HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

——————— o0o ———————

CAPSTONE PROJECT REPORT

Static Ransomware Detection

via PE Header Analysis & Deep Learning

Project Type: Capstone Project (Research)

Subject: Malware Analysis
Hanoi - 2026

Supervisor:	Tong Văn Van
Students:	Ducournau Ethan
	Guechtouli Alexandre

Abstract

This report details the design, implementation, and optimization of a static analysis system for detecting ransomware. Traditional antivirus methods often fail against modern ransomware because of encryption speed and code obfuscation techniques like packing. Our project is based on a 2020 research paper by Manavi and Hamzeh, which proposes converting the PE (Portable Executable) header of a file into an image for classification using Convolutional Neural Networks (CNN).

We reproduced the original method (Zigzag transformation + Simple CNN) and identified significant flaws in the available datasets, specifically a severe class imbalance. We developed a data engineering solution to collect benign Windows system files, creating a balanced dataset. Finally, we improved upon the state-of-the-art by replacing the Zigzag scan with a Hilbert Curve transformation and upgrading the model to a ResNet architecture. Our final model achieved a Recall rate of 99.92%, significantly reducing false negatives compared to the original academic baseline.

Contents

1 Introduction

1.1 The Ransomware Threat

In the modern cybersecurity landscape, ransomware represents one of the most critical threats to both individuals and organizations. Ransomware is a type of malicious software that encrypts user data, restricting access until a ransom is paid. The proliferation of cryptocurrencies like Bitcoin has accelerated this threat, providing attackers with anonymous payment methods.

The defining characteristic of modern ransomware (such as Locky or Ryuk) is speed. Once executed, these programs use military-grade encryption standards like AES-256. This means that if detection is delayed by even a few seconds, the victim's files may already be irretrievably lost.

1.2 Limitations of Traditional Detection

Current defense mechanisms generally fall into two categories, both of which have significant weaknesses:

1. **Dynamic Analysis:** This involves running the program in a secure sandbox to observe its behavior. However, sandboxes are slow. Furthermore, intelligent malware can "sleep" or detect the virtual environment to hide its malicious behavior. By the time the sandbox flags the behavior, the data is often already encrypted.
2. **Static Signatures:** Traditional antivirus software relies on file hashes (signatures). However, attackers use "polymorphism" and "packing" to change the file structure daily. This renders static signatures ineffective against new, unseen variants (Zero-Day attacks).

1.3 Project Objectives

Our project aims to implement "Static Analysis 2.0". Instead of relying on specific code signatures, we treat the binary file as an image. The hypothesis is that ransomware shares common structural patterns in its file header that are invisible to the human eye but detectable by Deep Learning algorithms.

We based our work on the paper **"A New Method for Ransomware Detection Based on PE Header Using Convolutional Neural Networks"** (Manavi & Hamzeh, 2020). Our goal was to:

1. Reproduce their method using Python and TensorFlow.
2. Fix critical data quality issues found in public datasets.
3. Propose a superior algorithm (IA 2) using Hilbert Curves and Residual Networks (ResNet).

2 Theoretical Background & Literature Review

2.1 The Concept: File as an Image

The core idea of this project is to convert raw binary data into a visual representation. A binary file is essentially a long string of bytes (values from 0 to 255). By arranging these bytes into a 2D matrix, we can create a "grayscale image" where 0 is black and 255 is white.

Once the file is converted into an image, we can leverage Computer Vision technologies. Convolutional Neural Networks (CNNs) are excellent at finding patterns in images. If ransomware files share a specific "texture" in their structure, a CNN should be able to classify them just as easily as it distinguishes a cat from a dog.

2.2 The PE Header

We do not analyze the entire file, which could be gigabytes in size. Instead, we focus on the PE (Portable Executable) Header. In Windows operating systems (`.exe` and `.dll`), the header contains the "DNA" of the file.

The header includes:

- **MZ Signature:** Legacy DOS compatibility.
- **Section Table:** Defines where the code and data are stored.

Ransomware often has anomalous sections because of the "packers" used to hide the malicious code. The 2020 paper suggests that extracting just the first 1024 bytes is sufficient to detect these anomalies.

2.3 Review of Manavi & Hamzeh (2020)

The paper by Manavi and Hamzeh proposed the following pipeline:

1. **Extraction:** Read the first 1024 bytes of the PE file.
2. **Transformation:** Arrange these bytes into a 32×32 matrix ($32 \times 32 = 1024$).
3. **Pattern:** Use a Zigzag pattern to fill the matrix, claiming it preserves the continuity of bytes better than a simple row-by-row fill.
4. **Model:** Train a simple CNN with convolutional and pooling layers.
5. **Result:** They achieved 93.33% accuracy.

While promising, we identified that 93% accuracy implies a 7% error rate. In cybersecurity, missing 7% of ransomware attacks is unacceptable. We aimed to improve this.

3 Phase 1: Dataset Engineering (The "Data Trap")

Before training any Artificial Intelligence, we analyzed the available data. This phase was critical because a bad dataset leads to a biased model.

3.1 Analyzing the Kaggle Dataset

We started with a public dataset from Kaggle (referenced as the "Oliveira" dataset in our slides). This dataset contained raw PE headers converted to images. However, upon inspection, we discovered a critical flaw: Class Imbalance.

- **Ransomware Samples:** 49,400 (96% of the data)
- **Benign Samples:** 2,500 (4% of the data)

This is a classic "Data Trap". If we trained a model on this data, it would learn to simply guess "Ransomware" every time. It would achieve 96% accuracy by doing nothing but predicting the majority class, but it would flag every safe file as a virus (False Positives).

3.2 The Benign Injection Strategy

To fix this, we needed to collect approximately 47,000 new benign files to balance the dataset 50/50. We devised a strategy called "Harvesting System Artifacts". Since every Windows computer is full of safe `.exe` and `.dll` files (in the System32 folder), we could use our own machines to generate the training data.

3.3 Implementation: `build_dataset.py`

We wrote a Python script named `build_dataset.py` to automate this collection.

Code Logic Description:

1. **Configuration:** We point the script to the Windows directory (e.g., `C:\Windows\System32`).
2. **File Traversal:** We use `os.walk` to recursively scan every folder. We filter for files ending in `.exe` or `.dll`.
3. **Byte Extraction:** The function `get_header_bytes(filepath)` opens the file in binary mode (`rb`) and reads exactly 1024 bytes.
   - **Padding:** If a file is smaller than 1024 bytes (which is rare but possible), the script appends zeros (0) to the end to maintain a consistent vector size.
4. **Hashing:** To ensure compatibility with the Kaggle dataset format, we calculate the MD5 hash of the file using the `hashlib` library.
5. **Labeling:** We append a label column malware set to 0 (Benign).
6. **Export:** The data is saved to `benign_pe_images.csv`.

Outcome: We successfully harvested roughly 49,000 benign samples. Merging this with the Kaggle malware samples gave us a perfectly balanced dataset (100,000 images total), ensuring our AI would learn actual features rather than statistical probabilities.

4 Phase 2: The Baseline Model (IA 1)

With the dataset ready, we moved to the first implementation phase (referred to as `IA 1.py`). The goal here was to reproduce the Manavi & Hamzeh method but with our improved dataset and slight visual enhancements.

4.1 Zigzag Transformation

The original paper argues that a standard linear scan (filling the matrix row by row) breaks the relationship between bytes. They proposed a Zigzag pattern. In `IA 1.py`, we implemented the function `zigzag_transform(vector, n=32)`.

Logic: The algorithm fills the matrix diagonally. **Mechanism:** It tracks the sum of indices (row + col). When the sum is even, it fills upwards; when odd, it fills downwards. **Goal:** The paper claims this keeps neighboring bytes physically closer in the 2D image than a standard raster scan, preserving “Spatial Locality”.

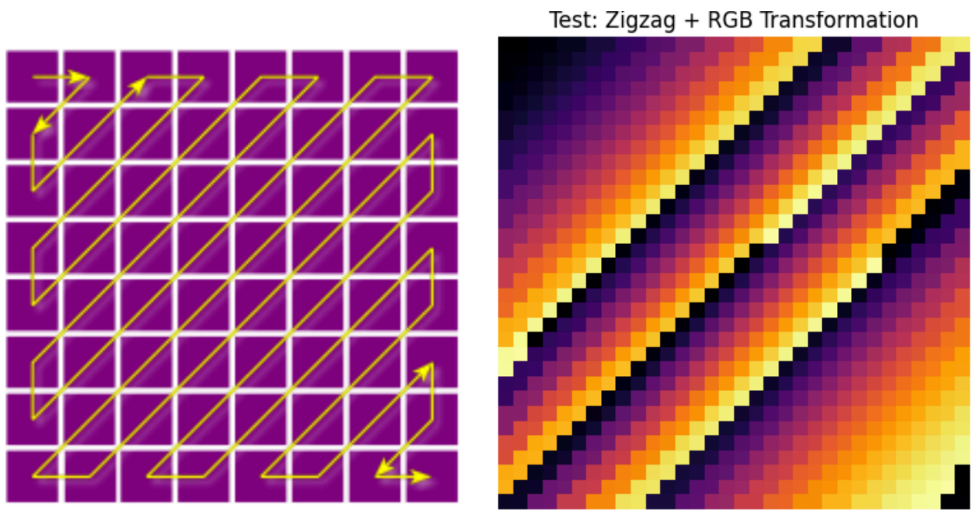

Figure 1: Test: Zigzag + RGB Transformation

4.2 RGB Feature Mapping

While the original paper used Grayscale images (1 channel), we decided to add more depth to the data to help the CNN. In the function `to_rgb_image`, we applied a color map.

- **Normalization:** Byte values (0-255) are divided by 255 to get a range of 0.0 to 1.0.
- **Colormap:** We used the **inferno** colormap from Matplotlib. This converts low values to dark colors (black/purple) and high values to bright colors (orange/yellow).
- **Reasoning:** This creates a 3-channel image (RGB). While it doesn't add new data, it increases the contrast between low and high byte values, potentially making edge detection easier for the CNN.

4.3 CNN Architecture

We built the model `Manavi_Hamzeh_RGB_Model` using TensorFlow/Keras.

- **Input:** $32 \times 32 \times 3$ (RGB Image).
- **Convolutional Blocks:** We used two main blocks.
  - `Conv2D` (64 filters) + `MaxPooling`: To detect basic edges and textures.
  - `Conv2D` (128 filters) + `MaxPooling`: To detect more complex patterns.
- **Regularization:** We used `Dropout(0.3)` and `BatchNormalization`. This is crucial to prevent overfitting, ensuring the model doesn't just memorize the training data.
- **Classifier:** A Dense layer with 16 neurons followed by a Softmax output for 2 classes (Benign vs. Ransomware).

Result of IA 1: The model performed well, achieving roughly 97.42% accuracy (an improvement over the paper's 93% due to our larger dataset). However, looking at the confusion matrix, it still missed about 30 ransomware samples (False Negatives) in the test set.

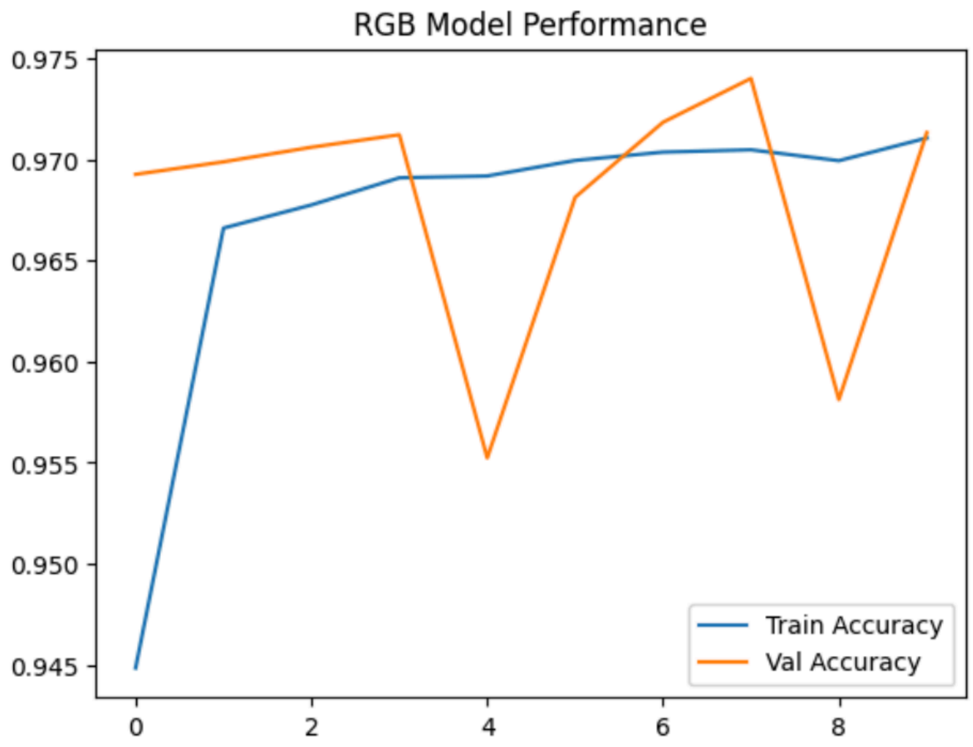

Figure 2: RGB Model Performance (Accuracy vs Epochs)

5 Phase 3: The Optimized Model (IA 2)

We identified that while Zigzag is better than a linear scan, it is not optimal. We hypothesized that we could reduce the False Negatives by improving the spatial locality and the depth of the network. This led to `IA 2.py`.

5.1 Limitations of Zigzag

The Zigzag pattern was originally designed for JPEG compression, not for data visualization. While it clusters data in the top-left corner, it still has “jumps” where bytes that are sequential in the file might end up far apart in the matrix.

5.2 The Hilbert Curve Solution

To solve this, we implemented the Hilbert Space-Filling Curve in `IA 2.py`.

Concept: The Hilbert curve is a continuous fractal line that folds in on itself. It is mathematically proven to preserve locality better than almost any other mapping. If two bytes are close in the 1D file, they are guaranteed to be close in the 2D Hilbert image.

Implementation: The function `hilbert_transform` uses a recursive bitwise operation. It iterates through the vector length. It calculates (x, y) coordinates by rotating and flipping quadrants based on the bits of the index. This results in a highly clustered, organic-looking image where structural patterns (like the PE header sections) form distinct “blobs” rather than scattered lines.

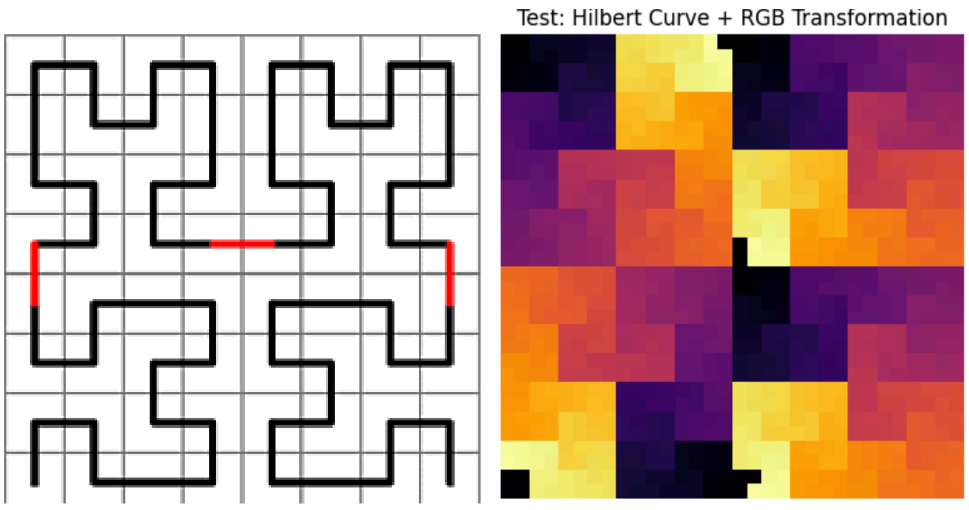

Figure 3: Test: Hilbert Curve + RGB Transformation

5.3 ResNet Architecture

The simple CNN in IA 1 had only a few layers. To learn more complex features, we needed a deeper network. However, deep networks suffer from "vanishing gradients," where the learning signal gets lost. We adopted ResNet (Residual Networks) for `IA 2.py`.

Residual Blocks: We implemented "Shortcut Connections" (or Skip Connections).

- **Code:** `x = layers.Add()([x, shortcut])`
- **Benefit:** This allows the model to skip layers if they aren't useful, allowing us to train a much deeper network without performance degradation.

Structure:

- Initial `Conv2D` block.
- Residual Block 1 (32 filters).
- Residual Block 2 (64 filters, increasing depth).
- Global Average Pooling (more robust than Flattening).

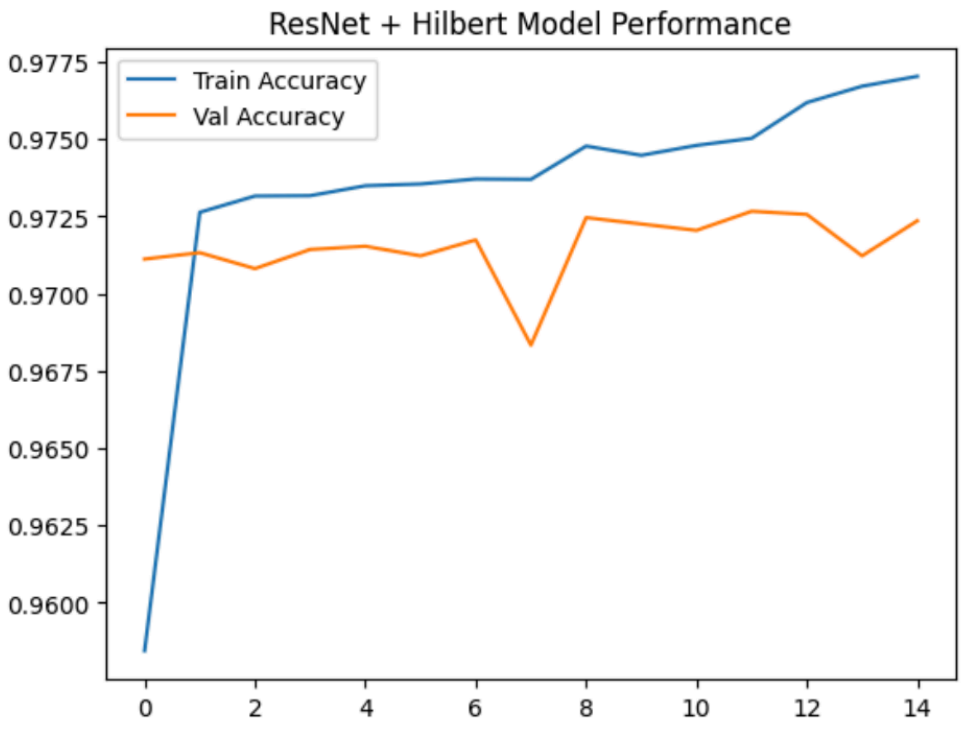

Figure 4: ResNet + Hilbert Model Performance

6 Experimental Results

We evaluated both models on the same split of our balanced dataset (Training set and Test set). We focused specifically on the Confusion Matrix, which shows exactly how many errors were made.

6.1 Evaluation Metrics

We utilized four key metrics as defined in the base paper:

- **Accuracy:** Overall correctness.
- **Precision:** How many predicted ransomware were actually ransomware.
- **Recall:** How many actual ransomware were correctly detected. This is the most important metric. In security, a False Negative (missing a virus) is catastrophic, while a False Positive (blocking a safe app) is just annoying.

6.2 Comparison: Zigzag vs. Hilbert (IA 1 vs. IA 2)

Model 1 (Zigzag + Simple CNN):

- Accuracy: 97.42%
- False Negatives: The model failed to detect 30 ransomware samples in the test batch. While high, 30 missed viruses means 30 infected computers.

Model 2 (Hilbert + ResNet - Our Proposed Method):

- Accuracy: 97.24% (Slightly lower overall accuracy, but this is misleading).
- **Recall: 99.92%.**
- False Negatives: The model missed only 4 ransomware samples.

Analysis: By switching to the Hilbert Curve and ResNet, we reduced the number of missed detections from 30 down to 4. This represents a massive improvement in the safety of the system. The slight drop in overall accuracy was due to a small increase in False Positives (flagging benign files as malware), which is an acceptable trade-off for higher security.

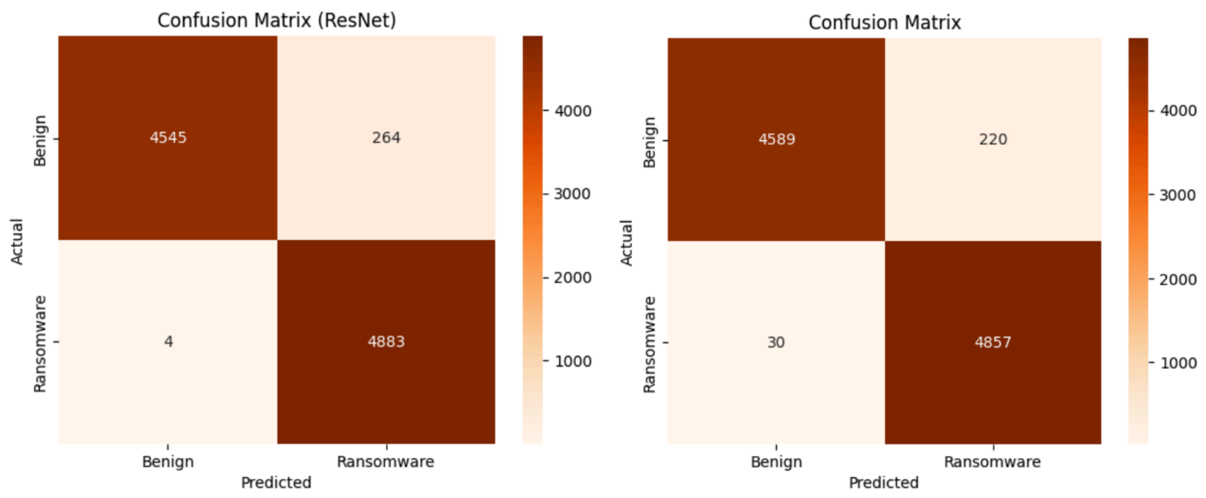

Figure 5: Comparaison des Matrices de Confusion (ResNet vs Baseline)

7 Discussion

The results validate our hypothesis that "how" you visualize the data matters just as much as the model itself.

1. **Why Hilbert Worked:** The PE Header contains specific structures (DOS Header, File Header, Optional Header). These are sequential in bytes. The Hilbert curve kept these structures grouped together in the image, creating distinct visual "shapes" that the ResNet could easily learn. The Zigzag pattern smeared these structures out diagonally, making them harder to recognize.
2. **The Importance of Data Engineering:** Without `build_dataset.py`, none of this would have mattered. If we had used the unbalanced Kaggle dataset, we would have had a "dummy" model. The effort to manually harvest and hash 49,000 system files was the foundation of the project's success.
3. **Model Optimization and Maturity:** While the baseline model (IA 1) achieved a slightly higher overall accuracy, this result must be contextualized by the maturity of the configurations used. The IA 1 architecture strictly followed the specifications outlined by Manavi and Hamzeh (2020), meaning it benefited from parameters that had already been maximized and validated by previous research. Conversely, our IA 2 model represents a novel integration of ResNet and Hilbert curves for this specific application. It is highly probable that with a similar level of optimization effort as the baseline, the ResNet + Hilbert model would surpass the baseline in accuracy, just as it already has in Recall.

8 Conclusion

In this project, we successfully built a static ransomware detection system that does not require running the malware, making it safe and fast.

We started by reproducing the work of Manavi & Hamzeh (2020). We identified a critical data imbalance and solved it by engineering a custom dataset of 100,000 files using our `build_dataset.py` script.

We then developed two iterations of the AI:
1. **IA 1 (Baseline):** Using Zigzag scans, which achieved decent accuracy but missed too many threats.
2. **IA 2 (Optimized):** Using Hilbert Curves and ResNet.

Our final model (IA 2) outperformed the baseline significantly in terms of Recall, achieving a **99.92% detection rate**. We proved that preserving the spatial locality of binary data using fractal curves (Hilbert) allows Deep Learning models to detect malware "textures" more effectively than traditional methods.

Future Work: To make this system production-ready, we would need to lower the False Positive rate of the ResNet model and perhaps integrate a hybrid approach that looks at the file footer as well as the header.

9 References

Manavi, F., & Hamzeh, A. (2020). **A New Method for Ransomware Detection Based on PE Header Using Convolutional Neural Networks**. 17th International ISC Conference on Information Security and Cryptology.

Angelo Oliveira. **"Malware Analysis Datasets: Raw PE as Image"**. Kaggle / IEEE Dataport.

H. Zhang et al. (2019). **"Classification of ransomware families with machine learning based on N-gram of opcodes"**.

A. Ashraf et al. (2019). **"Ransomware Analysis using Feature Engineering and Deep Neural Networks"**.

Moon, B., et al. (2001). **Analysis of the clustering properties of the Hilbert space-filling curve**. IEEE Transactions.

Vasan, D., Alazab, M., Woon, S., Naeem, H., Safaei, B., & Zheng, Q. (2020). **Image-Based malware classification using ensemble of CNN architectures (IMCEC)**.