Convolutional Neural Networks (CNNs) for Malware File Classification

A convolutional neural network (CNN) is a deep neural network architecture that uses convolution operations to extract local patterns from its inputs. Instead of treating every input feature independently, CNNs slide small learnable filters over the data. Each filter responds strongly when it encounters a pattern it has learned, such as a particular edge in an image or a specific arrangement of bytes in a file. This reuse of the same filter across the whole input (weight sharing) keeps the number of parameters manageable and allows the model to detect patterns regardless of their exact position.

A typical CNN layer takes an input tensor, applies several filters, and produces a stack of **feature maps**. After a non-linear activation function (like ReLU), a **pooling layer** often follows. Pooling summarises values in a local neighbourhood, for example by taking the maximum, which shrinks the spatial resolution and introduces invariance to small shifts and noise. Several convolution + pooling stages can be stacked, gradually building up from simple, low-level features to more abstract, high-level ones. Finally, the feature maps are flattened and passed through one or more fully connected layers to perform classification. Training is done using gradient-based optimisation to minimise a loss function such as cross-entropy.

While CNNs are famous for image recognition, the same idea applies to cybersecurity data. One popular trick is to **represent malware binaries as images**: bytes are interpreted as pixel intensities, reshaped into a 2D array. Malicious and benign binaries often produce visually different textures and structures. A 2D CNN can then learn to distinguish those patterns automatically, without handcrafted features like opcode statistics or imported function lists.

Example: 2D CNN on Synthetic “Malware Images”

Below is a minimal Python example that demonstrates such an approach using synthetic data. Each sample is a 32×32 grayscale “image” representing a file. In practice, you would load real binaries, map their bytes to pixel values, and normalise them.

Example data

- 600 artificial file images, each 32×32 pixels, one channel.
- Labels: `0` for benign files, `1` for malware.

Python code

```python
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# ------------------------------
# 1. Synthetic "malware image" dataset
# ------------------------------
np.random.seed(7)

n_samples = 600
img_h, img_w = 32, 32

# Random pixel intensities in [0, 1]
X = np.random.rand(n_samples, img_h, img_w, 1)

# Labels: 0 = benign, 1 = malware
y = np.random.choice([0, 1], size=n_samples, p=[0.6, 0.4])

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=7, stratify=y
)

# ------------------------------
# 2. Define a small 2D CNN
# ------------------------------
model = Sequential()
model.add(
    Conv2D(
        filters=16,
        kernel_size=(3, 3),
        activation="relu",
        input_shape=(img_h, img_w, 1),
    )
)
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.4))
model.add(Dense(1, activation="sigmoid"))
```

```
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)

model.summary()

# ------------------------------
# 3. Train and evaluate
# ------------------------------
history = model.fit(
    X_train,
    y_train,
    epochs=12,
    batch_size=32,
    validation_split=0.2,
    verbose=1,
)

y_pred_prob = model.predict(X_test)
y_pred = (y_pred_prob > 0.5).astype("int32")

print("Classification report on synthetic malware images:")
print(classification_report(y_test, y_pred, digits=3))
```