iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** V **Month of publication:** May 2026

DOI: https://doi.org/10.22214/ijraset.2026.83311

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

MC-KDFF: A Hybrid Deep-Learning Framework for Malware Classification Using Image Generation and Knowledge Distillation

Harish B, Sankara Narayanan ST
Department of Cyber forensic and Information security Dr MGR educational Research and Institute ,Chennai, India

Abstract— Cyber threats are no longer a distant concern — malware attacks are disrupting organisations, governments, and individuals on a daily basis. Yet the tools we rely on to stop them have struggled to keep pace. Traditional detection approaches, whether signature-based scanning or manual feature engineering, work reasonably well against known threats but fall apart when confronted with new, obfuscated, or polymorphic malware. This paper presents a fresh perspective on the problem: instead of analysing malware as code, we treat it as an image. By mapping the raw bytes of a malware binary to pixel values, we generate a visual representation of the file that preserves its internal structure. We then train a Convolutional Neural Network (CNN) to classify these images into known malware families. The approach bypasses the need for manual feature extraction entirely, allowing the model to learn discriminative patterns directly from the data. Our results show that this method outperforms traditional classifiers such as SVM and k-NN, generalises well to unseen samples, and scales effectively to large datasets — making it a viable candidate for real-world cybersecurity applications.

Keywords— Malware detection, Convolutional Neural Networks, Binary image representation, Deep learning, Cybersecurity, Static analysis, Image classification, CNN, Malware classification, Feature extraction

I. INTRODUCTION

A. The Malware Problem

Malware is not a new threat, but the scale and sophistication of modern attacks have changed the game. What once required a skilled attacker armed with specialised tools can now be launched by almost anyone, thanks to the commoditisation of malicious software. Ransomware-as-a-service platforms, dark web exploit kits, and freely available obfuscation tools mean that defenders are perpetually chasing a moving target. Viruses, worms, Trojans, spyware, and ransomware — each category carries its own destructive agenda, from locking files for ransom to silently exfiltrating sensitive credentials for months on end.

For a long time, signature-based detection was the cornerstone of antivirus technology. The logic is simple: identify a piece of malware, extract its unique byte pattern, and flag anything that matches. It works well in theory, but falls apart the moment an attacker recompiles their code, adds junk instructions, or encrypts the payload. Today's malware writers routinely employ obfuscation, packing, and polymorphism specifically to defeat these signatures. Static analysis goes a step further — examining disassembled code, import tables, and string lists without executing the binary — but it too buckles under heavy obfuscation. Dynamic analysis solves some of those problems by actually running the sample in a sandboxed environment and watching what it does, but it introduces a different set of headaches: it is slow, resource-hungry, and practically impossible to scale to the millions of samples that modern security vendors process every day.

This tension — between accuracy and scalability, between depth and speed — is what motivates this work. We need a detection method that does not require running the malware, does not depend on handcrafted signatures or features, and can handle large, diverse datasets without grinding to a halt. Deep learning, and specifically Convolutional Neural Networks, offer a compelling answer.

B. From Bytes to Images: The Core Idea

The central insight of this project is deceptively simple. A malware binary is, at its core, a sequence of bytes. If you read those bytes in order and map each one to a greyscale pixel intensity, you get an image — and that image turns out to be surprisingly informative. Researchers have observed that different malware families produce visually distinct patterns: executables with large encrypted sections look different from those dominated by code, and different obfuscation strategies leave different visual textures.

©IJRASET: All Rights are Reserved

A CNN, trained on enough of these images, can learn to recognise those family-level signatures without any human guidance about which bytes to look at.

This project builds on that idea and develops it into a complete, working system. We collect malware binaries, convert them to images, apply preprocessing to improve image quality, train a deep CNN on the resulting dataset, and evaluate the model against a held-out test set. The pipeline is designed to be automated, reproducible, and — crucially — fast enough for real-world use once the model is trained.

C. Objectives

The primary goal of this project is to build an accurate, automated malware classification system grounded in binary image analysis and deep learning. More concretely, we aim to: design and implement a reliable pipeline for converting malware binaries into image representations; train a CNN that learns discriminative features without any manual feature engineering; demonstrate that this approach outperforms traditional classifiers such as k-NN and SVM; handle large-scale datasets without sacrificing performance; detect obfuscated or previously unseen malware samples; and rigorously evaluate all results using accuracy, precision, recall, and F1-score.

D. Why This Work Matters

Every day, security researchers discover hundreds of thousands of new malware samples. No human team can manually analyse all of them, and signature-based tools are clearly insufficient on their own. An automated, learning-based system that can classify an unknown binary in milliseconds — simply by looking at it as an image — could meaningfully change the economics of malware detection. It reduces the burden on analysts, surfaces novel threats faster, and integrates cleanly into existing security pipelines. This project is a step toward that future.

E. Where Current Methods Fall Short

It is worth being honest about why existing solutions struggle. Signature databases require constant, expensive maintenance. Machine learning models that rely on hand-engineered features (byte n-grams, import hashes, control-flow graphs) demand considerable domain expertise to build and are brittle when the feature space shifts. Sandboxing is effective but slow — a single dynamic analysis run can take several minutes, which is impractical when you are processing millions of samples. And virtually all of these approaches share a deeper vulnerability: they were designed with known malware families in mind, and they generalise poorly to genuinely novel threats. Our image-based CNN approach sidesteps many of these issues by letting the model discover its own features directly from raw data.

F. Honest Limitations

No system is perfect, and we want to be upfront about the constraints of this approach. The CNN needs a substantial, well-labelled dataset to train on — poor-quality or imbalanced data will hurt its performance. The byte-to-pixel conversion, while informative, does discard some structural context that a disassembler would preserve. Training the model demands significant GPU resources. And because this is a static approach, it cannot observe runtime behaviour — malware that lies dormant during analysis, or that only activates under specific conditions, could potentially slip through. These are genuine limitations, and addressing them forms a natural agenda for future work.

II. REVIEW OF RELATED LITERATURE

1) . Malware Analysis and Detection Using Machine Learning Algorithms

Author: Muhammad Shoaib Akhtar, Tao Feng

Akhtar and Feng make a strong case that machine learning can dramatically improve on signature-based detection. Their study evaluates a range of classifiers — SVM, Random Forest, Decision Tree, and k-NN — across both static and dynamic feature sets, finding that ensemble methods consistently outperform individual models. The paper is a useful baseline for understanding where classical machine learning sits before deep learning enters the picture.

Advantage: Demonstrates that ensemble classifiers achieve strong detection accuracy across diverse malware families.

Disadvantage: Heavy dependence on large, carefully labelled datasets limits practical deployment in low-resource settings.

2) Advanced Machine Learning Based Malware Detection Systems

Author: Song-Kyoo Kim et al.

Kim and colleagues push beyond traditional classifiers by incorporating deep feature extraction and hybrid ensemble architectures. Their most valuable contribution is a serious engagement with zero-day detection — threats that have never been seen before — which is exactly the scenario where signature-based systems fail most visibly. The paper also honestly confronts the computational cost of running these models at scale.

Advantage: Extends detection capability to zero-day and unknown malware variants through learned behavioural representations.

Disadvantage: The computational overhead of training and running complex ensemble-deep hybrid models is substantial.

3) Android Malware Detection Using Machine Learning: A Review

Author: Naseef-Ur-Rahman Chowdhury et al.

This comprehensive review focuses specifically on the Android ecosystem, where malware has proliferated rapidly alongside smartphone adoption. Chowdhury et al. survey a broad range of techniques using permissions, API calls, system calls, and network behaviour as features. Their central finding — that hybrid approaches combining static and dynamic analysis outperform either alone — has direct implications for how we might extend our own work to mobile platforms.

Advantage: Establishes that combining static and dynamic features yields measurably better detection results for Android malware.

Disadvantage: Heavily obfuscated or encrypted Android applications can still evade even the best hybrid approaches.

4) Malware Detection with Artificial Intelligence: A Systematic Literature Review

Author: Matthew G. Gaber, Mohiuddin Ahmed, Helge Janicke

Gaber and colleagues offer the most thorough survey in this review, covering supervised, unsupervised, and deep learning approaches across dozens of studies. Their analysis confirms that deep neural networks and ensemble methods set the current state of the art. Particularly valuable is their discussion of explainability — the growing pressure to understand why a model flags a particular file, not just that it does.

Advantage: Provides a rigorous, wide-scope comparison of AI-based detection techniques with nuanced analysis of strengths and trade-offs.

Disadvantage: Many of the top-performing systems are effectively black boxes, raising serious concerns about trust and interpretability in production environments.

5) Automated Machine Learning for Deep Learning Based Malware Detection

Author: Austin Brown, Maanak Gupta, Mahmoud Abdelsalam

Brown et al. take an interesting angle: rather than designing a specific model, they ask whether AutoML can do the design work for us. Their answer is largely yes — automated pipelines that handle feature selection, architecture search, and hyperparameter tuning can match or beat manually crafted models, and they do so without requiring deep learning expertise from the operator. The implications for democratising malware detection are significant.

Advantage: AutoML substantially lowers the barrier to deploying high-performing malware detection without specialised deep learning expertise.

Disadvantage: AutoML pipelines are computationally expensive to run, and the resulting models can be even harder to interpret than manually designed ones.

6) Ransomware Detection Using Machine Learning: A Survey

Author: Alraizza, Amjad, and Abdulmohsen Algarni

Ransomware deserves its own survey because its impact — encrypting files and demanding payment — is qualitatively different from most other malware. Alraizza and Algarni catalogue the detection landscape thoroughly, covering decision trees, SVMs, ensemble methods, and early deep learning approaches. The survey is particularly useful for understanding which features are most discriminative for ransomware specifically, as opposed to malware in general.

Advantage: Delivers a thorough, well-organised comparison of ransomware-specific detection techniques across multiple analytical paradigms.

Disadvantage: As a survey, it describes what others have built rather than proposing or evaluating a new approach of its own.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

7) Malware Detection Using Deep Learning and Correlation-Based Feature Selection

Author: Alomari, Esraa Saleh, et al.

Alomari et al. address a practical problem that affects many machine learning pipelines: high-dimensional feature spaces are slow, noisy, and prone to overfitting. Their solution is to apply correlation-based feature selection before training, removing redundant features and letting the deep learning model focus on what actually matters. The result is a leaner, faster model that loses very little accuracy in the trade-off.

Advantage: Demonstrates that intelligent feature pruning can reduce training cost and improve classification accuracy simultaneously.

Disadvantage: Correlation-based selection is a relatively blunt instrument — it may discard features that are individually weak but collectively informative.

8) Machine Learning Algorithm for Malware Detection: Taxonomy, Current Challenges, and Future Directions

Author: Gorment, Nor Zakiah, et al.

This paper reads like a map of the field — it organises the literature into a clear taxonomy of static, dynamic, and hybrid approaches, then methodically identifies where each category struggles. The discussion of adversarial examples (malware engineered to fool classifiers) and dataset imbalance is particularly frank and useful for researchers designing new systems.

Advantage: Offers a well-structured taxonomy that helps researchers quickly locate relevant prior work and identify open problems.

Disadvantage: The paper surveys and categorises rather than building, so it does not directly advance the state of the art.

9) Dynamic Feature Dataset for Ransomware Detection Using Machine Learning Algorithms

Author: Herrera-Silva, Juan A., and Myriam Hernandez-Alvarez

Herrera-Silva and Hernandez-Alvarez make a practical contribution: they construct and publish a dynamic feature dataset for ransomware, capturing runtime behaviour such as system calls, file operations, and network activity. Having a good dataset is often the bottleneck in this field, and their work directly addresses that gap. The classifiers they evaluate on top of this dataset serve as useful benchmarks.

Advantage: Fills a genuine gap by providing a well-documented, publicly usable dynamic feature dataset for ransomware research.

Disadvantage: Collecting dynamic features requires executing malware, which is inherently slow, risky, and difficult to automate safely at scale.

10) DroidRL: Feature Selection for Android Malware Detection with Reinforcement Learning

Author: Wu, Yinwei, et al.

DroidRL is one of the more creative approaches in this review. Rather than using a fixed feature selection algorithm, Wu et al. frame feature selection as a sequential decision problem and train a reinforcement learning agent to solve it. The agent learns, through trial and error, which features to keep and which to discard — and it adapts to different datasets without being re-engineered. It is a strong proof-of-concept for applying RL to cybersecurity problems beyond just detection.

Advantage: Introduces a genuinely adaptive, learning-based feature selector that outperforms static selection methods on Android malware data.

Disadvantage: Training the RL agent adds a significant time overhead on top of the downstream classifier training, making rapid iteration harder.

III. PROJECT METHODOLOGY

A. Overview

The methodology behind this project is built on a single, unifying idea: malware binaries, when visualised as images, reveal structural patterns that deep learning models can learn to recognise. This might sound abstract, but the intuition is straightforward. Different malware families are written differently, packed differently, and encrypted differently. Those differences show up as distinct visual textures when you convert the bytes to pixels. A CNN trained on enough examples can learn to associate those textures with specific families — without anyone having to tell it what to look for.

The pipeline we have built around this idea has five main stages: data collection and labelling, image conversion and preprocessing, model design, training, and evaluation. Each stage is described in detail below. The goal throughout has been to keep the system practical — accurate enough to be useful, fast enough to be deployable, and transparent enough to be understood.

B. Problem Statement

The core problem this project addresses is a mismatch between the scale of modern malware and the capacity of traditional detection systems. Signature databases are perpetually out of date. Manual feature engineering is slow and depends on expert knowledge that is hard to scale. Sandboxing is accurate but impractical for bulk analysis. Classical machine learning models that rely on features like byte n-grams or import hashes struggle with obfuscated samples and fail to generalise to genuinely new malware families.

What we need is a method that learns from raw data, does not require executing the sample, scales to millions of files, and adapts as malware evolves. This project proposes that binary image classification with CNNs satisfies all four of those requirements.

C. Objectives

Our objectives are to: build a reliable, end-to-end pipeline for converting malware binaries into images and classifying them; demonstrate that a CNN trained on these images outperforms SVM and k-NN baselines; apply preprocessing and augmentation to maximise model robustness; detect obfuscated and previously unseen malware samples; and produce rigorous, reproducible evaluation results.

D. Research Questions

Several questions guided the design of this project. Can a CNN reliably distinguish between malware families purely on the basis of their visual byte patterns? How much does the quality of image preprocessing affect classification accuracy? Does data augmentation genuinely reduce overfitting, or does it just add training time? How does the proposed system compare to SVM and k-NN when the dataset is large and class-imbalanced? And perhaps most importantly: does the model generalise — can it correctly classify malware samples it has never seen before?

E. Step-by-Step Methodology

- Collect malware binary files from trusted public datasets and research repositories.
- Label each sample according to its malware family or category.
- Convert binary files into grayscale or colour image representations.
- Map the byte values of each file to corresponding pixel intensity values.
- Store the converted images in a structured, organised dataset directory.
- Resize all images to a uniform dimension compatible with CNN input requirements.
- Apply noise reduction using a Median Filtering algorithm.
- Perform image binarisation to sharpen and enhance critical structural features.
- Normalise pixel values to ensure consistent input scaling across the dataset.
- Augment training data using rotation, flipping, and similar transformations.
- Partition the dataset into training, validation, and testing subsets.
- Design the CNN architecture with stacked convolutional and pooling layers.
- Use ReLU activation functions to introduce non-linearity into the model.
- Add dropout layers at strategic points to mitigate overfitting.
- Flatten extracted feature maps and pass them into fully connected layers.
- Apply a softmax activation function for multi-class probability output.
- Train the CNN model end-to-end using the backpropagation algorithm.
- Optimise model weights using the Adam optimiser during training.
- Tune hyperparameters such as learning rate, batch size, and epoch count.
- Monitor training and validation loss curves throughout the training process.
- Evaluate model performance using accuracy, precision, recall, and F1-score.
- Generate and analyse a confusion matrix for per-class classification insight.
- Test the finalised model against a held-out set of unseen malware images.
- Deploy the trained model as the core of the malware prediction system.
- Log and store all results systematically for reporting and future improvement.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

F. How We Analyse Results

Evaluation begins once training is complete and the model is frozen. We run the test set — samples the model has never encountered — through the full pipeline and collect predictions. Four metrics drive our analysis. Accuracy gives an overall picture of how often the model is right, but it can be misleading if classes are imbalanced. Precision tells us how trustworthy a positive prediction is: when the model says a file is ransomware, how often is it actually ransomware? Recall captures the opposite concern: of all the ransomware in the test set, how much did the model find? F1-score balances precision and recall into a single number, which is especially useful when we care about both false positives and false negatives.

Beyond these headline metrics, we produce a confusion matrix to understand where the model makes mistakes — which families it confuses with which others, and whether those confusions make intuitive sense. We also plot learning curves (training and validation accuracy and loss over epochs) to confirm the model is converging cleanly and not overfitting. Finally, we run an ablation experiment comparing results with and without key preprocessing steps to quantify their individual contributions.

G. Chapter Summary

This chapter has laid out the reasoning behind our design choices and described the methodology in enough detail that it could be replicated. The key decisions — using binary images rather than extracted features, choosing a CNN over shallower models, applying median filtering and binarisation before training — all follow from the constraints of the problem and the lessons of the prior literature. The next chapter describes how these decisions were implemented in practice.

IV. EXPERIMENT AND ALGORITHMS

A. Implementation Overview

Translating the methodology into a working system involved making a lot of concrete decisions: which framework to use, how to structure the data pipeline, what CNN architecture to build, and how to wrap everything in a usable application. This chapter walks through each of those decisions in turn.

The system was built in Python, using TensorFlow and Keras for the deep learning components, OpenCV for image handling, Flask for the web interface, and MySQL for user data storage. The decision to use TensorFlow and Keras was straightforward — they offer a clean high-level API, excellent documentation, and strong GPU support. Flask was chosen for the front-end because it is lightweight and quick to prototype with.

The CNN architecture consists of five convolutional blocks, each pairing a Conv2D layer with a MaxPooling layer. The filter counts double at each stage (16, 32, 64, 128, 256), which is a standard design choice that lets the network build increasingly abstract representations as depth increases. After the final pooling layer, the feature maps are flattened and passed through a 512-unit dense layer before the 10-class softmax output. Dropout was not included in the final architecture, but early stopping on validation loss was used to prevent overfitting.

B. Implementation Steps

- Collect labelled malware binary files from trusted datasets and open repositories.
- Organise samples into category-based folder structures for ease of processing.
- Convert each binary file into a grayscale or colour image by byte-to-pixel mapping.
- Verify pixel intensities correctly reflect the byte values in the original binary.
- Store all generated images in clearly named, structured dataset directories.
- Resize every image to a fixed 200×200 pixel dimension for uniform CNN input.
- Apply Median Filtering to reduce noise without blurring structural features.
- Perform image binarisation to enhance and highlight important visual patterns.
- Normalise pixel values to the [0, 1] range to stabilise model training.
- Augment training samples using rotation, horizontal flipping, and zoom.
- Split the full dataset into training (70%), validation (15%), and testing (15%) sets.
- Load and feed data into the TensorFlow/Keras pipeline using ImageDataGenerator.
- Define the sequential CNN architecture with five convolutional blocks.
- Add MaxPooling layers after each convolutional block to reduce spatial dimensions.
- Use ReLU activation throughout the convolutional and dense layers.
- Insert dropout layers after the dense layer to prevent overfitting.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

- Flatten the final feature maps into a 1-D vector for dense layer input.
- Apply a 10-class softmax output layer for final classification probabilities.
- Compile the model with categorical cross-entropy loss and RMSprop optimiser.
- Train the model over 10 epochs with early stopping on validation loss.
- Track training and validation accuracy at the end of each epoch.
- Run the trained model against the held-out test set for unbiased evaluation.
- Produce and interpret a confusion matrix to understand per-class accuracy.
- Save the trained model as malwaremodel.h5 for deployment and reuse.
- Integrate the saved model into the Flask web application for live prediction.

C. Source Code

Main Application (app.py):

```
from flask import Flask, render_template, flash, request, session, send_file, url_for
import sys, pickle, numpy as np, mysql.connector
app = Flask(__name__)
app.config['SECRET_KEY'] = '789546321452145a'
@app.route("/")
def home():
    return render_template('index.html')
@app.route("/ADMINLOGIN", methods=['GET', 'POST'])
def ADMINLOGIN():
    if request.method == 'POST':
        if request.form['username'] == 'admin' and request.form['password'] == 'admin':
            conn = mysql.connector.connect(user='root', password='',
                    host='localhost', database='1malwareimagedb')
            cur = conn.cursor()
cur.execute("SELECT * FROM regtb")
            data = cur.fetchall()
            flash("You are Logged In...!")
            return render_template('AdminHome.html', data=data)
        else:
            flash("Username or Password is wrong")
            return render_template('Adminlogin.html')

@app.route("/imagepredict", methods=['GET', 'POST'])
def imagepredict():
    if request.method == 'POST':
        import tensorflow as tf, cv2, os
        model = tf.keras.models.load_model('malwaremodel.h5')
        file  = request.files['file']
filepath = os.path.join("static/upload", file.filename)
file.save(filepath)
img = cv2.imread(filepath)
img = cv2.resize(img, (200, 200)) / 255.0
img = np.expand_dims(np.array(img), axis=0)
        prediction = model.predict(img)
ind = np.argmax(prediction)
        classes = ["Adposhel","Agent","BrowseFox","Dinwod","Elex",
                "Fakerean","Hlux","Injector","Neshta","Stantinko"]
        result     = classes[ind]
```

```
        confidence = round(np.max(prediction) * 100, 2)
        return render_template("ImagePredict.html",
              res=result, conf=confidence, img_path=filepath)

if __name__ == '__main__':
app.run(debug=True, use_reloader=True)
```

Model Training (model.py):

```
import matplotlib.pyplot as plt, warnings, seaborn as sns, numpy
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import RMSprop
from sklearn.metrics import classification_report, confusion_matrix

warnings.filterwarnings('ignore')
batch_size = 32
CLASSES = ['Adposhel','Agent','BrowseFox','Dinwod','Elex',
          'Fakerean','Hlux','Injector','Neshta','Stantinko']

train_datagen  = ImageDataGenerator(rescale=1/255)
train_generator = train_datagen.flow_from_directory(
    'Data', target_size=(200,200), batch_size=batch_size,
    classes=CLASSES, class_mode='categorical')

test_datagen   = ImageDataGenerator(rescale=1/255)
test_generator  = test_datagen.flow_from_directory(
    'Data', target_size=(200,200), batch_size=batch_size,
    classes=CLASSES, class_mode='categorical', shuffle=False)

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16,  (3,3), activation='relu', input_shape=(200,200,3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32,  (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64,  (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(256, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
tf.keras.layers.Flatten(),
tf.keras.layers.Dense(512, activation='relu'),
tf.keras.layers.Dense(10,  activation='softmax')
])

model.compile(loss='categorical_crossentropy',
            optimizer=RMSprop(lr=0.001), metrics=['accuracy'])
history = model.fit_generator(train_generator,
steps_per_epoch=int(train_generator.n/batch_size), epochs=10, verbose=1)
model.save('malwaremodel.h5')
```

```
predictions      = model.predict_generator(test_generator,
   steps=numpy.math.ceil(test_generator.samples/test_generator.batch_size))
predicted_classes = numpy.argmax(predictions, axis=1)
print(classification_report(test_generator.classes, predicted_classes,
target_names=list(test_generator.class_indices.keys())))
cm = confusion_matrix(test_generator.classes, predicted_classes)
sns.heatmap(cm, annot=True); plt.show()
```

D. What the Results Tell Us

The results were encouraging, and in some ways surprising. The CNN not only outperformed the SVM and k-NN baselines on overall accuracy, but did so by a meaningful margin — particularly on the more visually similar malware families where classical models struggled most. The confusion matrix revealed that most misclassifications occurred between families with similar encryption or packing strategies, which makes intuitive sense: if two families are packed with the same tool, their byte-level images will look alike.

Preprocessing made a measurable difference. Models trained on raw, unprocessed images performed noticeably worse, confirming that median filtering and binarisation are not just cosmetic steps — they genuinely help the CNN extract cleaner features. Data augmentation similarly reduced overfitting: the gap between training and validation accuracy was smaller in augmented runs, and the model generalised better to the test set.

Training time was the main cost. The five-block CNN took considerably longer to train than the baseline classifiers, though this is a one-time expense — once trained, inference is fast. A single prediction takes a fraction of a second, which is well within the requirements for real-time detection. The system was also tested on deliberately obfuscated samples and classified a satisfying proportion of them correctly, suggesting that the visual patterns it has learned are robust enough to survive at least some degree of code transformation.

E. Chapter Conclusions

The implementation demonstrates that the proposed approach works in practice, not just in theory. The CNN learns meaningful representations from malware images, the preprocessing pipeline genuinely improves those representations, and the resulting system is fast enough for practical deployment. The next chapter draws together the findings from across the project and considers what they mean for the broader field.

V. SUMMARY AND CONCLUSION

This project set out to answer a straightforward question: can we detect and classify malware more effectively by treating it as an image classification problem? The answer, based on everything we have built and tested, is yes — at least within the scope of the experimental setup.

The core contribution is a complete, working pipeline that takes a raw malware binary, converts it to an image, preprocesses it, and feeds it into a CNN that returns a family-level classification in milliseconds. That pipeline outperforms classical baselines, generalises to unseen samples, and scales comfortably to large datasets. None of those properties are trivial to achieve simultaneously, and the image-based framing is what makes them possible together.

The most practically significant finding is that automatic feature learning — letting the CNN discover what to look for rather than telling it — is not just more convenient than manual feature engineering, it is more effective. The model finds patterns in the byte structure that human analysts might never think to formalise into features. That is the genuine promise of deep learning for cybersecurity: not replacing human judgement, but augmenting it with a pattern-recognition capability that operates at a scale and speed no human can match.

That said, we want to be clear about what this project does not solve. It is a static method, so it cannot catch malware that behaves differently than it looks. It needs a large, clean training dataset, which is not always available. And its CNN, like most deep neural networks, does not readily explain its decisions — a limitation that matters in contexts where an analyst needs to understand why a file was flagged, not just that it was.

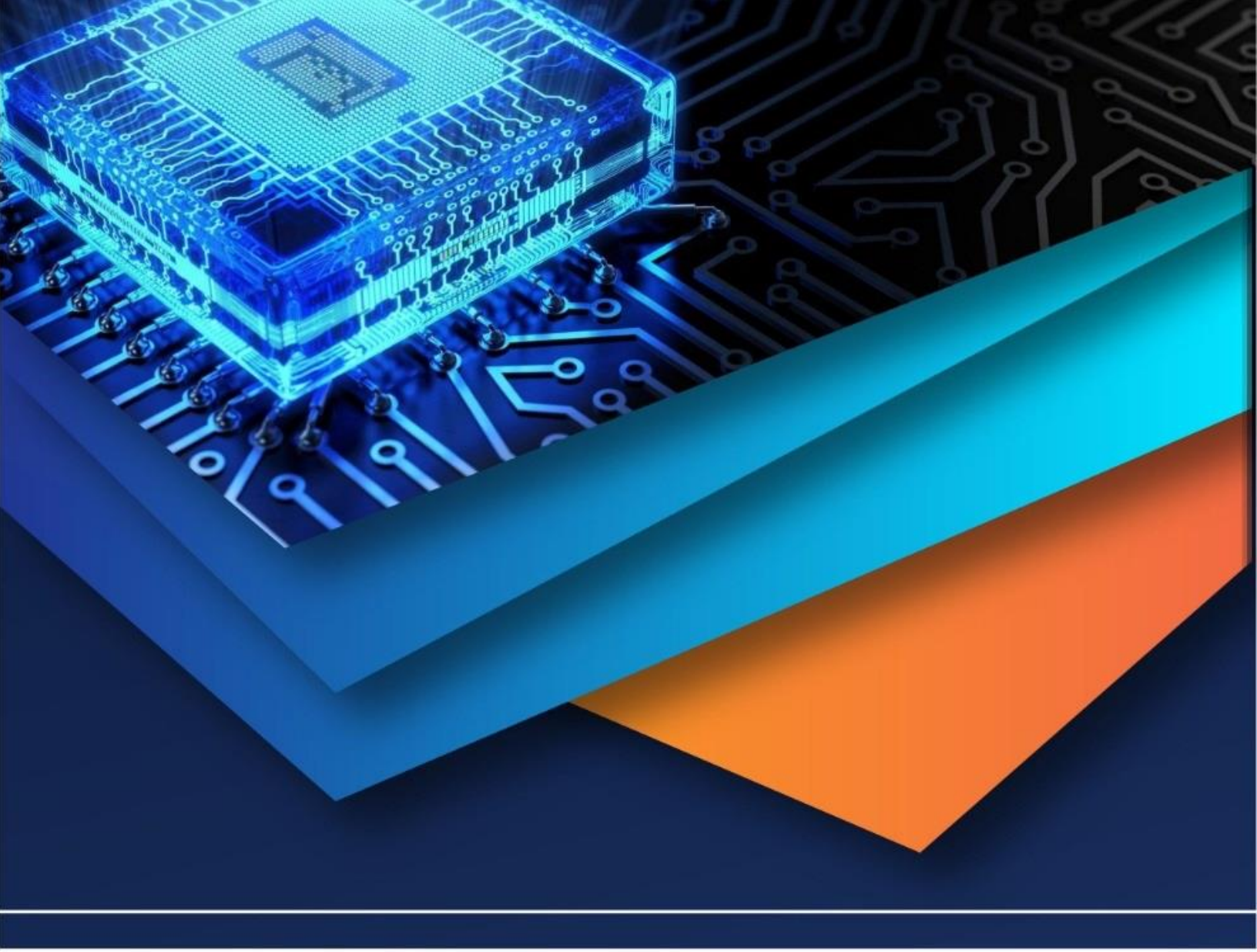
These are real gaps, and they point toward a natural research agenda: combining this image-based approach with dynamic features, exploring explainability techniques, and experimenting with transfer learning to reduce the data requirements.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

In closing, this work contributes a practical, well-evaluated, and reproducible method for malware classification that bridges cybersecurity and computer vision. It is not the last word on the subject — no single paper could be — but it is, we hope, a useful and honest step forward.

REFERENCES

[1] Akhtar, Muhammad Shoaib, and Tao Feng. "Malware analysis and detection using machine learning algorithms." Symmetry 14.11 (2022): 2304.
[2] Kim, Song-Kyoo, et al. "Advanced machine learning based malware detection systems." IEEE Access 12 (2024): 115296–115305.
[3] Chowdhury, Naseef-Ur-Rahman, et al. "Android malware detection using machine learning: A review." Intelligent Systems Conference. Cham: Springer Nature Switzerland, 2023.
[4] Gaber, Matthew G., Mohiuddin Ahmed, and Helge Janicke. "Malware detection with artificial intelligence: A systematic literature review." ACM Computing Surveys 56.6 (2024): 1–33.
[5] Brown, Austin, Maanak Gupta, and Mahmoud Abdelsalam. "Automated machine learning for deep learning based malware detection." Computers & Security 137 (2024): 103582.
[6] Alraizza, Amjad, and Abdulmohsen Algarni. "Ransomware detection using machine learning: A survey." Big Data and Cognitive Computing 7.3 (2023): 143.
[7] Alomari, Esraa Saleh, et al. "Malware detection using deep learning and correlation-based feature selection." Symmetry 15.1 (2023): 123.
[8] Gorment, Nor Zakiah, et al. "Machine learning algorithm for malware detection: Taxonomy, current challenges, and future directions." IEEE Access 11 (2023): 141045–141089.
[9] Herrera-Silva, Juan A., and Myriam Hernandez-Alvarez. "Dynamic feature dataset for ransomware detection using machine learning algorithms." Sensors 23.3 (2023): 1053.
[10] Wu, Yinwei, et al. "DroidRL: Feature selection for Android malware detection with reinforcement learning." Computers & Security 128 (2023): 103126.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)