Malware Analysis via CNN

February 13, 2026

Name: Shashwat Ahuja

Course: M.Sc. Cyber Security (Semester 1)

Date: 1 December 2025

1 Introduction

Traditional malware detection relies heavily on **signatures**—checking a file's specific hash against a database of known viruses. While effective for known threats, this method fails against **polymorphic malware** (malware that changes its code to evade detection) and **zero-day attacks**.

To solve this, researchers began treating **malware binaries as images**. Just as a CNN can classify a picture as a "cat" or "dog" based on visual features, it can classify a file as "Ransomware" or "Spyware" based on the visual patterns of its binary code. This approach leverages the powerful feature-extraction capabilities of Deep Learning to detect malicious *intent* rather than specific code strings.

---

2 Workflow: From Binary to Image

The process of converting a non-visual executable file into an image for a CNN follows a standard pipeline:

1. **Binary Read**: The malware executable (`.exe`) is read as a sequence of raw bytes (hex values from `00` to `FF`).
2. **Vectorization**: This sequence is treated as a stream of 8-bit integers (0-255).
3. **Reshaping**: The stream is reshaped into a 2D matrix. The width of the matrix is usually fixed based on the file size (e.g., small files have a width of 32 pixels, large files have 1024 pixels).
4. **Image Generation**: The 2D matrix is rendered as a grayscale image.
   - `00` = Black
   - `FF` = White
5. **Classification**: The generated image is fed into a CNN, which learns to recognize textures associated with specific malware families.

---

3 Why It Is Powerful

- **Resilience to Obfuscation**: Malware authors often "pack" or encrypt their code to hide text strings. However, these packers often leave distinct visual signatures (high entropy/noise patterns) that a CNN can easily recognize.
- **Variant Detection**: If an attacker changes a few variable names or adds junk code to change the file hash, the *overall visual structure* of the binary remains largely unchanged. A CNN will still see it as "99% similar" to the original virus.
- **Feature Autonomy**: Unlike traditional Machine Learning, where analysts must manually define features (e.g., "count the number of API calls"), a CNN automatically learns which features are important.

---

4 Limitations

- **Adversarial Attacks**: Attackers can inject specific "noise" bytes into a benign section of the file. To a human, the image looks the same; to a CNN, this noise can force a misclassification (e.g., making a virus look like a calculator app).
- **Loss of Context**: Converting code to pixels destroys the semantic meaning of instructions. A `JMP` instruction is critical logic, but in an image, it is just a single pixel, potentially ignored if the CNN pools it away.
- **Computational Cost**: converting thousands of files to images and training a deep network is significantly more resource-intensive than simple signature matching.

---

5 Example

This code demonstrates the entire pipeline

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, models
import os
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# ==========================================
# 1. CONFIGURATION
# ==========================================

DATA_DIR = "./malware_dataset"
IMG_SIZE = (64, 64)
BATCH_SIZE = 32
EPOCHS = 1

# Check if data exists
```

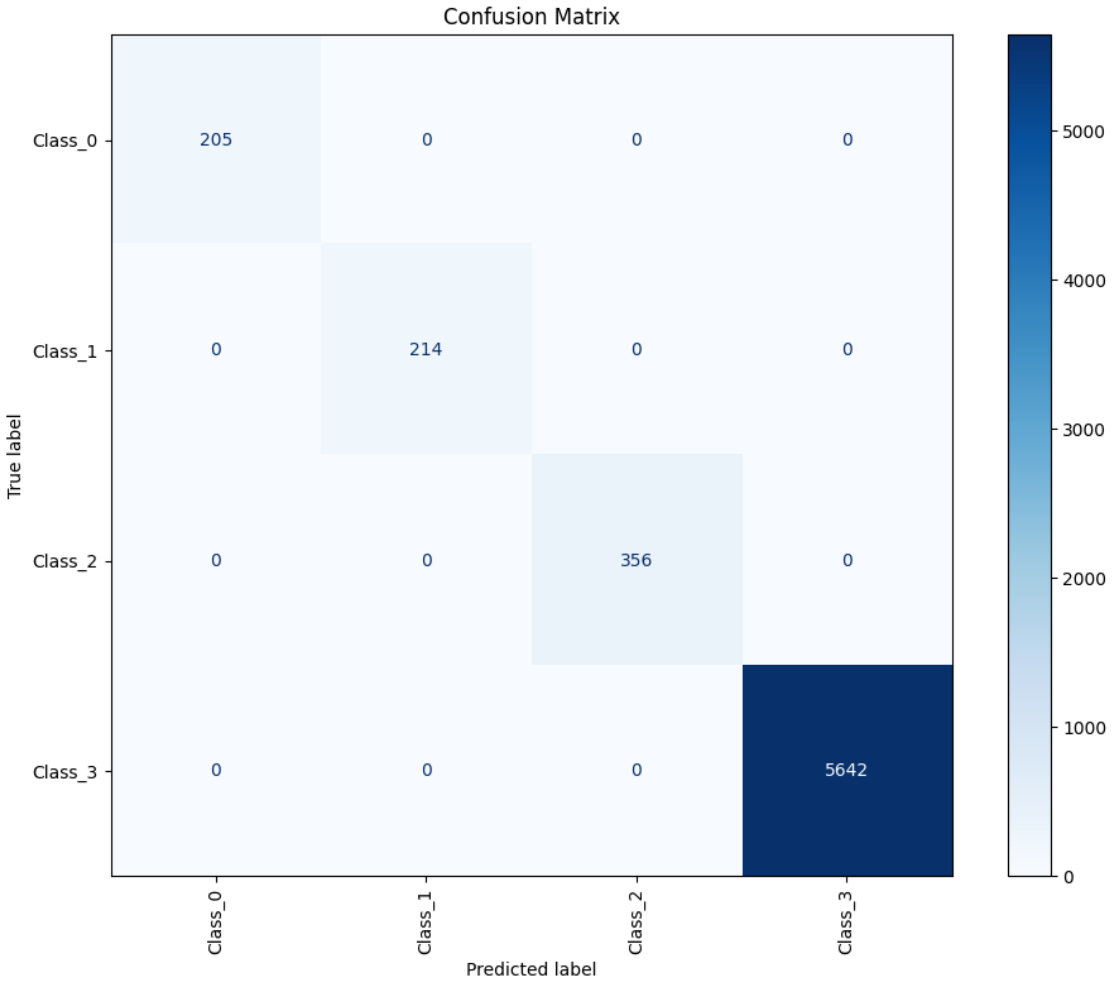
```
if not os.path.exists(DATA_DIR):
    raise FileNotFoundError(f"Directory {DATA_DIR} not found. Please run the␣
 ↪download script first.")

# ==========================================
# 2. LOAD DATA FROM FOLDERS
# ==========================================

print(f"Loading data from {DATA_DIR}...")

train_ds = tf.keras.utils.image_dataset_from_directory(
    DATA_DIR,
    validation_split=0.2,
    subset="training",
    seed=123,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    color_mode='grayscale'
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    DATA_DIR,
    validation_split=0.2,
    subset="validation",
    seed=123,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    color_mode='grayscale'
)

class_names = train_ds.class_names
num_classes = len(class_names)
print(f"\nFound {num_classes} classes: {class_names}")

# Performance tuning
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

# ==========================================
# 3. CNN MODEL
# ==========================================

model = models.Sequential([
    layers.Input(shape=(64, 64, 1)),
    layers.Rescaling(1./255),
```

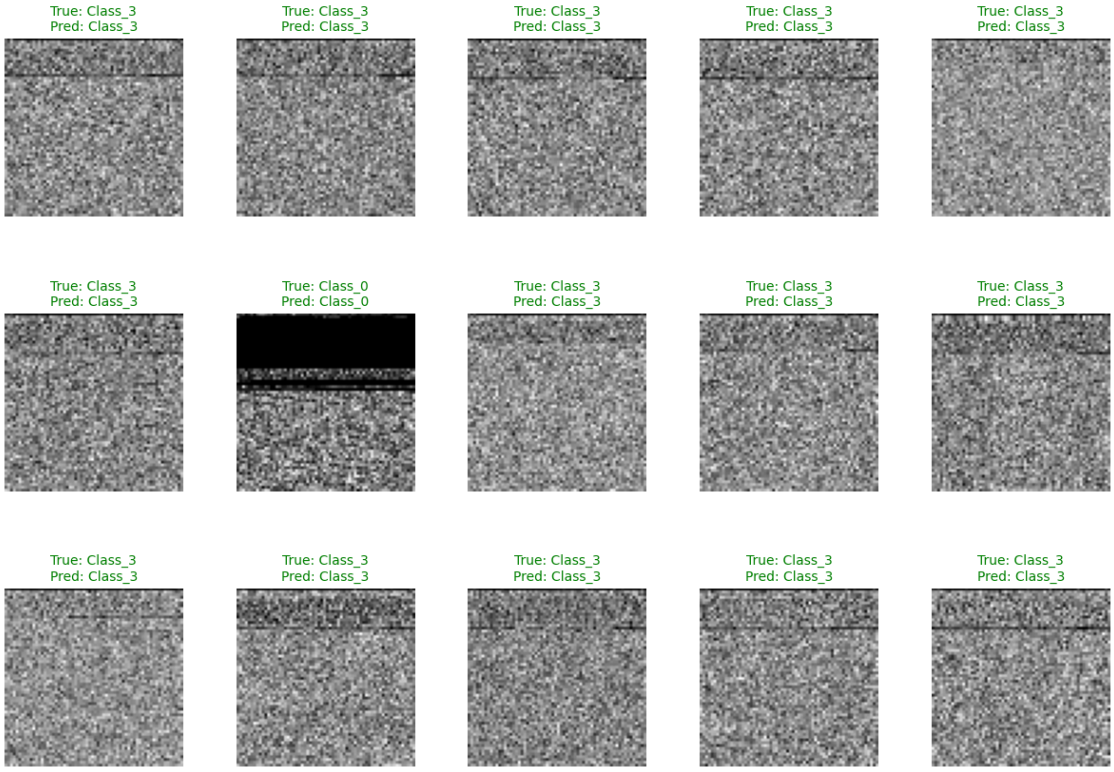
```
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),


    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(num_classes)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# ==========================================
# 4. TRAIN
# ==========================================

print("\nStarting training...")
history = model.fit(train_ds, validation_data=val_ds, epochs=EPOCHS)

# ==========================================
# 5. VISUALIZATION
# ==========================================

print("\nGenerating visualizations...")

# 5a. Collect all Validation Data and Predictions
# We iterate through the val_ds to get all images and labels in a single array
val_images = []
val_labels = []
val_predictions = []

for images, labels in val_ds:
    val_images.append(images.numpy())
    val_labels.append(labels.numpy())

    # Generate predictions for this batch
    batch_preds = model.predict(images, verbose=0)
    batch_pred_ids = np.argmax(batch_preds, axis=1)
    val_predictions.append(batch_pred_ids)

# Concatenate batches into single numpy arrays
val_images = np.concatenate(val_images)
val_labels = np.concatenate(val_labels)
val_predictions = np.concatenate(val_predictions)
```

```python
# ----------------------------------------
# Plot 1: Confusion Matrix
# ----------------------------------------
fig_cm, ax_cm = plt.subplots(figsize=(10, 8))
cm = confusion_matrix(val_labels, val_predictions)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)

# Plot with vertical x-labels for better readability if many classes exist
disp.plot(cmap=plt.cm.Blues, ax=ax_cm, xticks_rotation='vertical')
plt.title("Confusion Matrix")
plt.tight_layout()
plt.show()

# ----------------------------------------
# Plot 2: 3x5 Grid of Random Samples
# ----------------------------------------
plt.figure(figsize=(15, 10))
plt.subplots_adjust(hspace=0.5, wspace=0.3)

# Pick 15 unique random indices from the validation set
total_images = len(val_images)
random_indices = np.random.choice(total_images, size=15, replace=False)

for i, idx in enumerate(random_indices):
    image = val_images[idx]
    true_label_idx = val_labels[idx]
    pred_label_idx = val_predictions[idx]

    true_name = class_names[true_label_idx]
    pred_name = class_names[pred_label_idx]

    # Setup subplot
    ax = plt.subplot(3, 5, i + 1)
    plt.imshow(image.astype("uint8").reshape(64, 64), cmap='gray')

    # Color code title: Green if correct, Red if wrong
    color = 'green' if true_label_idx == pred_label_idx else 'red'
    plt.title(f"True: {true_name}\nPred: {pred_name}", color=color, fontsize=10)
    plt.axis('off')

plt.suptitle("Random Validation Predictions (3x5)", fontsize=16)
plt.show()
```

```
2025-12-01 20:20:01.750887: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:31]
Could not find cuda drivers on your machine, GPU will not be used.
2025-12-01 20:20:01.807022: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
```

```
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
2025-12-01 20:20:03.763207: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:31]
Could not find cuda drivers on your machine, GPU will not be used.

Loading data from ./malware_dataset…
Found 32086 files belonging to 4 classes.
Using 25669 files for training.

2025-12-01 20:20:05.368792: E
external/local_xla/xla/stream_executor/cuda/cuda_platform.cc:51] failed call to
cuInit: INTERNAL: CUDA error: Failed call to cuInit: CUDA_ERROR_NO_DEVICE: no
CUDA-capable device is detected

Found 32086 files belonging to 4 classes.
Using 6417 files for validation.

Found 4 classes: ['Class_0', 'Class_1', 'Class_2', 'Class_3']

Starting training…
803/803             27s 30ms/step -
accuracy: 0.9968 - loss: 0.0112 - val_accuracy: 1.0000 - val_loss: 2.9379e-05

Generating visualizations…

2025-12-01 20:20:44.247265: I tensorflow/core/framework/local_rendezvous.cc:407]
Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```


Confusion Matrix
Class_0
Class_1
Class_2
Class_3
True label
205
0
0
0
0
214
0
0
0
0
356
0
0
0
0
5642
Class_0
Class_1
Class_2
Class_3
Predicted label
5000
4000
3000
2000
1000
0


Random Validation Predictions (3x5)
True: Class_3
Pred: Class_3
True: Class_3
Pred: Class_3
True: Class_3
Pred: Class_3
True: Class_3
Pred: Class_3
True: Class_3
Pred: Class_3
True: Class_3
Pred: Class_3
True: Class_0
Pred: Class_0
True: Class_3
Pred: Class_3
True: Class_3
Pred: Class_3
True: Class_3
Pred: Class_3
True: Class_3
Pred: Class_3
True: Class_3
Pred: Class_3
True: Class_3
Pred: Class_3
True: Class_3
Pred: Class_3
True: Class_3
Pred: Class_3