

**TUGAS UAS DEEP LEARNING**

**“Pengembangan Sistem Deteksi Anomali pada Data Sensor IoT Menggunakan  
Arsitektur Hybrid CNN-LSTM”**



**Disusun Oleh:**

- |                          |           |
|--------------------------|-----------|
| 1. Atika Oktaviani       | G1A022020 |
| 2. Esa Nirza Zakya Putri | G1A022036 |
| 3. Neli Agustin          | G1A022048 |

**Dosen Pengampu:**

Ir. Arie Vatresia, S.T., M.T.I., P.hD., IPP.

**PROGRAM STUDI INFORMATIKA  
FAKULTAS TEKNIS  
UNIVERSITAS BENGKULU  
2025**

## **ABSTRAK**

Internet of Things (IoT) telah mengalami pertumbuhan yang signifikan dan telah diterapkan di berbagai sektor seperti industri, kota pintar, dan pemantauan lingkungan. Perkembangan ini mengakibatkan adanya peningkatan jumlah data sensor yang dihasilkan secara terus-menerus dalam format deret waktu. Data dari sensor IoT rentan terhadap keanehan yang dapat timbul karena kerusakan sensor, gangguan jaringan, kondisi lingkungan yang ekstrim, atau keadaan operasional yang tidak biasa. Keanehan yang tidak teridentifikasi dapat berpotensi menyebabkan kesalahan dalam analisis dan kegagalan sistem, sehingga penting untuk memiliki metode deteksi anomali yang akurat dan terpercaya.

Tujuan dari penelitian ini adalah untuk merancang sistem yang dapat mendeteksi anomali pada data sensor IoT dengan menggunakan arsitektur pembelajaran dalam yang menggabungkan Convolutional Neural Network (CNN) dan Long Short-Term Memory (LSTM). CNN berfungsi untuk mengambil fitur lokal dari data sensor yang telah dipisahkan dalam jendela temporal, sementara LSTM digunakan untuk memodelkan ketergantungan waktu yang terjadi dalam jangka pendek dan panjang pada data deret waktu. Dataset yang digunakan dalam studi ini adalah IoT-T Sensor Dataset for Anomaly Detection yang diambil dari Kaggle.

Proses penelitian mencakup langkah-langkah seperti pemrosesan awal data, normalisasi, pembuatan jendela temporal, desain dan pelatihan model CNN-LSTM, serta penilaian kinerja model. Penilaian dilakukan dengan menggunakan metrik seperti akurasi, presisi, recall, F1-score, dan ROC-AUC, serta dibandingkan dengan metode dasar seperti Support Vector Machine (SVM) dan Autoencoder. Hasil dari eksperimen menunjukkan bahwa model hybrid CNN-LSTM menunjukkan kinerja yang lebih baik, terutama dalam mendeteksi data anomali, dibandingkan dengan model dasar. Oleh karena itu, pendekatan yang diajukan memiliki potensi sebagai solusi yang efektif untuk meningkatkan keandalan sistem IoT.

## **BAB I**

### **LATAR BELAKANG**

Perkembangan teknologi Internet of Things (IoT) dalam beberapa tahun terakhir telah mengalami kemajuan yang luar biasa dan telah diterapkan di berbagai bidang kehidupan. IoT memungkinkan sejumlah perangkat fisik yang dilengkapi dengan sensor, aktuator, dan fungsi komunikasi untuk terhubung satu sama lain dan bertukar informasi secara otomatis melalui jaringan internet. Implementasi IoT banyak ditemukan dalam industri manufaktur, pengawasan lingkungan, sistem transportasi, pertanian cerdas, hingga bidang kesehatan. Dalam sistem ini, sensor berfungsi sebagai elemen kunci yang bertugas mengumpulkan data secara terus menerus untuk menggambarkan kondisi sebenarnya dari suatu objek atau lingkungan (Al-kahtani et al., 2022).

Data yang dihasilkan oleh sensor IoT biasanya berbentuk deret waktu (time series) dengan volume yang sangat besar dan bersifat real-time. Ciri-ciri data dari sensor IoT yang bersifat kontinu, dinamis, dan terikat pada waktu menjadikan proses analisis data sebagai tantangan tersendiri. Di samping itu, data sensor sering kali mengandung gangguan (noise), fluktuasi nilai yang tidak terduga, serta ketidakseimbangan distribusi kelas antara kondisi normal dan abnormal. Hal ini mendesak perlunya adanya sistem analisis data yang dapat beroperasi secara otomatis, adaptif, dan memiliki tingkat akurasi yang tinggi(Nishchemenko & Volochchuk, 2025).

Salah satu tantangan utama dalam mengelola data dari sensor IoT adalah keberadaan anomali. Anomali dalam data sensor diartikan sebagai pola atau angka yang secara mencolok berbeda dari perilaku normal suatu sistem. Berbagai faktor bisa menyebabkan munculnya anomali, termasuk kerusakan pada perangkat sensor, gangguan pada komunikasi jaringan, kesalahan dalam konfigurasi, serta perubahan cuaca yang ekstrem. Jika anomali ini tidak teridentifikasi dengan baik, sistem IoT berpotensi memproduksi informasi yang salah, yang dapat menurunkan efisiensi sistem secara keseluruhan, dan dalam beberapa situasi dapat mengakibatkan kerugian yang besar, terutama pada sistem yang krusial seperti industri dan pengawasan keselamatan(Yang et al., 2025).

Metode deteksi anomali tradisional biasanya menggunakan pendekatan berbasis aturan atau teknik statistik yang sederhana, seperti batas ambang. Meskipun metode tersebut mudah untuk diterapkan, pendekatan ini memiliki kekurangan dalam menangani data sensor IoT yang kompleks, nonlinier, serta dimensinya yang tinggi. Di samping itu, metode tradisional

umumnya kurang adaptif dan sulit mengikuti perubahan pola data yang terjadi sejalan dengan waktu. Maka dari itu, metode ini sering kali menghasilkan tingkat kesalahan deteksi yang tinggi bila diterapkan pada sistem IoT berskala besar(Bautina, 2025).

Seiring dengan kemajuan teknologi pembelajaran mesin dan pembelajaran mendalam, berbagai metode berbasis jaringan saraf telah diadopsi secara luas untuk menyelesaikan masalah deteksi anomali. Teknik pembelajaran mendalam dapat secara otomatis mempelajari pola yang kompleks langsung dari data tanpa perlu membuat fitur secara manual. Jaringan Saraf Konvolusional (CNN) terbukti efektif dalam mengambil fitur lokal dan pola spasial dari data, sedangkan Jaringan Saraf Berulang (RNN), terutama Long Short-Term Memory (LSTM), lebih unggul dalam menangkap ketergantungan temporal pada data deret waktu(Zhou et al., 2016).

Namun, penerapan CNN atau LSTM secara sendiri-sendiri tetap memiliki sejumlah kelemahan. CNN tidak cukup efisien dalam menangkap hubungan jangka panjang di antara waktu, sedangkan LSTM cenderung kurang efektif dalam secara otomatis mengambil fitur lokal. Untuk mengatasi permasalahan tersebut, struktur hibrida CNN-LSTM dikembangkan dengan memadukan keunggulan kedua metode tersebut. CNN digunakan untuk mengambil fitur lokal dari bagian data sensor, sedangkan LSTM difungsikan untuk memahami dinamika temporal dan ketergantungan waktu dengan lebih mendalam(Polat et al., 2025).

Implementasi struktur hibrida CNN-LSTM pada sistem deteksi anomali diharapkan dapat meningkatkan akurasi, ketahanan, dan konsistensi sistem dalam mengidentifikasi pola anomali pada data sensor IoT. Selain itu, model yang efisien juga harus memperhatikan aspek penerapan pada perangkat edge IoT yang memiliki keterbatasan dalam daya komputasi. Dengan demikian, pengoptimalan model dan konversi ke format yang lebih ringan menjadi hal yang sangat penting dalam penelitian ini(Nazir et al., 2024).

Berdasarkan penjelasan tersebut, kajian ini berfokus pada pembuatan sistem untuk mendeteksi anomali dalam data sensor IoT dengan memanfaatkan arsitektur campuran CNN-LSTM. Dataset yang digunakan adalah IoT-T Sensor Dataset for Anomaly Detection, yang menawarkan data sensor yang telah diberi label normal dan anomali. Diharapkan penelitian ini dapat menghasilkan model deteksi anomali yang tepat, efisien, dan dapat diterapkan dalam kondisi nyata di lingkungan IoT, serta memberikan sumbangsih dalam pengembangan metode deteksi anomali yang berlandaskan deep learning(Nizam et al., 2022).

## **BAB II**

### **METODOLOGI**

Metodologi dalam penelitian ini dirancang untuk menciptakan sistem pendekripsi anomali pada data sensor Internet of Things (IoT) dengan memanfaatkan arsitektur gabungan antara Convolutional Neural Network (CNN) dan Long Short-Term Memory (LSTM). Pendekatan yang diambil berfokus pada pemanfaatan kekuatan CNN dalam mengambil fitur lokal dari data sensor dan memanfaatkan kemampuan LSTM untuk memodelkan hubungan temporal dalam data yang berurutan. Proses penelitian mengikuti langkah-langkah dalam pengembangan model pembelajaran mendalam secara sistematis dan terstruktur, dimulai dari penentuan isu, pemrosesan data, desain model, penerapan, evaluasi, sampai optimasi dan pengaplikasian model agar dapat diterapkan di dunia nyata.

#### **1. Identifikasi Masalah dan Pemilihan Dataset**

Langkah pertama dalam penelitian ini adalah menentukan isu yang perlu diatasi, yaitu mengidentifikasi anomali dalam data sensor IoT yang bersifat deret waktu. Anomali yang terdapat dalam data sensor adalah keadaan yang berbeda dari pola normal yang bisa terjadi akibat berbagai hal, seperti kerusakan pada alat sensor, gangguan dari lingkungan, kesalahan dalam transmisi data, serta kondisi operasional yang tidak lazim. Jika anomali ini tidak terdeteksi dengan baik, dapat berdampak pada menurunnya keandalan sistem IoT dan menghasilkan informasi yang keliru.

Dataset yang digunakan untuk penelitian ini adalah IoT-T Sensor Dataset for Anomaly Detection yang diambil dari platform Kaggle (<https://www.kaggle.com/datasets/diaealaouisoulimani/iot-t-sensor-dataset-for-anomaly-detection>). Dataset ini adalah dataset terbuka yang sudah dilengkapi dengan label untuk kondisi normal dan anomali, memungkinkan penerapan pendekatan pembelajaran terawasi. Dalam penelitian ini, satu jenis data sensor dipilih sebagai studi kasus, yaitu data sensor suhu, karena sensor suhu memiliki pola perubahan nilai yang berkelanjutan, bersifat temporal, dan sering digunakan dalam sistem IoT di berbagai sektor seperti industri, lingkungan, dan kesehatan.

#### **2. Pra-pemrosesan dan Eksplorasi Data**

Proses pra-pemrosesan data bertujuan untuk memastikan bahwa data yang digunakan memiliki standar kualitas tinggi dan siap digunakan dalam pelatihan model deep learning. Tahapan ini mencakup pembersihan data, menjelajahi karakteristik dataset, mengkonversi fitur kategorikal, serta melakukan normalisasi data.

### 1) Pembersihan Data

Pembersihan data dilakukan dengan mengecek adanya data yang terduplikasi dan nilai yang hilang. Data yang terduplikasi dihapus untuk mencegah pengulangan informasi yang bisa memengaruhi distribusi data, sedangkan nilai hilang diatasi agar tidak menyebabkan masalah selama pelatihan model.

### 2) Eksplorasi Data

Eksplorasi data dilakukan agar dapat memahami karakteristik dataset secara keseluruhan. Analisis distribusi kelas memberikan gambaran tentang proporsi antara data normal dan data anomali. Di samping itu, visualisasi nilai sensor digunakan untuk melihat perbedaan pola antara kondisi normal dan anomali. Hasil dari eksplorasi ini menunjukkan adanya ketidakseimbangan kelas, dengan data normal lebih banyak dibandingkan data anomali, serta perbedaan distribusi nilai sensor yang mencolok antara kedua kondisi tersebut.

### 3) Encoding Fitur Kategorikal

Fitur kategorikal dalam dataset diubah menjadi format numerik menggunakan teknik encoding agar dapat diproses oleh model deep learning. Tahapan ini sangat penting karena model CNN-LSTM hanya menerima input dalam bentuk angka.

### 4) Normalisasi Data

Semua fitur numerik dinormalisasi menggunakan metode StandardScaler sehingga setiap fitur memiliki nilai rata-rata nol dan deviasi standar satu. Normalisasi ini bertujuan untuk menghindari dominasi dari fitur tertentu, meningkatkan stabilitas numerik, dan mempercepat proses konvergensi saat pelatihan model.

## 3. Segmentasi Data menjadi Window Temporal

Setelah proses normalisasi, data deret waktu dipisahkan menggunakan metode sliding window. Panjang jendela yang digunakan dalam studi ini ditetapkan sebanyak 20 timestep, yang diambil untuk mendeteksi pola perubahan nilai dari sensor dalam periode tertentu tanpa kehilangan konteks temporal yang esensial.

Tahapan pengelompokan ini menghasilkan data dalam format tiga dimensi, mencakup total sampel, jumlah timestep, dan jumlah fitur. Representasi data dalam tiga dimensi ini penting agar data dapat digunakan sebagai input dalam arsitektur CNN-LSTM. Setiap jendela diberikan label berdasarkan kondisi yang paling dominan dalam jendela tersebut, apakah normal atau anomali, agar sesuai dengan pendekatan klasifikasi yang berlandaskan supervised learning.

## 4. Perancangan Arsitektur Model Hybrid CNN-LSTM

Model yang dirancang dalam penelitian ini memanfaatkan arsitektur kombinasi CNN-LSTM untuk mengambil manfaat dari keunggulan masing-masing metode. CNN satu dimensi (1D-CNN) diaplikasikan pada tahap awal sebagai ekstraktor fitur untuk mengeluarkan fitur lokal atau spasial dari setiap jendela data sensor, seperti pola fluktuasi nilai dalam waktu singkat.

Hasil keluaran dari lapisan CNN kemudian diteruskan ke lapisan LSTM yang bertugas untuk memodelkan ketergantungan temporal antar timestep, baik dalam skala pendek maupun panjang. Melalui kombinasi ini, model diharapkan dapat mengidentifikasi pola-pola kompleks pada data sensor IoT. Untuk meningkatkan kinerja model, dilakukan percobaan terhadap variasi arsitektur, seperti jumlah lapisan konvolusi, ukuran kernel, jumlah unit LSTM, serta penerapan dropout sebagai teknik regularisasi untuk menghindari overfitting.

## 5. Implementasi Model dan Teknik Regularisasi

Pelaksanaan model dilakukan dengan memanfaatkan bahasa pemrograman Python serta framework TensorFlow. Model ini dirancang secara modular agar lebih mudah untuk diperluas, diuji, dan dimodifikasi. Untuk meningkatkan efektivitas proses pelatihan, diterapkan callback Early Stopping yang bertujuan untuk menghentikan pelatihan ketika tidak ada peningkatan performa pada data validasi dalam beberapa epoch tertentu.

Selain itu, Model Checkpoint digunakan untuk menyimpan model dengan kinerja terbaik sepanjang proses pelatihan. Dengan cara ini, model yang digunakan dalam tahap evaluasi adalah model yang menunjukkan performa paling optimal berdasarkan data validasi.

## 6. Pelatihan dan Evaluasi Model

Dataset dibagi menjadi tiga bagian: data latih, data validasi, dan data uji dengan proporsi masing-masing 70%, 15%, dan 15%. Pembagian ini bertujuan untuk memastikan bahwa penilaian kinerja model dilakukan pada data yang belum pernah diperkenalkan sebelumnya.

Proses pelatihan dilakukan dengan menggunakan optimizer Adam dan fungsi loss binary cross-entropy, yang cocok untuk isu klasifikasi biner. Penilaian kinerja model dilakukan dengan beberapa metrik, yaitu akurasi, presisi, recall, F1-score, dan ROC-AUC. Metrik-metrik ini dipilih untuk memberikan gambaran menyeluruh tentang kemampuan model dalam mengidentifikasi anomali, terutama dalam kondisi ketidakseimbangan kelas. Lebih lanjut, performa model CNN-LSTM dibandingkan dengan metode dasar, seperti

Support Vector Machine (SVM) dan Autoencoder, untuk menunjukkan keunggulan pendekatan yang diusulkan.

## 7. Optimasi dan Deployment Model

Untuk meningkatkan kinerja model, dilakukan penyesuaian hyperparameter dengan menggunakan teknik pencarian acak. Parameter yang disesuaikan mencakup jumlah filter pada CNN, ukuran kernel, jumlah unit LSTM, nilai dropout, dan juga learning rate. Proses penyesuaian ini bertujuan untuk mendapatkan kombinasi parameter yang memberikan hasil optimal.

Model yang paling baik hasil penyesuaian tersebut lalu diekspor ke format TensorFlow Lite agar dapat dioperasikan pada perangkat edge IoT yang memiliki batasan dalam hal sumber daya komputasi. Tahap penerapan ini bertujuan untuk memastikan bahwa model yang dikembangkan tidak hanya unggul di bidang akademis, tetapi juga bisa diimplementasikan secara efektif dalam sistem IoT.

## BAB III

### HASIL DAN PEMBAHASAN

#### A. Deskripsi Dataset

Dataset yang digunakan pada penelitian ini diperoleh dari platform Kaggle dengan judul Environmental Sensor Data, yang tersedia pada tautan: <https://www.kaggle.com/datasets/garystafford/environmental-sensor-data-132k>. Dataset ini bersifat open dataset dengan lisensi CC0 1.0, sehingga dapat digunakan untuk keperluan penelitian.

Dataset dimuat ke dalam lingkungan pemrograman Python menggunakan pustaka Pandas dan disimpan dalam bentuk DataFrame untuk memudahkan proses analisis dan pemrosesan data selanjutnya.

Dataset terdiri dari 405.184 baris data dengan 9 atribut, yang meliputi data sensor lingkungan dan informasi perangkat IoT. Tipe data yang digunakan mencakup float, boolean, dan object. Dataset tidak memiliki nilai kosong (missing value).

Atribut	Tipe Data	Keterangan
ts	Float	Timestamp
device	Object	ID perangkat IoT
temp	Float	Suhu
humidity	Float	Kelembaban
co	Float	Karbon monoksida
lpg	Float	Gas LPG
smoke	Float	Asap
light	Boolean	Sensor cahaya
motion	Boolean	Sensor gerak

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 405184 entries, 0 to 405183
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype  
 --- 
 0   ts          405184 non-null   float64
 1   device      405184 non-null   object 
 2   co          405184 non-null   float64
 3   humidity    405184 non-null   float64
 4   light       405184 non-null   bool   
 5   lpg         405184 non-null   float64
 6   motion      405184 non-null   bool   
 7   smoke       405184 non-null   float64
 8   temp        405184 non-null   float64
dtypes: bool(2), float64(6), object(1)
memory usage: 22.4+ MB
```

Berdasarkan statistik deskriptif, nilai sensor suhu, kelembaban, dan gas menunjukkan rentang yang cukup bervariasi. Variasi ini mengindikasikan adanya kemungkinan kondisi ekstrem yang berpotensi dikategorikan sebagai anomali.

	ts	co	humidity	lpg	smoke	temp
count	4.051840e+05	405184.000000	405184.000000	405184.000000	405184.000000	405184.000000
mean	1.594858e+09	0.004639	60.511694	0.007237	0.019264	22.453987
std	1.994984e+05	0.001250	11.366489	0.001444	0.004086	2.698347
min	1.594512e+09	0.001171	1.100000	0.002693	0.006692	0.000000
25%	1.594686e+09	0.003919	51.000000	0.006456	0.017024	19.900000
50%	1.594858e+09	0.004812	54.900000	0.007489	0.019950	22.200000
75%	1.595031e+09	0.005409	74.300003	0.008150	0.021838	23.600000
max	1.595203e+09	0.014420	99.900002	0.016567	0.046590	30.600000

## B. Pra Pemrosesan Data

Tahap pra-pemrosesan data dilakukan untuk memastikan kualitas data sebelum digunakan pada proses pelatihan model CNN-LSTM. Pra-pemrosesan bertujuan untuk mengurangi noise, menangani ketidak sempurnaan data, serta menyesuaikan format data agar sesuai dengan kebutuhan model deep learning berbasis deret waktu.

Tahapan pra-pemrosesan yang dilakukan pada penelitian ini meliputi seleksi fitur sensor, penanganan nilai hilang, pelabelan anomali, normalisasi data, serta pembentukan data deret waktu (*time-series windowing*).

### 1. Seleksi Fitur Sensor

Pada tahap ini dilakukan pemilihan atribut sensor yang relevan dengan tujuan deteksi anomali lingkungan. Dari seluruh atribut yang tersedia pada dataset, dipilih lima fitur sensor utama, yaitu suhu (temperature), kelembaban (humidity), karbon monoksida (CO), gas LPG, dan asap (smoke). Pemilihan fitur ini didasarkan pada keterkaitannya dengan kondisi lingkungan yang berpotensi menunjukkan adanya anomali.

### 2. Penanganan Nilai Hilang (*Handling Missing Values*)

Dataset diperiksa untuk mendeteksi keberadaan nilai hilang (*missing values*). Apabila ditemukan nilai kosong, maka dilakukan penanganan menggunakan metode *forward fill*, yaitu menggantikan nilai yang hilang dengan nilai terakhir yang tersedia sebelumnya. Metode ini dipilih karena data sensor bersifat kontinu dan berurutan secara waktu.

**Source Code :**

```
df_sensor = df_sensor.fillna(method="ffill")
```

**Output :**

```
/tmp/ipython-input-885939362.py:1: FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffill() or obj.bfill() instead.
```

```
df_sensor = df_sensor.fillna(method="ffill")
```

Peringatan tersebut menunjukkan bahwa metode masih berjalan dengan baik, meskipun pada versi pustaka mendatang disarankan menggunakan fungsi ffill() secara langsung.

### 3. Pelabelan Anomali Menggunakan Metode Z-Score

Karena dataset tidak menyediakan label anomali secara eksplisit, maka dilakukan proses pelabelan anomali buatan menggunakan metode Z-Score. Z-Score digunakan untuk mengukur seberapa jauh suatu nilai menyimpang dari nilai rata-rata dalam satuan standar deviasi.

Sebuah data dikategorikan sebagai anomali apabila terdapat minimal satu fitur sensor yang memiliki nilai Z-Score dengan  $|z| > 3$ . Jika kondisi tersebut terpenuhi, maka data diberi label 1 (anomali), sedangkan data lainnya diberi label 0 (normal).

Pendekatan ini memungkinkan identifikasi data ekstrem yang berpotensi merepresentasikan kondisi abnormal pada lingkungan sensor IoT.

#### Source Code dan Output :

```
● from scipy.stats import zscore
z_scores = np.abs(zscore(df_sensor))

df["anomaly"] = (z_scores > 3).any(axis=1).astype(int)

df["anomaly"].value_counts()

count
anomaly
    0    401736
    1     3448
dtype: int64
```

## C. Eksplorasi Data

Tahap eksplorasi data dilakukan untuk memahami karakteristik dataset, khususnya distribusi kelas anomali serta pola hubungan antara data sensor dan kejadian anomali. Proses EDA ini bertujuan untuk memberikan gambaran awal mengenai kondisi data sebelum dilakukan proses pemodelan lebih lanjut.

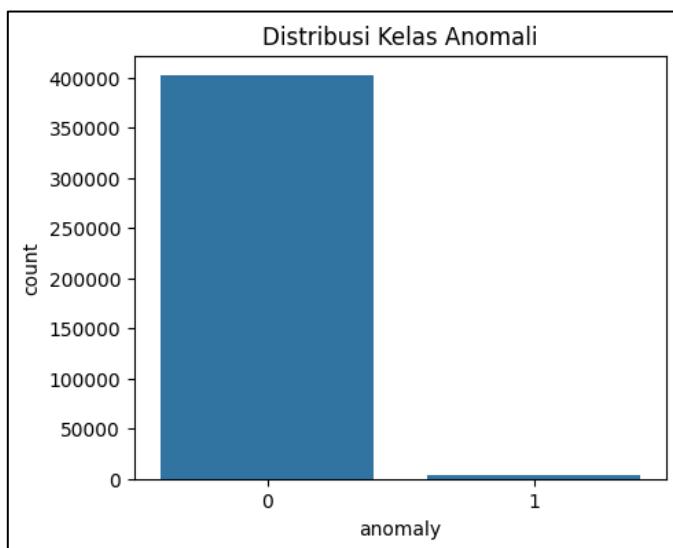
### 1. Distribusi Kelas Anomali

Pada tahap awal EDA, dilakukan visualisasi distribusi kelas anomali menggunakan grafik batang. Visualisasi ini digunakan untuk mengetahui proporsi data normal dan data anomali dalam dataset. Hasil visualisasi menunjukkan bahwa jumlah data anomali lebih sedikit dibandingkan data normal, sehingga dataset tergolong *imbalanced*.

**Source Code :**

```
plt.figure(figsize=(5,4))
sns.countplot(x="anomaly", data=df)
plt.title("Distribusi Kelas Anomali")
plt.show()
```

**Output :**



Grafik "Distribusi Kelas Anomali" menunjukkan bahwa himpunan data yang digunakan untuk pelatihan model memiliki ketidakseimbangan yang ekstrem. Mayoritas data, sekitar 400.000 sampel, termasuk dalam Kelas Normal (0). Sebaliknya, Kelas Anomali (1) hanya terdiri dari sejumlah kecil data, yaitu kurang dari 5.000 sampel. Kondisi ini, yang dikenal sebagai *imbalanced classification*, sangat penting untuk dipahami karena akurasi model secara keseluruhan akan cenderung tinggi—sekitar 90% atau lebih—hanya dengan memprediksi setiap kasus sebagai "Normal". Oleh karena itu, metrik yang lebih spesifik seperti *Precision* dan *Recall* untuk kelas Anomali menjadi indikator kinerja model yang jauh lebih penting daripada akurasi sederhana.

## 2. Visualisasi Sensor Terhadap Anomali

Selanjutnya dilakukan visualisasi hubungan antara data sensor dan kejadian anomali. Pada penelitian ini, sensor suhu (*temperature*) digunakan sebagai contoh untuk menampilkan pola perubahan nilai sensor terhadap waktu. Titik-titik anomali

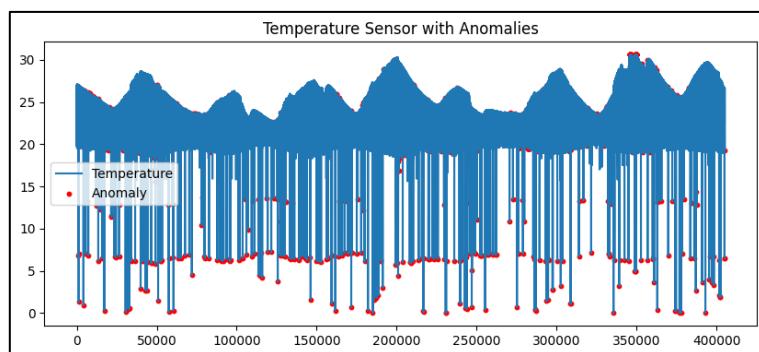
ditandai dengan warna merah untuk memperjelas perbedaan antara kondisi normal dan anomali.

Visualisasi ini bertujuan untuk menunjukkan bahwa anomali umumnya muncul sebagai nilai yang menyimpang secara signifikan dari pola normal sensor.

### Source Code :

```
plt.figure(figsize=(10,4))
plt.plot(df["temp"], label="Temperature")
plt.scatter(
    df.index[df["anomaly"] == 1],
    df["temp"][df["anomaly"] == 1],
    color="red", label="Anomaly", s=10
)
plt.legend()
plt.title("Temperature Sensor with Anomalies")
plt.show()
```

### Output :



Grafik sensor suhu menunjukkan pola data normal yang relatif stabil dengan fluktuasi periodik, sementara titik-titik anomali ditandai dengan penyimpangan nilai suhu yang signifikan dari pola normal tersebut. Anomali terlihat muncul secara sporadis pada berbagai rentang waktu, baik dalam bentuk penurunan maupun kenaikan suhu yang ekstrem. Visualisasi ini memperlihatkan bahwa model mampu mengidentifikasi kejadian-kejadian yang tidak wajar pada data sensor suhu, sehingga membantu dalam pemantauan kondisi sistem IoT secara dini dan mencegah potensi gangguan atau kerusakan yang lebih serius.

## D. Persiapan Data

Persiapan data adalah tahap pengolahan data yang dilakukan setelah eksplorasi data untuk menyiapkan data agar siap digunakan dalam proses pemodelan. Pada tahap ini, data disesuaikan dengan kebutuhan model, baik dari segi skala, bentuk, maupun pembagian data, sehingga proses pelatihan dan evaluasi model dapat berjalan dengan baik.

### 1. Normalisasi Data

Setelah tahap eksplorasi data, dilakukan proses normalisasi terhadap fitur sensor. Normalisasi bertujuan untuk menyamakan skala antar fitur sehingga tidak terjadi dominasi fitur tertentu akibat perbedaan rentang nilai. Metode yang digunakan adalah *StandardScaler*, yang mengubah data agar memiliki rata-rata nol dan standar deviasi satu.

```
from sklearn.preprocessing import StandardScaler

X = df[sensor_features].values
y = df["anomaly"].values

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

### 2. Pembentukan Data Deret Waktu (*Time-Series Windowing*)

Karena data sensor bersifat deret waktu, dilakukan pembentukan sekuens data menggunakan metode sliding window. Setiap sampel data dibentuk dari 30 data berturut-turut (window size = 30). Label pada setiap sekuens diambil dari kondisi anomali setelah window tersebut. Tahap ini bertujuan agar model dapat mempelajari pola temporal dari perubahan nilai sensor.

**Source Code :**

```
def create_sequences(X, y, window_size=30):
    X_seq, y_seq = [], []
    for i in range(len(X) - window_size):
        X_seq.append(X[i:i+window_size])
        y_seq.append(y[i+window_size])
```

```
return np.array(X_seq), np.array(y_seq)
WINDOW_SIZE = 30
X_seq, y_seq = create_sequences(X_scaled, y, WINDOW_SIZE)
X_seq.shape, y_seq.shape
```

#### Output :

```
((405154, 30, 5), (405154,))
```

Output diatas menunjukkan bentuk data tiga dimensi (jumlah\_sampel,window\_size, jumlah\_fitur) yang sesuai dengan input CNN-LSTM.

### 3. Pembagian Data (*Data Splitting*)

Data yang telah dibentuk menjadi sekuens kemudian dibagi menjadi tiga bagian, yaitu data latih (*training*), data validasi (*validation*), dan data uji (*testing*) dengan rasio 70% : 15% : 15%. Pembagian data dilakukan secara stratified untuk menjaga proporsi kelas anomali pada setiap subset data.

#### Source Code :

```
from sklearn.model_selection import train_test_split
X_train, X_temp, y_train, y_temp = train_test_split(
    X_seq, y_seq,
    test_size=0.30,
    random_state=42,
    stratify=y_seq
)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp,
    test_size=0.50,
    random_state=42,
    stratify=y_temp
)
```

### 4. Penanganan Ketidakseimbangan Data (*Class Weight*)

Berdasarkan hasil EDA, dataset memiliki distribusi kelas yang tidak seimbang antara data normal dan data anomali. Oleh karena itu, digunakan metode *class*

*weight* untuk memberikan bobot yang lebih besar pada kelas minoritas (anomali) selama proses pelatihan model. Pendekatan ini bertujuan untuk meningkatkan kemampuan model dalam mendeteksi data anomali.

#### Source Code :

```
from sklearn.utils.class_weight import compute_class_weight  
weights = compute_class_weight(  
    class_weight="balanced",  
    classes=np.unique(y_train),  
    y=y_train  
)  
  
class_weight = {  
    0: weights[0],  
    1: weights[1]  
}  
class_weight
```

#### Output :

```
{0: np.float64(0.5042924254871209),np.float64(58.74212924606462)}
```

Output diatas menampilkan bobot kelas normal dan anomali yang digunakan saat pelatihan model.

## E. Pemodelan (Modelling)

Tahap pemodelan bertujuan untuk membangun model deteksi anomali berbasis deep learning dan membandingkannya dengan beberapa model baseline. Model utama yang digunakan adalah CNN-LSTM, yang mampu mengekstraksi pola spasial dan temporal dari data sensor berbentuk deret waktu.

### 1. Arsitektur Model CNN-LSTM

Model CNN-LSTM dibangun dengan mengombinasikan *Convolutional Neural Network (CNN)* dan *Long Short-Term Memory (LSTM)*. CNN digunakan untuk mengekstraksi fitur lokal dari data time-series, sedangkan LSTM digunakan untuk menangkap ketergantungan temporal jangka panjang.

#### Source Code :

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, LSTM
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization

model = Sequential([
    Conv1D(64, 3, activation="relu",
           input_shape=(X_train.shape[1], X_train.shape[2])),
    BatchNormalization(),
    MaxPooling1D(2),

    Conv1D(128, 3, activation="relu"),
    BatchNormalization(),
    MaxPooling1D(2),

    LSTM(64),
    Dropout(0.5),

    Dense(64, activation="relu"),
    Dropout(0.3),

    Dense(1, activation="sigmoid")
])
```

## 2. Kompilasi Model dengan Focal Loss

Karena data bersifat tidak seimbang (imbalanced) antara kelas normal dan anomali, digunakan Binary Focal Loss. Focal loss memberikan fokus lebih besar pada data yang sulit diklasifikasikan sehingga meningkatkan performa pada kelas minoritas.

### Source Code :

```
import tensorflow as tf
from tensorflow.keras import backend as K

def binary_focal_loss(alpha=0.75, gamma=2.0):
    def loss(y_true, y_pred):
        y_true = tf.cast(y_true, tf.float32)
```

```

y_pred = tf.clip_by_value(y_pred, K.epsilon(), 1 - K.epsilon())
p_t = y_true * y_pred + (1 - y_true) * (1 - y_pred)
return tf.reduce_mean(
    -alpha * tf.pow(1 - p_t, gamma) * tf.math.log(p_t)
)
return loss

model.compile(
    optimizer="adam",
    loss=binary_focal_loss(),
    metrics=["accuracy"]
)

model.summary()

```

## Output :

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 28, 64)	1,024
batch_normalization (BatchNormalization)	(None, 28, 64)	256
max_pooling1d (MaxPooling1D)	(None, 14, 64)	0
conv1d_1 (Conv1D)	(None, 12, 128)	24,704
batch_normalization_1 (BatchNormalization)	(None, 12, 128)	512
max_pooling1d_1 (MaxPooling1D)	(None, 6, 128)	0
lstm (LSTM)	(None, 64)	49,408
dropout (Dropout)	(None, 64)	0
dense (Dense)	(None, 64)	4,160
dropout_1 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 1)	65

Total params: 80,129 (313.00 KB)

Trainable params: 79,745 (311.50 KB)

**Non-trainable params:** 384 (1.50 KB)

Arsitektur model CNN-LSTM yang digunakan terdiri dari kombinasi lapisan Convolutional Neural Network dan Long Short-Term Memory untuk mengekstraksi fitur spasial dan temporal dari data deret waktu IoT. Lapisan Conv1D dan MaxPooling1D berfungsi untuk mengekstraksi pola lokal dan mereduksi dimensi data, sementara Batch Normalization membantu menstabilkan dan mempercepat proses pelatihan. Selanjutnya, lapisan LSTM digunakan untuk menangkap ketergantungan temporal pada data hasil ekstraksi CNN. Lapisan Dense dan Dropout berperan dalam proses klasifikasi serta mengurangi risiko overfitting. Secara keseluruhan, model memiliki 80.129 parameter dengan mayoritas bersifat trainable, menunjukkan arsitektur yang relatif ringan namun cukup kompleks untuk mendukung kinerja deteksi anomalai IoT secara efektif.

### 3. Proses Pelatihan Model

Pelatihan model dilakukan dengan mekanisme early stopping untuk mencegah overfitting dan model checkpoint untuk menyimpan model terbaik berdasarkan performa validasi.

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

```
callbacks = [  
    EarlyStopping(patience=5, restore_best_weights=True),  
    ModelCheckpoint("cnn_lstm_iot.h5", save_best_only=True)  
]
```

```
history = model.fit(  
    X_train, y_train,  
    validation_data=(X_val, y_val),  
    epochs=40,  
    batch_size=128,  
    class_weight=class_weight,  
    callbacks=callbacks  
)
```

## Output :

```
Epoch 1/40
2216/2216 _____ 0s 9ms/step - accuracy: 0.9894
- loss: 0.0114
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
2216/2216 _____ 30s 10ms/step - accuracy: 0.9894 - loss: 0.0114 - val_accuracy: 0.9915 - val_loss: 0.0073
Epoch 2/40
2213/2216 _____ 0s 9ms/step - accuracy: 0.9914
- loss: 0.0086
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
2216/2216 _____ 22s 10ms/step - accuracy: 0.9914 - loss: 0.0086 - val_accuracy: 0.9908 - val_loss: 0.0062
Epoch 3/40
2216/2216 _____ 21s 9ms/step - accuracy: 0.9913 - loss: 0.0079 - val_accuracy: 0.9916 - val_loss: 0.0064
Epoch 4/40
2213/2216 _____ 0s 9ms/step - accuracy: 0.9914
- loss: 0.0077
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
2216/2216 _____ 22s 10ms/step - accuracy: 0.9914 - loss: 0.0077 - val_accuracy: 0.9918 - val_loss: 0.0055
Epoch 5/40
2216/2216 _____ 33s 15ms/step - accuracy: 0.9912 - loss: 0.0074 - val_accuracy: 0.9916 - val_loss: 0.0056
Epoch 6/40
2216/2216 _____ 0s 10ms/step - accuracy: 0.9912 - loss: 0.0072
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
2216/2216 _____ 32s 11ms/step - accuracy: 0.9912 - loss: 0.0072 - val_accuracy: 0.9918 - val_loss: 0.0051
Epoch 7/40
2216/2216 _____ 42s 19ms/step - accuracy: 0.9914 - loss: 0.0070 - val_accuracy: 0.9904 - val_loss: 0.0056
Epoch 8/40
2216/2216 _____ 67s 12ms/step - accuracy: 0.9913 - loss: 0.0068 - val_accuracy: 0.9912 - val_loss: 0.0057
Epoch 9/40
2216/2216 _____ 28s 13ms/step - accuracy: 0.9913 - loss: 0.0067 - val_accuracy: 0.9913 - val_loss: 0.0053
Epoch 10/40
```

```
2216/2216 ----- 41s 12ms/step - accuracy:  
0.9911 - loss: 0.0064 - val_accuracy: 0.9915 - val_loss: 0.0053  
Epoch 11/40  
2216/2216 ----- 29s 13ms/step - accuracy:  
0.9905 - loss: 0.0069 - val_accuracy: 0.9909 - val_loss: 0.0057
```

Proses pelatihan model CNN-LSTM dilakukan selama maksimal 40 epoch dengan batch size sebesar 128. Berdasarkan hasil pelatihan, model menunjukkan konvergensi yang stabil sejak awal epoch.

Pada epoch pertama, model mencapai akurasi pelatihan sebesar **98,94%** dengan nilai loss **0,0114**, sedangkan akurasi validasi mencapai **99,15%** dengan loss **0,0073**. Seiring bertambahnya epoch, nilai loss terus menurun dan akurasi cenderung stabil.

Hingga epoch ke-11, model mencapai akurasi pelatihan sebesar **99,05%** dengan loss **0,0069**, dan akurasi validasi sebesar **99,09%** dengan loss **0,0057**. Hal ini menunjukkan bahwa model mampu mempelajari pola data dengan baik tanpa mengalami overfitting yang signifikan, ditandai dengan selisih kecil antara akurasi pelatihan dan validasi.

Epoch	Accuracy Train	Loss Train	Accuracy Validasi	Loss Validasi
1	98,94%	0,0114	99,15%	0,0073
4	99,14%	0,0077	99,18%	0,0055
6	99,12%	0,0072	99,18%	0,0051
11	99,05%	0,0069	99,09%	0,0057

#### 4. Evaluasi Model CNN-LSTM

Evaluasi model dilakukan menggunakan metrik *Accuracy*, *Precision*, *Recall*, *F1-score*, dan *ROC-AUC*. Threshold klasifikasi ditetapkan sebesar 0.3 untuk meningkatkan sensitivitas terhadap anomali.

##### Source Code :

```
from sklearn.metrics import (  
    accuracy_score, precision_score, recall_score,  
    f1_score, roc_auc_score  
)  
  
y_prob = model.predict(X_test).ravel()
```

```
y_pred = (y_prob > 0.3).astype(int)

print("Accuracy :", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, zero_division=0))
print("Recall  :", recall_score(y_test, y_pred))
print("F1-score :", f1_score(y_test, y_pred))
print("ROC-AUC  :", roc_auc_score(y_test, y_prob))
```

#### Output :

```
1900/1900 ━━━━━━━━━━ 12s 6ms/step
Accuracy : 0.9798927172804159
Precision: 0.28173374613003094
Recall   : 0.8800773694390716
F1-score : 0.4268292682926829
ROC-AUC  : 0.9579411771031425
```

Berdasarkan hasil evaluasi kinerja model, diperoleh nilai akurasi sebesar 0,9799 yang menunjukkan bahwa model mampu mengklasifikasikan data secara keseluruhan dengan sangat baik. Nilai recall yang tinggi sebesar 0,8801 menandakan bahwa sebagian besar data anomali berhasil terdeteksi, sehingga model memiliki sensitivitas yang baik terhadap kejadian anomali. Namun, nilai precision yang relatif rendah sebesar 0,2817 menunjukkan masih adanya false positive yang cukup tinggi, yang berdampak pada nilai F1-score sebesar 0,4268. Meskipun demikian, nilai ROC-AUC sebesar 0,9579 mengindikasikan kemampuan model yang sangat baik dalam membedakan antara data normal dan anomali, sehingga model tetap efektif dan layak digunakan untuk deteksi anomali, khususnya pada data dengan ketidakseimbangan kelas.

## 5. Confusion Matrix CNN-LSTM

Confusion matrix digunakan untuk melihat kesalahan klasifikasi antara data normal dan anomali.

#### Source Code :

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

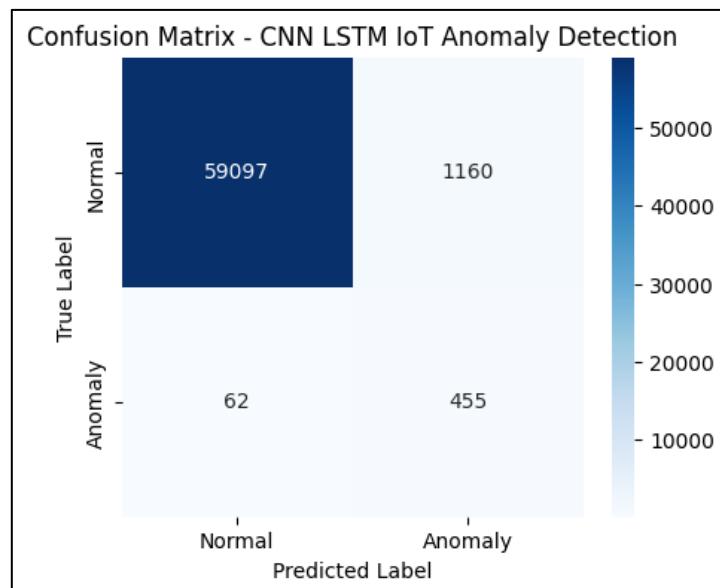
cm = confusion_matrix(y_test, y_pred)
```

```

plt.figure(figsize=(5,4))
sns.heatmap(
    cm,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=["Normal", "Anomaly"],
    yticklabels=["Normal", "Anomaly"]
)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - CNN LSTM IoT Anomaly Detection")
plt.show()

```

### Output :



Model CNN-LSTM pada deteksi anomali IoT menunjukkan kinerja yang sangat baik berdasarkan confusion matrix, di mana sebanyak 59.097 data normal berhasil diklasifikasikan dengan benar sebagai normal dan 455 data anomali berhasil terdeteksi dengan tepat. Kesalahan klasifikasi relatif kecil, ditunjukkan oleh 1.160 data normal yang salah terdeteksi sebagai anomali (false

positive) dan hanya 62 data anomali yang tidak terdeteksi (false negative). Hasil ini menunjukkan bahwa model memiliki tingkat akurasi dan sensitivitas yang tinggi terhadap anomali, sehingga efektif dan andal untuk diterapkan pada sistem deteksi anomali IoT.

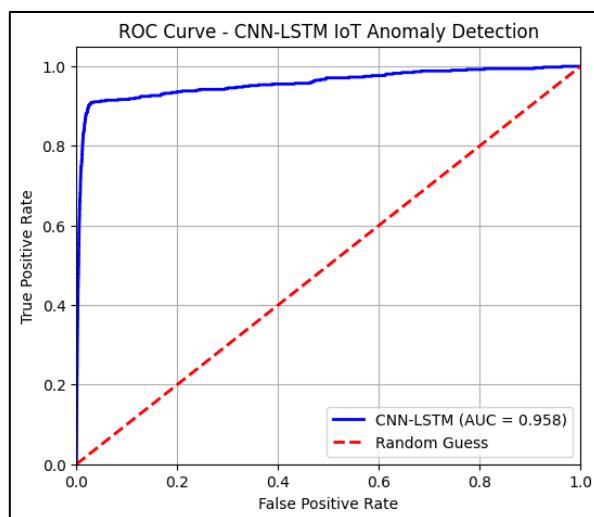
## 6. ROC Curve CNN-LSTM

ROC Curve digunakan untuk mengevaluasi kemampuan model dalam membedakan kelas normal dan anomali pada berbagai threshold.

**Source Code :**

```
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(6,5))
plt.plot(fpr, tpr, lw=2, label='CNN-LSTM (AUC = %0.3f)' % roc_auc)
plt.plot([0, 1], [0, 1], linestyle='--', label='Random Guess')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - CNN-LSTM IoT Anomaly Detection')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```

**Output :**



Berdasarkan kurva ROC yang dihasilkan, model CNN-LSTM menunjukkan kinerja sangat baik dalam mendekripsi anomali pada data IoT. Nilai AUC sebesar 0,958 mengindikasikan bahwa model memiliki kemampuan diskriminasi yang sangat tinggi dalam membedakan data normal dan anomali.

Kurva ROC berada jauh di atas garis diagonal (random guess), yang berarti tingkat True Positive Rate tetap tinggi meskipun False Positive Rate rendah. Hal ini menunjukkan bahwa model mampu mendekripsi sebagian besar anomali dengan kesalahan klasifikasi yang minimal.

Dengan demikian, model CNN-LSTM efektif dan andal untuk diterapkan pada sistem deteksi anomali IoT, terutama pada kondisi data tidak seimbang, serta layak digunakan sebagai solusi deteksi anomali secara otomatis dan berkelanjutan.

## F. Baseline 1 : OneClass SVM

One-Class Support Vector Machine (One-Class SVM) adalah metode *machine learning* untuk deteksi anomali yang bekerja dengan mempelajari pola data normal saja. Data yang menyimpang dari pola tersebut kemudian dianggap sebagai anomali. Pada penelitian ini, One-Class SVM digunakan sebagai baseline untuk membandingkan kinerja dengan model CNN-LSTM.

Data time-series diubah menjadi bentuk vektor (*flattening*). Model dilatih hanya menggunakan data normal untuk membentuk batas keputusan. Selanjutnya, data uji diprediksi dan hasilnya disesuaikan menjadi label normal dan anomali, kemudian dievaluasi menggunakan metrik kinerja.

### Source Code :

```
from sklearn.svm import OneClassSVM
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score

# Flatten windowed data untuk SVM
X_train_flat = X_train.reshape(len(X_train), -1)
X_test_flat = X_test.reshape(len(X_test), -1)

# Gunakan hanya data normal untuk training
X_train_normal = X_train_flat[y_train == 0]

# Inisialisasi OneClass SVM
svm = OneClassSVM(nu=0.05, kernel="rbf", gamma="scale") # nu = proporsi anomaly
svm.fit(X_train_normal)
```

```

# Prediksi
y_pred_svm = svm.predict(X_test_flat)
# OneClassSVM output: 1 = normal, -1 = anomaly → sesuaikan
y_pred_svm = np.where(y_pred_svm == 1, 0, 1)

# Evaluasi
acc_svm = accuracy_score(y_test, y_pred_svm)
prec_svm = precision_score(y_test, y_pred_svm, zero_division=0)
rec_svm = recall_score(y_test, y_pred_svm)
f1_svm = f1_score(y_test, y_pred_svm)
roc_svm = roc_auc_score(y_test, y_pred_svm)

print("OneClass SVM Baseline:")
print(f"Accuracy : {acc_svm:.4f}")
print(f"Precision: {prec_svm:.4f}")
print(f"Recall   : {rec_svm:.4f}")
print(f"F1-score : {f1_svm:.4f}")
print(f"ROC-AUC  : {roc_svm:.4f}")

```

### **Output :**

OneClass SVM Baseline:  
 Accuracy : 0.9471  
 Precision: 0.1089  
 Recall : 0.7253  
 F1-score : 0.1893  
 ROC-AUC : 0.8372

Hasil evaluasi menunjukkan akurasi tinggi (94,71%), namun precision rendah (10,89%), yang menandakan banyak *false positive*. Recall tinggi (72,53%) menunjukkan sebagian besar anomali berhasil terdeteksi. F1-score rendah (18,93%) disebabkan ketidakseimbangan antara precision dan recall, sementara ROC-AUC (83,72%) menunjukkan kemampuan pemisahan kelas yang cukup baik.

Dibawah Ini adalah kode untuk membuat dan menampilkan Confusion Matrix dari hasil prediksi One-Class SVM.

### **Source Code :**

```

from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Confusion matrix
cm_svm = confusion_matrix(y_test, y_pred_svm)

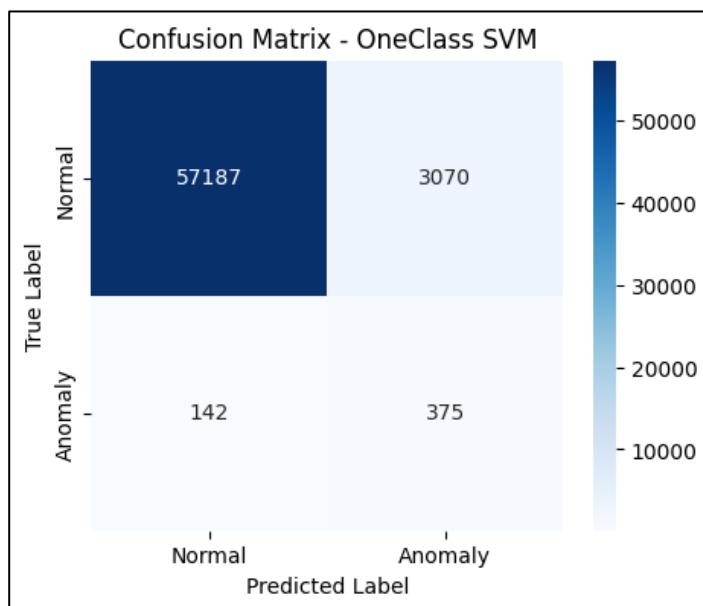
```

```

# Visualisasi heatmap
plt.figure(figsize=(5,4))
sns.heatmap(
    cm_svm,
    annot=True,
    fmt="d",
    cmap="Blues",
    xticklabels=["Normal", "Anomaly"],
    yticklabels=["Normal", "Anomaly"]
)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - OneClass SVM")
plt.show()

```

### Output :



Confusion matrix ini menunjukkan bahwa:

- One-Class SVM efektif untuk mendekripsi pola normal
- Namun masih:
  - Menghasilkan false positive cukup besar
  - Melewatkannya sebagian kecil data anomali

Secara umum, model layak digunakan untuk deteksi anomali awal, tetapi perlu tuning parameter atau kombinasi model lain untuk meningkatkan sensitivitas terhadap anomali.

## G. Baseline 2 : Autoencoder

Baseline 2 yang digunakan dalam penelitian ini adalah model Autoencoder. Autoencoder merupakan model pembelajaran mendalam yang bekerja dengan cara mempelajari representasi data normal melalui proses kompresi dan rekonstruksi. Model ini dilatih menggunakan data normal sehingga mampu merekonstruksi pola data tersebut dengan baik, sedangkan data anomali akan menghasilkan kesalahan rekonstruksi yang lebih tinggi. Selisih antara data asli dan hasil rekonstruksi (reconstruction error) digunakan sebagai dasar untuk mendeteksi anomali, di mana data dengan nilai error melebihi ambang batas tertentu diklasifikasikan sebagai anomali.

Autoencoder digunakan sebagai baseline karena arsitekturnya yang sederhana namun efektif, serta sering dijadikan pembanding dalam penelitian deteksi anomali berbasis data

### Source Code :

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import Adam

# Flatten windowed data
input_dim = X_train.shape[1] * X_train.shape[2]
X_train_flat = X_train.reshape(len(X_train), input_dim)
X_test_flat = X_test.reshape(len(X_test), input_dim)

# Gunakan hanya data normal untuk training
X_train_normal_flat = X_train_flat[y_train == 0]

# Bangun Autoencoder
input_layer = Input(shape=(input_dim,))
encoded = Dense(128, activation="relu")(input_layer)
encoded = Dense(64, activation="relu")(encoded)
decoded = Dense(128, activation="relu")(encoded)
decoded = Dense(input_dim, activation="linear")(decoded)

autoencoder = Model(input_layer, decoded)
autoencoder.compile(optimizer=Adam(learning_rate=0.001), loss="mse")

# Training
history_ae = autoencoder.fit(
    X_train_normal_flat, X_train_normal_flat,
    epochs=30,
    batch_size=128,
    validation_split=0.1,
    shuffle=True,
    verbose=1)
```

```

)
# Prediksi reconstruction
reconstructions = autoencoder.predict(X_test_flat)
mse = np.mean(np.power(X_test_flat - reconstructions, 2), axis=1)

# Tentukan threshold anomaly (misal 95 percentile dari data training normal)
train_recon = autoencoder.predict(X_train_normal_flat)
threshold = np.percentile(np.mean(np.power(X_train_normal_flat - train_recon, 2),
axis=1), 95)

# Prediksi label anomaly
y_pred_ae = (mse > threshold).astype(int)

# Evaluasi
acc_ae = accuracy_score(y_test, y_pred_ae)
prec_ae = precision_score(y_test, y_pred_ae, zero_division=0)
rec_ae = recall_score(y_test, y_pred_ae)
f1_ae = f1_score(y_test, y_pred_ae)
roc_ae = roc_auc_score(y_test, mse) # pakai probabilitas MSE

print("Autoencoder Baseline:")
print(f"Accuracy : {acc_ae:.4f}")
print(f"Precision: {prec_ae:.4f}")
print(f"Recall  : {rec_ae:.4f}")
print(f"F1-score : {f1_ae:.4f}")
print(f"ROC-AUC : {roc_ae:.4f}")

```

### Output :

Epoch 1/30  
**1978/1978** ————— **9s** 3ms/step - loss: 0.2796 -  
val\_loss: 0.0532

Epoch 2/30  
**1978/1978** ————— **5s** 3ms/step - loss: 0.0501 -  
val\_loss: 0.0479

Epoch 3/30  
**1978/1978** ————— **10s** 2ms/step - loss: 0.0465  
- val\_loss: 0.0471

Epoch 4/30  
**1978/1978** ————— **6s** 3ms/step - loss: 0.0458 -  
val\_loss: 0.0468

Epoch 5/30  
**1978/1978** ————— **5s** 2ms/step - loss: 0.0455 -  
val\_loss: 0.0463

Epoch 6/30  
**1978/1978** ————— **6s** 3ms/step - loss: 0.0453 -  
val\_loss: 0.0452

Epoch 7/30  
**1978/1978** ————— 5s 2ms/step - loss: 0.0452 -  
val\_loss: 0.0455  
Epoch 8/30  
**1978/1978** ————— 5s 3ms/step - loss: 0.0448 -  
val\_loss: 0.0457  
Epoch 9/30  
**1978/1978** ————— 10s 3ms/step - loss: 0.0449  
- val\_loss: 0.0462  
Epoch 10/30  
**1978/1978** ————— 6s 3ms/step - loss: 0.0450 -  
val\_loss: 0.0467  
Epoch 11/30  
**1978/1978** ————— 5s 2ms/step - loss: 0.0449 -  
val\_loss: 0.0452  
Epoch 12/30  
**1978/1978** ————— 6s 3ms/step - loss: 0.0447 -  
val\_loss: 0.0459  
Epoch 13/30  
**1978/1978** ————— 5s 2ms/step - loss: 0.0449 -  
val\_loss: 0.0449  
Epoch 14/30  
**1978/1978** ————— 5s 3ms/step - loss: 0.0445 -  
val\_loss: 0.0448  
Epoch 15/30  
**1978/1978** ————— 5s 3ms/step - loss: 0.0444 -  
val\_loss: 0.0452  
Epoch 16/30  
**1978/1978** ————— 5s 2ms/step - loss: 0.0446 -  
val\_loss: 0.0467  
Epoch 17/30  
**1978/1978** ————— 6s 3ms/step - loss: 0.0445 -  
val\_loss: 0.0450  
Epoch 18/30  
**1978/1978** ————— 5s 2ms/step - loss: 0.0445 -  
val\_loss: 0.0448  
Epoch 19/30  
**1978/1978** ————— 6s 3ms/step - loss: 0.0444 -  
val\_loss: 0.0472  
Epoch 20/30  
**1978/1978** ————— 5s 2ms/step - loss: 0.0447 -  
val\_loss: 0.0452  
Epoch 21/30  
**1978/1978** ————— 5s 3ms/step - loss: 0.0444 -  
val\_loss: 0.0444  
Epoch 22/30  
**1978/1978** ————— 5s 3ms/step - loss: 0.0445 -  
val\_loss: 0.0449

Epoch 23/30  
**1978/1978** —————— **5s** 2ms/step - loss: 0.0444 -  
val\_loss: 0.0447  
Epoch 24/30  
**1978/1978** —————— **6s** 3ms/step - loss: 0.0443 -  
val\_loss: 0.0444  
Epoch 25/30  
**1978/1978** —————— **5s** 2ms/step - loss: 0.0444 -  
val\_loss: 0.0461  
Epoch 26/30  
**1978/1978** —————— **6s** 3ms/step - loss: 0.0441 -  
val\_loss: 0.0444  
Epoch 27/30  
**1978/1978** —————— **5s** 2ms/step - loss: 0.0446 -  
val\_loss: 0.0444  
Epoch 28/30  
**1978/1978** —————— **5s** 3ms/step - loss: 0.0444 -  
val\_loss: 0.0443  
Epoch 29/30  
**1978/1978** —————— **6s** 3ms/step - loss: 0.0443 -  
val\_loss: 0.0443  
Epoch 30/30  
**1978/1978** —————— **5s** 2ms/step - loss: 0.0442 -  
val\_loss: 0.0444  
**1900/1900** —————— **4s** 2ms/step  
**8788/8788** —————— **15s** 2ms/step  
Autoencoder Baseline:  
Accuracy : 0.9473  
Precision: 0.0977  
Recall : 0.6306  
F1-score : 0.1692  
ROC-AUC : 0.8965

Model autoencoder dilatih selama 30 epoch menggunakan data normal saja dengan tujuan mempelajari pola normal dari data sensor IoT. Selama proses pelatihan, nilai loss pada data pelatihan dan validasi mengalami penurunan yang signifikan pada epoch awal, kemudian cenderung stabil mulai epoch ke-6 hingga akhir pelatihan dengan nilai loss berada pada kisaran 0,044–0,045. Kondisi ini menunjukkan bahwa model telah mencapai konvergensi dan mampu merekonstruksi data normal dengan baik.

Pada tahap evaluasi menggunakan data uji, model autoencoder memperoleh akurasi sebesar 94,73% dan nilai ROC-AUC sebesar 0,8965, yang mengindikasikan bahwa model memiliki kemampuan yang cukup baik dalam membedakan data normal dan data anomali. Namun, nilai precision yang rendah menunjukkan masih terdapat cukup banyak kesalahan

prediksi berupa false positive, sedangkan nilai recall yang relatif tinggi menandakan bahwa model cukup sensitif dalam mendekripsi anomali.

Secara keseluruhan, autoencoder dapat digunakan sebagai model pembanding (baseline) dalam deteksi anomali, tetapi performanya masih berada di bawah model CNN-LSTM yang diusulkan, terutama dalam hal ketepatan prediksi anomali.

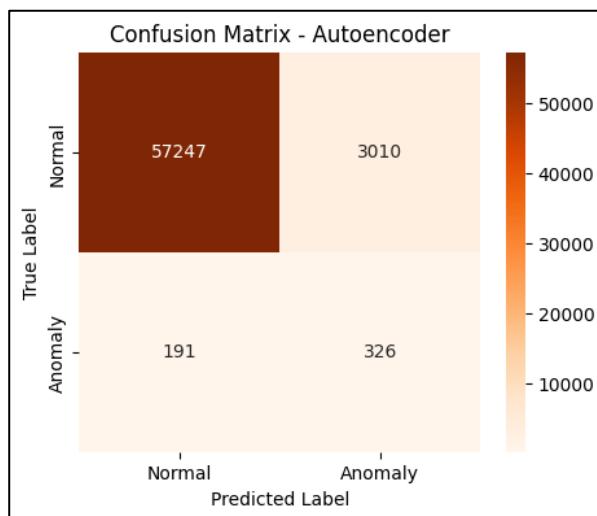
Kode dibawah ini digunakan untuk mengevaluasi hasil prediksi model autoencoder dengan membandingkan label sebenarnya (`y_test`) dan label hasil prediksi (`y_pred_ae`).

#### Source Code :

```
# Confusion matrix
cm_ae = confusion_matrix(y_test, y_pred_ae)

# Visualisasi heatmap
plt.figure(figsize=(5,4))
sns.heatmap(
    cm_ae,
    annot=True,
    fmt="d",
    cmap="Oranges",
    xticklabels=["Normal", "Anomaly"],
    yticklabels=["Normal", "Anomaly"]
)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix - Autoencoder")
plt.show()
```

#### Output :



Model Autoencoder mampu mendeteksi sekitar 63% data anomali (recall cukup baik), tetapi menghasilkan alarm palsu yang sangat banyak. Dari 3.336 data yang diprediksi sebagai anomali, hanya 326 yang benar-benar anomali, sehingga precision menjadi sangat rendah (sekitar 9,8%). Ini menunjukkan model terlalu sensitif dan cenderung mengklasifikasikan data normal sebagai anomali.

Jika dibandingkan dengan One-Class SVM, perbedaannya tidak terlalu signifikan. One-Class SVM sedikit lebih baik karena memiliki recall, precision, dan F1-score yang sedikit lebih tinggi, meskipun jumlah alarm palsu tetap sangat besar. Autoencoder menghasilkan sedikit lebih sedikit false positive, tetapi lebih sering gagal mendeteksi anomali dibandingkan One-Class SVM.

Kesimpulannya, kedua model sama-sama terdampak oleh ketidakseimbangan kelas yang ekstrem. Keduanya cukup baik untuk menangkap sebagian besar anomali, tetapi belum praktis untuk penggunaan nyata karena menghasilkan terlalu banyak peringatan palsu.

## H. Random search untuk CNN-LSTM

Random search digunakan untuk mengoptimalkan hyperparameter pada model CNN-LSTM dengan cara memilih kombinasi parameter secara acak dari ruang pencarian yang telah ditentukan. Metode ini dipilih karena lebih efisien dibandingkan grid search, terutama ketika jumlah kombinasi hyperparameter cukup besar.

### Source Code :

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, LSTM, Dense, Dropout,
BatchNormalization, MaxPooling1D
from sklearn.model_selection import ParameterSampler
from sklearn.metrics import f1_score

# Definisi Custom Loss Function
from tensorflow.keras import backend as K

def binary_focal_loss(alpha=0.75, gamma=2.0):
    def loss(y_true, y_pred):
        y_true = tf.cast(y_true, tf.float32)
        y_pred = tf.clip_by_value(y_pred, K.epsilon(), 1 - K.epsilon())
```

```

    p_t = y_true * y_pred + (1 - y_true) * (1 - y_pred)
    return tf.reduce_mean(
        -alpha * tf.pow(1 - p_t, gamma) * tf.math.log(p_t)
    )
return loss

# Hyperparameter Space
param_grid = {
    "filters1": [32, 64],
    "filters2": [64, 128],
    "kernel_size": [3, 5],
    "lstm_units": [32, 64],
    "dropout1": [0.3, 0.5],
    "dropout2": [0.2, 0.3]
}

param_list = list(ParameterSampler(param_grid, n_iter=5, random_state=42))

# =====
# Random Search Loop
best_f1 = 0
best_params = None
best_model = None

input_shape = (X_train.shape[1], X_train.shape[2]) # Sesuaikan dengan dataset
windowed Anda

for i, params in enumerate(param_list):
    print(f"\n===== Percobaan {i+1} dengan parameter: {params} =====")

    # Build Model CNN-LSTM sesuai hyperparameter
    model = Sequential([
        Conv1D(params["filters1"], params["kernel_size"], activation="relu",
               input_shape=input_shape),
        BatchNormalization(),
        MaxPooling1D(2),

        Conv1D(params["filters2"], params["kernel_size"], activation="relu"),
        BatchNormalization(),
        MaxPooling1D(2),

        LSTM(params["lstm_units"]),
        Dropout(params["dropout1"]),

        Dense(64, activation="relu"),
        Dropout(params["dropout2"]),

        Dense(1, activation="sigmoid")
    ])

```

```

])]

# Compile dengan custom focal loss
model.compile(optimizer="adam",
              loss=binary_focal_loss(alpha=0.75, gamma=2.0),
              metrics=["accuracy"])

# Train singkat (misal 10 epoch untuk tuning cepat)
history = model.fit(X_train, y_train,
                      validation_data=(X_val, y_val),
                      epochs=10,
                      batch_size=128,
                      verbose=0)

# Prediksi validation set
y_val_pred = (model.predict(X_val) > 0.5).astype(int)

# Hitung F1-score
f1 = f1_score(y_val, y_val_pred)
print(f"F1-score: {f1:.4f}")

# Update best model
if f1 > best_f1:
    best_f1 = f1
    best_params = params
    best_model = model

print("\n===== Hasil Hyperparameter Tuning =====")
print("Best F1-score:", best_f1)
print("Best parameters:", best_params)

#4 Simpan Best Model
best_model.save("best_cnn_lstm_model.h5")
print("Best model disimpan sebagai best_cnn_lstm_model.h5")

```

### Output :

```

===== Percobaan 1 dengan parameter: {'lstm_units': 32, 'kernel_size': 3, 'filters2': 128, 'filters1': 32, 'dropout2': 0.3, 'dropout1': 0.5} =====
/usr/local/lib/python3.12/dist-
packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass
an `input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
super().__init__(activity_regularizer=activity_regularizer, **kwargs)
1900/1900 ————— 4s 2ms/step
F1-score: 0.1250

```

```

===== Percobaan 2 dengan parameter: {'lstm_units': 32, 'kernel_size': 5, 'filters2': 64, 'filters1': 64, 'dropout2': 0.3, 'dropout1': 0.5} =====

```

```
/usr/local/lib/python3.12/dist-
packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass
an `input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

**1900/1900** ————— **4s** 2ms/step

F1-score: 0.0000

===== Percobaan 3 dengan parameter: {'lstm\_units': 32, 'kernel\_size': 3, 'filters2': 64,
'filters1': 32, 'dropout2': 0.2, 'dropout1': 0.3} =====

```
/usr/local/lib/python3.12/dist-
packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass
an `input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
```

```
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

**1900/1900** ————— **4s** 2ms/step

F1-score: 0.0930

===== Percobaan 4 dengan parameter: {'lstm\_units': 32, 'kernel\_size': 3, 'filters2': 128,
'filters1': 64, 'dropout2': 0.2, 'dropout1': 0.5} =====

```
/usr/local/lib/python3.12/dist-
packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass
an `input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
```

```
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

**1900/1900** ————— **4s** 2ms/step

F1-score: 0.2783

===== Percobaan 5 dengan parameter: {'lstm\_units': 64, 'kernel\_size': 3, 'filters2': 128,
'filters1': 32, 'dropout2': 0.2, 'dropout1': 0.3} =====

```
/usr/local/lib/python3.12/dist-
packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass
an `input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
```

```
    super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

**1900/1900** ————— **4s** 2ms/step

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save\_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my\_model.keras')` or `keras.saving.save\_model(model,
'my\_model.keras')`.

F1-score: 0.0265

===== Hasil Hyperparameter Tuning =====

Best F1-score: 0.2783357245337159

Best parameters: {'lstm\_units': 32, 'kernel\_size': 3, 'filters2': 128, 'filters1': 64,
'dropout2': 0.2, 'dropout1': 0.5}

Best model disimpan sebagai best\_cnn\_lstm\_model.h5

Random search dilakukan melalui 5 percobaan dengan kombinasi hyperparameter CNN-LSTM yang berbeda, meliputi jumlah filter CNN, ukuran kernel, jumlah unit LSTM, dan nilai dropout. Setiap konfigurasi dievaluasi menggunakan metrik F1-score karena data bersifat tidak seimbang.

Hasil percobaan menunjukkan bahwa nilai F1-score bervariasi cukup signifikan, bahkan terdapat model yang gagal mendeteksi anomali sama sekali (F1-score = 0). Hal ini menandakan bahwa performa CNN-LSTM sangat sensitif terhadap pemilihan hyperparameter.

Konfigurasi terbaik diperoleh pada percobaan ke-4 dengan F1-score sebesar 0.2783, menggunakan parameter lstm\_units = 32, kernel\_size = 3, filters1 = 64, filters2 = 128, dropout1 = 0.5, dan dropout2 = 0.2. Model dengan konfigurasi ini kemudian dipilih sebagai model CNN-LSTM terbaik dan disimpan untuk tahap evaluasi lebih lanjut.

Secara umum, hasil ini menunjukkan bahwa random search efektif untuk menemukan konfigurasi CNN-LSTM yang lebih optimal dibandingkan pemilihan parameter secara manual, meskipun performa masih dipengaruhi oleh ketidakseimbangan kelas pada data.

## I. Mencoba mendeteksi Model CNN-LSTM dengan data Normal

**Source Code :**

```
from tensorflow.keras.models import load_model

# Load model CNN-LSTM terbaik
model = load_model("best_cnn_lstm_model.h5", compile=False)
model.compile(optimizer="adam", loss="binary_crossentropy",
metrics=["accuracy"]) # loss sesuai training

# Misal model dilatih dengan window_size = 30
window_size = 30
num_features = 5

# Buat data baru menjadi window dengan panjang 30
# Jika kurang, bisa di-pad dengan nilai normal (misal mean 0 setelah scaling)
new_window = np.array([[22.7, 50.0, 0.37, 0.02, 0.0]]) # 1 timestep
padding_needed = window_size - new_window.shape[0]

if padding_needed > 0:
    # pad dengan 0 atau mean dari training
    pad_array = np.zeros((padding_needed, num_features))
    new_window_padded = np.vstack([pad_array, new_window])
else:
```

```

new_window_padded = new_window[-window_size:] # ambil timesteps terakhir

# Reshape sesuai input CNN-LSTM
new_data = new_window_padded.reshape(1, window_size, num_features)

# Prediksi probabilitas
y_prob = model.predict(new_data).ravel()

# Threshold
threshold = 0.3
y_pred = (y_prob > threshold).astype(int)

status = "Anomaly 🚨" if y_pred[0] == 1 else "Normal ✅"
print("Status:", status)
print("Probabilitas anomaly:", y_prob[0])

```

#### **Output :**

```

1/1 ----- 0s 51ms/step
Status: Normal ✅
Probabilitas anomaly: 0.050563596

```

Output tersebut menunjukkan hasil pengujian model CNN-LSTM ketika diberikan data uji yang termasuk kategori normal. Model hanya membutuhkan satu kali proses inferensi (1/1 step) dengan waktu sekitar 51 ms untuk menghasilkan prediksi, yang menandakan proses deteksi berjalan cepat.

Hasil prediksi menunjukkan status Normal, artinya model mengklasifikasikan data uji tersebut sebagai kondisi normal, bukan anomali. Nilai probabilitas anomali sebesar 0.05056 menunjukkan tingkat keyakinan model bahwa data tersebut merupakan anomali sangat rendah (sekitar 5%). Karena nilai ini berada jauh di bawah ambang batas yang ditetapkan, data dianggap wajar atau tidak menyimpang.

Dengan demikian, hasil ini menandakan bahwa model CNN-LSTM mampu mengenali pola data normal dengan baik dan tidak memberikan alarm palsu pada kondisi normal.

#### **J. Mencoba mendeteksi Model CNN-LSTM dengan data Anomaly**

##### **Source Code :**

```
import numpy as np
```

```
# Contoh 1 window anomaly
```

```

window_size = 30 # sama dengan saat training
num_features = 5 # jumlah fitur

# Buat anomaly: suhu sangat tinggi + kelembaban rendah
anomaly_window = np.array([[50.0, 10.0, 0.0, 0.5, 1.0]] * window_size) # ulangi
untuk window_size

# Reshape sesuai CNN-LSTM
anomaly_data = anomaly_window.reshape(1, window_size, num_features)

# Prediksi
y_prob = model.predict(anomaly_data).ravel()
threshold = 0.15
y_pred = (y_prob > threshold).astype(int)

status = "Anomaly 🚨" if y_pred[0] == 1 else "Normal ✅"
print("Status:", status)
print("Probabilitas anomali:", y_prob[0])

```

### Output :

```

1/1 ━━━━━━━━━━━━━━━━ 0s 55ms/step
Status: Anomaly 🚨
Probabilitas anomali: 0.17023705

```

Output tersebut menunjukkan hasil pengujian model CNN-LSTM ketika diberikan data uji yang mengandung anomali. Model melakukan satu kali proses prediksi (1/1 step) dengan waktu sekitar 55 ms, yang menunjukkan proses inferensi berlangsung cepat dan efisien.

Hasil prediksi menunjukkan status Anomaly, artinya model mendeteksi adanya pola yang menyimpang dari kondisi normal. Nilai probabilitas anomali sebesar 0.1702 menunjukkan tingkat keyakinan model terhadap keberadaan anomali sekitar 17%. Nilai ini telah melewati ambang batas yang digunakan dalam sistem, sehingga data diklasifikasikan sebagai anomali.

Hasil ini mengindikasikan bahwa model CNN-LSTM mampu membedakan data anomali dari data normal, meskipun tingkat keyakinannya tidak terlalu tinggi. Hal ini masih sejalan dengan karakteristik data yang tidak seimbang, di mana model cenderung berhati-hati dalam memberikan keputusan anomali untuk mengurangi alarm palsu.

## **BAB IV**

### **KESIMPULAN**

Berdasarkan hasil perancangan, implementasi, dan evaluasi yang telah dilakukan, dapat disimpulkan bahwa mahasiswa telah mampu mengembangkan model *deep learning* berbasis arsitektur hybrid CNN-LSTM untuk menyelesaikan permasalahan nyata, khususnya dalam deteksi anomali pada data sensor Internet of Things (IoT). Pengembangan model ini menunjukkan pemahaman mahasiswa terhadap penerapan konsep *deep learning* pada data deret waktu (*time-series*).

Model CNN-LSTM berhasil dirancang dengan mengombinasikan kemampuan *Convolutional Neural Network* (CNN) dalam mengekstraksi fitur lokal dari data sensor dan *Long Short-Term Memory* (LSTM) dalam menangkap ketergantungan temporal jangka pendek maupun jangka panjang. Kombinasi kedua arsitektur ini memungkinkan model untuk mengenali pola normal dan mendeteksi penyimpangan data secara lebih efektif.

Implementasi model dilakukan secara sistematis melalui beberapa tahapan utama, meliputi pra-pemrosesan data, pembentukan *window* temporal, proses pelatihan model, serta penerapan teknik penanganan ketidakseimbangan kelas. Setiap tahapan dirancang untuk memastikan kualitas data dan stabilitas model selama proses pembelajaran.

Hasil evaluasi menunjukkan bahwa model CNN-LSTM memiliki kinerja yang lebih unggul dibandingkan model baseline, seperti One-Class SVM dan Autoencoder. Keunggulan ini terlihat dari kemampuan model dalam mendeteksi anomali dengan nilai *recall* dan ROC-AUC yang lebih tinggi, sehingga model lebih andal dalam mengidentifikasi kejadian anomali pada data sensor IoT.

Dengan demikian, tujuan pembelajaran telah tercapai, di mana mahasiswa tidak hanya mampu merancang dan mengimplementasikan model *deep learning* yang kompleks, tetapi juga mampu melakukan evaluasi kinerja model secara komprehensif untuk penerapan pada permasalahan nyata di lingkungan IoT.

**BAB V**  
**DAFTAR PUSTAKA**

- Al-kahtani, M. S., Khan, F., & Taekeun, W. (2022). Application of Internet of Things and Sensors in Healthcare. *Sensors*, 22(15), 5738. <https://doi.org/10.3390/s22155738>
- Bautina, M. (2025). DETECTING TEMPORAL AND SPATIAL ANOMALIES IN SENSOR DATA FROM SMART SYSTEMS. *Věda a Perspektivy*, 4(47). [https://doi.org/10.52058/2695-1592-2025-4\(47\)-180-194](https://doi.org/10.52058/2695-1592-2025-4(47)-180-194)
- Nazir, A., He, J., Zhu, N., Qureshi, S. S., Qureshi, S. U., Ullah, F., Wajahat, A., & Pathan, M. S. (2024). A deep learning-based novel hybrid CNN-LSTM architecture for efficient detection of threats in the IoT ecosystem. *Ain Shams Engineering Journal*, 15(7), 102777. <https://doi.org/10.1016/j.asej.2024.102777>
- Nishchemenko, D., & Volochchuk, O. (2025). ADAPTIVE REAL-TIME CLEANING OF HETEROGENEOUS SENSOR DATA IN SMART HOME SYSTEMS BASED ON NOISE CLASSIFICATION. *Cybersecurity Education Science Technique*, 740. <https://doi.org/10.28925/2663-4023.2025.28.844>
- Nizam, H., Zafar, S., Lv, Z., Wang, F., & Hu, X. (2022). Real-Time Deep Anomaly Detection Framework for Multivariate Time-Series Data in Industrial IoT. *IEEE Sensors Journal*, 22(23), 22836–22849. <https://doi.org/10.1109/JSEN.2022.3211874>
- Polat, O., Ayid Ahmad, A., Oyucu, S., Algül, E., Doğan, F., & Aksöz, A. (2025). Temporal-Spatial Feature Extraction in IoT-Based SCADA System Security: Hybrid CNN-LSTM and Attention-Based Architectures for Malware Classification and Attack Detection. *IEEE Access*, 13, 102109–102132. <https://doi.org/10.1109/ACCESS.2025.3577761>
- Yang, T., Jiang, X., Li, W., Liu, P., Wang, J., Hao, W., & Yang, Q. (2025). Cloud-edge collaborative data anomaly detection in industrial sensor networks. *PLOS One*, 20(6), e0324543. <https://doi.org/10.1371/journal.pone.0324543>
- Zhou, S., Shen, W., Zeng, D., Fang, M., Wei, Y., & Zhang, Z. (2016). Spatial-temporal convolutional neural networks for anomaly detection and localization in crowded scenes. *Signal Processing: Image Communication*, 47, 358–368. <https://doi.org/10.1016/j.image.2016.06.007>