Task 1 — Convolutional Neural Network (CNN)

Course: AI & ML for Cybersecurity — Final Exam Retake
Student: Bekar Oikashvili

1) Comprehensive Description of CNN

A **Convolutional Neural Network (CNN)** is a deep-learning architecture designed to learn hierarchical patterns from grid-like data (images, spectrograms, packet-flow "maps", etc.). Its central operation is the **convolution**, where a small learnable filter (kernel) slides across an input tensor, computing dot products that emphasize local relationships (e.g., edges, bursts, or clustered activations). Because the same filter weights are **shared** across all locations, CNNs drastically reduce the number of parameters compared to fully connected networks and improve generalization.

A typical CNN comprises stacked **Conv2D** layers (for feature extraction), **nonlinear activations** (e.g., ReLU) to model complex functions, and **Pooling** layers (e.g., max pooling) to down-sample representations, provide translation invariance, and reduce computation. After several convolutional blocks, **fully connected** layers combine the learned high-level features to produce output probabilities through **sigmoid** (binary) or **softmax** (multi-class). Additional components—**batch normalization**, **dropout**, and **residual connections**—improve training stability and robustness.

In **cybersecurity**, CNNs are effective because many security signals can be represented as structured arrays. Examples include packet bytes, flow statistics, PE/ELF byte images, or log windows. By arranging features into 2-D grids, CNNs can learn spatial co-occurrence patterns—such as unusual flag combinations, periodicity in packet sizes, or opcode motifs typical of malware families—without hand-crafted signatures. This enables detection of **intrusions**, **malware classification**, **phishing-page screenshot analysis**, and **steganography/obfuscation** recognition. Compared to rule-based systems, CNNs adapt as threats evolve: the filters are learned directly from data and can capture previously unseen patterns. The mini-project below demonstrates end-to-end application in an intrusion-like setting: we generate a synthetic dataset of network flows as 8×8 feature maps, then train a compact CNN to classify **normal vs attack**. The entire pipeline—**data generation → training → evaluation**—is reproducible and self-contained.

2) Practical Cybersecurity Example (Data + Python Code)

2.1 Generate Dataset (CSV)

File: task_1/generate_intrusion_data.py

```
#!/usr/bin/env python3

import numpy as np

import pandas as pd

from pathlib import Path


def main(n_normal=500, n_attack=500, seed=42, out_csv="task_1/data/flows.csv"):

    rng = np.random.default_rng(seed)


    # 8x8 feature maps flattened -> 64 features per sample

    # Benign flows ~ N(0, 1), Attack flows ~ N(2, 1)

    X_normal = rng.normal(loc=0.0, scale=1.0, size=(n_normal, 64))

    X_attack = rng.normal(loc=2.0, scale=1.0, size=(n_attack, 64))

    X = np.vstack([X_normal, X_attack])

    y = np.array([0]*n_normal + [1]*n_attack)  # 0=normal, 1=attack


    cols = [f"f{i}" for i in range(64)]

    df = pd.DataFrame(X, columns=cols)

    df["label"] = y


    out_path = Path(out_csv)

    out_path.parent.mkdir(parents=True, exist_ok=True)
```

```
    df.to_csv(out_path, index=False)

    # Embeddable previews for the PDF
    print("\n=== Data preview (first 8 rows) ===")
    print(df.head(8).to_string(index=False))

    print("\n=== Data preview (last 8 rows) ===")
    print(df.tail(8).to_string(index=False))

    print("\n=== Class balance ===")
    print(df['label'].value_counts().rename({0:'normal',1:'attack'}).to_string())

    print(f"\nSaved dataset to: {out_path.resolve()} (shape={df.shape})")

if __name__ == "__main__":
    main()
```

How to run

python3 task_1/generate_intrusion_data.py

2.2 Train & Evaluate CNN on the Generated Data

File: task_1/cnn_intrusion_demo.py

```
#!/usr/bin/env python3
import numpy as np
```

```
import pandas as pd
from pathlib import Path
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt

def load_csv_as_images(csv_path, side=8):
    df = pd.read_csv(csv_path)
    y = df["label"].values.astype(int)
    X = df.drop(columns=["label"]).values.astype(np.float32)
    X = X.reshape(-1, side, side, 1)  # (N, 8, 8, 1)
    return X, y

def build_cnn(input_shape=(8,8,1)):
    return models.Sequential([
        layers.Conv2D(16, (3,3), activation='relu', input_shape=input_shape),
        layers.MaxPooling2D((2,2)),
        layers.Conv2D(32, (3,3), activation='relu'),
        layers.Flatten(),
        layers.Dense(32, activation='relu'),
        layers.Dense(1, activation='sigmoid')
    ])
```

```
def main(csv_path="task_1/data/flows.csv", epochs=10, batch_size=32):
    X, y = load_csv_as_images(csv_path)
    Xtr, Xte, ytr, yte = train_test_split(
        X, y, test_size=0.30, stratify=y, random_state=42
    )

    model = build_cnn(input_shape=Xtr.shape[1:])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    history = model.fit(Xtr, ytr, epochs=epochs, batch_size=batch_size,
                        validation_split=0.2, verbose=0)

    y_prob = model.predict(Xte)
    y_pred = (y_prob > 0.5).astype(int).ravel()

    print("\n=== Evaluation ===")
    acc = np.mean(y_pred == yte)
    print("Accuracy:", acc)
    cm = confusion_matrix(yte, y_pred, labels=[0,1])
    print("Confusion Matrix:\n", cm)
    print(classification_report(yte, y_pred, target_names=["normal","attack"]))

    # Optional: save confusion matrix figure
    fig, ax = plt.subplots()
```

```
    im = ax.imshow(cm, cmap='Blues')
    ax.set_title('Confusion Matrix')
    ax.set_xticks([0,1]); ax.set_xticklabels(['normal','attack'])
    ax.set_yticks([0,1]); ax.set_yticklabels(['normal','attack'])
    for i in range(2):
        for j in range(2):
            ax.text(j, i, cm[i, j], ha='center', va='center')
    fig.colorbar(im, ax=ax)
    fig.tight_layout()
    Path("task_1").mkdir(exist_ok=True)
    fig.savefig("task_1/confusion_matrix.png", dpi=150)
    print("Saved: task_1/confusion_matrix.png")


if __name__ == "__main__":
    main()
```

How to run

after generating data

python3 task_1/cnn_intrusion_demo.py

3) Embedded Data (Preview)

Below is an excerpt (first 8 rows) from the generated CSV (task_1/data/flows.csv), as printed by the generator script:

=== Data preview (first 8 rows) ===

```
      f0       f1       f2       f3       f4       f5       f6       f7       f8       f9      f10      f11      f12
f13      f14      f15      f16      f17      f18      f19      f20      f21      f22      f23      f24      f25
```

f26 f27 f28 f29 f30 f31 f32 f33 f34 f35 f36 f37 f38
f39 f40 f41 f42 f43 f44 f45 f46 f47 f48 f49 f50 f51
f52 f53 f54 f55 f56 f57 f58 f59 f60 f61 f62 f63 label

0.304717 -1.039984 0.750451 0.940565 -1.951035 -1.302180 0.127840 -0.316243 -0.016801
-0.853044 0.879398 0.777792 0.066031 1.127241 0.467509 -0.859292 0.368751 -0.958883
0.878450 -0.049926 -0.184862 -0.680930 1.222541 -0.154529 -0.428328 -0.352134 0.532309
0.365444 0.412733 0.430821 2.141648 -0.406415 -0.512243 -0.813773 0.615979 1.128972
-0.113947 -0.840156 -0.824481 0.650593 0.743254 0.543154 -0.665510 0.232161 0.116686
0.218689 0.871429 0.223596 0.678914 0.067579 0.289119 0.631288 -1.457156 -0.319671
-0.470373 -0.638878 -0.275142 1.494941 -0.865831 0.968278 -1.682870 -0.334885
0.162753 0.586222 0

0.711227 0.793347 -0.348725 -0.462352 0.857976 -0.191304 -1.275686 -1.133287 -0.919452
0.497161 0.142426 0.690485 -0.427253 0.158540 0.625590 -0.309347 0.456775 -0.661926
-0.363054 -0.381738 -1.195840 0.486972 -0.469402 0.012494 0.480747 0.446531 0.665385
-0.098485 -0.423298 -0.079718 -1.687334 -1.447112 -1.322700 -0.997247 0.399774
-0.905479 -0.378163 1.299228 -0.356264 0.737516 -0.933618 -0.205438 -0.950022
-0.339033 0.840308 -1.727320 0.434424 0.237736 -0.594150 -1.446058 0.072130 -0.529493
0.232676 0.021852 1.601779 -0.239356 -1.023497 0.179276 0.219997 1.359188 0.835111
0.356871 1.463303 -1.188763 0

-0.639752 -0.926576 -0.389810 -1.376686 0.635151 -0.222223 -1.470806 -1.015579
0.313514 0.838127 1.996731 2.913862 0.414409 -0.989538 -2.132046 0.267711 -0.812941
-0.415357 -0.612097 -0.140791 1.065980 0.157049 -0.158635 -1.035654 -1.674683
-0.486308 -0.053783 1.767930 0.130275 0.982740 -0.499296 -1.184944 -0.965117 -0.725226
2.128470 -0.821387 0.838489 -0.902927 0.931573 0.384951 -0.156638 -0.040763 -0.654788
0.446072 -0.454983 -1.225606 -1.277938 0.172588 1.579091 0.159992 -0.118638 0.285826
1.306002 0.219383 -0.410927 1.106289 0.428756 1.535756 0.183234 -1.224469 -1.368159
1.650928 1.723666 -0.179519 0

-0.383187 1.461444 -1.107046 -0.894727 0.643327 -0.394605 -0.005122 -0.163443
0.337575 1.407482 0.090585 0.643939 -2.050172 -0.048718 -0.843230 -1.218813 -0.878152
-0.334123 0.915903 -1.326393 0.030631 -0.484169 -0.327673 1.002758 0.538115 1.337398
-0.154506 -0.695943 -0.223859 0.242497 0.176573 -1.084388 0.090490 0.228228 2.517474
1.876845 -0.853243 -0.287383 -1.463442 -0.590707 0.315605 1.205854 -0.729084 -0.654146
-2.147289 -0.162666 -1.062414 -0.529439 -0.876861 -0.094263 -1.757728 -1.467045
2.129247 -1.287423 -1.096786 1.836914 2.905067 -1.171567 -0.368249 0.341556 1.728698
-0.986857 -0.245278 0.777338 0

0.434766 -0.376156 -0.133823 -1.374896 -0.238174 -0.266387 0.232170 -0.555327
0.471539 1.012716 0.155429 0.351756 0.053155 0.000084 -0.721558 0.316494 -0.097287
2.093168 1.573355 0.385847 -0.763057 -1.112411 1.191143 0.262749 0.480143 -1.744586
0.927438 0.454420 -1.110431 -0.471525 0.263717 0.052467 -0.292171 -0.103488 -0.251977
0.152563 1.471492 -2.566658 -0.236850 0.176512 0.295994 -0.371915 -1.756722 0.327995

1.727350 -1.533861 0.863828 -0.328525 -0.061324 -1.052899 -0.334456 1.300045 0.582655
1.732312 1.177412 0.439087 1.743935 0.438993 0.827988 -0.296571 0.066546 -0.697424
0.989584 -1.178304   0

 0.782350 -0.190651 1.171247 0.750869 1.820646 0.730775 -1.572040 -0.066953 -1.172007
-0.518280 1.511228 0.637534 -0.698930 -1.013717 0.032782 -1.216560 -0.671140 0.312009
1.155312 0.608761 -2.291290 0.304367 0.072034 0.413890 1.616210 -2.063238 -0.591103
0.590906 -1.581594 1.475949 0.368357 0.846584 -0.570944 0.813764 1.068472 0.232878
0.234401 0.270343 -0.863345 -0.147529 -0.152523 0.383394 0.999824 -1.058536 -0.125009
1.481456 -0.743588 -0.822250 0.202306 0.844385 0.011426 1.328961 0.856794 0.841820
0.554117 2.327653 -0.205162 -2.003522 1.604254 -0.457699 0.107880 1.309551 -1.602260
-1.251647   0

-1.601278 -0.794136 0.439637 0.524188 0.276274 -1.412766 -2.310103 0.054354 -0.471776
0.459386 0.701954 0.138241 0.760133 0.229211 0.530065 -0.704673 -0.179611 0.196776
0.820528 -0.393741 0.521167 -0.265839 -0.117542 0.829519 -1.993060 -1.296472 -1.482185
-2.333616 -0.678264 0.749434 -0.284884 0.197790 1.089217 1.327686 -0.069138 1.353586
0.092127 -0.837398 -0.594400 -1.480537 -0.888134 -0.358017 0.803585 1.720770 -1.382182
0.392827 -1.040544 0.474697 -0.131087 -1.830906 0.928297 -0.605001 -0.533900 -1.069752
-0.654283 0.427890 -0.189244 0.328662 0.361922 1.320662 -0.342786 -1.476858 1.067222
-0.331488   0

 1.114592 0.383377 -0.131138 0.348776 1.951013 2.076981 0.069381 0.160191 1.076240
-0.845661 0.333070 -0.025863 0.313908 -0.833369 -1.589567 -2.072983 -1.117384 -0.458675
-0.293192 1.937231 1.105993 -0.962091 0.347708 -0.407078 -0.284364 0.185326 0.619171
-0.339258 1.063852 -1.141938 0.006339 2.597674 0.223080 1.433215 0.091520 0.580777
-0.056783 -0.170408 -0.779482 0.430301 -0.851537 0.665585 1.085287 0.366531 -0.286249
0.453966 -0.308673 0.935547 -1.831406 -0.335607 -1.990812 -1.495061 1.363862 0.895185
-0.719480 -1.502503 -2.964529 -0.543496 2.420415 0.434884 -0.559572 0.465080
-1.560958 -0.297323   0

4) Embedded Evaluation Output

From task_1/cnn_intrusion_demo.py:

10/10 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step

=== Evaluation ===

Accuracy: 1.0

Confusion Matrix:

[[150 0]

[0 150]]

	precision	recall	f1-score	support
normal	1.00	1.00	1.00	150
attack	1.00	1.00	1.00	150
accuracy			1.00	300
macro avg	1.00	1.00	1.00	300
weighted avg	1.00	1.00	1.00	300

Why this is high: classes are generated from clearly separated distributions (benign ~ N(0,1) vs attack ~ N(2,1)), so a compact CNN fits them perfectly.

5) Reproducibility

Environment

python>=3.10

tensorflow

numpy

pandas

scikit-learn

matplotlib

Run order

python3 task_1/generate_intrusion_data.py

python3 task_1/cnn_intrusion_demo.py

Notes

- Input shape: (8, 8, 1) from 64 features.
- 70/30 stratified split, random_state=42.
- Model: Conv2D(16) → MaxPool → Conv2D(32) → Dense(32) → Sigmoid.