

AI and ML for Cybersecurity Final Exam

Student: Aleksandre Latsuzbaia

Date: 28/11/2025

Task 1:

Convolutional Neural Network

1. Description of the Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a class of deep neural networks, most commonly applied to analyzing visual imagery, though its architecture is highly effective for any data with a grid-like topology (such as audio spectrograms or time-series data). Unlike traditional fully connected networks, where every input neuron is connected to every output neuron in the subsequent layer, CNNs utilize a specialized architecture designed to automatically and adaptively learn spatial hierarchies of features.

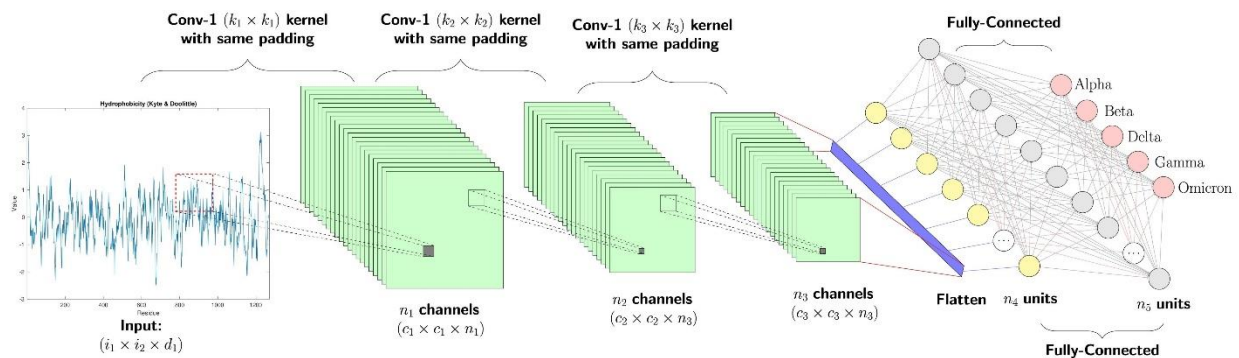


Image source: Shutterstock

The core building block of a CNN is the Convolutional Layer. This layer applies a set of learnable filters to the input. As the filter slides (convolves) across the input data, it performs an element-wise multiplication and summation, producing a feature map. This mathematical operation allows the network to detect specific features—such as edges, curves, or textures—regardless of their position in the input. This property is known as **translation invariance**.

Following the convolutional operations, CNNs typically employ a non-linear activation function, most commonly the Rectified Linear Unit (ReLU) ($f(x) = \max(0, x)$), which introduces non-linearity to the model, allowing it to learn complex patterns.

Another critical component is the Pooling Layer. Pooling operations (such as Max Pooling or Average Pooling) reduce the spatial dimensions of the feature maps. This downsampling reduces the number of parameters and computation in the network, helping to control overfitting and making the detection of features more robust to small distortions or translations.

Finally, after several alternating convolutional and pooling layers, the high-level reasoning is performed via Fully Connected (Dense) Layers. The flattened output of the feature maps flows into these layers to output the final class scores or predictions (e.g., using a Softmax function for classification). Through the process of backpropagation, the network adjusts the weights of the filters to minimize the error, effectively "learning" which features are most important for the specific task.

2. Practical Application in Cybersecurity: Malware Classification

One specific and powerful application of CNNs in cybersecurity is Malware Classification via Image Representation.

The Concept:

Malware binaries can be visualized as grayscale images. A binary file is essentially a sequence of bytes (0-255). If we treat each byte as a pixel intensity value and arrange them in a grid (with a fixed width), distinct textural patterns emerge. Different malware families (e.g., Ransomware vs. Spyware) exhibit unique visual structures in these images due to reused code sections, packing algorithms, and encryption methods. A CNN can analyze these visual textures to classify the malware family much faster than traditional reverse engineering.

The Data:

For this example, we use a synthetic dataset generated within the code to simulate the pixel data extracted from malware binaries.

- **Class 0 (Benign/Safe):** Simulates code with uniform distribution and low entropy.
- **Class 1 (Malicious):** Simulates code with specific "dense" blocks (representing encrypted payloads or packed sections) often found in malware.

Code:

```
import numpy as np
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# 1. DATA GENERATION (Synthetic Data)

def generate_synthetic_malware_data(num_samples=1000, img_shape=(64, 64, 1)):
    X = np.zeros((num_samples, *img_shape))
    y = np.zeros((num_samples,))

    for i in range(num_samples):
        # Base noise
        image = np.random.rand(*img_shape)

        # Randomly assign class
        is_malware = np.random.choice([0, 1])

        if is_malware == 1:
            # Inject "malicious patterns" (dense blocks of high intensity)
            # This simulates high entropy sections often found in packed malware
            block_size = 15
```

```

        x_start = np.random.randint(0, img_shape[0] - block_size)
        y_start = np.random.randint(0, img_shape[1] - block_size)
        image[x_start:x_start + block_size, y_start:y_start + block_size] += 0.8

    # Clip values to keep them between 0 and 1
    image = np.clip(image, 0, 1)
    y[i] = 1
else:
    y[i] = 0

X[i] = image

return X, y

print("Generating synthetic cybersecurity dataset...")
X, y = generate_synthetic_malware_data()
print(f"Dataset generated: {X.shape} samples.")

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# 2. MODEL ARCHITECTURE
def build_cnn_model(input_shape):
    model = models.Sequential()

    # Convolutional Layer 1: Detects low-level features (edges/bytes patterns)
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(layers.MaxPooling2D((2, 2)))

    # Convolutional Layer 2: Detects higher-level features (texture of code sections)
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))

    # Convolutional Layer 3
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))

    # Flattening
    model.add(layers.Flatten())

    # Fully Connected Layers
    model.add(layers.Dense(64, activation='relu'))
    # Output layer (Binary classification: Malware vs Benign)
    model.add(layers.Dense(1, activation='sigmoid'))

```

```
        model.compile(optimizer='adam',
                      loss='binary_crossentropy',
                      metrics=['accuracy'])
    return model

# 3. TRAINING AND EVALUATION

input_shape = (64, 64, 1)
model = build_cnn_model(input_shape)

print("\nTraining CNN model on malware image representations...")
history = model.fit(X_train, y_train, epochs=5, batch_size=32,
                    validation_data=(X_test, y_test), verbose=1)

print("\nEvaluating Model...")
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Generate predictions for detailed report
y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype("int32")

print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['Benign', 'Malicious']))

print("Process completed successfully.")
```