Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Image-Based Malware Family Classification: A Corrected, Controlled Comparison of CNN, CNN-to-LSTM and Fully Connected Networks

Abiola Richie Imafidon

imafidonaby@gmail.com

Independent Researcher

Research Article

Keywords: malware classification, convolutional neural network, LSTM, deep learning, malware visualization, cybersecurity

Posted Date: November 25th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11817145/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Abstract

Malware authors produce new variants by repacking and modifying existing code, yet variants of one family often keep a recognizable byte structure. Converting binaries into grayscale images lets standard image classifiers learn that structure without hand-crafted features. This study compares three neural-network designs on the same data: a convolutional neural network (CNN), a fully connected artificial neural network (ANN), and a CNN-to-LSTM pipeline in which a long short-term memory network classifies the CNN's outputs. The models were trained and validated on 8,684 images from nine Windows malware families and evaluated on a held-out test set of 2,176 images over three random seeds. The CNN achieved 94.24% ± 0.82% accuracy and a macro F1-score of 0.905 ± 0.019, while the ANN reached only 30.47% ± 3.46% accuracy and a macro F1-score of 0.112, close to always predicting the largest family. Adding the LSTM stage gave 94.73% accuracy against 94.19% for the same CNN code, a difference within seed-to-seed variation. Per-family results show near-perfect classification of Kelihos_ver3 (F1 0.995) and the weakest results for Simda, the smallest family, and for Tracur. The study also corrects an evaluation error in earlier master's thesis work on these models. Code, logs and evaluation scripts are publicly released.

1. Introduction

Malicious software remains one of the main tools of cybercrime, used to steal credentials and financial data, encrypt files for ransom, and recruit machines into botnets [1], [2]. The volume is large: McAfee [3] reported more than 600 million malicious samples in the first quarter of 2017 alone, and independent test laboratories record steady year-on-year growth [4].

Traditional antivirus tools match files against databases of known signatures. Signature matching is fast and precise for samples already catalogued, but it must be updated for every new variant [5]. Static analysis inspects code without running it but is weakened by obfuscation, while dynamic analysis observes behaviour in a sandbox at a high cost in time and computation; hybrid approaches combine the two [6], [7].

Most new samples, however, are not new families. They are variants of existing families produced with packers, encryption, compression or small code changes [7], [8]. Variants of one family tend to keep similar structure, and that structure becomes visible when the raw bytes of a binary are drawn as a grayscale image [5]. Image-based classification therefore offers a way to group unseen variants with their family without executing them or engineering features by hand.

Deep learning has made this approach practical. Convolutional neural networks learn spatial features directly from pixels [9], [10], and several studies have applied them to malware images [7], [11], [12]. Recurrent networks and fully connected networks are also used in security tasks, but they are less often compared with CNNs on identical image data under an identical training pipeline.

This paper addresses that gap. Its contributions are:

1. A controlled comparison of a CNN, a fully connected ANN and a CNN-to-LSTM pipeline on the same nine-family malware image dataset, with the same preprocessing, optimizer, loss and number of training epochs.

2. Evaluation on a held-out test set over three random seeds, reporting accuracy, macro-averaged F1-score, per-family precision, recall and F1-score, and a majority-class baseline.

3. An analysis of why each architecture succeeds or fails on byte-level images.

4. A corrected evaluation of earlier thesis experiments, which had compared predictions with misaligned labels, and public release of the code, logs and evaluation scripts for reproduction.

The remainder of the paper reviews related work (Section 2), describes the data and models (Section 3), reports results (Section 4), discusses limitations (Section 5) and concludes (Section 6).

2. Related work

2.1 Machine learning for malware analysis

Surveys trace the shift from signature matching toward learned models that can label malware without an explicit signature [13]–[15]. Early systems relied on hand-built features such as n-grams, opcodes and API calls; deep learning reduced that effort by learning representations from raw or lightly processed input [6], [16]. Android malware has received particular attention [17], [18].

2.2 Image-based malware classification

Nataraj et al. [5] showed that malware binaries rendered as grayscale images display family-specific textures and could be classified with standard image features. Later work replaced hand-crafted image features with CNNs: Gibert et al. [11] trained an end-to-end network on binary content, Lad and Adamuthe [12] improved a CNN for malware images, and Ravi and Alazab [7] added attention to a CNN for robust classification. Transfer learning from networks pre-trained on natural images has also been used [19]–[21], as has data augmentation to address small and imbalanced families [22]. Venkatraman et al. [4] combined image analysis with hybrid deep learning, and Yadav et al. [23] applied a two-stage image framework to Android malware.

2.3 Recurrent and hybrid architectures

Recurrent networks, and LSTMs in particular, model order in sequences and have been applied to API-call sequences [24], opcode sequences [25] and network traffic [26]. Convolutional-recurrent hybrids have reported strong results on behavioural data [27]. Lin and Yeh [28] treated binaries as one-dimensional byte sequences for a 1-D CNN. These studies suggest each architecture suits a different input form, but they rarely test the architectures against one another on the same image data.

2.4 Robustness concerns

Learned detectors can be evaded by inputs crafted to mislead them [29], and image-based malware classifiers in particular have been tested against samples generated by adversarial networks [30]. These findings motivate careful, reproducible evaluation of any architecture before it is relied on in practice.

3. Materials and methods

3.1 Dataset

The study uses the public *Malware-Classification-Dataset-Images* collection on Kaggle [31], version 1. It contains 10,860 grayscale images of Windows malware from nine families, already generated from the binaries and distributed in separate labelled training and test folders. According to its Kaggle description, the images are from the Microsoft Malware Classification Challenge, whose labelled training set covers the same nine families [32]. The families cover information stealers, adware, spam botnets, Trojans and obfuscated droppers. Table 1 gives the number of images per family.

Table 1
Images per family in the dataset's training and test folders.

Family	Training folder	Test folder	Total
Kelihos_ver3	2,353	589	2,942
Lollipop	1,982	496	2,478
Ramnit	1,226	307	1,533
Obfuscator.ACY	982	246	1,228
Gatak	810	203	1,013
Tracur	600	151	751
Vundo	380	95	475
Kelihos_ver1	318	80	398
Simda	33	9	42
Total	**8,684**	**2,176**	**10,860**

The classes are strongly imbalanced: Kelihos_ver3 makes up 27.1% of the test set, while Simda has only 9 test images. Macro-averaged F1 is therefore reported alongside accuracy, because accuracy alone rewards a model that favours large families.

3.2 Preprocessing and data split

Each image was resized to 150 × 150 pixels, loaded as three channels (the grayscale value repeated) and scaled to the range [0, 1]. No augmentation was applied. For each random seed, the training folder was divided into 80% for training and 20% for validation, stratified by family. The test folder was never used

for training or model selection. A check of file hashes found one test image identical to a training image; it was retained.

3.3 Models

The three designs follow the original thesis study [33].

- **CNN.** Three convolution blocks with 32, 64 and 128 filters (3 × 3 kernels, ReLU), each followed by 2 × 2 max pooling; then a 128-unit dense layer with ReLU, dropout of 0.5 and a nine-way softmax output.
- **ANN.** The image is flattened and passed through dense layers of 512 and 256 units (ReLU), each followed by dropout of 0.5, then a nine-way softmax output.
- **CNN-to-LSTM.** After the CNN is trained, its nine-value output for each image is treated as a one-step sequence and passed to two stacked LSTM layers of 128 and 64 units, with dropout of 0.5 between them, and a nine-way softmax output. The LSTM stage is trained on the CNN outputs for the training images. It is therefore a second-stage classifier on CNN predictions, not a recurrent model of the image itself.

3.4 Training

All networks used the Adam optimizer with its default learning rate of 0.001, sparse categorical cross-entropy loss and a batch size of 32, and were trained for a fixed 10 epochs without early stopping; the LSTM stage was also trained for 10 epochs. Each experiment was repeated with random seeds 42, 7 and 2024.

Two experiments were run. The first ("corrected evaluation") trained the CNN and ANN in a single TensorFlow data pipeline. The second ("original-code replication") ran the thesis code for the CNN and CNN-to-LSTM, with one change: an evaluation on the held-out test folder was added. Both ran on Kaggle with an NVIDIA T4 GPU, Python 3.13.16 and TensorFlow 2.21.0, in November 2026. The code and full run logs for both experiments are publicly available [34].

3.5 Evaluation

Each model was scored on the test set by accuracy, macro-averaged F1-score (each family weighted equally), and per-family precision, recall and F1-score. Results are reported as mean ± standard deviation over the three seeds. Predictions were made in a single unshuffled pass over the test set, so every prediction is compared with its own label. As a reference point, a classifier that always predicts the largest family, Kelihos_ver3, would score 27.07% accuracy and a macro F1-score of 0.047.

The 2024 thesis [33] reported precision, recall and F1-scores from an earlier version of these experiments. That code compared predictions made on a shuffled data loader with labels in their original order, which produces near-chance class-level scores; the LSTM stage was also trained on misaligned labels. All results in this paper come from the corrected evaluation and supersede the thesis figures. A large language model (Claude, Anthropic) was used to help prepare the corrected evaluation code and to

draft the text of this article; the author ran all experiments, checked every reported figure against the run logs and takes full responsibility for the content.

4. Results

4.1 Overall performance

The CNN clearly outperformed the fully connected ANN (Table 2). Across three seeds it reached 94.24% ± 0.82% test accuracy and a macro F1-score of 0.905 ± 0.019. The ANN reached 30.47% ± 3.46% accuracy and a macro F1-score of 0.112 ± 0.053, only slightly above a classifier that always predicts the largest family.

Table 2
Corrected evaluation on the held-out test set (2,176 images), mean ± standard deviation over seeds 42, 7 and 2024.

Model	Accuracy (%)	Macro F1
CNN	94.24 ± 0.82	0.905 ± 0.019
ANN	30.47 ± 3.46	0.112 ± 0.053
Majority-class baseline (always Kelihos_ver3)	27.07	0.047

The original thesis code, run with the same seeds and test set, gave almost the same CNN result, 94.19% ± 0.99% (Table 3). This confirms that the earlier thesis figures were affected by the evaluation, not by the model. Adding the LSTM stage raised mean accuracy to 94.73% ± 0.45% and macro F1 to 0.916 ± 0.013. The gains, 0.54 percentage points and 0.013, are smaller than the seed-to-seed standard deviation of the CNN alone, so they do not show a reliable improvement.

Table 3
Original-code replication on the same test set, mean ± standard deviation over seeds 42, 7 and 2024.

Model	Accuracy (%)	Macro F1
CNN (thesis code)	94.19 ± 0.99	0.902 ± 0.017
CNN-to-LSTM (thesis code)	94.73 ± 0.45	0.916 ± 0.013

4.2 Per-family performance

The CNN classified the two Kelihos families almost perfectly and was weakest on Simda, the smallest family, followed by Tracur, Vundo and Obfuscator.ACY (Table 4). Kelihos_ver3 reached an F1-score of 0.995 and Kelihos_ver1 0.971. Simda had the lowest F1-score, 0.769: the CNN found 6 of its 9 test images in every seed, a recall of 0.667. Tracur (0.843), Vundo (0.869) and Obfuscator.ACY (0.884) were the next weakest; for Obfuscator.ACY the weakness was recall (0.840), meaning the CNN missed about one in six of its samples.

Table 4
CNN per-family results on the test set, corrected evaluation, mean ± standard deviation over three seeds.

Family	Test images	Precision	Recall	F1
Kelihos_ver3	589	0.995 ± 0.002	0.994 ± 0.003	0.995 ± 0.002
Kelihos_ver1	80	0.983 ± 0.020	0.958 ± 0.019	0.971 ± 0.019
Lollipop	496	0.961 ± 0.017	0.974 ± 0.007	0.967 ± 0.006
Gatak	203	0.898 ± 0.078	0.967 ± 0.011	0.930 ± 0.038
Ramnit	307	0.893 ± 0.028	0.957 ± 0.034	0.923 ± 0.002
Obfuscator.ACY	246	0.935 ± 0.046	0.840 ± 0.035	0.884 ± 0.008
Vundo	95	0.888 ± 0.072	0.856 ± 0.058	0.869 ± 0.023
Tracur	151	0.893 ± 0.030	0.804 ± 0.089	0.843 ± 0.040
Simda	9	0.917 ± 0.144	0.667 ± 0.000	0.769 ± 0.054

The ANN behaved very differently. It labelled every Kelihos_ver3 test image correctly (recall 1.000) but scored an F1 of zero on Gatak, Lollipop, Ramnit, Simda and Vundo in every seed. In other words, it predicted the largest family for most inputs.

4.3 Training behaviour

In the original-code run with seed 42, the CNN's validation accuracy rose from 82.3% after the first epoch to 94.5% after the tenth. The LSTM stage reached 94.4% validation accuracy by its second epoch while its training accuracy exceeded 99%, as expected for a classifier whose inputs are already near-correct CNN predictions.

5. Discussion

5.1 Why the CNN performs best

The 64-point gap between the CNN and the ANN follows from how each design treats the image. Convolution kernels learn local patterns, such as the textures that packed sections, padding or embedded resources produce in a byte image, and pooling layers combine them into larger structures while tolerating small shifts in position. Because variants within a family keep much of their layout, these patterns transfer across variants [5], [7].

The ANN discards spatial arrangement when the image is flattened. With 67,500 inputs feeding a 512-unit layer and no weight sharing, it has tens of millions of parameters to fit from about 7,000 training

images. Within 10 epochs it settled on predicting the dominant family, which explains an accuracy close to the majority-class baseline and a macro F1 near zero.

5.2 What the LSTM stage adds

The CNN-to-LSTM pipeline did not reliably improve on the CNN. Its input is the CNN's nine-value output treated as a single time step, so the recurrent layers have no sequence to model; they act as a small second classifier on predictions that are already about 94% correct. The slightly lower variance across seeds may reflect this smoothing. Testing whether recurrence itself helps would require feeding the LSTM a genuine sequence, such as the image rows or the raw byte stream [25], [28].

5.3 Correcting the earlier evaluation

The thesis [33] reported 93.94% CNN accuracy together with precision, recall and F1-scores of 17%, a combination that is implausible when predictions and labels are correctly paired. The cause was that predictions from a shuffled data loader were compared with labels in their original order. With the pairing corrected, the CNN's class-level scores are consistent with its accuracy. The case illustrates a simple safeguard: predictions should be made in a single unshuffled pass, and accuracy computed from those predictions should match the accuracy the framework reports.

5.4 Limitations

- **Classification, not detection.** Every sample is malware. The models assign known malware to a family; they were not tested on benign software and do not establish detection performance or false-positive rates.
- **One older dataset.** All samples are Windows executables from nine families in a 2015-era collection. Performance on newer samples, other families or other platforms is untested.
- **Small families.** Simda has only 9 test images, so its per-family scores are uncertain; one image changes its recall by 0.11.
- **No robustness testing.** Packed, obfuscated or adversarially modified variants were not tested, though they are known to reduce the accuracy of learned classifiers [29], [30], including deep-learning detectors that work on image representations of executables [35].
- **Fixed, untuned training.** All models were trained for 10 epochs with default settings to keep the comparison fair. Longer or tuned training, deeper networks or pre-trained models would likely score higher; recent CNN designs report over 97% accuracy on image-format malware data [36]. The results compare designs, not best attainable accuracy.
- **Resizing.** Scaling every image to 150 × 150 pixels distorts binaries of different lengths by different amounts and may remove information from large files.

6. Conclusion and future work

This study compared a CNN, a fully connected ANN and a CNN-to-LSTM pipeline for classifying grayscale images of Windows malware into nine families, using one dataset, a held-out test set and identical training settings. The CNN reached 94.24% test accuracy and a macro F1-score of 0.905, while the ANN reached 30.47% and 0.112, barely above always predicting the largest family. Adding an LSTM stage to the CNN's outputs gave no reliable gain. The results support convolutional feature extraction as the right fit for byte-level malware images, show that a design without spatial structure fails on this task, and correct the class-level figures reported in the earlier thesis [33].

Future work will extend the evaluation in four directions: testing malicious-versus-benign detection and reporting false-positive and false-negative rates; measuring robustness to packed, obfuscated and adversarially modified samples; evaluating on newer samples to measure performance drift over time; and testing recurrent models on genuine sequences, along with deeper and pre-trained architectures, against the baseline released here.

Declarations

Author Contribution

A.R.I. conceived and designed the study, wrote the original code, ran all experiments, analysed the results, prepared the figures and tables, and wrote and reviewed the manuscript, with AI assistance in preparing the evaluation code and drafting the text as described in Section 3.5. A.R.I. is the sole author and takes full responsibility for the content.

Data Availability

The image dataset analysed in this study is publicly available on Kaggle (Malware-Classification-Dataset-Images, version 1: https://www.kaggle.com/datasets/mindtrinket/malwareclassificationdatasetimages). The code, run logs, per-family results and confusion matrices generated in this study are openly available on Zenodo (https://doi.org/10.5281/zenodo.22666836). No malware files are redistributed with this article.

References

1. Mat, S.R.T., Ab Razak, M.F., Kahar, M.N.M., Arif, J.M., Mohamad, S., Firdaus, A.: Towards a systematic description of the field using bibliometric analysis: malware evolution. Scientometrics. **126**, 2013–2055 (2021)
2. Huang, L.C., Chang, C.H., Hwang, M.S.: Research on malware detection and classification based on artificial intelligence. Int. J. Netw. Secur. **22**(5), 717–727 (2020)

3. McAfee: McAfee Labs threats report, June 2017. (2017). https://www.mcafee.com/us/resources/reports/rp-quarterly-threats-jun-2017.pdf
4. Venkatraman, S., Alazab, M., Vinayakumar, R.: A hybrid deep learning image-based analysis for effective malware detection. J. Inf. Secur. Appl. **47**, 377–389 (2019)
5. 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, pp 1–7 (2011)
6. Vinayakumar, R., Alazab, M., Soman, K.P., Poornachandran, P., Venkatraman, S.: Robust intelligent malware detection using deep learning. IEEE Access. **7**, 46717–46738 (2019)
7. Ravi, V., Alazab, M.: Attention-based convolutional neural network deep learning approach for robust malware classification. Comput. Intell. **39**(1), 145–168 (2023)
8. Alazab, M.: Profiling and classifying the behavior of malicious codes. J. Syst. Softw. **100**, 91–102 (2015)
9. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature. **521**(7553), 436–444 (2015)
10. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. Commun. ACM. **60**(6), 84–90 (2017)
11. Gibert, D., Mateu, C., Planes, J.: An end-to-end deep learning architecture for classification of malware's binary content. In: International Conference on Artificial Neural Networks. Springer, Cham, pp 383–391 (2018)
12. Lad, S.S., Adamuthe, A.C.: Malware classification with improved convolutional neural network model. Int. J. Comput. Netw. Inf. Secur. **12**(6), 30–43 (2020)
13. Souri, A., Hosseini, R.: A state-of-the-art survey of malware detection approaches using data mining techniques. Hum-centric Comput. Inf. Sci. **8**(1), 1–22 (2018)
14. Ucci, D., Aniello, L., Baldoni, R.: Survey of machine learning techniques for malware analysis. Comput. Secur. **81**, 123–147 (2019)
15. Singh, J., Singh, J.: A survey on machine learning-based malware detection in executable files. J. Syst. Archit. **112**, 101861 (2021)
16. Ali, R., Ali, A., Iqbal, F., Hussain, M., Ullah, F.: Deep learning methods for malware and intrusion detection: a systematic literature review. Secur Commun Netw 2022 (2022)
17. Senanayake, J., Kalutarage, H., Al-Kadri, M.: Android mobile malware detection using machine learning: a systematic review. Electronics. **10**(13), 1606 (2021)
18. Yuan, Z., Lu, Y., Wang, Z., Xue, Y.: Droid-Sec: deep learning in Android malware detection. In: Proceedings of the 2014 ACM Conference on SIGCOMM, pp 371–372 (2014)
19. Prima, B., Bouhorma, M.: Using transfer learning for malware classification. Int. Arch. Photogramm Remote Sens. Spat. Inf. Sci. **44**, 343–349 (2020)
20. AlGarni, M.D., AlRoobaea, R., Almotiri, J., Ullah, S.S., Hussain, S., Umar, F.: An efficient convolutional neural network with transfer learning for malware classification. Wirel. Commun. Mob. Comput.

2022, 1–8 (2022)

21. Naeem, H., Alshammari, B.M., Ullah, F.: Explainable artificial intelligence-based IoT device malware detection mechanism using image visualization and fine-tuned CNN-based transfer learning model. Comput Intell Neurosci 2022 (2022)
22. Marastoni, N., Giacobazzi, R., Dalla Preda, M.: Data augmentation and transfer learning to classify malware images in a deep learning context. J. Comput. Virol. Hacking Tech. **17**, 279–297 (2021)
23. 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. **38**(5), 1748–1771 (2022)
24. Catak, F.O., Yazı, A.F., Elezaj, O., Ahmed, J.: Deep learning based sequential model for malware analysis using Windows exe API calls. PeerJ Comput. Sci. **6**, e285 (2020)
25. Jeon, S., Moon, J.: Malware-detection method with a convolutional recurrent neural network using opcode sequences. Inf. Sci. **535**, 1–15 (2020)
26. Fallah, S., Bidgoly, A.J.: Android malware detection using network traffic based on sequential deep learning models. Softw. Pract. Exp. **52**(9), 1987–2004 (2022)
27. Alsulami, B., Mancoridis, S.: Behavioral malware classification using convolutional recurrent neural networks. In: 2018 13th International Conference on Malicious and Unwanted Software (MALWARE) (2018)
28. Lin, W.C., Yeh, Y.R.: Efficient malware classification by binary sequences with one-dimensional convolutional neural networks. Mathematics. **10**(4), 608 (2022)
29. Biggio, B., Roli, F.: Wild patterns: ten years after the rise of adversarial machine learning. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp 2154–2156 (2018)
30. Reilly, C., O'Shaughnessy, S., Thorpe, C.: Robustness of image-based malware classification models trained with generative adversarial networks. In: Proceedings of the 2023 European Interdisciplinary Cybersecurity Conference, pp 92–99 (2023)
31. Dietle, J.: Malware-Classification-Dataset-Images, version 1 [dataset]. Kaggle. (2023). https://www.kaggle.com/datasets/mindtrinket/malwareclassificationdatasetimages
32. Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., Ahmadi, M.: Microsoft Malware Classification Challenge. arXiv:1802.10135. (2018). https://doi.org/10.48550/arXiv.1802.10135
33. Imafidon, A.R.: Malware detection and classification using different neural networks. Master's thesis, Western Illinois University. ProQuest Dissertations & Theses, publication no. 31292990 (2024)
34. [Anonymized for review]: Neural-network malware family classification: code and corrected evaluation, version 2.0 [software]. Zenodo. [Link removed for anonymous review] (2026)
35. Greco, C., Ianni, M., Guzzo, A., Fortino, G.: Evasion of deep learning malware detection via adversarial selective obfuscation. In: 2025 IEEE International Conference on Cyber Security and Resilience

(CSR), pp 17–22. (2025). https://doi.org/10.1109/CSR64739.2025.11130150

36. Sercek, I., Gun, M.V., Dogan, S., Tuncer, T.: KANPoolConvNeXt: an innovative malware detection method. J. Comput. Virol. Hacking Tech. **22**, 14 (2026). https://doi.org/10.1007/s11416-025-00589-7

Figures

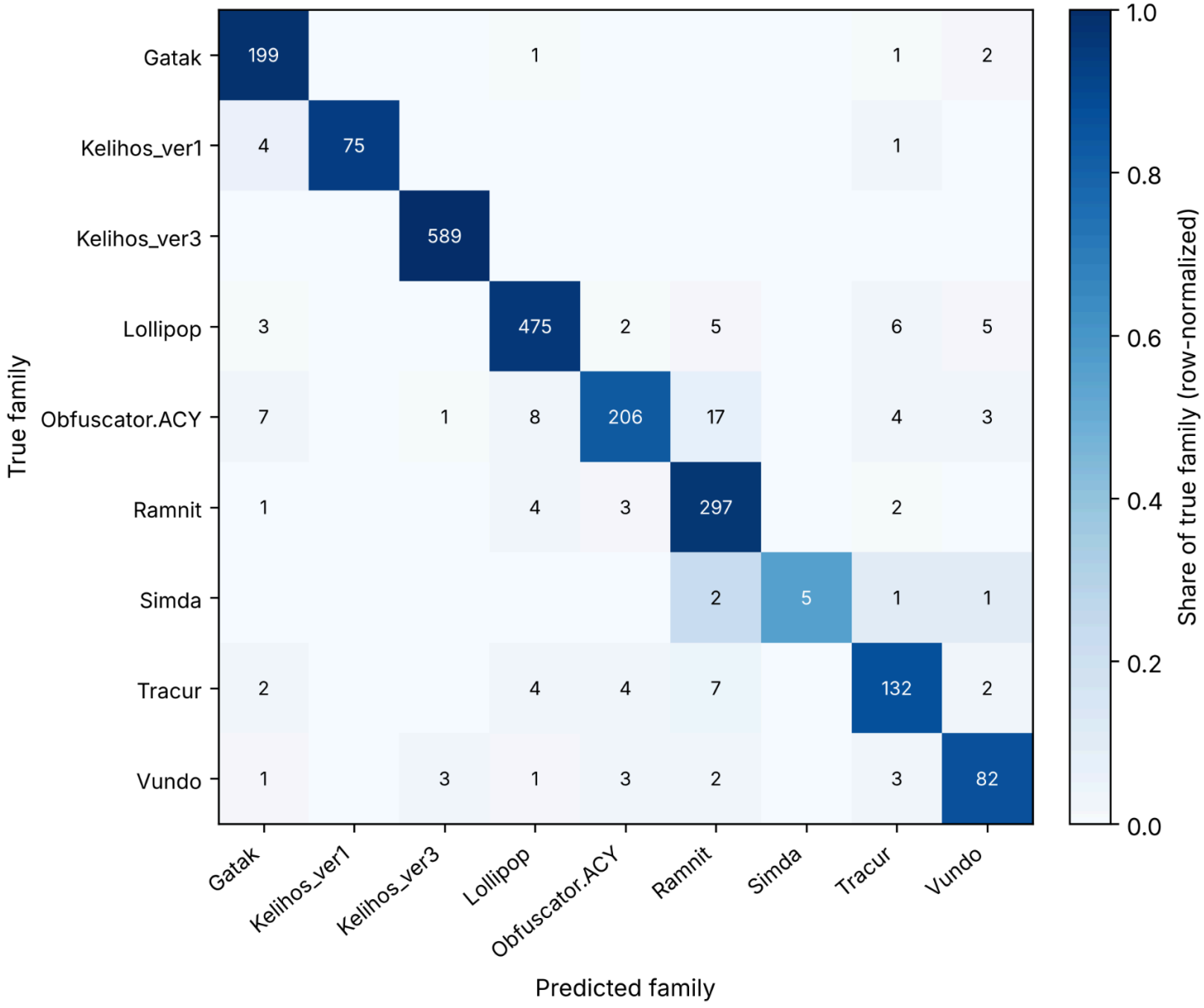

Figure 1

Confusion matrix of the CNN on the held-out test set (2,176 images), from the original-code replication with seed 42 (accuracy 94.67%; Table 3). Rows are true families and columns predicted families;

numbers are image counts, and shading shows the share of each true family. The largest single error is 17 Obfuscator.ACY images predicted as Ramnit.