

Kelompok 9

# Machine Learning

LuNet: A Deep Neural Network for  
Network Intrusion Detection (2019)

5025221160- Fairna Mustika Hermafiddhanti

5025221281 - Aulia Putri Salsabila

5025221290 - Nisrina Salma Robbani

[Link Notebook Model LuNet](#)

[Link Notebook Model Benchmark](#)



# LuNet: A Deep Neural Network for Network Intrusion Detection

## LuNet: A Deep Neural Network for Network Intrusion Detection

Peilun Wu\* and Hui Guo†

\*†The University of New South Wales, Sydney, Australia  
Email: {\*z5100023, †huig}@cse.unsw.edu.au

**Abstract**—Network attack is a significant security issue for modern society. From small mobile devices to large cloud platforms, almost all computing products, used in our daily life, are networked and potentially under the threat of network intrusion. With the fast-growing network users, network intrusions become more and more frequent, volatile and advanced. Being able to capture intrusions in time for such a large scale network is critical and very challenging. To this end, the machine learning (or AI) based network intrusion detection (NID), due to its intelligent capability, has drawn increasing attention in recent years. Compared to the traditional signature-based approaches, the AI-based solutions are more capable of detecting variants of advanced network attacks. However, the high detection rate achieved by the existing designs is usually accompanied by a high rate of false alarms, which may significantly discount the overall effectiveness of the intrusion detection system. In this paper, we consider the existence of spatial and temporal features in the network traffic data and propose a hierarchical CNN+RNN neural network, LuNet. In LuNet, the convolutional neural network (CNN) and the recurrent neural network (RNN) learn input traffic data in sync with a gradually increasing granularity such that both spatial and temporal features of the data can be effectively extracted. Our experiments on two network traffic datasets show that compared to the state-of-the-art network intrusion detection techniques, LuNet not only offers a high level of detection capability but also has a much low rate of false positive-alarm.

**Keywords**—network intrusion detection; convolutional neural network; LuNet; recurrent neural network

<https://doi.org/10.1109/SSCI44817.2019.9003126>

909.10031v2 [cs.AI] 6 Oct 2019

only able to quickly and correctly identify known attacks but also adaptive and intelligent enough for the unknown and evolved attacks, which leads to AI-based solutions. The artificial intelligence gained from machine learning enables the detection system to discover network attacks without much need for human intervention [1].

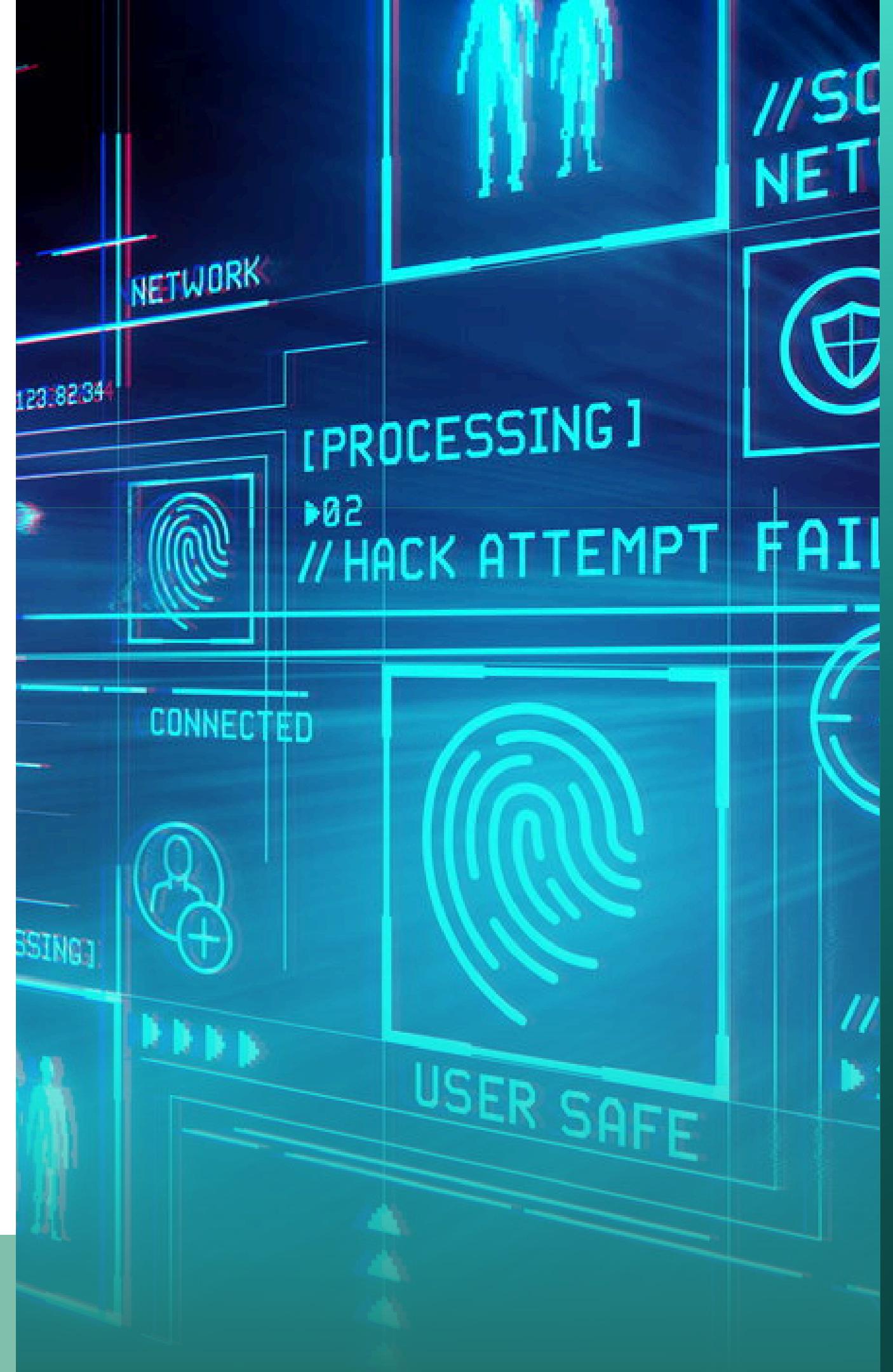
So far, investigations on the AI-based solutions are mainly based on two schemes: anomaly detection and misuse detection. The anomaly detection identifies an attack based on its anomalies deviated from the profile of normal traffic. Nevertheless, this scheme may have a high false positive rate if the normal traffic is not well profiled, and the profile used in the detection is not fully representative [2]. Furthermore, to obtain a fully representative normal traffic profile for a dynamically evolving and expanding network is unlikely possible.

The misuse detection, on the other hand, focuses on the abnormal behaviour directly. The scheme can learn features of attacks based on a labelled training dataset, where both normal and attacked traffic data are marked. Given sufficient labelled data, a misuse-detection design can effectively generate an abstract hypothesis of the boundary between normal and malicious traffic. The hypothesis can then be used to detect attacks for future unknown traffic. Therefore the misuse detection is more feasible and effective than the anomaly detection [2], [3] and has been adopted in real-world systems, such as Google, Qihoo 360 and Aliyun.

Tujuan utama dari paper ini adalah untuk mengatasi tingginya tingkat false positive (FP rate) dalam sistem deteksi intrusi jaringan (Network Intrusion Detection Systems – NIDS) yang ada, khususnya yang berbasis misuse detection.

# LuNet: Kombinasi Hirarkis CNN dan RNN

LuNet terdiri dari beberapa tingkat sub-jaringan konvolusi dan rekuren yang digabungkan. Pada setiap tingkat, data masukan dipelajari oleh jaringan CNN dan RNN, dan seiring dengan kemajuan pembelajaran dari tingkat pertama hingga tingkat terakhir, granularitas pembelajaran menjadi semakin rinci. Sinergi CNN dan RNN dapat secara efektif dimanfaatkan untuk ekstraksi fitur spasial dan temporal. Hasil pengujian pada dua dataset menunjukkan bahwa LuNet memiliki tingkat deteksi tinggi dengan tingkat false positive yang jauh lebih rendah dari metode deteksi intruksi lainnya.





## Kelompok 9

# DATASET

### UNSW-NB15 Dataset

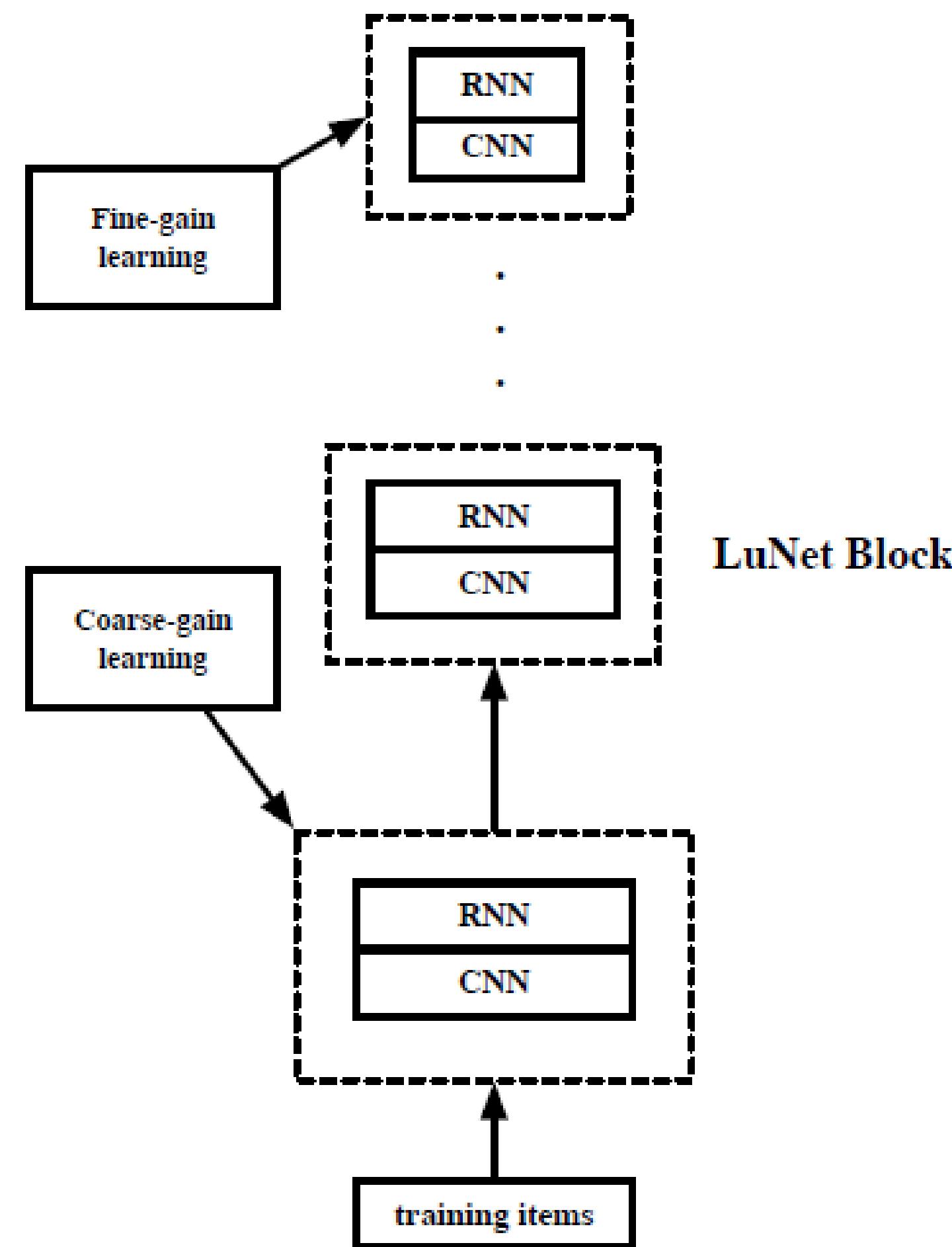
Mengandung campuran lalu lintas normal yang realistik dan sembilan jenis serangan modern, termasuk Exploits, Fuzzers, Backdoors, Analysis, Reconnaissance, Generic, Shellcode, Worms, dan DoS. Dataset ini terdiri dari 175,341 data training dan 82,332 data testing.

[] df_train.head()	
→	id dur proto service state spkts dpkts sbytes nbytes rate ... ct_dst_sport_ltm ct_dst_src_ltm is_ftp_login ct_ftp_cmd ct_flw_http_mthd ct_src_ltm ct_srv_dst is_sm_ips_ports attack_cat label
0	1 0.000011 udp - INT 2 0 496 0 90909.0902 ... 1 2 0 0 0 1 2 0 Normal 0
1	2 0.000008 udp - INT 2 0 1762 0 125000.0003 ... 1 2 0 0 0 1 2 0 Normal 0
2	3 0.000005 udp - INT 2 0 1068 0 200000.0051 ... 1 3 0 0 0 1 3 0 Normal 0
3	4 0.000006 udp - INT 2 0 900 0 166666.6608 ... 1 3 0 0 0 2 3 0 Normal 0
4	5 0.000010 udp - INT 2 0 2126 0 100000.0025 ... 1 3 0 0 0 2 3 0 Normal 0

5 rows × 45 columns

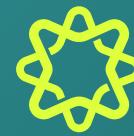
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 82332 entries, 0 to 82331
Data columns (total 45 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   id              82332 non-null   int64  
 1   dur             82332 non-null   float64 
 2   proto            82332 non-null   object  
 3   service           82332 non-null   object  
 4   state            82332 non-null   object  
 5   spkts            82332 non-null   int64  
 6   dpkts            82332 non-null   int64  
 7   sbytes           82332 non-null   int64  
 8   nbytes            82332 non-null   int64  
 9   rate              82332 non-null   float64 
 10  sttl             82332 non-null   int64  
 11  ttl              82332 non-null   int64  
 12  sload            82332 non-null   float64 
 13  dload            82332 non-null   float64 
 14  sloss             82332 non-null   int64  
 15  dloss             82332 non-null   int64  
 16  sinpkt            82332 non-null   float64 
 17  dinpkt            82332 non-null   float64 
 18  sjit              82332 non-null   float64 
 19  djit              82332 non-null   float64 
 20  swin              82332 non-null   int64  
 21  stcpb             82332 non-null   int64  
 22  dtcpb             82332 non-null   int64  
 23  dwin              82332 non-null   int64  
 24  tcprtt            82332 non-null   float64 
 25  synack            82332 non-null   float64 
 26  ackdat            82332 non-null   float64 
 27  smean             82332 non-null   int64  
 28  dmean             82332 non-null   int64  
 29  trans_depth        82332 non-null   int64  
 30  response_body_len 82332 non-null   int64  
 31  ct_srv_src         82332 non-null   int64  
 32  ct_state_ttl        82332 non-null   int64  
 33  ct_dst_ltm          82332 non-null   int64  
 34  ct_src_dport_ltm      82332 non-null   int64  
 35  ct_dst_sport_ltm      82332 non-null   int64  
 36  ct_dst_src_ltm        82332 non-null   int64  
 37  is_ftp_login          82332 non-null   int64  
 38  ct_ftp_cmd            82332 non-null   int64  
 39  ct_flw_http_mthd       82332 non-null   int64  
 40  ct_src_ltm             82332 non-null   int64  
 41  ct_srv_dst             82332 non-null   int64  
 42  is_sm_ips_ports        82332 non-null   int64  
 43  attack_cat             82332 non-null   object  
 44  label                  82332 non-null   int64  
dtypes: float64(11), int64(30), object(4)
memory usage: 28.3+ MB
```

# LuNet



## LuNet Architecture

- **CNN** berfokus pada **pola spasial** (kombinasi fitur serangan di setiap titik)
- **RNN** berfokus pada **pola temporal** (urutan serangan)
- **LuNet** : Kombinasi CNN dan RNN yang disusun secara bertumpuk di setiap levelnya dan disebut **LuNet Block**
- CNN menangkap *high level features* dan mempertahankan pola temporal yang langsung diteruskan ke RNN.
- CNN dan RNN belajar di tingkat detail yang sama serta memproses data secara independen tanpa kehilangan *learning capacity*.



# LuNet Block Architecture

Hidden layer Lunet terdiri dari 5 lapisan

### 1. Convolutional Neural Network (CNN)

- Convolutional : Mengubah data input menjadi feature map (fitur yang penting)
- Pooling : Mengurangi data yang tidak relevan dan meningkatkan learning capacity

### 2. Batch Normalization

- Scaling data

### 3. LSTM

- Control akumulasi error RNN

### 4. Dimension Reshape

- Reshape output dari satu level block agar sesuai dengan block selanjutnya

### 5. Overfitting Prevention

- Dropout layer 0.5

Deep Architecture of LuNet

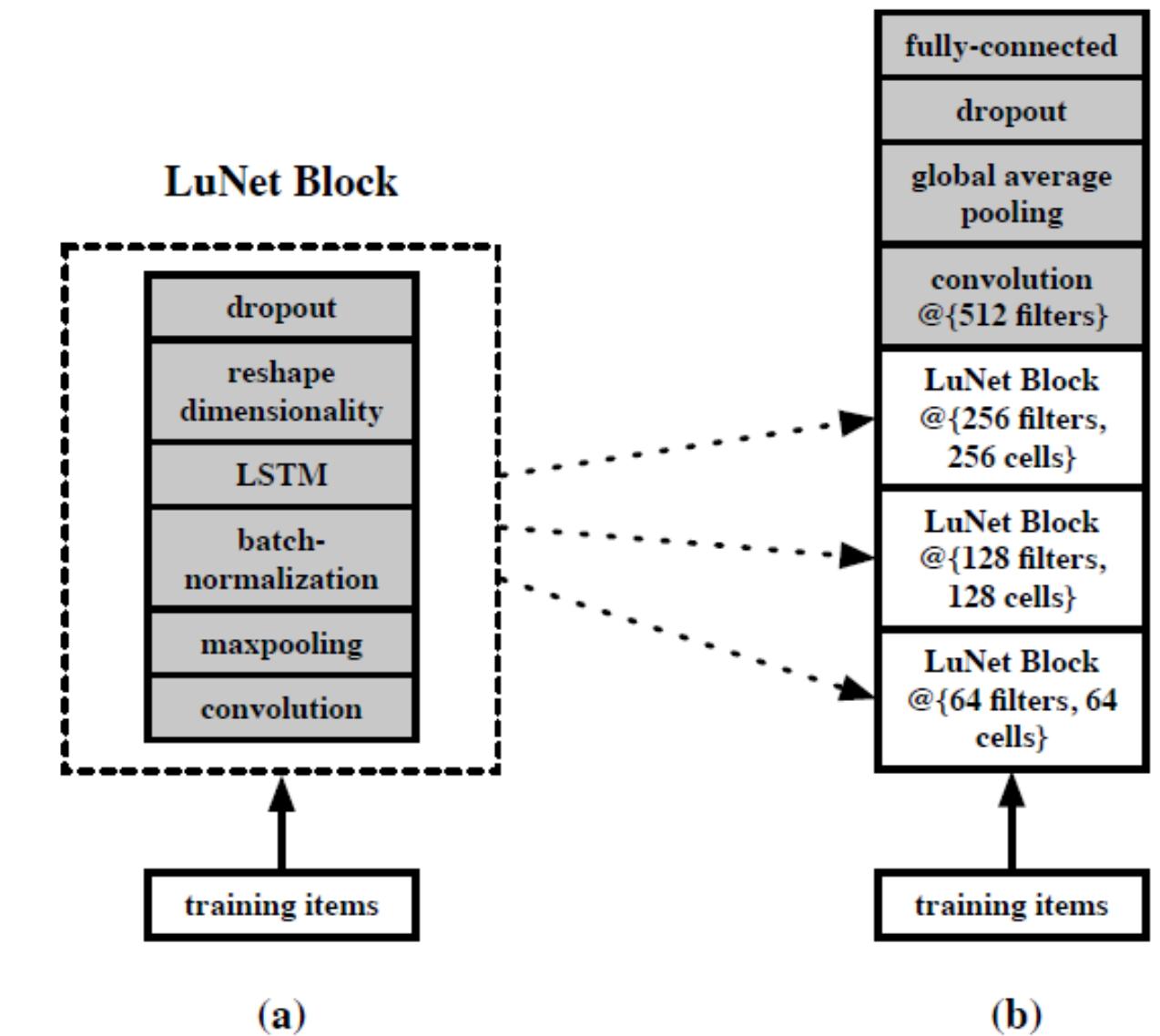


Fig. 2: LuNet. (a) a LuNet Block. (b) the overall deep architecture of LuNet.

# METODOLOGI PENELITIAN

## 1. Pra-pemrosesan Data

- Encoding Fitur Kategorikal
- Normalisasi/Standardisasi Fitur
- Stratified K-Fold Cross Validation

## 2. Implementasi Model

- LuNet diimplementasikan menggunakan TensorFlow backend, Keras, dan paket scikit-learn.
- Algoritma optimasi RMSprop digunakan untuk melatih bobot dan bias LuNet, dengan learning rate diatur ke 0.001 dan dropout rate pada lapisan dropout diatur ke 0.5.

## 3. Skenario Eksperimen

- Model yang sudah diimplementasikan kemudian diuji kemampuannya dalam skenario klasifikasi biner (membedakan serangan dan normal) dan klasifikasi multi-kelas (mengidentifikasi jenis serangan spesifik atau normal).
- Pengujian ini dilakukan dengan menggunakan strategi Stratified K-Fold Cross Validation.

## 4. Evaluasi Metriks

Kinerja LuNet dievaluasi menggunakan tiga metrik utama:

- ACC (validation accuracy)
- DR (detection rate)
- FPR (false positive alarm)

# Praprocessing Data

```
1 ATTACK_CAT_TO_ID = {  
2     'Normal': 0,  
3     'Reconnaissance': 1,  
4     'Backdoor': 2,  
5     'DoS': 3,  
6     'Exploits': 4,  
7     'Analysis': 5,  
8     'Fuzzers': 6,  
9     'Worms': 7,  
10    'Shellcode': 8,  
11    'Generic': 9,  
12 }
```

```
1 def sigmoid(x):  
2     return np.divide(1, (1 + np.exp(np.negative(x))))
```

```
1 def double_sided_log(x):  
2     return np.sign(x) * np.log(1 + np.abs(x))
```

```
1 def state_to_one_hot(x):  
2     state_val = ['INT', 'FIN', 'REQ', 'ACC', 'CON', 'RST', 'CLO', 'ECO', 'PAR', 'URN', 'no']  
3  
4     one_hot = [0] * len(state_val)  
5     one_hot[state_val.index(x)] = 1  
6  
7     return one_hot
```

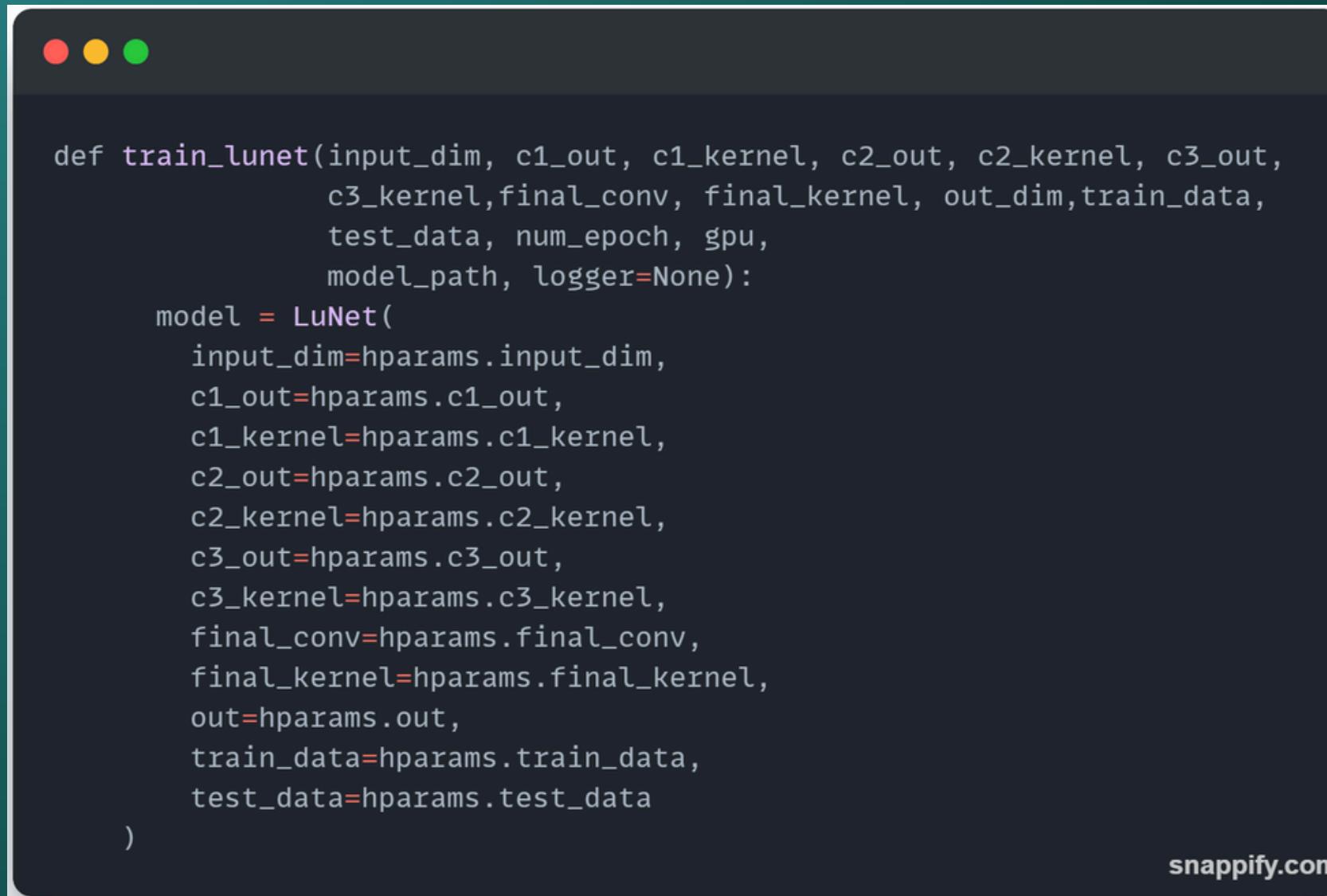
# Pragmatic Data Processing

```
1 def preprocess_data(data):
2     # extract label and drop id
3     attack_cat = data['attack_cat'].apply(func=(lambda x: ATTACK_CAT_TO_ID.get(x))).to_numpy()
4     .reshape(-1, 1)
5     label = data['label'].to_numpy().reshape(-1, 1)
6     data = data.drop(['id', 'attack_cat', 'label'], axis=1)
7
8     # categorical to one-hot-encoding
9     # extract and transform categorical to one-hot
10    proto = np.array(data['proto'].apply(lambda x: proto_to_one_hot(x)).to_list())
11    service = np.array(data['service'].apply(lambda x: service_to_one_hot(x)).to_list())
12    state = np.array(data['state'].apply(lambda x: state_to_one_hot(x)).to_list())
13    categorical = np.concatenate([proto, service, state], axis=1)
14    data = data.drop(['proto', 'service', 'state'], axis=1)
15
16    # transform to np matrix
17    data = data.to_numpy()
18
19    # merge categorical feature back to data
20    data = np.concatenate([data, categorical], axis=1)
21
22    # double-sided log: to attenuate outliers
23    data = double_sided_log(data)
24
25    # sigmoid transform
26    data = sigmoid(data)
27
28    # merge label back to data
29    data = np.concatenate([data, attack_cat, label], axis=1)
30
31    return data
```

# Model

```
1      # LuNet Block 1
2      hparams_lu_block_1 = Namespace(
3          input_dim=input_dim,
4          conv_out=c1_out,
5          kernel=c1_kernel
6      )
7      self.lu_block_1 = LuNetBlock(hparams_lu_block_1)
8
9      dummy_x = torch.randn(1, 1, input_dim, requires_grad=False)
10     dummy_x = self.lu_block_1(dummy_x)
11
12     # LuNet Block 2
13     hparams_lu_block_2 = Namespace(
14         input_dim=dummy_x.shape[2],
15         conv_out=c2_out,
16         kernel=c2_kernel
17     )
18     self.lu_block_2 = LuNetBlock(hparams_lu_block_2)
19     dummy_x = self.lu_block_2(dummy_x)
20
21     # LuNet Block 3
22     hparams_lu_block_3 = Namespace(
23         input_dim=dummy_x.shape[2],
24         conv_out=c3_out,
25         kernel=c3_kernel
26     )
27     self.lu_block_3 = LuNetBlock(hparams_lu_block_3)
28     dummy_x = self.lu_block_3(dummy_x)
29
30     # Final conv + pooling
31     self.conv = nn.Sequential(
32         nn.Conv1d(
33             in_channels=1,
34             out_channels=final_conv,
35             kernel_size=final_kernel,
36         )
37     )
38
39     dummy_x = self.conv(dummy_x)
40
41     self.avg_pool = nn.AvgPool1d(kernel_size=final_conv)
42     dummy_x = self.avg_pool(dummy_x)
43
44     self.drop_out = nn.Dropout(p=0.5)
45
46     # Output layer
47     if self.out_dim == 2:
48         self.out = nn.Sequential(
49             nn.Linear(dummy_x.shape[1] * dummy_x.shape[2], 1),
50             nn.Sigmoid()
51         )
52     else:
53         self.out = nn.Linear(dummy_x.shape[1] * dummy_x.shape[2],
54         self.out_dim)
```

# Training



```
def train_lunet(input_dim, c1_out, c1_kernel, c2_out, c2_kernel, c3_out,
                 c3_kernel, final_conv, final_kernel, out_dim, train_data,
                 test_data, num_epoch, gpu,
                 model_path, logger=None):
    model = LuNet(
        input_dim=hparams.input_dim,
        c1_out=hparams.c1_out,
        c1_kernel=hparams.c1_kernel,
        c2_out=hparams.c2_out,
        c2_kernel=hparams.c2_kernel,
        c3_out=hparams.c3_out,
        c3_kernel=hparams.c3_kernel,
        final_conv=hparams.final_conv,
        final_kernel=hparams.final_kernel,
        out=hparams.out,
        train_data=hparams.train_data,
        test_data=hparams.test_data
    )
snappyf.com
```

- Model di training untuk 2 jenis kelas : Binary dan Multiclass
- Pada binary class, output dimension = 2 yaitu normal atau intrusion
- Pada multi class, output dimension = 10, yaitu 9 jenis serangan dan normal
- Model di-*train* dalam **10 epoch** dengan 2 **batch size** yaitu **16** dan **8**

## Argumen Training

- **input\_dim=196,**
- **c1\_out=64,**
- **c1\_kernel=64,**
- **c2\_out=128,**
- **c2\_kernel=128,**
- **c3\_out=256,**
- **c3\_kernel=256,**
- **final\_conv=512,**
- **final\_kernel=512,**
- **out\_dim=2 atau 10**
- **train\_data=train\_data,**
- **test\_data=test\_data,**
- **num\_epoch=10,**

- *C\_out : jumlah filter output feature map*
- *C\_kernel : panjang kernel filter*

# Testing & Evaluasi LuNet

- Proses testing masing-masing kelas dilakukan dalam 2 kondisi yaitu menggunakan **batch train 16 dan 8**.

## Metriks Evaluasi

- **Accuracy** : Mengukur prediksi benar secara keseluruhan.
- **Detection Rate** : Berapa banyak serangan yang terdeteksi. Setinggi mungkin.
- **False Positive Rate** : Berapa banyak **data normal yang terdeteksi sebagai attack**. Semakin rendah semakin bagus.

### Binary Batch Train 16

 Evaluation Metrics (scikit-learn):  
Accuracy : 0.6806  
Detection Rate (DR) : 1.0000  
False Pos. Rate : 1.0000  
Execution Time : 4467.60 seconds

### Binary Batch Train 8

 Evaluation Metrics (scikit-learn):  
Accuracy : 0.8318  
Detection Rate (DR) : 0.7578  
False Pos. Rate : 0.0105  
Execution Time : 4318.45 seconds

*Pada model binary, model LuNet menghasilkan FPR terbaik pada batch train 8 yaitu sebesar 1% dengan akurasi 83%*

# Testing & Evaluasi LuNet

## Multi Class Batch Train 16

```
📊 Evaluation Metrics (scikit-learn):  
Accuracy : 0.6200  
Detection Rate (DR) : 0.2947  
False Pos. Rate : 0.0457  
Execution Time : 4083.78 seconds
```

Pada model multiclass, model LuNet menghasilkan **FPR terbaik pada batch train 16 yaitu sebesar 4,5% dengan akurasi 62%**.

## Multi Class Batch Train 8

```
📊 Evaluation Metrics (scikit-learn):  
Accuracy : 0.4775  
Detection Rate (DR) : 0.1901  
False Pos. Rate : 0.0660  
Execution Time : 1858.77 seconds
```

# HASIL PAPER

Binary Classification

UNSW-NB15		
DR%	ACC%	FPR%
98.50	96.65	6.60
98.04	97.51	3.44
98.12	97.62	3.26
97.78	97.67	2.54
98.45	97.57	3.96
98.18	97.40	3.96

Multi-Class Classification

UNSW-NB15		
DR%	ACC%	FPR%
95.37	84.61	1.69
95.08	84.78	1.45
96.10	84.93	2.09
95.83	85.21	1.32
97.43	85.35	2.89
95.96	84.98	1.89

Pada paper, *model binary terbaik menghasilkan FPR 2,54% dengan rata rata akurasi 97% sedangkan pada multiclass menghasilkan FPR terbaik yaitu 2,09% dengan rata-rata akurasi 84%*.

*Hasil ini berbeda dengan implementasi karena terdapat kemungkinan:*

- ukuran batch pada paper tidak sama dengan implementasi yang dilakukan.
- kendala resource GPU sehingga implementasi hanya bisa dilakukan sampai percobaan pada batch train 16.
- tidak memungkinkan untuk dilakukan tuning karena resource tidak memadai

# BENCHMARK

## CNN (CONVOLUTIONAL NEURAL NETWORK)

```
● ● ●  
1 def cnn_model(input_dim,  
2                 c1_out, c1_kernel,  
3                 c2_out, c2_kernel,  
4                 c3_out, c3_kernel,  
5                 final_conv, final_kernel,  
6                 out_classes):  
7     model = Sequential()  
8     # Input Layer  
9     model.add(Input(shape=(input_dim, 1)))  
10    # Block 1  
11    model.add(Conv1D(filters=c1_out, kernel_size=c1_kernel, activation='relu'))  
12    # Block 2  
13    model.add(Conv1D(filters=c2_out, kernel_size=c2_kernel, activation='relu'))  
14    # Block 3  
15    model.add(Conv1D(filters=c3_out, kernel_size=c3_kernel, activation='relu'))  
16    # Final Conv  
17    model.add(Conv1D(filters=final_conv, kernel_size=final_kernel, activation='relu'))  
18    # Average Pooling  
19    model.add(AveragePooling1D(pool_size=final_conv))  
20    # Dropout  
21    model.add(Dropout(0.5))  
22    # Flatten  
23    model.add(Flatten())  
24    # Output Layer  
25    if out_classes == 2:  
26        model.add(Dense(1, activation='sigmoid'))  
27    else:  
28        model.add(Dense(out_classes, activation='softmax'))  
29  
30 model.summary()
```

- Model ini dimulai dengan empat lapisan konvolusi (Conv1D) yang berfungsi untuk mengekstraksi pola atau fitur dari data input.
- Setelah itu, lapisan **AveragePooling1D** digunakan untuk merangkum dan mengurangi dimensi fitur yang telah dipelajari.
- Untuk mencegah overfitting, lapisan Dropout ditambahkan yang secara acak menonaktifkan sebagian neuron selama pelatihan.
- Lapisan Flatten kemudian mengubah data menjadi format satu dimensi agar bisa diproses oleh lapisan output.
- Terakhir, lapisan output (Dense) menentukan hasil klasifikasi untuk masalah biner atau masalah multikelas.

# BENCHMARK

## RNN (RECURRENT NEURAL NETWORK)

```
● ● ●  
1 def rnn_model(input_shape, out_classes):  
2     model = Sequential()  
3     # LSTM Block 1  
4     model.add(LSTM(64, return_sequences=True, input_shape=input_shape))  
5     # LSTM Block 2  
6     model.add(LSTM(128, return_sequences=True))  
7     # LSTM Block 3  
8     model.add(LSTM(256, return_sequences=True))  
9     # Conv1D Layer  
10    model.add(Conv1D(filters=512, kernel_size=3, padding='same', activation='relu'))  
11    # Global Average Pooling  
12    model.add(GlobalAveragePooling1D())  
13    # Dropout  
14    model.add(Dropout(0.5))  
15    # Fully Connected  
16    if out_classes == 1:  
17        model.add(Dense(1, activation='sigmoid'))  
18    else:  
19        model.add(Dense(out_classes, activation='softmax'))  
20    return model  
21
```

- Model RNN ini adalah arsitektur hibrida yang menggabungkan lapisan LSTM (Long Short-Term Memory) dan Conv1D.
- Data pertama kali dianalisis secara berurutan oleh tiga lapisan LSTM untuk mengenali pola sekuensial atau yang bergantung pada urutan waktu.
- Hasil dari LSTM kemudian dimasukkan ke lapisan Conv1D untuk mengekstraksi fitur-fitur lokal yang lebih spesifik.
- Setelah melalui lapisan pooling dan dropout, sebuah lapisan output akhir membuat keputusan klasifikasi, baik untuk masalah biner maupun multikelas.

# BENCHMARK

## HYBRID CNN + RNN

```
● ● ●  
1 model_binary = tf.keras.Sequential([  
2     tf.keras.layers.Input(shape=(num_features,)),  
3     tf.keras.layers.Dense(128, activation='relu'),  
4     tf.keras.layers.Reshape((16, 8)),  
5     tf.keras.layers.Conv1D(64, 3, activation='relu'),  
6     tf.keras.layers.Conv1D(64, 3, activation='relu'),  
7     tf.keras.layers.MaxPooling1D(2),  
8     tf.keras.layers.LSTM(64, return_sequences=True),  
9     tf.keras.layers.LSTM(64, return_sequences=True),  
10    tf.keras.layers.Conv1D(128, 3, activation='relu'),  
11    tf.keras.layers.Conv1D(128, 3, activation='relu'),  
12    tf.keras.layers.MaxPooling1D(2),  
13    tf.keras.layers.Flatten(),  
14    tf.keras.layers.Dense(256, activation='relu'),  
15    tf.keras.layers.Dense(128, activation='relu'),  
16    tf.keras.layers.Dense(1, activation='sigmoid')  
17 ])
```

- Model hibrida CNN + RNN ini bekerja dengan cara mengubah bentuk data input (Input dan Reshape) terlebih dahulu agar sesuai untuk dianalisis.
- Selanjutnya, model menggunakan beberapa lapisan konvolusi (Conv1D) yang diselingi dengan MaxPooling untuk mengekstraksi fitur-fitur lokal yang penting dari data.
- Fitur-fitur ini kemudian diproses oleh dua lapisan LSTM untuk mengenali pola yang bersifat sekuensial atau bergantung pada urutan.
- Setelah melalui blok konvolusi terakhir dan lapisan Flatten, data dimasukkan ke lapisan Dense untuk melakukan klasifikasi.

# STUDI KOMPARATIF

Model	Target	Train Batch Size	Accuracy	Detection Rate	False Positive Rate
LuNet	Binary	16	0.6806	1.000	1.000
		8	0.8318	0.7578	0.0105
	Multiclass	16	0.62	0.2947	0.0457
		8	0.4775	0.1901	0.066
CNN+RNN	Binary	16	0.9277	0.9674	0.1568
		8	0.6806	1.000	1.000
	Multiclass	16	0.7342	0.4246	0.0299
		8	0.6854	0.3662	0.0361
CNN	Binary	16	0.8988	0.8678	0.0352
		8	0.8949	0.8597	0.0302
	Multiclass	16	0.7253	0.3705	0.0349
		8	0.7383	0.3555	0.0341
RNN	Binary	16	0.8745	0.8291	0.0287
		8	0.8288	0.8024	0.115
	Multiclass	16	0.6772	0.3373	0.0425
		8	0.5992	0.2361	0.0543

Meminimalkan False  
Positive Rate (FPR)



**LuNet**  
Binary Batch 8

# ANALISIS HASIL

Secara umum, semua model menunjukkan kinerja terbaik mereka saat menangani tugas klasifikasi biner. Kinerja semua model menurun secara signifikan ketika beralih ke tugas multiclass yang lebih kompleks. Hal ini wajar karena mengklasifikasikan dua kelas (Biner) secara fundamental lebih mudah daripada banyak kelas (Multiclass).

## BINARY

- **ACCURACY** : Dicapai oleh model CNN (hingga 0.8988), menjadikannya paling andal secara keseluruhan.
- **DETECTION RATE** : Dicapai oleh CNN+RNN. Beberapa model lain mendapat skor sempurna namun dengan FPR yang tidak realistik (1.000).
- **FALSE POSITIVE RATE (FPR)** : Dicapai oleh model LuNet (0.0105), menjadikannya terbaik untuk menghindari "alarm palsu".

## MULTICLASS

- **ACCURACY** : Tetap dicapai oleh model CNN (0.7383), meskipun dengan skor yang lebih rendah dari binary.
- **KESEIMBANGAN TERBAIK** : Model CNN+RNN menawarkan keseimbangan yang baik antara FPR terendah dan Tingkat Deteksi tertinggi di kategori ini.
- **KESIMPULAN** : Model CNN dan CNN+RNN terbukti lebih mampu menangani kompleksitas tugas multiclass.

# ANALISIS HASIL

Analisis menunjukkan tidak ada ukuran batch yang unggul secara universal. Pilihan batch size yang optimal sangat bergantung pada arsitektur model dan metrik apa yang menjadi prioritas.

- **BATCH 16**

Secara umum, batch size 16 cenderung menghasilkan model dengan kinerja yang lebih stabil dan akurasi lebih tinggi, terutama untuk arsitektur **CNN** dan **RNN**.

- **BATCH 8**

Penggunaan ukuran batch pelatihan 8 menunjukkan peran yang sangat terspesialisasi, di mana ia mampu menghasilkan kinerja optimal untuk konfigurasi model dan tujuan metrik yang sangat spesifik. Hal ini terbukti pada model LuNet dalam tugas biner, yang berhasil mencapai False Positive Rate (FPR) terendah secara absolut (0.0105), menjadikannya pilihan superior untuk meminimalkan kesalahan deteksi. Selain itu, model CNN pada tugas multiclass juga mencapai akurasi tertinggi (0.7383) saat menggunakan ukuran batch ini.

Namun, pendekatan ini tidak efektif secara universal, karena pada model lain seperti RNN dan CNN+RNN, penggunaan batch size 8 justru berdampak negatif dengan memperburuk kinerja dan meningkatkan nilai FPR.

# KESIMPULAN

- Secara umum, semua model menunjukkan kinerja terbaik mereka saat menangani tugas klasifikasi biner. Kinerja semua model menurun secara signifikan ketika beralih ke tugas multiclass yang lebih kompleks.
- Model LuNet adalah pemenang mutlak dalam kategori kelas biner.
- Keunggulan paling menonjol dari LuNet adalah kemampuannya mencapai nilai FPR terendah secara absolut di seluruh studi, yaitu 0.0105. Ini dicapai pada konfigurasi Target Biner dengan Batch Size 8.
- Tidak ada ukuran batch size yang "terbaik" secara universal; pilihan yang tepat sangat bergantung pada arsitektur model dan tujuan akhir.