

In Table 4, time is the number of seconds the model takes to classify 1000 testing samples. The CNNs perform as expected: the parametrically smaller plain CNN is the fastest, followed by the efficiency-optimised MobileNetV2, ResNet18, and the larger DenseNet121. GNNs were comparable to the plain CNN, while GIN slightly outpaced GCN. For the ensembles, classification times were generally equal to or less than the sum of the base model times, highlighting the efficiency trade-off in the multimodal approach. Nevertheless, all classification times were rapid—a few milliseconds per sample—falling well within acceptable ranges for practical applications. Table 3 indicates that the optimised meta-classifier achieved the shortest inference time, although differences among fusion methods were minimal.

5.3. Confusion Matrices

For a more detailed performance analysis of the plain CNN-GCN ensemble, which saw the largest accuracy improvement, Figure 11 shows the confusion matrices for the CNN, GCN, and ensemble on the test dataset. While the CNN and GNN misclassify between malicious types, the ensemble mitigates this. However, it occasionally remains susceptible to obfuscation, which presents obstacles to detection by allowing the malware to resemble innocuous software patterns. Along with the increased overall accuracy, we also notice some discrepancies between the base models in capturing class-specific patterns flattening out in the ensemble.

Figure 12 shows the confusion matrices for the best model—the DenseNet121-GIN ensemble—and its base models on the test dataset. While highly capable of distinguishing between malware types, there are likewise detection failures which indicate the use of obfuscation. While optimising the meta-classifier for accuracy yields superior overall performance for the ensemble, this focus results in missing trojan and adware samples, suggesting that re-tuning the MLP to prioritise recall might be beneficial.

Table 5 shows strong performance across the classes, with particularly high accuracy for addisplay (95.5%) and downloader (99.5%). However, trojan sees a notable drop in recall (78.5%), as some samples are misclassified—primarily as benign. Nevertheless, the model's overall accuracy of 90.6% reinforces its effectiveness.

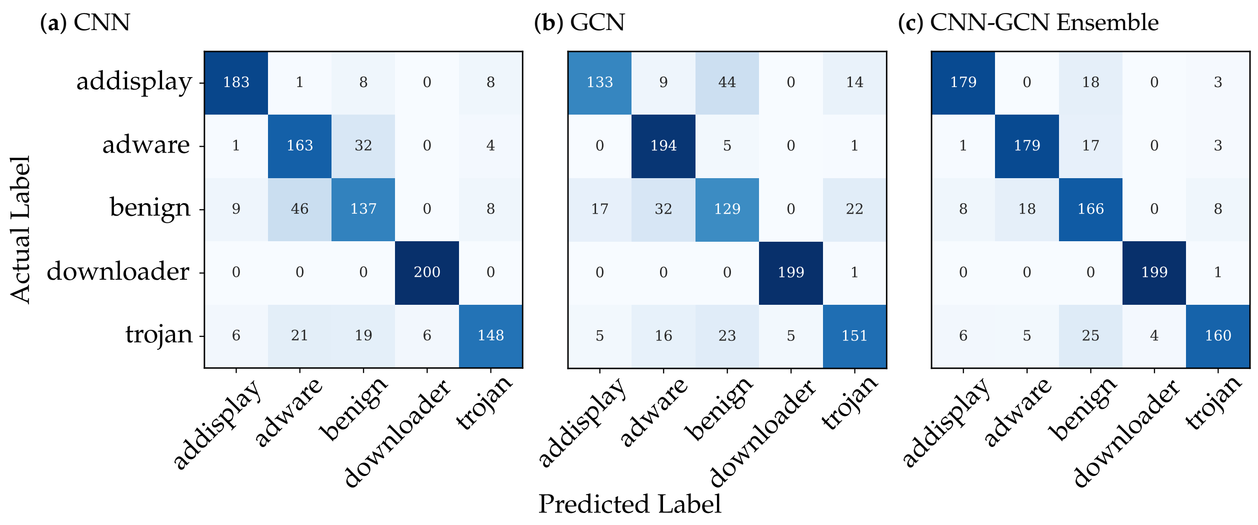

Figure 11. Base and ensemble confusion matrices for plain CNN-GCN on the test dataset.

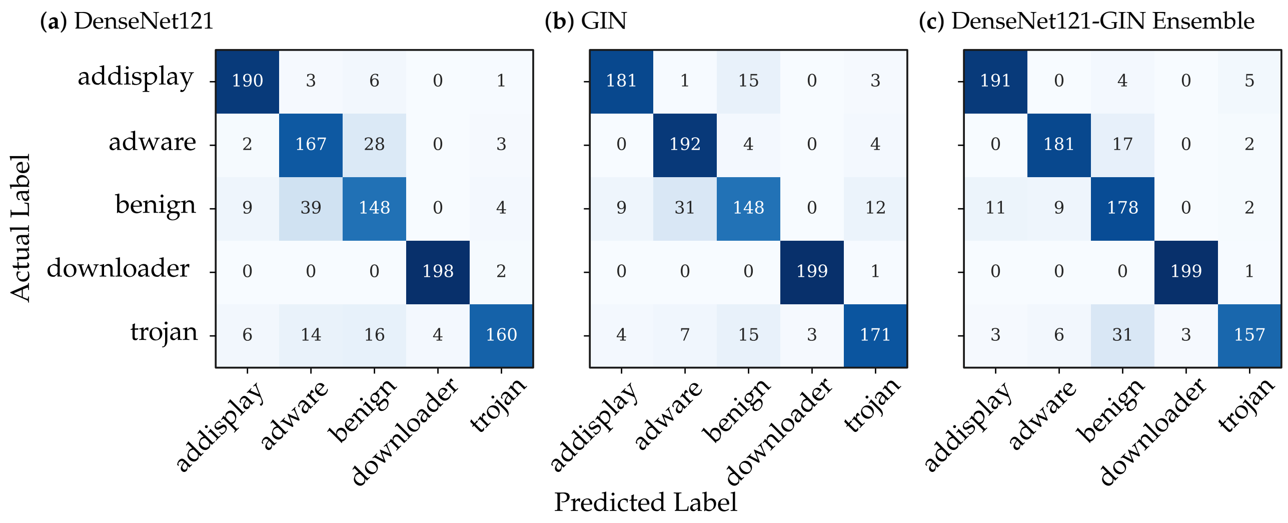

Figure 12. Base and ensemble confusion matrices for DenseNet121-GIN on the test dataset.

Table 5. Metrics for the DenseNet121-GIN ensemble on the test dataset for each class.

Class	Accuracy (%)	Precision (%)	Recall (%)	F_1 (%)
addisplay	95.5	93.2	95.5	94.3
adware	90.5	92.3	90.5	91.4
benign	89.0	77.4	89.0	82.8
downloader	99.5	99.5	98.5	99.0
trojan	78.5	94.0	78.5	85.6

5.4. UMAP Analysis

Uniform Manifold Approximation and Projection (UMAP) [81] is a dimensionality reduction technique well-suited for visualising high-dimensional data. By preserving both local and global structures, UMAP is particularly effective for analysing the relationships between embeddings in complex machine learning models. In this study, UMAP was used to project the high-dimensional dense layer embeddings (89 dimensions) from the test set of the best-performing DenseNet121-GIN model into a two-dimensional space (Figure 13). These visualisations provide insights into the feature utility and the model's ability to distinguish between classes.

(**a**) Coloured by Classification (**b**) Coloured by Detection

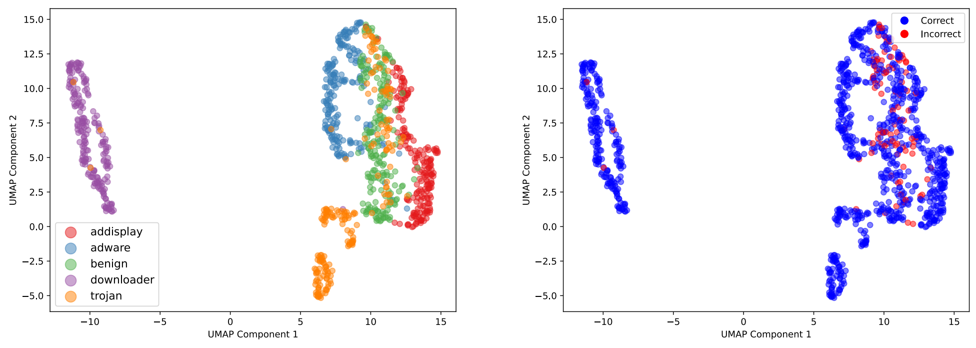

Figure 13. UMAP visualisation of DenseNet121-GIN embeddings on the test dataset.

With respect to class separability (Figure 13a), the embeddings show well-defined and generally distinct clusters for most classes, particularly downloader (purple). This indicates that the underlying features used in the multimodal fusion strategy successfully capture class-specific patterns. However, the overlap observed between benign and other classes, particularly trojan (orange), highlights the difficulty of correctly detecting malware. This is reinforced in Figure 13b, which shows that misclassified samples (red) are concentrated in the overlapping regions of benign. A small number of misclassifications are scattered throughout the other clusters, suggesting shared characteristics in their FCG topologies and/or binary image textures. These functional similarities underscore the added complexity of multi-type malware classification compared with detection solely. However, it is observable that most test samples are classified correctly, as indicated by the dense presence of blue points across clusters.

The outcome from the UMAP analysis, supported by Figure 12, is that the visualisations validate the effectiveness of both the underlying features and the late fusion strategy in integrating spatial and structural features for classification, enabling robust separation of malware classes. However, the overlap between benign and other classes, notably trojan, highlights the limitations of the current feature representations in handling obfuscation. Incorporating additional modalities, such as dynamic execution traces or contextual metadata, could further enhance class separability. Additionally, advanced fusion techniques, such as attention-based mechanisms, might help emphasise discriminative features. From a practical perspective, the implications of this analysis demonstrate the model's ability to generalise to unseen data, suggesting its readiness for deployment in real-world malware detection systems.

5.5. ROC Curve

The receiver operating characteristic (ROC) curve depicts the trade-off between True Positive (TP) and False Positive (FP) rates as the discrimination threshold is varied. In binary classification, calculating the area under the curve (AUC) provides the likelihood that the classifier will rank a randomly selected positive sample above a negative one. Figure 14a shows an ROC graph with AUC calculations for the strongest model, the DenseNet121-GIN ensemble, on the test dataset. Figure 14b focuses on the top-left corner of the ROC curve, the region of high sensitivity and low FP rate. Included are the curves for chance level, individual classes, and the macro- and micro-averages over five classes.

Addisplay (dotted turquoise), adware (dotted orange), and downloader (dotted green) curves remain close to the top left corner, suggesting excellent model performance for these classes. However, for trojan (dotted pink) and benign (dotted purple) the trade-off with respect to the cost in false positives becomes apparent, albeit at different rates, with increasing the threshold, thus further testifying to the challenge of reliably detecting trojan malware. Nevertheless, the AUC for the macro- (solid blue) and micro-average (solid red) ROC curves are both 0.98, confirming that our proposal is highly effective at classifying Android malware.

(a) ROC Curve **(b)** Zoomed-in View of Top-Left Corner

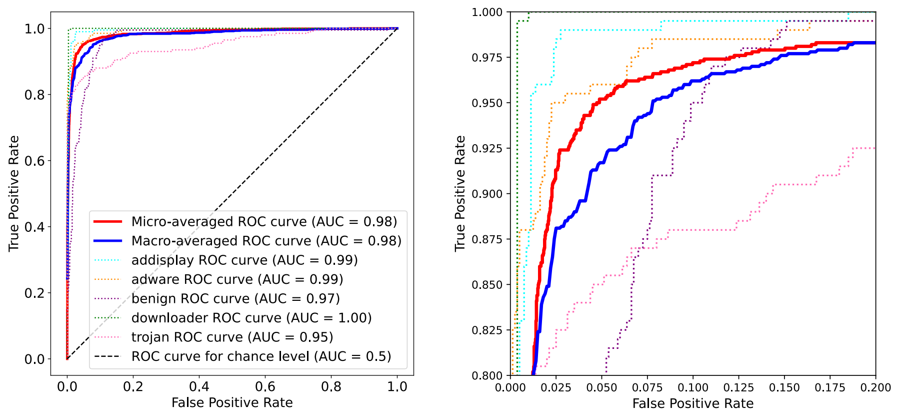

Figure 14. ROC Curve for DenseNet121-GIN on the Test Dataset.

5.6. Explaining Base Model Contributions

Although training an MLP on base model predictions outperformed simple averaging, it introduced opacity into the data fusion step. To counteract this, we compute SHAP (SHapley Additive exPlanations) values [82] to elucidate the base models' contributions to the final decision. SHAP borrows from game theory to attribute an importance score to each feature, providing an objective quantification. Figure 15 shows one-vs-all SHAP values for each class based on DenseNet121-GIN's test predictions. A positive (negative) SHAP value indicates a contributing (detracting) effect on the final decision, with its magnitude representing the strength of this influence. Base model predictions are shown in descending order according to the absolute mean SHAP value for each feature.

The SHAP analysis suggests that the MLP often reinforces its predictions by supplmenting higher base probabilities for a target class with lower probabilities for others. GIN downloader typically ranks highest across classes, exerting more influence on the MLP's decision than the target probabilities, with addisplay as the sole exception. This influence is generally detracting,though not consistently so. Downloader probabilities, particularly from GIN, contribute strongly to predicting this class. Overall, we observe the classifier prioritising GIN's highly accurate downloader probabilities, a strategy consistent with the distinct clustering observed for this class in Figure 13. Additionally, GIN addisplay and especially GIN trojan contribute to benign classifications. This reveals a means for the MLP to enhance overall accuracy following optimisation at the expense of reduced detection for these malware types (see Figure 12).

Figure 16 displays global feature importance, calculated as the mean absolute SHAP values for base predictions across test samples. This result confirms the model's reliance on the GIN's downloader probabilities and also indicates a greater overall dependence on GIN compared to DenseNet121, consistent with GIN's superior performance.

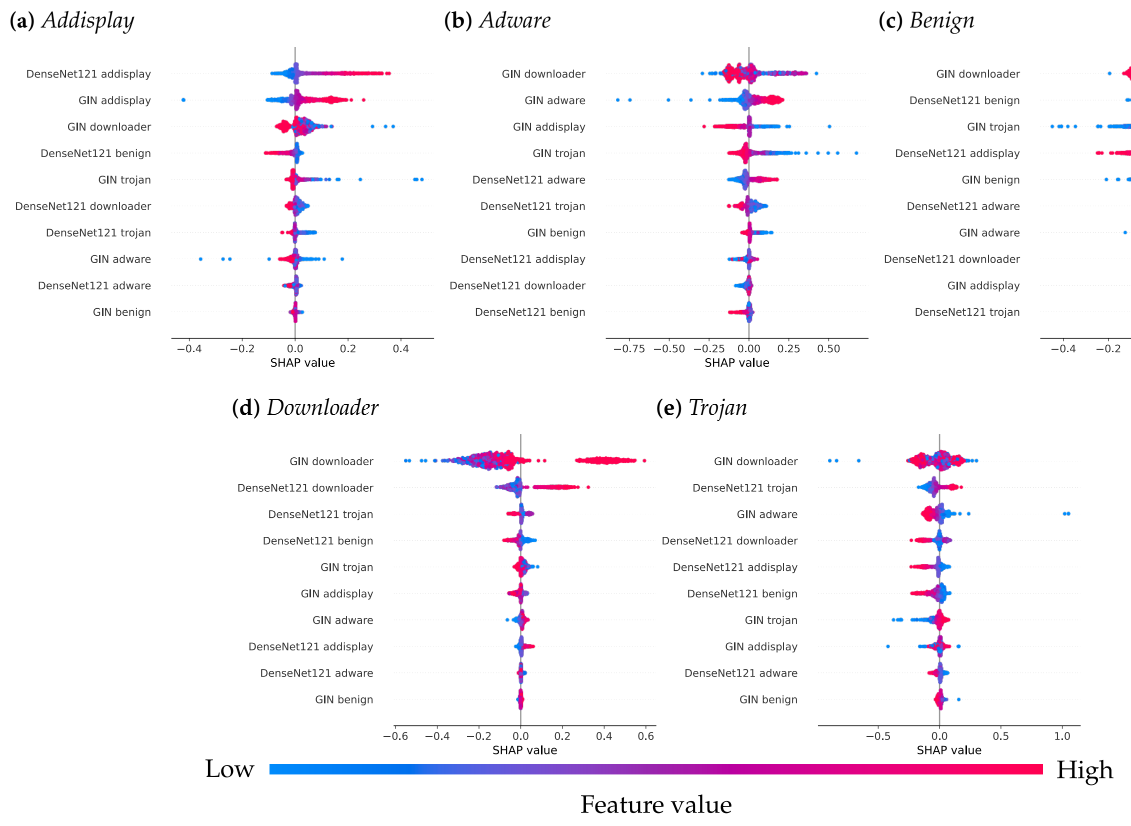

Figure 15. 5× One-vs-all SHAP graphs for DenseNet121-GIN on the test dataset.

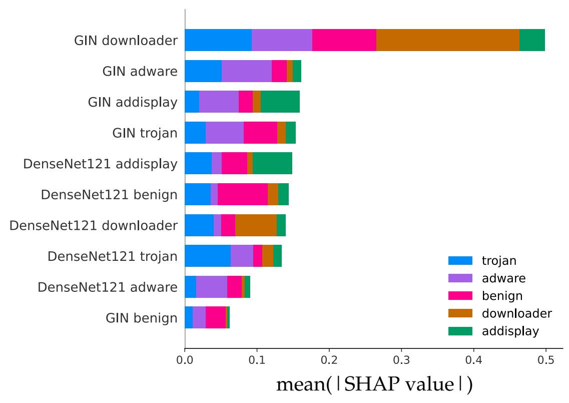

Figure 16. SHAP global feature importance graph for DenseNet121-GIN on the test dataset.

5.7. Comparison with Previous Works

Table 6 presents the accuracy of other methods applied to similar samples. In the absence of multimodal approaches sharing our data, we compare our results with graph-based methods using MalNet-Graph Tiny [8], which differs from MalNet-Image Tiny [9] in malware volume and types. Several studies [8,14,48] employ GCN and/or GIN-based methods, enabling a more direct comparison with the performance gains of our approach, while some [14,48,49,83–85] utilised alternative architectures.

DenseNet121-GCN outperforms DenseNet121 by 2.6% accuracy and GCN [8] by 7.9% accuracy. Moreover, while our reimplemented GIN model achieves 89.1%, and DenseNet121 attains 86.3%, fusing these models reaches 90.6%—surpassing the 90% reported for GIN, the strongest model in [8]. Jumping Knowledge (JK) [73] boosts GCN's accuracy to 89.7% [14], exceeding DenseNet121-GCN by 0.8%. However, our DenseNet121-GIN model achieves 90.6%, outperforming GIN-JK by 0.6%. Notably, JK did not enhance the GIN architecture [14], whereas our method consistently improves performance. These results further substantiate the effectiveness of our multimodal approach.

Table 6. Accuracy compared with previous works evaluated with MalNet-Graph Tiny.

Model	Category	Accuracy (%)
GCN [8]	Spectral-based GNN	81 *
GCN GST + EFD [48]	Spectral-based GNN	88.8
GCN-JK [14]	Spectral-based GNN	89.7
GIN [8]	Spatial-based GNN	90 *
GIN-JK [14]	Spatial-based GNN	90.0
GraphSAGE [48]	Spatial-based GNN	88.1
GraphSAGE GST + EFD [48]	Spatial-based GNN	89.2
GraphSAGE-JK [14]	Spatial-based GNN	94.4
HGNN [83]	Spectral-based HNN	87.6
HyperGCN [83]	Spectral-based HNN	79.3
HGNNP [83]	Spectral-based HNN	91.1
UniGIN [83]	Spatial-based HNN	89.0
GraphGPS [48]	Graph Transformer	90.8
GraphGPS GST + EFD [48]	Graph Transformer	92.5
Exphormer [49]	Graph Transformer	94.0
GraphCL [84]	Graph Contrastive Learning	87.6
Inferential SIR-GN [85]	Graph Structural Representation Learning	92 *
DenseNet121	CNN	86.3
GCN †	Spectral-based GNN	80.6
GIN †	Spacial-based GNN	89.1
DenseNet121 + GCN	CNN-GNN Ensemble	88.9
DenseNet121 + GIN	CNN-GNN Ensemble	90.6

* Presented with a precision of two significant figures, as specified in the referenced publication. † Our reproduction.

Other architectures have also been benchmarked on MalNet-Graph Tiny. GraphSAGE [57] obtained 88.1% accuracy, increasing to 89.2% with GST+EFD [48]—both values below DenseNet121-GIN. However, GraphSAGE-JK achieved 94.4% accuracy [14], though a GraphSAGE baseline was not provided in that study. DenseNet121-GIN outperforms three Hypergraph Neural Network (HNN) approaches, although it is surpassed by HGNNP [83]. GraphCL [86] shares a drawback with our approach by requiring two models to learn from the same samples; however, it only achieved an accuracy of 87.6% [84], 3.0% lower than DenseNet121-GIN [84]. However, Inferential SIR-GN [87] and Transformer-based methods [47] have exceeded the performance of the base and ensemble models used to validate our approach [48,49,85]

Although some architectures surpass our method in absolute performance, our multimodal approach generally compares favourably. Further optimising our base models (e.g., DenseNet121) may enhance ensemble performance. Nonetheless, the modular design of our late-fusion approach ensures compatibility with other unimodal techniques. Ablation experiments show consistent performance improvements (see Table 4), with the ensemble benefiting from stronger base models. Consequently, our framework can be readily extended to incorporate different unimodal algorithms to directly assess the performance gains from multimodal fusion.

6. Discussion

Addressing RQ1, the results confirm that simple multimodal fusion improves Android malware classification compared with unimodal models. Irrespective of the unimodal algorithm used, late fusion of binary images and FCGs consistently outperforms unimodal baselines across all key evaluation metrics. The notable accuracy gains, such as the 5.2% improvement seen with the CNN-GCN ensemble, not only justify the added computation but also suggest that leveraging complementary features from different modalities can enhance model robustness and adaptability to varied malware strains. Meanwhile, neither modality consistently outperformed the other, as unimodal performance depended on the architecture and parameters used—aligning with literature identifying both CNN-based textural bytecode analysis and GNN-based topological analysis of API calls as effective methods for malware feature extraction.

Our approach maintains the efficient code coverage characteristic of static analysis, as evidenced by classification times of a few milliseconds per sample—related to its simplicity, the enhanced multimodal classifications can require less time than the combined times of the unimodal components. Additionally, the meta-classifier is efficient to train, converging in a few epochs (e.g., 5 for DenseNet121-GIN). Moreover, on a single core of Google Cloud's general-purpose N2 machine, processing an RGB binary image or its corresponding FCG from an APK takes an average of 7.7 s (extrapolated from the reported one-week processing time on 16 N2 cores for the full MalNet datasets [8,9]) and, in contrast to dynamic analysis, this processing is hardware-unspecific. Additionally, 256×256 images and FCG edge lists take up minimal disc space. Thus, our approach—from raw APK through to classification—is highly streamlined, making it practical for real-world deployment. Yet, the holistic nature of our method also partially mitigates common challenges in static analysis, improving the detection of sophisticated malware that may evade other methods. By avoiding signature detection, our approach is less vulnerable to polymorphism. Further, the multimodal strategy may leverage graph-based detectors' resilience to obfuscation and concept drift, along with image-based robustness to APK-level adversarial attacks [65], resulting in protection against broader threats.

Further supporting practical uptake, our framework is straightforward to implement. The use of advanced deep-learning models at each stage automates feature extraction and analysis, eliminating the need for manual feature engineering which typically demands specialised expertise. Our simple fusion method also avoids complex and computationally intensive preprocessing steps (e.g., tokenisation and normalisation) required by other multimodal approaches [53]. In addition, our approach introduces a minimal number of tuneable parameters, reducing the time required for further optimisation. By pretraining unimodal algorithms independently, the search space increases linearly with additional algorithms. Moreover, optimising the meta-classifier is highly efficient. On an NVIDIA Tesla P100, Optuna [80] required only 3 h and 20 min to optimise DenseNet121-GIN. Advanced CNNs (e.g., DenseNet [18]) required minimal tuning, making this the only additional tuning beyond that for the GNN. Finally, we also propose simple averaging as a practical alternative that eliminates optimisation for the data fusion step.

Regarding RQ2, while we did not experiment with early fusion due to the heterogeneity of our modalities and models, our evaluation indicates that late fusion outperforms intermediate fusion strategies. This superiority likely arises from preserving modality-specific learning until the final aggregation step, which promotes an effective combination of diverse predictions and implies that modular architectures could benefit other multimodal tasks as well. Specifically, it suggests that integrating binary images and FCGs offers primarily complementary rather than cooperative or redundant features, with each modality largely influencing the target variable independently. This follows from the different information captured by either modality: while binary images represent low-level bytecode patterns, FCGs explicitly model a program's high-level behaviour, logical structure, and control flow. Their fusion thus provides primarily non-overlapping information, thereby reducing the emphasis on cross-modality relationships. This enabled us to benefit from the advantages of late fusion—namely, reducing model complexity and overfitting risk [79]. Yet, the superior performance of meta-learning over simple averaging suggests that it partly compensates for not explicitly modelling redundant and cooperative relationships. By learning complex relationships between base model predictions, the MLP effectively models (non-)linear interactions without directly learning correlations between features of different modalities.

In relation to RQ3, algorithmic heterogeneity within ensembles generally enhances classification performance [10–12]. Diverse learning approaches—as exhibited across models developed for different modalities—help mitigate individual model biases and errors, yielding more robust detection, especially when weaker base models are incorporated. This likely contributed to our finding that all multimodal models outperformed their unimodal counterparts. However, introducing an additional model to each modality did not further improve overall performance. This demonstrates the efficiency of our approach: the performance gains of combining binary images and FCGs can be realised by extracting marginal representations with one algorithm per modality, thereby minimising increases in computation and the hyperparameter search space. Nevertheless, it also emphasises the need for strategic selection of complementary models rather than simply increasing ensemble size.

Study Limitations and Future Work

This study is limited by its use of a curated subset of the MalNet dataset, which does not capture the full complexity of larger-scale datasets. Future work should extend the proposed framework to the complete MalNet dataset to validate scalability and generalisability.

Additionally, the use of static analysis, although efficient, limits the ability to handle obfuscation, dynamic loading, and other evasive techniques. While our multimodal deep-learning approach compensates for this to some degree, future work could explore combining our methods with dynamic analysis to address these limitations.

Moreover, our experiments focus primarily on simple fusion strategies and a limited set of CNN and GNN architectures. Expanding the exploration to include more advanced fusion techniques, such as attention-based mechanisms and diverse deep-learning architectures, could further enhance performance. Additionally, finding the optimal fusion strategy is non-trivial and highly dependent on the problem and modalities used. Automating this process is an active area of research [2] and is yet to be explored in the context of malware detection.

Another limitation is the increase in the number of tuneable parameters; however, since this grows linearly, the complexity remains manageable. Moreover, although our approach adds computational overhead by processing and integrating multiple modalities, multimodal classification times remained within a few milliseconds per sample, representing a favourable trade-off for enhanced performance. However, the use of deep learning may become a bottleneck in resource-constrained environments, where the smaller improvements to stronger unimodal models (e.g., GIN) might not justify the added computational cost. This aligns with [31] which argues that performance enhancements should be balanced with practical feasibility according to the application.

Finally, while this work demonstrates the efficacy of multimodal integration and employs SHAP [82] to interpret the fusion step, incorporating modal explainable AI methods, such as GradCAM [88] (binary images) or SubgraphX [89] (FCGs), in future research could offer more comprehensive interpretability, improving trust and adoption in real-world applications.

7. Conclusions

We propose a multimodal deep-learning approach for Android malware classification. We employ CNNs and GNNs to analyse binary images and function call graphs for subsequent fusion. Experimenting with four simple fusion methods—two intermediate and two late—we find that concatenating competing predictions to form inputs for an MLP meta-classifier performs best. This modular strategy fully exploits complementary features and consistently outperforms unimodal baselines across diverse architectures—for example, a plain CNN-GNN ensemble boosted overall accuracy by 5.2%. Meanwhile, the 90.6% accuracy of DenseNet121-GIN highlights the effectiveness of both the underlying features and the late fusion strategy in integrating their marginal representations for classification. Due to its static nature, our approach achieves these improvements with high efficiency, while alleviating several challenges common to other static techniques. Our setup automatically extracts features and analyses samples without laborious feature engineering or preprocessing—the latter often encountered in other multimodal deep-learning approaches. Finally, our experiments indicate that the performance gains from combining binary images and FCGs can be realised with just one algorithm per modality. Overall, our results demonstrate that multimodal deep learning provides a framework for developing algorithms that capture and integrate distinct application characteristics for robust Android malware classification.

Author Contributions: Conceptualization, J.A. and J.J.-J.; methodology, software, investigation, writing—original draft preparation, visualization, J.A.; formal analysis, writing—review and editing, J.A. and T.S.; supervision, funding acquisition, T.S. and J.J.-J.; All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Cyber Security Research Programme, Artificial Intelligence for Automating Response to Threats from the Ministry of Business, Innovation, and Employment (MBIE) of New Zealand as a part of the Catalyst Strategy Funds under the Grant Number MAUX1912.

Data Availability Statement: The source code for this research is openly available in the GitHub repository at https://github.com/jarrowsm/multimodal-malware-classification. The image and graph data are available at https://www.mal-net.org/.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Baltrušaitis, T.; Ahuja, C.; Morency, L.P. Multimodal machine learning: A survey and taxonomy. *IEEE Trans. Pattern Anal. Mach. Intell.* **2018**, *41*, 423–443. [CrossRef] [PubMed]
2. Ramachandram, D.; Taylor, G.W. Deep multimodal learning: A survey on recent advances and trends. *IEEE Signal Process. Mag.* **2017**, *34*, 96–108. [CrossRef]
3. Mobile Operating System Market Share Worldwide. Available online: https://gs.statcounter.com/os-market-share/mobile/worldwide (accessed on 30 December 2024).
4. Chen, L.; Xia, C.; Lei, S.; Wang, T. Detection, Traceability, and Propagation of Mobile Malware Threats. *IEEE Access* **2021**, *9*, 14576–14598. [CrossRef]
5. McAfee. *McAfee Labs Threat Report*; Technical Report; McAfee: San Jose, CA, USA, 2021.
6. Gong, L.; Lin, H.; Li, Z.; Qian, F.; Li, Y.; Ma, X.; Liu, Y. Systematically Landing Machine Learning onto Market-Scale Mobile Malware Detection. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 1615–1628. [CrossRef]
7. Abijah Roseline, S.; Geetha, S. A comprehensive survey of tools and techniques mitigating computer and mobile malware attacks. *Comput. Electr. Eng.* **2021**, *92*, 107143. [CrossRef]
8. Freitas, S.; Dong, Y.; Neil, J.; Chau, D.H. A Large-Scale Database for Graph Representation Learning. *CoRR* **2020**, arXiv:2011.07682.
9. Freitas, S.; Duggal, R.; Chau, D.H. MalNet: A Large-Scale Cybersecurity Image Database of Malicious Software. *CoRR* **2021**, arXiv:2102.01072.

10. Kuncheva, L.I.; Whitaker, C.J. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Mach. Learn.* **2003**, *51*, 181–207. [CrossRef]
11. Brown, G.; Wyatt, J.; Harris, R.; Yao, X. Diversity creation methods: A survey and categorisation. *Inf. Fusion* **2005**, *6*, 5–20. [CrossRef]
12. Ueda, N.; Nakano, R. Generalization error of ensemble estimators. In Proceedings of the International Conference on Neural Networks (ICNN'96), Washington, DC, USA, 3–9 June 1996; IEEE: Piscataway, NJ, USA, 1996; Volume 1, pp. 90–95.
13. Bengio, Y.; Courville, A.; Vincent, P. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* **2013**, *35*, 1798–1828. [CrossRef]
14. Lo, W.W.; Layeghy, S.; Sarhan, M.; Gallagher, M.; Portmann, M. Graph neural network-based android malware classification with jumping knowledge. In Proceedings of the 2022 IEEE Conference on Dependable and Secure Computing (DSC), Edinburgh, UK, 22–24 June 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1–9.
15. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Yu, P.S. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2020**, *32*, 4–24. [CrossRef]
16. Li, L.; Gao, J.; Hurier, M.; Kong, P.; Bissyandé, T.F.; Bartel, A.; Klein, J.; Traon, Y.L. AndroZoo++: Collecting Millions of Android Apps and Their Metadata for the Research Community. *CoRR* **2017**, arXiv:1709.05281.
17. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
18. Huang, G.; Liu, Z.; Van Der Maaten, L.; Weinberger, K.Q. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–27 July 2017; pp. 4700–4708.
19. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 4510–4520.
20. Kipf, T.N.; Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. *CoRR* **2016**, arXiv:1609.02907.
21. Xu, K.; Hu, W.; Leskovec, J.; Jegelka, S. How Powerful are Graph Neural Networks? *arXiv* **2018**, arXiv:1810.00826.
22. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami, FL, USA, 20–25 June 2019; IEEE: Piscataway, NJ, USA, 2009; pp. 248–255.
23. Kouliaridis, V.; Kambourakis, G. A comprehensive survey on machine learning techniques for android malware detection. *Information* **2021**, *12*, 185. [CrossRef]
24. Bacci, A.; Bartoli, A.; Martinelli, F.; Medvet, E.; Mercaldo, F.; Visaggio, C.A. Impact of Code Obfuscation on Android Malware Detection based on Static and Dynamic Analysis. In Proceedings of the International Conference on Information Systems Security and Privacy ICISSP, Funchal, Portugal, 22–24 January 2018; pp. 379–385.
25. Kouliaridis, V.; Kambourakis, G.; Geneiatakis, D.; Potha, N. Two anatomists are better than one—Dual-level android malware detection. *Symmetry* **2020**, *12*, 1128. [CrossRef]
26. Petsas, T.; Voyatzis, G.; Athanasopoulos, E.; Polychronakis, M.; Ioannidis, S. Rage against the virtual machine: Hindering dynamic analysis of android malware. In Proceedings of the Seventh European Workshop on System Security, Amsterdam, The Netherlands, 13 April 2014; pp. 1–6.
27. Tam, K.; Feizollah, A.; Anuar, N.B.; Salleh, R.; Cavallaro, L. The evolution of android malware and android analysis techniques. *ACM Comput. Surv. (CSUR)* **2017**, *49*, 76. [CrossRef]
28. McIntosh, T.; Kayes, A.; Chen, Y.P.P.; Ng, A.; Watters, P. Ransomware mitigation in the modern era: A comprehensive review, research challenges, and future directions. *ACM Comput. Surv. (CSUR)* **2021**, *54*, 197. [CrossRef]
29. You, I.; Yim, K. Malware obfuscation techniques: A brief survey. In Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications, Fukuoka, Japan, 4–6 November 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 297–300.
30. Qiang, C.Q.; Ping, L.J.; Gang, S.; Hui, W.Z. Ensemble Method For Net Traffic Classification Based On Deep Learning. In Proceedings of the 2021 18th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), Chengdu, China, 17–19 December 2021; pp. 466–470. [CrossRef]
31. McIntosh, T.; Susnjak, T.; Liu, T.; Xu, D.; Watters, P.; Liu, D.; Hao, Y.; Ng, A.; Halgamuge, M. Ransomware reloaded: Re-examining its trend, research and mitigation in the era of data exfiltration. *ACM Comput. Surv.* **2024**, *57*, 18. [CrossRef]
32. Nataraj, L.; Karthikeyan, S.; Jacob, G.; Manjunath, B.S. Malware images: Visualization and automatic classification. In Proceedings of the 8th International Symposium on Visualization for Cyber Security, Pittsburgh, PA, USA, 20 July 2011; pp. 1–7.
33. Nataraj, L.; Yegneswaran, V.; Porras, P.; Zhang, J. A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, Chicago, IL, USA, 21 October 2011; pp. 21–30.

34. Gibert, D.; Mateu, C.; Planes, J.; Vicens, R. Using convolutional neural networks for classification of malware represented as images. *J. Comput. Virol. Hacking Tech.* **2019**, *15*, 15–28. [CrossRef]
35. Kancherla, K.; Mukkamala, S. Image visualization based malware detection. In Proceedings of the 2013 IEEE Symposium on Computational Intelligence in Cyber Security (CICS), Singapore, 16–19 April 2013; IEEE: Piscataway, NJ, USA, 2013; pp. 40–44.
36. Ahmadi, M.; Ulyanov, D.; Semenov, S.; Trofimov, M.; Giacinto, G. Novel feature extraction, selection and fusion for effective malware family classification. In Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, New Orleans, LA, USA, 9–11 March 2016; pp. 183–194.
37. Rezende, E.; Ruppert, G.; Carvalho, T.; Ramos, F.; De Geus, P. Malicious software classification using transfer learning of resnet-50 deep neural network. In Proceedings of the 2017 16th IEEE International Conference on Machine Learning And Applications (ICMLA), Cancun, Mexico, 18–21 December 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1011–1014.
38. Yadav, P.; Menon, N.; Ravi, V.; Vishvanathan, S.; Pham, T.D. A Two-Stage Deep Learning Framework for Image-Based Android Malware Detection and Variant Classification. *Comput. Intell.* **2020**, *36*, 1163–1181. [CrossRef]
39. Daoudi, N.; Samhi, J.; Kabore, A.K.; Allix, K.; Bissyandé, T.F.; Klein, J. Dexray: A simple, yet effective deep learning approach to android malware detection based on image representation of bytecode. In *Deployable Machine Learning for Security Defense: Proceedings of the Second International Workshop, MLHat 2021, Virtual, 15 August 2021, Proceedings 2*; Springer: Berlin/Heidelberg, Germany, 2021; pp. 81–106.
40. Gennissen, J.; Cavallaro, L.; Moonsamy, V.; Batina, L. Gamut: Sifting Through Images to Detect Android Malware. Bachelor Thesis, Royal Holloway University, London, UK, 2017.
41. Kinable, J.; Kostakis, O. Malware classification based on call graph clustering. *J. Comput. Virol.* **2011**, *7*, 233–245. [CrossRef]
42. Hassen, M.; Chan, P.K. Scalable function call graph-based malware classification. In Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, Scottsdale, AZ, USA, 22–24 Matrch 2017; pp. 239–248.
43. Pektaş, A.; Acarman, T. Deep Learning for Effective Android Malware Detection Using API Call Graph Embeddings. *Soft Comput.* **2020**, *24*, 1027–1043. [CrossRef]
44. Gascon, H.; Yamaguchi, F.; Arp, D.; Rieck, K. Structural detection of android malware using embedded call graphs. In Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security, Berlin, Germany, New York, NY, USA, 2013; pp. 45–54.
45. Xu, P.; Eckert, C.; Zarras, A. Detecting and Categorizing Android Malware with Graph Neural Networks. In Proceedings of the 36th Annual ACM Symposium on Applied Computing, SAC '21, New York, NY, USA, 22–26 March 2021; pp. 409–412. [CrossRef]
46. Gao, H.; Cheng, S.; Zhang, W. GDroid: Android malware detection and classification with graph convolutional network. *Comput. Secur.* **2021**, *106*, 102264. [CrossRef]
47. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.U.; Polosukhin, I. Attention is All you Need. In *Advances in Neural Information Processing Systems*; Guyon, I.; Luxburg, U.V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; Garnett, R., Eds.; Curran Associates, Inc.: Newry, Northern Ireland, 2017; Volume 30.
48. Cao, K.; Phothilimthana, P.M.; Abu-El-Haija, S.; Zelle, D.; Zhou, Y.; Mendis, C.; Leskovec, J.; Perozzi, B. Large graph property prediction via graph segment training. *Adv. Neural Inf. Process. Syst.* **2023**, *36*, 23345–23361.
49. Shirzad, H.; Velingker, A.; Venkatachalam, B.; Sutherland, D.J.; Sinop, A.K. Exphormer: Sparse transformers for graphs. In Proceedings of the International Conference on Machine Learning. PMLR, Honolulu, HI, USA, 23–29 July 2023; pp. 31613–31632.
50. Cai, H.; Meng, N.; Ryder, B.; Yao, D. Droidcat: Effective android malware detection and categorization via app-level profiling. *IEEE Trans. Inf. Forensics Secur.* **2018**, *14*, 1455–1470. [CrossRef]
51. Lou, S.; Cheng, S.; Huang, J.; Jiang, F. TFDroid: Android malware detection by topics and sensitive data flows using machine learning techniques. In Proceedings of the 2019 IEEE 2nd International Conference on Information and Computer Technologies (ICICT), Kahului, HI, USA, 14–17 March 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 30–36.
52. Gibert, D.; Mateu, C.; Planes, J. HYDRA: A multimodal deep learning framework for malware classification. *Comput. Secur.* **2020**, *95*, 101873. [CrossRef]
53. Kim, T.; Kang, B.; Rho, M.; Sezer, S.; Im, E.G. A multimodal deep learning method for android malware detection using various features. *IEEE Trans. Inf. Forensics Secur.* **2018**, *14*, 773–788. [CrossRef]
54. de Oliveira, A.S.; Sassi, R.J. Chimera: An Android Malware Detection Method Based on Multimodal Deep Learning and Hybrid Analysis. *TechRxiv* **2020**. [CrossRef]
55. Song, J.; Li, R.; Zhang, Z. A Multi-modality Feature Fusion Method for Android Malware Detection. In Proceedings of the 2023 International Conference on Advances in Artificial Intelligence and Applications, Istanbul, Turkey, 13–15 October 2023; pp. 380–384.
56. Li, X.; Liu, L.; Liu, Y.; Liu, H. Detecting Android malware: A multimodal fusion method with fine-grained feature. *Inf. Fusion* **2025**, *114*, 102662. [CrossRef]
57. Hamilton, W.; Ying, Z.; Leskovec, J. Inductive representation learning on large graphs. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 1025–1035.

58. Lee, J.; Lee, I.; Kang, J. Self-attention graph pooling. In Proceedings of the International Conference on Machine Learning. PMLR, Long Beach, CA, USA, 9–15 June 2019; pp. 3734–3743.
59. Woo, S.; Park, J.; Lee, J.Y.; Kweon, I.S. Cbam: Convolutional block attention module. In Proceedings of the European conference on computer vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 3–19.
60. Mahdavifar, S.; Kadir, A.F.A.; Fatemi, R.; Alhadidi, D.; Ghorbani, A.A. Dynamic android malware category classification using semi-supervised deep learning. In Proceedings of the 2020 IEEE International Conference on Dependable, Autonomic and Secure Computing, International Conference on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, International Conference on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), Calgary, AB, Canada, 22–26 June 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 515–522.
61. Allix, K.; Bissyandé, T.F.; Klein, J.; Le Traon, Y. Androzoo: Collecting millions of android apps for the research community. In Proceedings of the 13th International Conference on Mining Software Repositories, Austin, TX, USA, 14–15 May 2016; pp. 468–471.
62. Desnos, A.; Gueguen, G. Android: From Reversing to Decompilation. *Proc. Black Hat Abu Dhabi* **2011**, *1*, 18–21.
63. Guo, D.; Lu, S.; Duan, N.; Wang, Y.; Zhou, M.; Yin, J. Unixcoder: Unified cross-modal pre-training for code representation. *arXiv* **2022**, arXiv:2203.03850.
64. Dosovitskiy, A. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv* **2020**, arXiv:2010.11929.
65. Gao, C.; Huang, G.; Li, H.; Wu, B.; Wu, Y.; Yuan, W. A Comprehensive Study of Learning-based Android Malware Detectors under Challenging Environments. In Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, Lisbon, Portugal, 14–20 April 2024; pp. 1–13.
66. Cai, C.; Wang, Y. A simple yet effective baseline for non-attribute graph classification. *arXiv* **2018**, arXiv:1811.03508.
67. Gu, J.; Wang, Z.; Kuen, J.; Ma, L.; Shahroudy, A.; Shuai, B.; Liu, T.; Wang, X.; Wang, G.; Cai, J.; et al. Recent advances in convolutional neural networks. *Pattern Recognit.* **2018**, *77*, 354–377. [CrossRef]
68. Nair, V.; Hinton, G.E. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (ICML-10), Haifa, Israel, 21–24 June 2010; pp. 807–814.
69. Bronstein, M.M.; Bruna, J.; LeCun, Y.; Szlam, A.; Vandergheynst, P. Geometric deep learning: Going beyond Euclidean data. *CoRR* **2016**, arXiv:1611.08097. [CrossRef]
70. Boureau, Y.L.; Ponce, J.; LeCun, Y. A theoretical analysis of feature pooling in visual recognition. In Proceedings of the 27th International Conference on Machine Learning (ICML-10), Haifa, Israel, 21–24 June 2010; pp. 111–118.
71. Howard, A.G. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017** arXiv:1704.04861.
72. Chollet, F. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 1251–1258.
73. Xu, K.; Li, C.; Tian, Y.; Sonobe, T.; Kawarabayashi, K.I.; Jegelka, S. Representation learning on graphs with jumping knowledge networks. In Proceedings of the International Conference on Machine Learning. PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 5453–5462.
74. Ying, R.; You, J.; Morris, C.; Ren, X.; Hamilton, W.L.; Leskovec, J. Hierarchical Graph Representation Learning with Differentiable Pooling. *CoRR* **2018**, arXiv:1806.08804.
75. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. *arXiv* **2014**, arXiv:1412.6980.
76. Leman, A.; Weisfeiler, B. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Tech. Informatsiya* **1968**, *2*, 12–16.
77. Wu, F.; Zhang, T.; Souza, A.H.d.; Fifty, C.; Yu, T.; Weinberger, K.Q. Simplifying Graph Convolutional Networks. *arXiv* **2019**, arXiv:1902.07153.
78. Castanedo, F. A review of data fusion techniques. *Sci. World J.* **2013**, *2013*, 704504. [CrossRef]
79. Wang, W.; Tran, D.; Feiszli, M. What makes training multi-modal classification networks hard? In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 12695–12705.
80. Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; Koyama, M. Optuna: A next-generation hyperparameter optimization framework. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage, AK, USA, 4–8 August 2019; pp. 2623–2631.
81. McInnes, L.; Healy, J.; Melville, J. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv* **2018**, arXiv:1802.03426.
82. Lundberg, S.M.; Lee, S. A unified approach to interpreting model predictions. *CoRR* **2017**, arXiv:1705.07874.
83. Zhang, D.; Wu, X.; He, E.; Guo, X.; Yang, X.; Li, R.; Li, H. Android Malware Detection Based on Hypergraph Neural Networks. *Appl. Sci.* **2023**, *13*, 12629. [CrossRef]
84. Gao, Y.; Hasegawa, H.; Yamaguchi, Y.; Shimada, H. Malware Self-Supervised Graph Contrastive Learning with Data Augmentation. *Int. J. Adv. Secur.* **2023**, *16*, 116–125.

85. Cuzzocrea, A.; Quebrado, M.; Hafsaoui, A.; Serra, E. A Graph-Representation-Learning Framework for Supporting Android Malware Identification and Polymorphic Evolution. In Proceedings of the 2023 10th IEEE Swiss Conference on Data Science (SDS), Zurich, Switzerland, 22–23 June 2023; pp. 34–41. [CrossRef]
86. You, Y.; Chen, T.; Sui, Y.; Chen, T.; Wang, Z.; Shen, Y. Graph contrastive learning with augmentations. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 5812–5823.
87. Layne, J.; Serra, E. Inferential sir-gn: Scalable graph representation learning. *arXiv* **2021**, arXiv:2111.04826.
88. Selvaraju, R.R.; Cogswell, M.; Das, A.; Vedantam, R.; Parikh, D.; Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 618–626.
89. Yuan, H.; Yu, H.; Wang, J.; Li, K.; Ji, S. On explainability of graph neural networks via subgraph explorations. In Proceedings of the International Conference on Machine Learning, PMLR, Virtual, 18–24 July 2021; pp. 12241–12252.