LAPORAN TUGAS AKHIR
JARINGAN SARAF TIRUAN

Teknik Informatika - Kelas D

Dosen Pengampu Mata Kuliah:
Dr. Eng. Budi Darma Setiawan, S.Kom., M.Cs.

Disusun Oleh:

Krisna Arinugraha Liantara	225150207111022
Muhammad Hasan Fadhlillah	225150207111026
Muhammad Husain Fadhlillah	225150207111027

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2024

1. TAUTAN *GOOGLE COLLAB*

v3 Project Akhir JST - Kelompok 9.ipynb

2. PROJECT OVERVIEW

2.1 Latar Belakang

- Dengan meningkatnya penggunaan smartphone yang dilengkapi berbagai sensor (accelerometer dan gyroscope), terbuka peluang untuk mengembangkan sistem yang dapat mengenali aktivitas manusia secara otomatis.
- Sensor pada smartphone dapat mengumpulkan data gerakan yang detail dan dapat digunakan untuk mengklasifikasikan berbagai aktivitas fisik.
- Kemampuan mengenali aktivitas manusia memiliki aplikasi penting di berbagai bidang seperti:
    - Healthcare monitoring
    - Fitness tracking
    - Elder care
    - Sports analysis
    - Smart home automation

2.2 Masalah

Pengenalan aktivitas manusia secara akurat menjadi tantangan utama karena adanya aktivitas yang mirip serta adanya variasi individu dalam melakukan gerakan. Sensor ponsel memberikan data yang kaya, tetapi tantangannya adalah bagaimana memilih fitur-fitur yang signifikan dan menerapkan model yang mampu membedakan aktivitas-aktivitas yang tampak serupa.

2.3 Tujuan

1. Mengembangkan model machine learning yang dapat mengklasifikasikan 3 aktivitas dasar manusia dengan akurasi tinggi.
2. Memahami pola karakteristik dari setiap aktivitas berdasarkan data sensor.
3. Mengidentifikasi fitur-fitur yang paling berpengaruh dalam membedakan aktivitas.
4. Menganalisis performa model dalam membedakan aktivitas yang memiliki karakteristik mirip.

2.4 Manfaat

1. Bidang Kesehatan:
    - Monitoring aktivitas pasien
    - Deteksi anomali gerakan

- Tracking rehabilitasi fisik
- Pencatatan aktivitas harian untuk analisis kesehatan

2. Fitness & Sports:
    - Tracking latihan otomatis
    - Analisis performa olahraga
    - Pemantauan intensitas aktivitas
    - Feedback real-time untuk perbaikan gerakan

3. Elderly Care:
    - Deteksi jatuh
    - Monitoring aktivitas sehari-hari
    - Alert system untuk situasi darurat
    - Tracking pola aktivitas untuk deteksi perubahan kesehatan

4. Research & Development:
    - Pengembangan algoritma HAR yang lebih baik
    - Studi pola aktivitas manusia
    - Pengembangan aplikasi healthcare berbasis smartphone

3. DATA *UNDERSTANDING*

3.1 Sumber Dataset

Dataset yang digunakan berasal dari UCI Machine Learning Repository, dengan nama "Human Activity Recognition Using Smartphones". Dataset ini juga tersedia di Kaggle sebagai alternatif.

- URL UCI: https://archive.ics.uci.edu/dataset/240
- URL Kaggle: https://www.kaggle.com/datasets/uciml/human-activity-recognition-with-smartphones/data

3.2 Deskripsi Dataset

Dataset ini dikumpulkan melalui eksperimen yang melibatkan 30 relawan dengan rentang usia 19-48 tahun. Para relawan melakukan 6 aktivitas dasar, yaitu:

1. WALKING (berjalan di permukaan datar)
2. WALKING_UPSTAIRS (menaiki tangga)
3. WALKING_DOWNSTAIRS (menuruni tangga)
4. SITTING (duduk)
5. STANDING (berdiri)
6. LAYING (berbaring)

Aktivitas direkam menggunakan smartphone Samsung Galaxy S II yang dipasang di pinggang setiap relawan. Data dikumpulkan dari sensor accelerometer dan gyroscope pada frekuensi sampling 50 Hz.

3.3 Struktur Dataset Secara Umum

Dataset ini terdiri dari file data fitur (X_train.txt dan X_test.txt) yang mencakup 561 kolom fitur dari sinyal sensor dan file label aktivitas (y_train.txt dan y_test.txt) yang menunjukkan aktivitas spesifik yang dilakukan. File features.txt menyediakan deskripsi dari setiap fitur, sedangkan activity_labels.txt mendefinisikan kode untuk setiap aktivitas.

3.4 Statistik Dataset

- Total samples: 10,299
- Training samples: 7,352 (71.5%)
- Test samples: 2,947 (28.5%)
- Balanced distribution antar aktivitas
- Feature scaling sudah diterapkan (normalized & bounded within [-1,1])

3.5 Karakteristik Dataset

- Jumlah Fitur Total: 561
- Tipe Data: Numerik (representasi sinyal dari sensor)
- Pembagian Dataset:
    - Training Set: 70%
    - Testing Set: 30%
- Preprocessing:
    - Noise filtering diterapkan pada sinyal mentah untuk mengurangi noise.
    - Fitur diekstrak dengan teknik domain waktu dan domain frekuensi, seperti mean, standar deviasi, magnitude, dan Fast Fourier Transform (FFT).

3.6 Fitur yang Digunakan

Dalam proyek ini, hanya subset fitur tertentu yang digunakan untuk memudahkan analisis dan pengembangan model. Fokus utama adalah fitur dari sensor gyroscope dengan nama prefiks tBodyGyro (Time Domain Signals).

- Fitur yang Dipilih:
    - Gyroscope Minimum Values:
        - tBodyGyro-min()-X
        - tBodyGyro-min()-Y
        - tBodyGyro-min()-Z

    - Gyroscope Standard Deviation:
        - tBodyGyro-std()-X
        - tBodyGyro-std()-Y
        - tBodyGyro-std()-Z
- Alasan Pemilihan Fitur:
  Fitur minimum dan standar deviasi dari gyroscope dipilih karena memberikan informasi penting tentang perubahan gerakan angular, baik dalam pola regular maupun perubahan mendadak. Fitur ini relevan untuk membedakan aktivitas seperti WALKING, WALKING_DOWNSTAIRS, dan LAYING. Statistik minimum (min()) digunakan untuk menangkap nilai terendah, sedangkan deviasi standar (std()) menggambarkan variabilitas gerakan pada tiap axis.

3.7 Peran dan Signifikansi Fitur

- LAYING: Gyroscope hampir tidak menunjukkan perubahan, sehingga fitur min() sangat kecil.
- WALKING: Variasi tinggi pada gyroscope (std() besar) karena pola gerakan berulang.
- WALKING_DOWNSTAIRS: Gyroscope memiliki variasi tinggi dengan pola amplitudo lebih besar daripada WALKING.

3.8 Aktivitas yang Difokuskan

Hanya 3 dari 6 aktivitas dalam dataset yang dianalisis:

1. LAYING: Representasi aktivitas statis, dengan sinyal gyroscope minimal.
2. WALKING: Aktivitas dinamis dengan pola gyroscope periodik.
3. WALKING_DOWNSTAIRS: Aktivitas dinamis dengan pola gyroscope lebih intens karena adanya gaya dorong tambahan.

Distribusi Data:

- Training Set: 3619 sampel
- Testing Set: 1453 sampel

3.9 Transformasi Data

- Outlier Removal: Diterapkan metode IQR (Interquartile Range) untuk menghilangkan data ekstrim.
- Standardisasi Data: Semua fitur dinormalisasi menggunakan StandardScaler untuk menghindari skala yang tidak konsisten.

3.10 Kelas Data

Kelas pada dataset direpresentasikan dalam format numerik (1 untuk WALKING, 3 untuk WALKING_DOWNSTAIRS, dan 6 untuk LAYING), yang kemudian dimapping ke format label untuk kemudahan interpretasi model:

- 1 → WALKING
- 3 → WALKING_DOWNSTAIRS
- 6 → LAYING

Dataset akhir setelah preprocessing memiliki ukuran sebagai berikut:

- Training Set: 2928 sampel, 26 fitur.
- Testing Set: 1233 sampel, 26 fitur.

3.11 Keperluan Model

Dataset ini digunakan untuk melatih model backpropagation neural network yang dibuat from scratch. Fitur-fitur gyroscope memberikan masukan penting untuk fungsi aktivasi yang dirancang manual sesuai persyaratan proyek.

4. PENGOLAHAN DATA

4.1 *Step 1: Import Required Libraries*

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import confusion_matrix,
classification_report
import warnings
warnings.filterwarnings('ignore')

# Set random seed dan style visualisasi
np.random.seed(42)
```

Langkah awal dimulai dengan mengimpor pustaka-pustaka penting yang dibutuhkan untuk analisis dan pengolahan data. Pustaka-pustaka yang diimpor mencakup:

1. Numpy untuk operasi numerik dan manipulasi array,
2. Pandas untuk pengolahan dan analisis data berbasis tabel,
3. Seaborn dan Matplotlib untuk pembuatan visualisasi data yang informatif,

4. Pustaka dari scikit-learn untuk pembagian dataset, preprocessing data, dan evaluasi model, termasuk modul seperti train_test_split untuk membagi data menjadi set pelatihan dan pengujian, StandardScaler untuk normalisasi data, LabelEncoder untuk mengubah label kategori menjadi numerik, serta modul metrics untuk menghitung matriks kebingungan dan laporan klasifikasi.

Program juga menyertakan modul peringatan (warnings) untuk mengabaikan pesan peringatan yang tidak penting selama eksekusi. Kemudian, seed acak diatur menggunakan np.random.seed(42) untuk memastikan hasil yang konsisten dan dapat direproduksi. Selain itu, gaya visualisasi disesuaikan menggunakan seaborn untuk memberikan tampilan grafik yang lebih estetis.

4.2 *Step 2: Load Data*

```
# Download and unzip dataset if not available
import os
if not os.path.exists('UCI HAR Dataset'):
                                                    !wget
https://archive.ics.uci.edu/ml/machine-learning-databases/00240
/UCI%20HAR%20Dataset.zip
    !unzip 'UCI HAR Dataset.zip'

# Load data
print("Loading data...")
X_train = pd.read_csv('UCI HAR Dataset/train/X_train.txt',
delim_whitespace=True, header=None)
y_train = pd.read_csv('UCI HAR Dataset/train/y_train.txt',
header=None)
X_test = pd.read_csv('UCI HAR Dataset/test/X_test.txt',
delim_whitespace=True, header=None)
y_test = pd.read_csv('UCI HAR Dataset/test/y_test.txt',
header=None)

# Load features dan activity labels
features = pd.read_csv('UCI HAR Dataset/features.txt',
delim_whitespace=True, header=None)[1]
activity_labels = pd.read_csv('UCI HAR
Dataset/activity_labels.txt',
```

```
                                                    delim_whitespace=True,
header=None)[1]

# Set column names
X_train.columns = features
X_test.columns = features
```

Langkah kedua adalah mengunduh, memuat, dan mempersiapkan dataset yang akan digunakan dalam analisis.

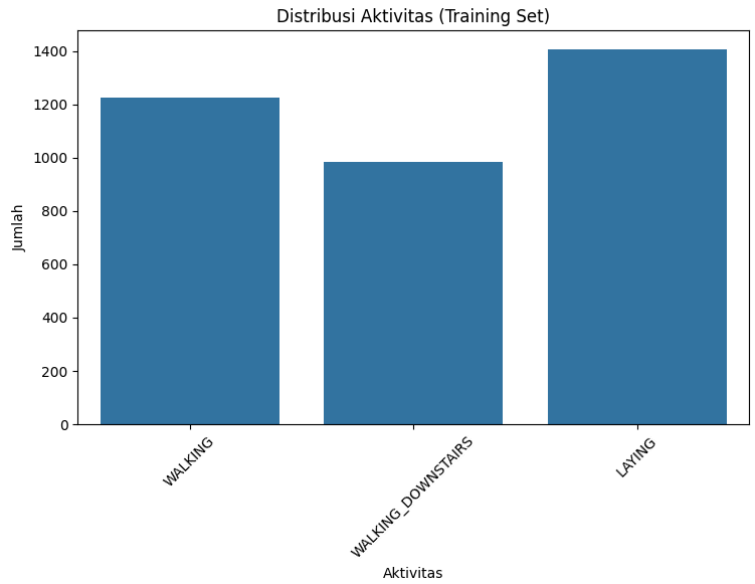
1. **Memeriksa keberadaan dataset:** Program menggunakan modul `os` untuk memeriksa apakah folder dataset bernama "UCI HAR Dataset" sudah tersedia di direktori kerja. Jika tidak ditemukan, program akan mengunduh dataset dari repositori UCI menggunakan perintah `wget`, dan kemudian mengekstraknya dengan `unzip`.
2. **Memuat data pelatihan dan pengujian:** Data yang terdiri dari features (variabel independen) dan labels (target) untuk set pelatihan dan pengujian dimuat menggunakan `pandas.read_csv`. Dataset pelatihan berada di file `'train/X_train.txt'` dan `'train/y_train.txt'`, sementara dataset pengujian berada di file `'test/X_test.txt'` dan `'test/y_test.txt'`. Format file menggunakan delimitasi spasi (`delim_whitespace=True`) tanpa header.
3. **Memuat fitur dan label aktivitas:** Program memuat daftar nama fitur dari file `'features.txt'` dan label aktivitas dari file `'activity_labels.txt'`. Informasi ini penting untuk memberikan nama kolom yang bermakna pada dataset dan memberikan makna pada label aktivitas yang berupa angka.
4. **Mengatur nama kolom:** Nama-nama fitur yang diambil dari `'features.txt'` diterapkan sebagai nama kolom untuk data pelatihan (`X_train`) dan data pengujian (`X_test`). Hal ini bertujuan untuk meningkatkan interpretabilitas dataset sehingga setiap kolom mewakili fitur yang relevan sesuai dengan deskripsi aslinya.

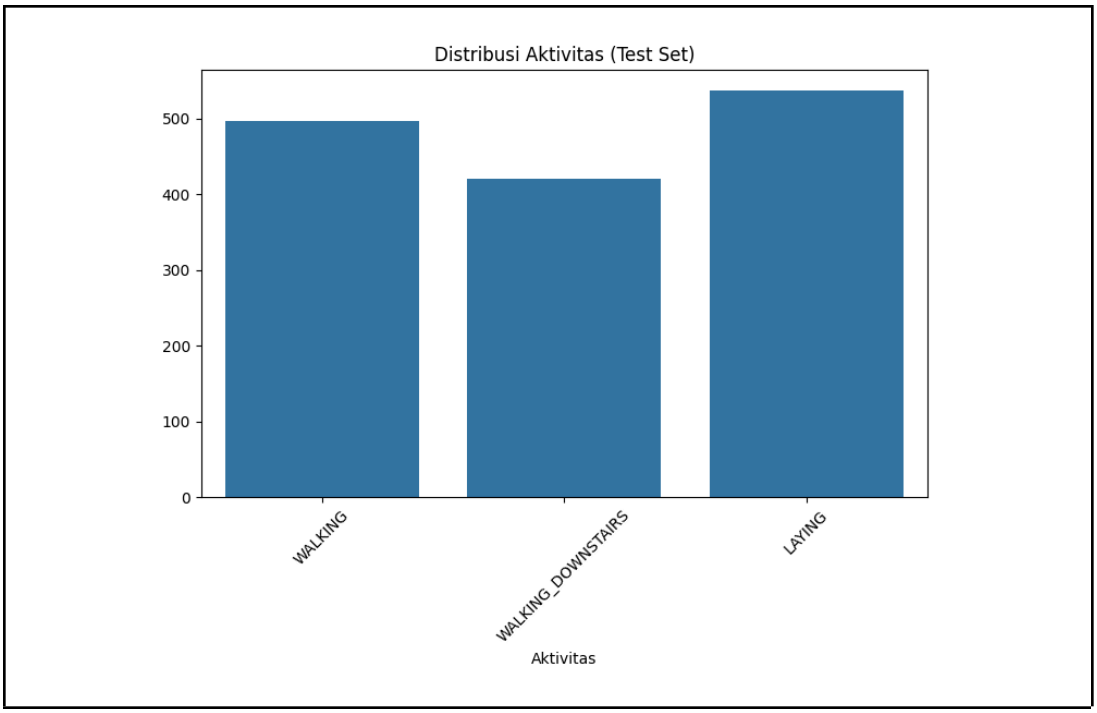
4.3 *Step 3: Exploratory Data Analysis (EDA)*

```
# Visualisasi distribusi aktivitas awal
print("Visualisasi distribusi aktivitas awal...")
plt.figure(figsize=(15, 6))

# Training Set
plt.subplot(1, 2, 1)
```

```
sns.countplot(x=y_train.iloc[:, 0], order=[1, 3, 6])  # Fokus
pada aktivitas tertentu
plt.title('Distribusi Aktivitas (Training Set)')
plt.xlabel('Aktivitas')
plt.ylabel('Jumlah')
plt.xticks([0,  1,  2],  ['WALKING',  'WALKING_DOWNSTAIRS',
'LAYING'], rotation=45)

# Test Set
plt.subplot(1, 2, 2)
sns.countplot(x=y_test.iloc[:, 0], order=[1, 3, 6])  # Fokus
pada aktivitas tertentu
plt.title('Distribusi Aktivitas (Test Set)')
plt.xlabel('Aktivitas')
plt.ylabel('Jumlah')
plt.xticks([0,  1,  2],  ['WALKING',  'WALKING_DOWNSTAIRS',
'LAYING'], rotation=45)

plt.tight_layout()
plt.show()
```

```
Visualisasi distribusi aktivitas awal...
```





Langkah selanjutnya adalah tahap *Exploratory Data Analysis* (EDA) menggunakan dataset aktivitas manusia bertujuan untuk memahami distribusi awal data berdasarkan label aktivitas tertentu.

1. Visualisasi Distribusi Aktivitas:
    - Program dimulai dengan mencetak pesan "Visualisasi distribusi aktivitas awal...".
    - Grafik dibuat dengan ukuran figur 15 x 6 menggunakan `plt.figure(figsize=(15, 6))`.
    - Visualisasi dibagi menjadi dua bagian:
        - Training Set:
            - Menggunakan `sns.countplot` untuk menghitung jumlah data aktivitas dari dataset `y_train`. Hanya aktivitas dengan label 1 (WALKING), 3 (WALKING_DOWNSTAIRS), dan 6 (LAYING) yang divisualisasikan, ditentukan oleh parameter `order=[1, 3, 6]`.
            - Judul grafik: "Distribusi Aktivitas (Training Set)". Label sumbu X adalah "Aktivitas", dan sumbu Y adalah "Jumlah".
            - Label aktivitas diubah menjadi nama aktivitas (`['WALKING', 'WALKING_DOWNSTAIRS', 'LAYING']`), dengan rotasi 45 derajat.

    - Test Set:
        - Langkah serupa dilakukan untuk dataset `y_test`, menampilkan aktivitas yang sama.
        - Judul grafik: "Distribusi Aktivitas (Test Set)". Label sumbu X dan Y sama seperti grafik sebelumnya.
- `plt.tight_layout()` digunakan untuk mengatur tata letak grafik agar tidak saling tumpang tindih.
- Grafik ditampilkan menggunakan `plt.show()`.

2. Distribusi Data dalam Bentuk Teks:
    - Distribusi jumlah aktivitas pada Training Set dan Test Set dihitung dan ditampilkan menggunakan fungsi `value_counts()` dengan urutan tertentu `reindex([1, 3, 6])`.
    - Hasilnya mencetak distribusi jumlah aktivitas untuk setiap dataset.

Penjelasan Grafik:

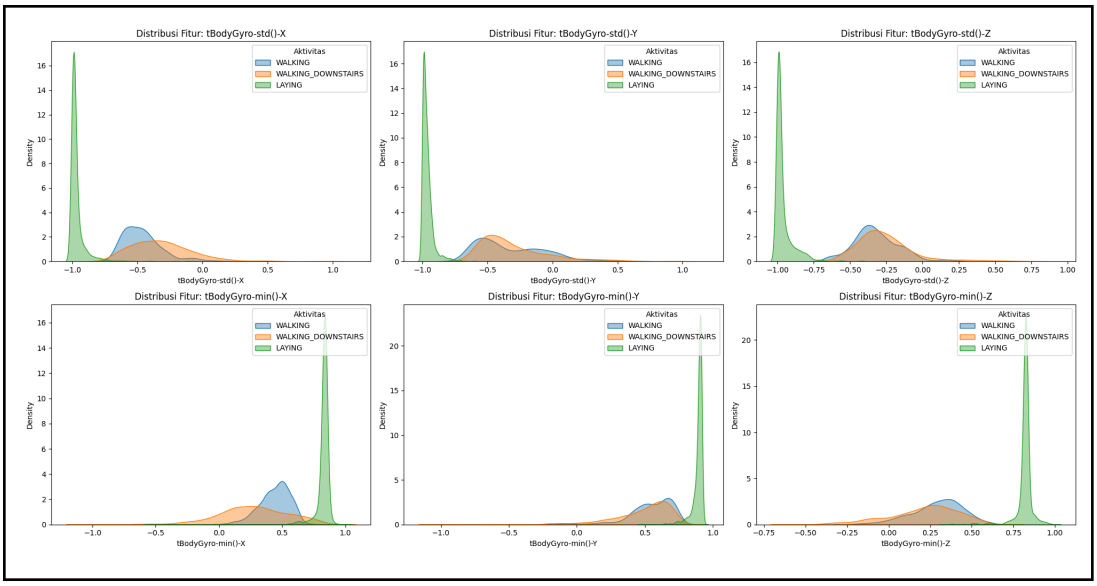
1. Grafik Training Set:
    - Menampilkan distribusi tiga jenis aktivitas:
        - WALKING: 1.226 data
        - WALKING_DOWNSTAIRS: 986 data
        - LAYING: 1.407 data
    - Aktivitas LAYING memiliki jumlah data terbanyak, diikuti WALKING, dan terakhir WALKING_DOWNSTAIRS.
    - Grafik ini memberikan gambaran distribusi aktivitas dalam dataset training.

2. Grafik Test Set:
    - Distribusi aktivitas pada dataset test memiliki pola yang sama, tetapi jumlahnya lebih kecil:
        - WALKING: 496 data
        - WALKING_DOWNSTAIRS: 420 data
        - LAYING: 537 data
    - Seperti Training Set, aktivitas LAYING tetap menjadi yang paling banyak, diikuti oleh WALKING, dan WALKING_DOWNSTAIRS yang paling sedikit.

    Dalam visualisasi ini, dapat dilihat bahwasanya distribusi 3 kelas aktivitas ini sudah cukup balance sehingga tidak perlu dilakukan teknik balancing seperti undersampling atau oversampling.

```
# Filter gyroscope features
print("Menyiapkan fitur gyroscope...")
gyro_features = [col for col in X_train.columns if 'Gyro' in
col and ('min()' in col or 'std()' in col)]
X_train_gyro = X_train[gyro_features]
X_test_gyro = X_test[gyro_features]

# Visualisasi karakteristik fitur gyroscope untuk setiap
aktivitas
print("\nVisualisasi karakteristik fitur gyroscope untuk setiap
aktivitas...")
plt.figure(figsize=(20, 10))
for i, feature in enumerate(gyro_features[:6]):  # Ambil hingga
6 fitur pertama gyroscope
    plt.subplot(2, 3, i + 1)
        for activity, label in zip([1, 3, 6], ['WALKING',
'WALKING_DOWNSTAIRS', 'LAYING']):
        # Filter data untuk aktivitas tertentu
            activity_data = X_train_gyro[y_train.iloc[:, 0] ==
activity][feature]
        sns.kdeplot(data=activity_data, label=label, fill=True,
alpha=0.4)
    plt.title(f'Distribusi Fitur: {feature}')
    plt.xlabel(feature)
    plt.ylabel('Density')
    plt.legend(title='Aktivitas')

plt.tight_layout()
plt.show()
```

```
Menyiapkan fitur gyroscope...

Visualisasi karakteristik fitur gyroscope untuk setiap
aktivitas…
```



Langkah selanjutnya adalah proses analisis fitur gyroscope:

1. Penyaringan Fitur Gyroscope

- Langkah pertama adalah memfilter kolom fitur yang terkait dengan sensor gyroscope. Kolom yang dipilih mengandung kata kunci *"Gyro"* serta karakteristik statistik tertentu seperti *min()* (nilai minimum) atau *std()* (standar deviasi).
- Dari *training set* dan *test set*, fitur yang memenuhi kriteria ini diekstrak menjadi subset data baru bernama X_train_gyro dan X_test_gyro.
- Penyaringan ini bertujuan untuk fokus pada karakteristik gyroscope yang relevan dengan pola aktivitas manusia.

2. Visualisasi Distribusi Fitur Gyroscope

- Program melakukan visualisasi distribusi nilai fitur gyroscope pada tiga aktivitas utama: *WALKING*, *WALKING_DOWNSTAIRS*, dan *LAYING*.
- Sebanyak 6 fitur pertama dari fitur gyroscope yang telah dipilih divisualisasikan menggunakan grafik *Kernel Density Estimation* (KDE). Grafik ini menunjukkan distribusi probabilitas fitur untuk setiap aktivitas.
- Setiap subplot menunjukkan distribusi satu fitur gyroscope dengan perbandingan aktivitas.

3. Detail Visualisasi

- Grafik terbagi menjadi 6 subplot (2 baris x 3 kolom). Setiap subplot merepresentasikan satu fitur gyroscope.
- Pada setiap subplot:
  - Sumbu X mewakili nilai fitur gyroscope.
  - Sumbu Y mewakili *density* atau kepadatan nilai fitur.

- Tiga garis berwarna menggambarkan distribusi fitur untuk tiga aktivitas (*WALKING*, *WALKING_DOWNSTAIRS*, dan *LAYING*), dengan area bawah kurva diwarnai (*fill=True*) untuk memberikan visualisasi lebih jelas.

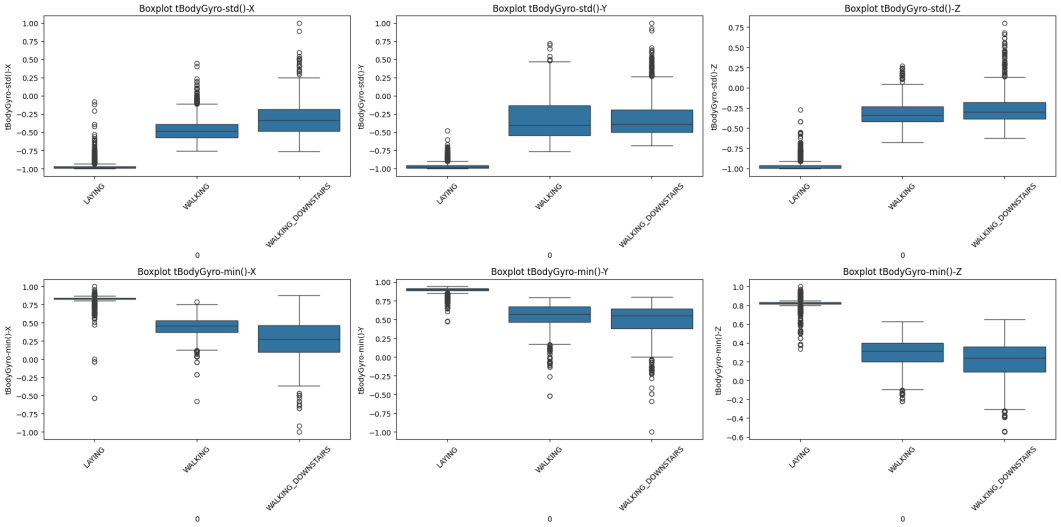
Interpretasi Visualisasi

- Fitur pertama menunjukkan bahwa aktivitas LAYING memiliki distribusi yang lebih sempit (konsisten), sedangkan WALKING_DOWNSTAIRS menunjukkan variasi lebih besar.
- Fitur lain menunjukkan tumpang tindih antara distribusi WALKING dan WALKING_DOWNSTAIRS, yang dapat menjadi tantangan dalam proses klasifikasi.
- Grafik distribusi membantu memahami pola gyroscope yang khas untuk masing-masing aktivitas.
- Pola distribusi ini dapat membantu proses pengklasifikasian aktivitas di tahap berikutnya. Distribusi yang jelas dan tidak tumpang tindih menunjukkan fitur yang dapat membedakan aktivitas dengan baik.

```
# Filter aktivitas
print("Memfilter aktivitas yang dipilih...")
selected_activities = [1, 3, 6]
train_mask = y_train[0].isin(selected_activities)
test_mask = y_test[0].isin(selected_activities)

X_train_filtered = X_train_gyro[train_mask]
y_train_filtered = y_train[train_mask]
X_test_filtered = X_test_gyro[test_mask]
y_test_filtered = y_test[test_mask]

# Map labels
activity_map = {
    1: 'WALKING',
    3: 'WALKING_DOWNSTAIRS',
    6: 'LAYING'
}

y_train_filtered[0]                                                    =
pd.Series(y_train_filtered[0]).map(activity_map)
```

```
y_test_filtered[0]                                                    =
pd.Series(y_test_filtered[0]).map(activity_map)

# 5. Visualisasi data sebelum preprocessing
print("\nVisualisasi data sebelum preprocessing...")
plt.figure(figsize=(20, 10))
for i, feature in enumerate(gyro_features[:6]):
    plt.subplot(2, 3, i+1)
                sns.boxplot(data=X_train_filtered,    y=feature,
x=y_train_filtered[0])
    plt.title(f'Boxplot {feature}')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Print informasi data
print("\nUkuran data sebelum preprocessing:")
print(f"X_train_filtered shape: {X_train_filtered.shape}")
print(f"y_train_filtered shape: {y_train_filtered.shape}")
print(f"X_test_filtered shape: {X_test_filtered.shape}")
print(f"y_test_filtered shape: {y_test_filtered.shape}")
```

```
Memfilter aktivitas yang dipilih...

Visualisasi data sebelum preprocessing...
```




```
Ukuran data sebelum preprocessing:
X_train_filtered shape: (3619, 26)
```

```
y_train_filtered shape: (3619, 1)
X_test_filtered shape: (1453, 26)
y_test_filtered shape: (1453, 1)
```

Langkah selanjutnya adalah:

1. **Memfilter Aktivitas yang Dipilih**:
   - Program dimulai dengan memfilter data berdasarkan tiga jenis aktivitas yang dipilih: *WALKING* (1), *WALKING_DOWNSTAIRS* (3), dan *LAYING* (6). Aktivitas ini disimpan dalam variabel selected_activities.
   - Masking dilakukan pada dataset y_train dan y_test menggunakan metode .isin(selected_activities) untuk memilih baris yang hanya berisi aktivitas yang relevan.
   - Data fitur (X_train_gyro dan X_test_gyro) dan label (y_train dan y_test) difilter berdasarkan mask tersebut, menghasilkan dataset baru X_train_filtered, y_train_filtered, X_test_filtered, dan y_test_filtered.
2. **Mapping Label Aktivitas**:
   - Program membuat *mapping* label numerik (1, 3, 6) menjadi label string yang lebih mudah dipahami menggunakan activity_map:
     - 1 → *WALKING*
     - 3 → *WALKING_DOWNSTAIRS*
     - 6 → *LAYING*
   - Dataset label (y_train_filtered dan y_test_filtered) diperbarui menggunakan pd.Series.map(activity_map) sehingga label numerik diganti dengan string sesuai *mapping*.
3. **Visualisasi Data Sebelum *Preprocessing***:
   - Program memvisualisasikan distribusi enam fitur pertama dari dataset *gyroscope* (gyro_features[:6]) menggunakan *boxplot*. Grafik ini dibuat untuk menunjukkan persebaran nilai fitur berdasarkan aktivitas.
   - Grafik dibuat dalam figur berukuran 20 x 10 dengan enam *subplot* dalam dua baris.
   - Setiap *subplot* menggunakan sns.boxplot, dengan sumbu X adalah label aktivitas (dari y_train_filtered) dan sumbu Y adalah nilai fitur terkait.
   - Rotasi sumbu X (aktivitas) diatur menjadi 45 derajat untuk mempermudah membaca.
4. **Informasi Ukuran Data**:
   - Setelah pemfilteran, program mencetak ukuran dataset *training* dan *test* yang telah difilter:
     - X_train_filtered memiliki 3.619 baris dan 26 kolom fitur.

- y_train_filtered memiliki 3.619 baris dan 1 kolom label.
- X_test_filtered memiliki 1.453 baris dan 26 kolom fitur.
- y_test_filtered memiliki 1.453 baris dan 1 kolom label.

Deskripsi Grafik:

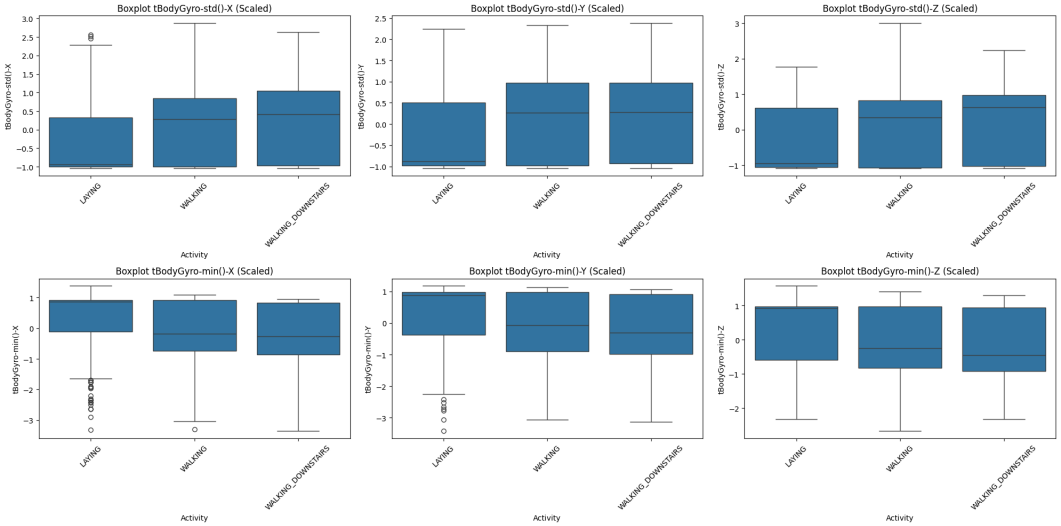
- Grafik ini bertujuan menunjukkan persebaran nilai enam fitur pertama dari data *gyroscope* sebelum dilakukan *preprocessing*, untuk setiap aktivitas (*WALKING*, *WALKING_DOWNSTAIRS*, dan *LAYING*). Disini grafik menunjukkan bahwasanya masih terdapat *outliers* dalam persebaran nilai dari tiap fitur dataset ini.
- **Karakteristik Grafik**:
  - Terdapat enam *subplot*:
    - Setiap *subplot* adalah *boxplot* yang menggambarkan persebaran nilai suatu fitur untuk tiga aktivitas.
    - Sumbu X menunjukkan tiga aktivitas yang difilter (dalam bentuk string hasil *mapping*).
    - Sumbu Y menunjukkan nilai numerik dari fitur terkait.
  - Judul pada setiap *subplot* berupa nama fitur (misalnya, *Boxplot X-Axis Gyro Mean* jika fitur pertama adalah rata-rata gyroscope pada sumbu X).

4.4 *Step 4: Data Preprocessing*

```
# Preprocessing
print("Melakukan preprocessing data...")

# Hitung Q1 dan Q3 untuk setiap fitur
q1 = X_train_filtered.quantile(0.25)
q3 = X_train_filtered.quantile(0.75)
iqr = q3 - q1

# Tentukan batas bawah dan atas
lower = q1 - 1.5 * iqr
upper = q3 + 1.5 * iqr

# Hapus outlier dari data
X_train_no_outliers = X_train_filtered[(X_train_filtered < upper).all(axis=1) & (X_train_filtered > lower).all(axis=1)]
y_train_no_outliers = y_train_filtered[(X_train_filtered < upper).all(axis=1) & (X_train_filtered > lower).all(axis=1)]
```

```
X_test_no_outliers    =    X_test_filtered[(X_test_filtered    <
upper).all(axis=1) & (X_test_filtered > lower).all(axis=1)]
y_test_no_outliers    =    y_test_filtered[(X_test_filtered    <
upper).all(axis=1) & (X_test_filtered > lower).all(axis=1)]

# Standarisasi data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_no_outliers)
X_test_scaled = scaler.transform(X_test_no_outliers)

# Visualisasi data setelah preprocessing
print("\nVisualisasi data setelah preprocessing...")
plt.figure(figsize=(20, 10))
scaled_df = pd.DataFrame(X_train_scaled, columns=gyro_features)
scaled_df['Activity'] = y_train_no_outliers[0]
for i, feature in enumerate(gyro_features[:6]):
    plt.subplot(2, 3, i+1)
    sns.boxplot(data=scaled_df, y=feature, x='Activity')
    plt.title(f'Boxplot {feature} (Scaled)')
    plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Print informasi data setelah preprocessing
print("\nUkuran data setelah preprocessing:")
print(f"X_train_scaled shape: {X_train_scaled.shape}")
print(f"y_train_no_outliers                                shape:
{y_train_no_outliers.shape}")
print(f"X_test_scaled shape: {X_test_scaled.shape}")
print(f"y_test_no_outliers shape: {y_test_no_outliers.shape}")
```

```
Melakukan preprocessing data...

Visualisasi data setelah preprocessing…
```




```
Ukuran data setelah preprocessing:
X_train_scaled shape: (2928, 26)
y_train_no_outliers shape: (2928, 1)
X_test_scaled shape: (1233, 26)
y_test_no_outliers shape: (1233, 1)
```

Langkah selanjutnya adalah tahap *Data Preprocessing*, tujuan dari langkah ini adalah untuk membersihkan dan menyiapkan data sebelum melanjutkan ke tahap pemodelan lebih lanjut.

1. **Proses *Preprocessing***:
    - **Menghitung Kuartil 1 (Q1) dan Kuartil 3 (Q3)**:
        - Program menghitung Q1 dan Q3 untuk setiap fitur dalam dataset *training* (X_train_filtered) menggunakan fungsi .quantile(). Q1 adalah persentil ke-25, sementara Q3 adalah persentil ke-75.
        - Selisih antara Q3 dan Q1 disebut *Interquartile Range* (IQR), yang dihitung untuk mendeteksi outlier.
    - **Menentukan Batas Bawah dan Atas**:
        - Batas bawah dihitung dengan rumus Q1 - 1.5 * IQR, sedangkan batas atas dihitung dengan rumus Q3 + 1.5 * IQR. Nilai-nilai di luar batas ini dianggap sebagai outlier.
    - **Menghapus Outlier**:
        - Data yang berada di luar batas bawah atau atas dihapus. Hal ini dilakukan pada dataset *training* (X_train_filtered) dan test (X_test_filtered) menggunakan logika (X < upper).all(axis=1) & (X > lower).all(axis=1).

        - Dataset hasilnya disimpan dalam variabel baru: X_train_no_outliers, y_train_no_outliers, X_test_no_outliers, dan y_test_no_outliers.
    - **Standarisasi Data**:
        - Standarisasi dilakukan menggunakan StandardScaler dari sklearn, yang mengubah data sehingga memiliki rata-rata 0 dan standar deviasi 1.
        - Dataset *training* distandarisasi menggunakan scaler.fit_transform(), sedangkan dataset *test* hanya diubah dengan scaler.transform() untuk mencegah *data leakage*.
        - Data yang sudah distandarisasi disimpan dalam X_train_scaled dan X_test_scaled.
2. **Visualisasi Data Setelah *Preprocessing***:
    - Grafik dibuat untuk memvisualisasikan enam fitur pertama dari dataset yang telah distandarisasi (X_train_scaled).
    - Data distandarisasi dimasukkan ke dalam *DataFrame* baru (scaled_df) yang dilengkapi dengan label aktivitas (Activity).
    - Enam *boxplot* dibuat (satu untuk setiap fitur) menggunakan sns.boxplot untuk menampilkan persebaran nilai fitur berdasarkan aktivitas (*WALKING*, *WALKING_DOWNSTAIRS*, dan *LAYING*).
    - Grafik ini membantu melihat distribusi data yang telah dinormalisasi setelah outlier dihapus.
3. **Mencetak Informasi Data**:
    - Setelah *preprocessing*, program mencetak ukuran dataset yang sudah diproses:
        - X_train_scaled: 2.928 baris dan 26 fitur.
        - y_train_no_outliers: 2.928 baris dengan 1 kolom label.
        - X_test_scaled: 1.233 baris dan 26 fitur.
        - y_test_no_outliers: 1.233 baris dengan 1 kolom label.

Deskripsi Grafik:

- Grafik menunjukkan persebaran nilai fitur yang telah distandarisasi untuk tiga aktivitas (*WALKING*, *WALKING_DOWNSTAIRS*, dan *LAYING*), setelah dilakukan preprocessing.
- **Karakteristik Grafik**:
    - Terdapat enam *subplot*, masing-masing adalah *boxplot* untuk fitur tertentu.
    - Sumbu X menunjukkan tiga aktivitas yang difilter, sedangkan sumbu Y menunjukkan nilai standar dari fitur terkait.

- Judul setiap *subplot* menunjukkan nama fitur (misalnya, *Boxplot X-Axis Gyro Mean (Scaled)* jika fitur pertama adalah rata-rata gyroscope pada sumbu X).

- **Interpretasi Grafik**:
  - Grafik menunjukkan data lebih seragam karena telah distandarisasi.
  - Outlier yang berlebihan telah dihapus, sehingga distribusi data lebih merata.
  - Beberapa fitur mungkin tetap menunjukkan variasi distribusi berdasarkan aktivitas, yang mencerminkan pola unik setiap aktivitas.

```
# Prepare labels for neural network
print("Preparing labels for neural network...")
le = LabelEncoder()
y_train_encoded = le.fit_transform(y_train_no_outliers[0])
y_test_encoded = le.transform(y_test_no_outliers[0])

def to_one_hot(y, num_classes):
    return np.eye(num_classes)[y]

y_train_onehot          =          to_one_hot(y_train_encoded,
len(np.unique(y_train_encoded)))
y_test_onehot           =           to_one_hot(y_test_encoded,
len(np.unique(y_test_encoded)))

print("\nUkuran data final untuk neural network:")
print(f"X_train_scaled shape: {X_train_scaled.shape}")
print(f"y_train_onehot shape: {y_train_onehot.shape}")
print(f"X_test_scaled shape: {X_test_scaled.shape}")
print(f"y_test_onehot shape: {y_test_onehot.shape}")
```

```
Preparing labels for neural network...

Ukuran data final untuk neural network:
X_train_scaled shape: (2928, 26)
y_train_onehot shape: (2928, 3)
X_test_scaled shape: (1233, 26)
y_test_onehot shape: (1233, 3)
```

Langkah selanjutnya berfokus pada mempersiapkan label data untuk digunakan dalam model neural network.

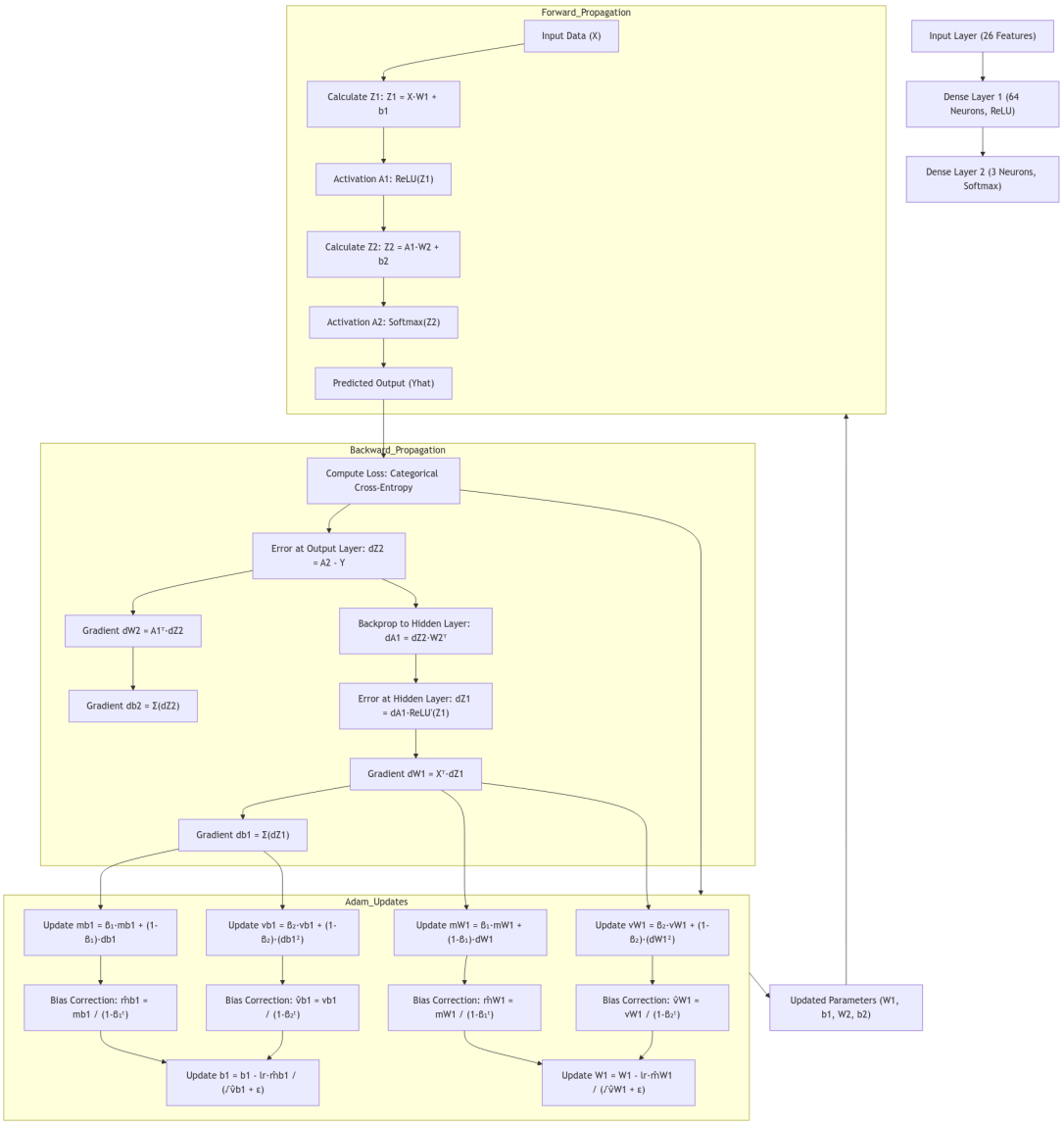
1. **Encoding Label**:

- Data label (y_train_no_outliers dan y_test_no_outliers) masih berbentuk teks atau kategori (*WALKING*, *WALKING_DOWNSTAIRS*, dan *LAYING*). Label ini diubah menjadi format numerik menggunakan *LabelEncoder* dari *sklearn*.
- Prosesnya adalah:
    - le.fit_transform() digunakan untuk mengonversi label pada data *training* menjadi nilai numerik: misalnya, *WALKING* menjadi 0, *WALKING_DOWNSTAIRS* menjadi 1, dan *LAYING* menjadi 2.
    - le.transform() digunakan untuk data *test*, mengikuti pola *encoding* yang sama seperti data *training*.
- Hasil dari langkah ini adalah dua array numerik: y_train_encoded dan y_test_encoded.

2. **Konversi ke Format *One-Hot Encoding***:
    - Label yang telah di-*encode* numerik dikonversi menjadi format *one-hot encoding* menggunakan fungsi to_one_hot().
    - *One-hot encoding* adalah representasi di mana setiap kategori diubah menjadi vektor biner dengan panjang sama dengan jumlah kategori. Contohnya, label numerik 0, 1, dan 2 akan menjadi:
        - 0 → [1, 0, 0]
        - 1 → [0, 1, 0]
        - 2 → [0, 0, 1]
    - Fungsi to_one_hot() menggunakan metode matriks identitas (np.eye()) untuk membuat representasi biner berdasarkan nilai kategori.
    - Hasilnya adalah dua array baru: y_train_onehot dan y_test_onehot, yang siap digunakan sebagai target untuk *neural network*.
3. **Mencetak Ukuran Data Final**:
    - Setelah semua persiapan selesai, program mencetak ukuran dataset final:
        - **X_train_scaled**: Dataset *training* dengan 2.928 baris dan 26 fitur.
        - **y_train_onehot**: Label data *training* dalam format *one-hot encoding* dengan 2.928 baris dan 3 kolom (satu kolom untuk setiap kategori aktivitas).
        - **X_test_scaled**: Dataset *test* dengan 1.233 baris dan 26 fitur.
        - **y_test_onehot**: Label data *test* dalam format *one-hot encoding* dengan 1.233 baris dan 3 kolom.

5. *MODELLING* (PEMBUATAN ARSITEKTUR *NEURAL NETWORK*)

5.1 *Step 5: Implement Neural Network from Scratch with Backpropagation*



Gambar arsitektur dapat diakses melalui tautan berikut:

Arsitektur dan Alur Kerja Neural Network.png

```
Neural Network Architecture Summary
================================================================================
Layer (Type)              Activation      Output Shape          Param #
================================================================================
Input Layer                    -              (None, 26)           0
Dense Layer 1                ReLU           (None, 64)           1728
Dense Layer 2                Softmax        (None, 3)            195
================================================================================
Total Trainable Parameters: 1,923
Additional Configurations:
- Dropout Rate: Not Implemented
- L2 Regularization (λ): 0.01
- Optimizer: Adam (β1=0.9, β2=0.999, ε=1e-08)
================================================================================
```

Gambar arsitektur dapat diakses melalui tautan berikut:

Neural Network Model Summary.jpg

```python
class EnhancedNeuralNetwork:
    def __init__(self, input_size, hidden_sizes, output_size, learning_rate, beta1=0.9, beta2=0.999, epsilon=1e-8):
        self.layers_dims = [input_size] + hidden_sizes + [output_size]
        self.learning_rate = learning_rate
        self.beta1 = beta1
        self.beta2 = beta2
        self.epsilon = epsilon
        self.parameters = {}
        self.m = {}
        self.v = {}
        self.activations = {}
        self.initialize_parameters()
        self.loss_history = []
        self.val_loss_history = []
        self.accuracy_history = []
        self.t = 0  # Time step for Adam optimizer

    def initialize_parameters(self):
        """Initialize weights using He initialization for ReLU"""
        for l in range(1, len(self.layers_dims)):
```

```
                                        self.parameters[f'W{l}'] =
np.random.randn(self.layers_dims[l-1], self.layers_dims[l]) *
np.sqrt(2. / self.layers_dims[l-1])
                        self.parameters[f'b{l}'] = np.zeros((1,
self.layers_dims[l]))
                            self.m[f'dW{l}'], self.m[f'db{l}'] =
np.zeros_like(self.parameters[f'W{l}']),
np.zeros_like(self.parameters[f'b{l}'])
                            self.v[f'dW{l}'], self.v[f'db{l}'] =
np.zeros_like(self.parameters[f'W{l}']),
np.zeros_like(self.parameters[f'b{l}'])

    def relu(self, x):
        """ReLU activation function"""
        return np.maximum(0, x)

    def relu_derivative(self, x):
        """Derivative of ReLU"""
        return (x > 0).astype(float)

    def softmax(self, x):
                """Softmax activation function with numerical
stability"""
        exps = np.exp(x - np.max(x, axis=1, keepdims=True))
        return exps / np.sum(exps, axis=1, keepdims=True)

    def forward(self, X, training=True):
        """Forward propagation"""
        self.activations['A0'] = X
        for l in range(1, len(self.layers_dims) - 1):
                        Z = np.dot(self.activations[f'A{l-1}'],
self.parameters[f'W{l}']) + self.parameters[f'b{l}']
            self.activations[f'Z{l}'] = Z
            self.activations[f'A{l}'] = self.relu(Z)

        # Output layer
        L = len(self.layers_dims) - 1
                        Z = np.dot(self.activations[f'A{L-1}'],
self.parameters[f'W{L}']) + self.parameters[f'b{L}']
```

```
        self.activations[f'Z{L}'] = Z
        self.activations[f'A{L}'] = self.softmax(Z)
        return self.activations[f'A{L}']

    def backward(self, X, y):
        """Backward propagation with Adam optimizer"""
        m = y.shape[0]
        L = len(self.layers_dims) - 1
        lambda_reg = 0.01

        dZ = self.activations[f'A{L}'] - y
        for l in reversed(range(1, L + 1)):
             dW = np.dot(self.activations[f'A{l-1}'].T, dZ) / m
+ (lambda_reg * self.parameters[f'W{l}']) / m
            db = np.sum(dZ, axis=0, keepdims=True) / m

            if l > 1:
                dA = np.dot(dZ, self.parameters[f'W{l}'].T)
                                                    dZ   =   dA   *
self.relu_derivative(self.activations[f'Z{l-1}'])

            # Adam updates
            self.t += 1
             self.m[f'dW{l}'] = self.beta1 * self.m[f'dW{l}'] +
(1 - self.beta1) * dW
             self.m[f'db{l}'] = self.beta1 * self.m[f'db{l}'] +
(1 - self.beta1) * db
             self.v[f'dW{l}'] = self.beta2 * self.v[f'dW{l}'] +
(1 - self.beta2) * (dW ** 2)
             self.v[f'db{l}'] = self.beta2 * self.v[f'db{l}'] +
(1 - self.beta2) * (db ** 2)

            # Bias correction
              m_hat_dw = self.m[f'dW{l}'] / (1 - self.beta1 **
self.t)
              m_hat_db = self.m[f'db{l}'] / (1 - self.beta1 **
self.t)
              v_hat_dw = self.v[f'dW{l}'] / (1 - self.beta2 **
self.t)
```

```
                v_hat_db = self.v[f'db{l}'] / (1 - self.beta2 **
self.t)

            # Update parameters
                self.parameters[f'W{l}'] -= self.learning_rate *
m_hat_dw / (np.sqrt(v_hat_dw) + self.epsilon)
                self.parameters[f'b{l}'] -= self.learning_rate *
m_hat_db / (np.sqrt(v_hat_db) + self.epsilon)

    def compute_loss(self, y_true, y_pred):
        """Compute categorical cross-entropy loss"""
        m = y_true.shape[0]
        loss = -np.mean(np.sum(y_true * np.log(y_pred + 1e-15),
axis=1))
        return loss

    def train(self, X, y, X_val, y_val, epochs, batch_size):
        """Training loop"""
        for epoch in range(epochs):
            indices = np.random.permutation(X.shape[0])
            X_shuffled, y_shuffled = X[indices], y[indices]

            for i in range(0, X.shape[0], batch_size):
                X_batch = X_shuffled[i:i + batch_size]
                y_batch = y_shuffled[i:i + batch_size]
                self.forward(X_batch)
                self.backward(X_batch, y_batch)

            # Metrics
            train_loss = self.compute_loss(y, self.forward(X))
                        val_loss  =  self.compute_loss(y_val,
self.forward(X_val))
            val_preds = np.argmax(self.forward(X_val), axis=1)
            val_true = np.argmax(y_val, axis=1)
            val_accuracy = np.mean(val_preds == val_true)

            self.loss_history.append(train_loss)
            self.val_loss_history.append(val_loss)
            self.accuracy_history.append(val_accuracy)
```

```
            if epoch % 100 == 0:
                                print(f"Epoch {epoch}: Train Loss =
{train_loss:.4f}, Val Loss = {val_loss:.4f}, Val Accuracy =
{val_accuracy:.4f}")

    def model_summary(self):
        """Print detailed model architecture summary"""
        print("Neural Network Architecture Summary")
        print("=" * 80)
              print(f"{'Layer (Type)':<25} {'Activation':<15}
{'Output Shape':<20} {'Param #':<15}")
        print("=" * 80)

        # Input Layer
             print(f"Input Layer{'':<19} {'-':<15} {str((None,
self.layers_dims[0])):<20} {'0':<15}")

        total_params = 0
        for l in range(1, len(self.layers_dims)):
            # Calculate parameters
                              weights = self.layers_dims[l-1] *
self.layers_dims[l]  # Weight matrix
            biases = self.layers_dims[l]  # Bias vector
            layer_params = weights + biases
            total_params += layer_params

            # Determine activation
                               activation = "Softmax" if l ==
len(self.layers_dims) - 1 else "ReLU"

            # Print layer details
                  print(f"Dense Layer {l:<15} {activation:<15}
{str((None, self.layers_dims[l])):<20} {layer_params:<15}")

        print("=" * 80)
        print(f"Total Trainable Parameters: {total_params:,}")
        print("Additional Configurations:")
```

```
            print(f"- Dropout Rate: {'Not Implemented' if not
hasattr(self, 'dropout_rate') else '20% (during training
only)'}")
         print(f"- L2 Regularization (λ): {0.01 if 'lambda_reg'
in globals() or locals() else 'Not Set'}")
                print(f"- Optimizer: Adam (β1={self.beta1},
β2={self.beta2}, ε={self.epsilon})")
        print("=" * 80)
```

Langkah selanjutnya adalah implementasi dari kelas *EnhancedNeuralNetwork*, yaitu sebuah model *neural network* yang dilengkapi dengan berbagai fitur untuk pelatihan, seperti inisialisasi parameter yang optimal, *forward propagation*), *backward propagation*, pengoptimalan, dan lainnya.

1. Inisialisasi Model

- Ketika objek kelas ini dibuat, parameter penting seperti ukuran input, ukuran lapisan tersembunyi, ukuran output, laju pembelajaran (*learning rate*), dan parameter optimizer (*Adam optimizer*: β1, β2, dan ε) diatur.
- Arsitektur model didefinisikan melalui layers_dims, yang merupakan daftar ukuran setiap lapisan (input → lapisan tersembunyi → output).
- Parameter seperti bobot (*weights* W) dan bias (*bias* b) diinisialisasi menggunakan metode *He initialization*, yang dirancang khusus untuk lapisan dengan fungsi aktivasi ReLU.
- *Adam optimizer* memerlukan dua momen (momentum): *m* untuk rata-rata gradien pertama dan *v* untuk rata-rata gradien kedua, yang juga diinisialisasi di sini.

2. Fungsi Aktivasi

- **ReLU (Rectified Linear Unit)** digunakan di lapisan tersembunyi, yang memetakan nilai negatif menjadi nol dan membiarkan nilai positif apa adanya.
- **Softmax** digunakan di lapisan output untuk menghasilkan probabilitas kelas yang dijamin bernilai antara 0 dan 1, dengan jumlah total 1.

3. *Forward Propagation*

- Data masukan (X) dikalikan dengan bobot, ditambahkan bias, dan melalui fungsi aktivasi untuk menghasilkan output di setiap lapisan.
- Lapisan tersembunyi menggunakan fungsi ReLU, sedangkan lapisan akhir menggunakan Softmax.
- Output akhir berupa probabilitas prediksi untuk setiap kelas.

4. *Backward Propagation*

- *Backward propagation* menghitung gradien dari fungsi loss terhadap parameter model untuk setiap lapisan, dimulai dari lapisan output ke lapisan pertama (metode *chain rule*).

- Untuk mengurangi overfitting, dilakukan regularisasi L2 dengan parameter $\lambda = 0.01$.
- Gradien diperbarui menggunakan algoritma *Adam optimizer*, yang mengkombinasikan *momentum* dan *adaptive learning rate*. Ini mencakup:
    - Pembaruan rata-rata gradien pertama (m) dan kedua (v).
    - Koreksi bias pada momen pertama dan kedua.
    - Pembaruan bobot dan bias berdasarkan gradien yang telah diperbaiki.

5. *Fungsi Loss*

- Fungsi loss menggunakan *categorical cross-entropy*, yang menghitung perbedaan antara distribusi probabilitas prediksi model dan label sebenarnya. Loss ini cocok untuk klasifikasi multi-kelas.

6. Pelatihan Model

- Model dilatih melalui beberapa epoch menggunakan *mini-batch gradient descent*. Data diacak sebelum dibagi menjadi batch.
- Pada setiap iterasi, dilakukan:
    - Forward Propagation untuk menghitung prediksi.
    - Backward Propagation untuk memperbarui parameter.
- Setelah setiap epoch, loss pada data pelatihan dan validasi dihitung, serta akurasi validasi dihitung dengan membandingkan prediksi dan label.

```
# Train model dengan data validasi
model = EnhancedNeuralNetwork(
    input_size=X_train_scaled.shape[1],
    hidden_sizes=[64],
    output_size=len(np.unique(y_train_encoded)),
    learning_rate=0.0001
)

# Tampilkan summary model
model.model_summary()

# Training dengan batch size
print("\nTraining model...")
model.train(
    X_train_scaled, y_train_onehot,
    X_test_scaled, y_test_onehot,
    epochs=5000,
    batch_size=16,
```

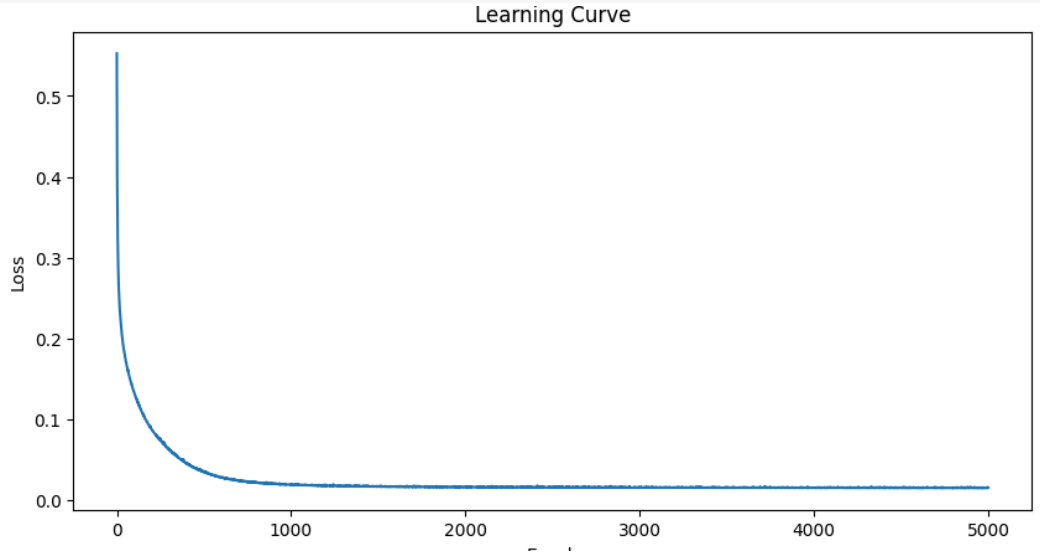
```
)

# Plot learning curve
plt.figure(figsize=(10, 5))
plt.plot(model.loss_history)
plt.title('Learning Curve')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```

```
Neural Network Architecture Summary
=================================================================
=================
Layer (Type)                    Activation          Output Shape
Param #
=================================================================
=================
Input Layer                        -                (None, 26)
0
Dense Layer 1                   ReLU                (None, 64)
1728
Dense Layer 2                   Softmax             (None, 3)
195
=================================================================
=================
Total Trainable Parameters: 1,923
Additional Configurations:
- Dropout Rate: Not Implemented
- L2 Regularization (λ): 0.01
- Optimizer: Adam (β1=0.9, β2=0.999, ε=1e-08)
=================================================================
=================

Training model...
Epoch 0: Train Loss = 0.5525, Val Loss = 0.6080, Val Accuracy =
0.7242
Epoch 100: Train Loss = 0.1325, Val Loss = 0.2550, Val Accuracy
= 0.8929
```

```
Epoch 200: Train Loss = 0.0875, Val Loss = 0.2460, Val Accuracy
= 0.8978
Epoch 300: Train Loss = 0.0624, Val Loss = 0.2494, Val Accuracy
= 0.8970
Epoch 400: Train Loss = 0.0452, Val Loss = 0.2510, Val Accuracy
= 0.8954
Epoch 500: Train Loss = 0.0351, Val Loss = 0.2562, Val Accuracy
= 0.8954
Epoch 600: Train Loss = 0.0278, Val Loss = 0.2547, Val Accuracy
= 0.8978
Epoch 700: Train Loss = 0.0239, Val Loss = 0.2568, Val Accuracy
= 0.8970
Epoch 800: Train Loss = 0.0221, Val Loss = 0.2620, Val Accuracy
= 0.8946
Epoch 900: Train Loss = 0.0202, Val Loss = 0.2580, Val Accuracy
= 0.8986
Epoch 1000: Train Loss = 0.0190, Val Loss = 0.2591, Val
Accuracy = 0.8962
Epoch 1100: Train Loss = 0.0183, Val Loss = 0.2571, Val
Accuracy = 0.8994
Epoch 1200: Train Loss = 0.0180, Val Loss = 0.2556, Val
Accuracy = 0.9027
Epoch 1300: Train Loss = 0.0173, Val Loss = 0.2571, Val
Accuracy = 0.9019
Epoch 1400: Train Loss = 0.0170, Val Loss = 0.2562, Val
Accuracy = 0.9011
Epoch 1500: Train Loss = 0.0169, Val Loss = 0.2515, Val
Accuracy = 0.9043
Epoch 1600: Train Loss = 0.0165, Val Loss = 0.2518, Val
Accuracy = 0.9019
Epoch 1700: Train Loss = 0.0171, Val Loss = 0.2558, Val
Accuracy = 0.8986
Epoch 1800: Train Loss = 0.0163, Val Loss = 0.2513, Val
Accuracy = 0.9059
Epoch 1900: Train Loss = 0.0162, Val Loss = 0.2478, Val
Accuracy = 0.9108
Epoch 2000: Train Loss = 0.0162, Val Loss = 0.2475, Val
Accuracy = 0.9100
```

```
Epoch 2100: Train Loss = 0.0161, Val Loss = 0.2503, Val Accuracy = 0.9011
Epoch 2200: Train Loss = 0.0161, Val Loss = 0.2491, Val Accuracy = 0.9100
Epoch 2300: Train Loss = 0.0158, Val Loss = 0.2520, Val Accuracy = 0.9019
Epoch 2400: Train Loss = 0.0158, Val Loss = 0.2518, Val Accuracy = 0.9027
Epoch 2500: Train Loss = 0.0156, Val Loss = 0.2507, Val Accuracy = 0.9075
Epoch 2600: Train Loss = 0.0155, Val Loss = 0.2502, Val Accuracy = 0.9043
Epoch 2700: Train Loss = 0.0155, Val Loss = 0.2510, Val Accuracy = 0.9035
Epoch 2800: Train Loss = 0.0157, Val Loss = 0.2545, Val Accuracy = 0.8994
Epoch 2900: Train Loss = 0.0155, Val Loss = 0.2530, Val Accuracy = 0.9019
Epoch 3000: Train Loss = 0.0155, Val Loss = 0.2515, Val Accuracy = 0.9067
Epoch 3100: Train Loss = 0.0153, Val Loss = 0.2530, Val Accuracy = 0.9027
Epoch 3200: Train Loss = 0.0153, Val Loss = 0.2531, Val Accuracy = 0.9043
Epoch 3300: Train Loss = 0.0153, Val Loss = 0.2534, Val Accuracy = 0.9051
Epoch 3400: Train Loss = 0.0152, Val Loss = 0.2536, Val Accuracy = 0.9043
Epoch 3500: Train Loss = 0.0154, Val Loss = 0.2533, Val Accuracy = 0.9059
Epoch 3600: Train Loss = 0.0158, Val Loss = 0.2601, Val Accuracy = 0.9002
Epoch 3700: Train Loss = 0.0152, Val Loss = 0.2571, Val Accuracy = 0.9011
Epoch 3800: Train Loss = 0.0154, Val Loss = 0.2549, Val Accuracy = 0.9043
Epoch 3900: Train Loss = 0.0152, Val Loss = 0.2574, Val Accuracy = 0.9019
```

```
Epoch 4000: Train Loss = 0.0151, Val Loss = 0.2555, Val Accuracy = 0.9002
Epoch 4100: Train Loss = 0.0153, Val Loss = 0.2535, Val Accuracy = 0.9035
Epoch 4200: Train Loss = 0.0151, Val Loss = 0.2559, Val Accuracy = 0.9011
Epoch 4300: Train Loss = 0.0151, Val Loss = 0.2547, Val Accuracy = 0.9043
Epoch 4400: Train Loss = 0.0151, Val Loss = 0.2565, Val Accuracy = 0.8986
Epoch 4500: Train Loss = 0.0152, Val Loss = 0.2582, Val Accuracy = 0.9002
Epoch 4600: Train Loss = 0.0150, Val Loss = 0.2562, Val Accuracy = 0.9035
Epoch 4700: Train Loss = 0.0154, Val Loss = 0.2548, Val Accuracy = 0.9059
Epoch 4800: Train Loss = 0.0154, Val Loss = 0.2549, Val Accuracy = 0.9035
Epoch 4900: Train Loss = 0.0151, Val Loss = 0.2562, Val Accuracy = 0.9027
```



Langkah selanjutnya adalah melatih sebuah *neural network* menggunakan data yang telah di-*scale* untuk memprediksi kategori dari data input.

1. **Inisialisasi Model**
   Sebuah objek model EnhancedNeuralNetwork diinisialisasi dengan parameter:

- input_size sesuai dengan jumlah fitur pada data (26 fitur dari X_train_scaled).
- Satu *hidden layer* dengan 64 neuron.
- Tiga neuron di lapisan keluaran (*output layer*), sesuai jumlah kelas dari label target (y_train_encoded).
- *Learning rate* sebesar 0.0001.

2. **Summary Model**
   Model mencetak arsitektur jaringan saraf:
   - Lapisan masukan memiliki 26 unit, tanpa fungsi aktivasi.
   - Lapisan tersembunyi menggunakan 64 unit dengan fungsi aktivasi ReLU.
   - Lapisan keluaran memiliki 3 unit dengan fungsi aktivasi Softmax. Total parameter yang dilatih adalah 1,923, termasuk bobot dan bias.
3. **Proses Pelatihan**
   - Model dilatih menggunakan *training data* (X_train_scaled dan y_train_onehot) serta divalidasi menggunakan *test data* (X_test_scaled dan y_test_onehot).
   - Pelatihan dilakukan selama 5000 *epoch* dengan ukuran *batch* 16.
   - Setiap *epoch*, model menghitung *train loss*, *validation loss*, dan *validation accuracy*. Nilai-nilai ini disimpan dalam *history* untuk analisis lebih lanjut.
   - Optimasi dilakukan menggunakan *Adam optimizer* dengan regularisasi L2 untuk mencegah *overfitting*.
4. **Plot Learning Curve**
   Grafik *learning curve* dibuat untuk memvisualisasikan perubahan *loss* selama pelatihan.

Penjelasan Grafik:
Grafik yang dihasilkan menunjukkan *learning curve* dengan sumbu-x sebagai jumlah *epoch* dan sumbu-y sebagai nilai *loss*. Berikut adalah penjelasan pola grafik:

- **Penurunan Awal yang Cepat**
  Di awal pelatihan, baik *train loss* maupun *validation loss* turun dengan cepat, menunjukkan bahwa model belajar dengan baik.
- **Konsistensi Loss Setelah Epoch Tertentu**
  Setelah beberapa *epoch* (~1000), *train loss* menjadi sangat kecil, lalu stabil. Hal ini menunjukkan bahwa model telah mencapai konvergensi tanpa *overfitting* yang signifikan.

6. CONTOH PERHITUNGAN MANUAL

Untuk perhitungan manual terlampir pada tautan berikut:

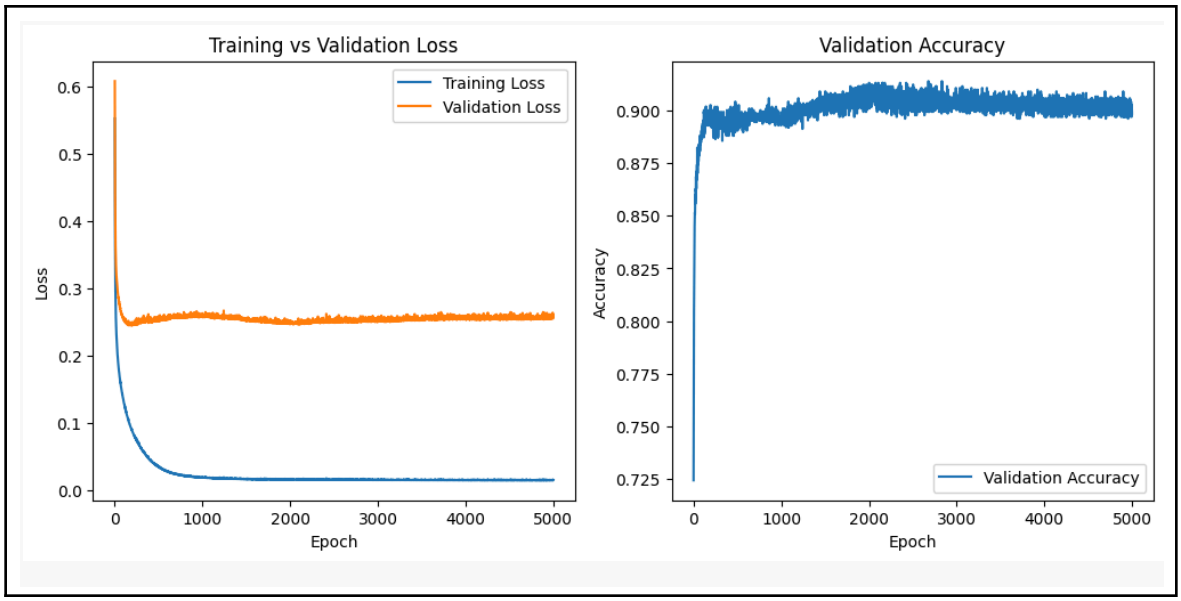
Sample Perhitungan Manual JST_Kelompok 9

Penjelasan: Perhitungan manual backpropagation adalah proses untuk mengupdate bobot dan bias dalam sebuah model jaringan saraf (neural network) secara manual dengan menggunakan algoritma **backpropagation**. Proses ini dilakukan untuk meminimalkan kesalahan (error) antara prediksi model dan target (nilai yang diharapkan). Backpropagation adalah bagian dari algoritma **Gradient Descent**, yang digunakan untuk optimasi dalam pelatihan jaringan saraf.. Fungsi dari perhitungan manual backpropagation adalah untuk memahami secara mendalam cara kerja algoritma pembelajaran pada jaringan saraf tiruan (neural network) dan untuk memastikan setiap langkah dari proses pembaruan parameter model dilakukan dengan benar.

7. HASIL ANALISIS PERFORMA MODEL

7.1 *Step 6: Evaluate the Model*

```
# Plot Training vs Validation Loss
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(model.loss_history, label='Training Loss')
plt.plot(model.val_loss_history, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.title('Training vs Validation Loss')

# Plot Training vs Validation Accuracy
plt.subplot(1, 2, 2)
plt.plot(model.accuracy_history, label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Validation Accuracy')
plt.show()
```



Langkah selanjutnya bertujuan untuk memvisualisasikan performa model selama proses pelatihan dengan menampilkan grafik perbandingan training loss dan validation loss serta grafik akurasi validasi.

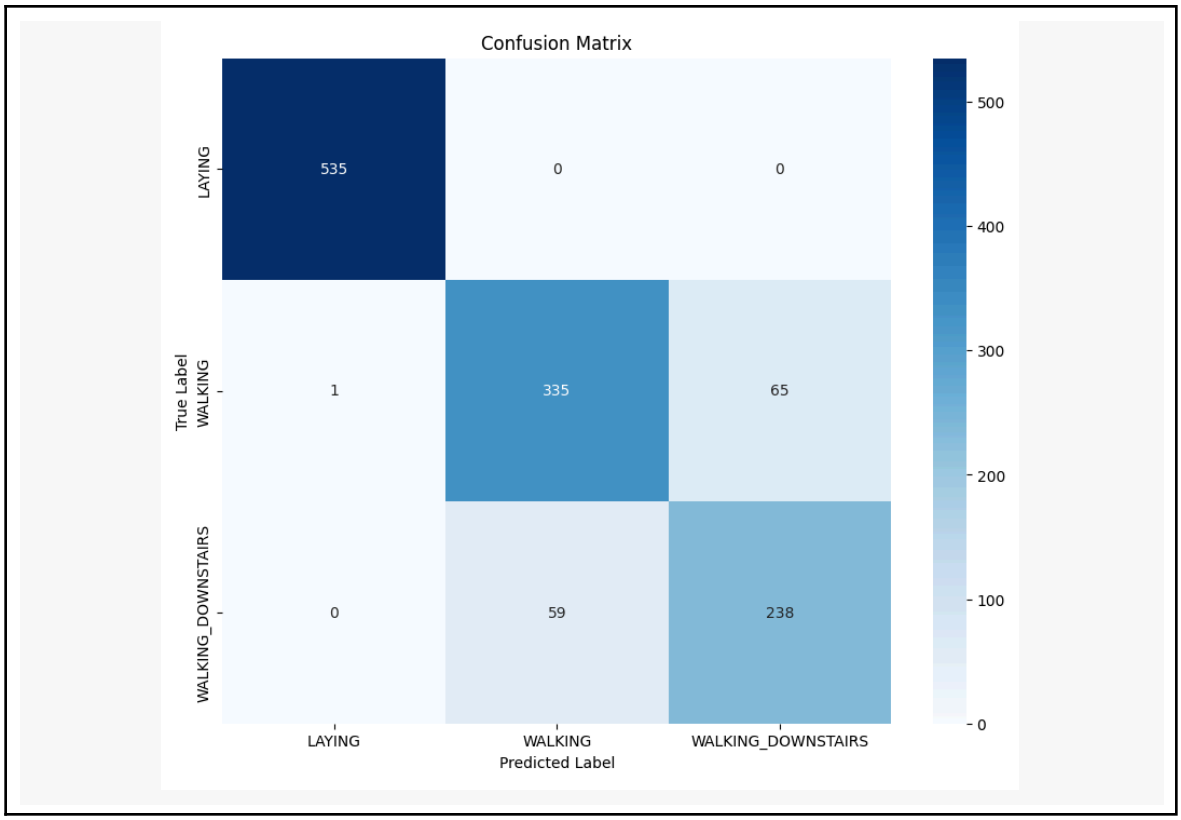
1. **Membuat Grafik *Training Loss* dan *Validation Loss***
   Grafik pertama dibuat untuk membandingkan *training loss* (kerugian selama pelatihan) dan *validation loss* (kerugian pada data validasi) terhadap jumlah *epoch*.
   - Sumbu-X: Jumlah *epoch*.
   - Sumbu-Y: Nilai *loss*.
   - Dua garis digambar: satu untuk *training loss* (dari model.loss_history) dan satu lagi untuk *validation loss* (dari model.val_loss_history).
   - Grafik ini membantu memantau apakah terjadi *overfitting* atau *underfitting*.
2. **Membuat Grafik Akurasi Validasi**
   Grafik kedua menunjukkan akurasi validasi terhadap jumlah *epoch*:
   - Sumbu-X: Jumlah *epoch*.
   - Sumbu-Y: Akurasi validasi (dari model.accuracy_history).
   - Grafik ini digunakan untuk memantau seberapa baik model dapat memprediksi data validasi selama pelatihan.
3. **Menampilkan Grafik**
   - Kedua grafik ditempatkan berdampingan (*subplot*) untuk mempermudah analisis.
   - Grafik pertama menunjukkan *loss* (*training* dan *validation*), sementara grafik kedua menunjukkan akurasi validasi saja.

Penjelasan Grafik

1. **Grafik *Training vs Validation Loss*:**
   - **Pola Penurunan Loss**
     Pada awal pelatihan, *training loss* dan *validation loss* menurun dengan cepat, menunjukkan bahwa model belajar dari data.
   - **Stabilitas di Akhir Pelatihan**
     Setelah beberapa *epoch*, *training loss* terus menurun hingga sangat kecil, sedangkan *validation loss* menjadi stabil. Jika *validation loss* meningkat sementara *training loss* terus menurun, ini dapat menunjukkan *overfitting*.
2. **Grafik Akurasi Validasi:**
   - **Peningkatan Awal**
     Akurasi validasi meningkat tajam selama tahap awal pelatihan, menunjukkan bahwa model menjadi lebih baik dalam memprediksi data validasi.
   - **Stabilitas Akhir**
     Setelah beberapa *epoch*, akurasi validasi cenderung stabil di sekitar 90%, menandakan model mencapai *generalization* yang baik terhadap data validasi.

```
# 10. Model Evaluation
print("Evaluasi model...")
y_pred = model.forward(X_test_scaled)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test_onehot, axis=1)

# Confusion Matrix
plt.figure(figsize=(10, 8))
cm = confusion_matrix(y_true_classes, y_pred_classes)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=le.classes_,
            yticklabels=le.classes_)
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```

```
Evaluasi model…
```



Langkah selanjutnya adalah mengevaluasi performa model yang telah dilatih pada data uji melalui prediksi kelas dan analisis dengan *confusion matrix*.

1. **Prediksi Kelas Output pada Data Uji**
    - Model menghasilkan prediksi probabilitas untuk setiap kelas pada data uji menggunakan fungsi forward. Output ini disimpan dalam variabel y_pred.
    - Kelas prediksi diperoleh dengan mencari indeks nilai probabilitas maksimum di setiap baris y_pred menggunakan np.argmax, menghasilkan variabel y_pred_classes (kelas yang diprediksi).
    - Kelas sebenarnya (true label) pada data uji juga diperoleh dengan cara yang sama dari representasi *one-hot encoding* data target (y_test_onehot), menghasilkan variabel y_true_classes.
2. **Membuat *Confusion Matrix***
    - Matriks kebingungan dihitung menggunakan confusion_matrix dari Scikit-learn, yang membandingkan y_true_classes (kelas sebenarnya) dengan y_pred_classes (kelas prediksi).
    - Matriks ini memberikan informasi tentang jumlah prediksi benar (*true positives*) dan salah (*false positives*, *false negatives*, dll.) untuk setiap kelas.
3. **Memvisualisasikan *Confusion Matrix***

- Matriks kebingungan divisualisasikan menggunakan *heatmap* dari Seaborn (sns.heatmap).
- Anotasi jumlah data pada setiap sel ditampilkan dengan parameter annot=True, dan kolom/baris dilabeli dengan nama kelas menggunakan xticklabels dan yticklabels yang memanfaatkan.

4. **Deskripsi Tampilan Grafik**
   - Grafik diberi judul "Confusion Matrix".
   - Sumbu X diberi label sebagai "Predicted Label" (kelas yang diprediksi oleh model), dan sumbu Y sebagai "True Label" (kelas sebenarnya dari data uji).

Confusion matrix pada visualisasi menggambarkan performa model dalam memprediksi tiga kelas aktivitas: **LAYING**, **WALKING**, dan **WALKING_DOWNSTAIRS**.

1. **Kinerja Model untuk Kelas "LAYING"**
   Model memiliki performa yang sangat baik untuk memprediksi aktivitas "LAYING". Semua data uji (sebanyak **535 instance**) berhasil diklasifikasikan dengan benar tanpa kesalahan, yang ditunjukkan oleh nilai diagonal yang tinggi (535) dan tidak adanya nilai prediksi pada kolom lainnya.
2. **Kinerja Model untuk Kelas "WALKING"**
   Untuk kelas "WALKING", model menghasilkan **335 prediksi benar** tetapi terdapat sejumlah kesalahan klasifikasi. Model salah memprediksi **65 instance** sebagai "WALKING_DOWNSTAIRS" dan **1 instance** sebagai "LAYING". Hal ini menunjukkan bahwa meskipun model cukup akurat untuk kelas ini, masih terdapat beberapa tumpang tindih dengan aktivitas "WALKING_DOWNSTAIRS".
3. **Kinerja Model untuk Kelas "WALKING_DOWNSTAIRS"**
   Pada kelas ini, model memprediksi dengan benar **238 instance** dari total data. Namun, terdapat **59 instance** yang salah diprediksi sebagai "WALKING". Hal ini menunjukkan bahwa model mengalami kesulitan untuk membedakan aktivitas "WALKING" dan "WALKING_DOWNSTAIRS", kemungkinan karena kemiripan fitur pada kedua aktivitas tersebut.
4. **Overall Performance**
   Keakuratan tinggi terlihat pada prediksi untuk kelas "LAYING", sementara performa untuk kelas "WALKING" dan "WALKING_DOWNSTAIRS" menunjukkan bahwa model masih perlu ditingkatkan, terutama untuk mengurangi kesalahan prediksi antar kedua kelas tersebut.

```
# Classification Report
print("Classification Report:")
print(classification_report(y_true_classes, y_pred_classes,
                            target_names=le.classes_))
```

```
Classification Report:
                    precision    recall  f1-score   support

            LAYING       1.00      1.00      1.00       535
           WALKING       0.85      0.84      0.84       401
WALKING_DOWNSTAIRS       0.79      0.80      0.79       297

          accuracy                           0.90      1233
         macro avg       0.88      0.88      0.88      1233
      weighted avg       0.90      0.90      0.90      1233
```

Langkah selanjutnya menghasilkan laporan klasifikasi (classification report) untuk mengevaluasi performa model berdasarkan metrik evaluasi klasifikasi.

1. **Menghitung *Classification Report***
    - Fungsi classification_report dari Scikit-learn digunakan untuk menghitung metrik evaluasi utama, yaitu *precision*, *recall*, *f1-score*, dan *support* untuk setiap kelas dalam dataset.
    - Input fungsi ini adalah:
        - y_true_classes: label sebenarnya dari data uji.
        - y_pred_classes: label prediksi yang dihasilkan oleh model.
        - target_names: nama kelas yang sesuai dengan label numerik, diperoleh dari le.classes_.
2. **Metrik Evaluasi dalam Laporan**

    Laporan klasifikasi mencakup metrik berikut:
    - **Precision**: Proporsi prediksi positif yang benar. Precision tinggi menunjukkan model jarang salah memprediksi kelas tertentu sebagai positif.
    - **Recall**: Proporsi sampel dari kelas tertentu yang berhasil dikenali dengan benar oleh model. Recall tinggi berarti model tidak banyak melewatkan sampel dari kelas tersebut.
    - **F1-Score**: Harmonik rata-rata dari *precision* dan *recall*. F1-score digunakan ketika keseimbangan antara *precision* dan *recall* diperlukan.
    - **Support**: Jumlah sampel aktual di setiap kelas.

3. **Akurasi Global dan Rata-Rata**
    - Akurasi: Proporsi total prediksi yang benar di seluruh kelas.
    - *Macro Avg*: Rata-rata metrik di semua kelas tanpa memperhatikan jumlah sampel di setiap kelas.
    - *Weighted Avg*: Rata-rata metrik di semua kelas, mempertimbangkan jumlah sampel di setiap kelas.

Analisis Performa Kelas Per Kelas

1. **LAYING**

- **Precision, Recall, dan F1-Score = 1.00:**
  Model mengenali aktivitas ini dengan sempurna tanpa kesalahan. Artinya, setiap prediksi untuk LAYING benar, dan semua data uji untuk LAYING terdeteksi dengan baik.
- **Support = 535:**
  Aktivitas ini memiliki jumlah sampel tertinggi di dataset, yang mungkin membantu model belajar pola dengan lebih baik.

2. **WALKING**

- **Precision = 0.85, Recall = 0.84, F1-Score = 0.84:**
  Model cukup baik mengenali aktivitas ini, meskipun ada kesalahan yang mengurangi performa.
- **Kesalahan Umum**: Prediksi model mungkin bingung antara WALKING dan aktivitas lain seperti WALKING_DOWNSTAIRS, karena keduanya memiliki pola gerakan yang mirip.
- **Support = 401**:
  Aktivitas ini memiliki jumlah data yang cukup seimbang, tetapi kesalahan bisa disebabkan oleh overlapping fitur antar kelas.

3. **WALKING_DOWNSTAIRS**

- **Precision = 0.79, Recall = 0.80, F1-Score = 0.79:**
  Performa model paling rendah pada kelas ini. Recall yang lebih rendah menunjukkan bahwa model gagal mengenali beberapa sampel dari kelas ini, mungkin karena data yang lebih sedikit atau pola yang lebih sulit dibedakan dari kelas lain.
- **Support = 297:**
  Jumlah data yang lebih sedikit dibandingkan LAYING dan WALKING kemungkinan membuat model kesulitan mempelajari pola dengan baik.

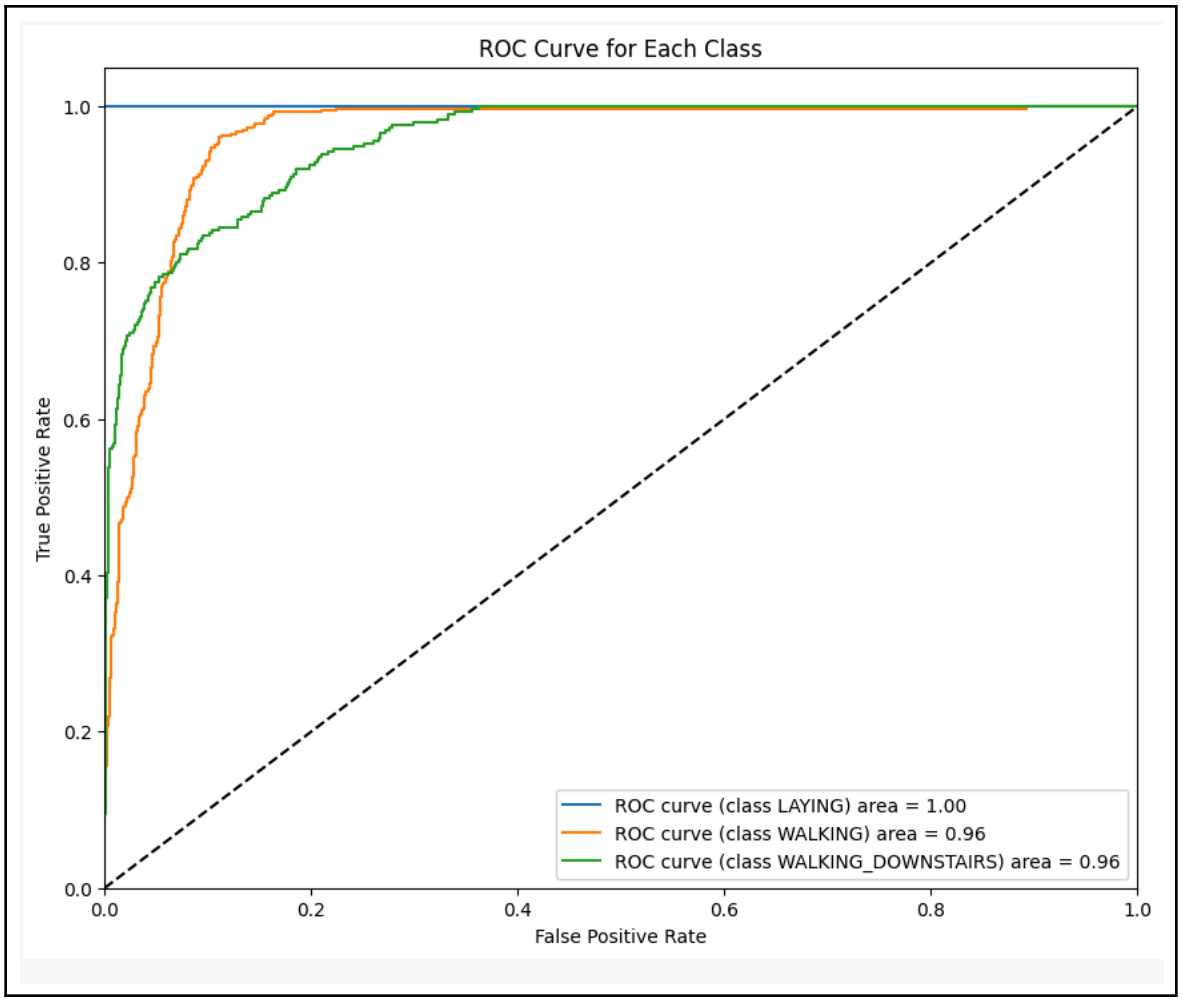
Analisis Performa Keseluruhan

1. **Akurasi Global = 0.90**
   Model mampu membuat prediksi yang benar untuk 90% dari total 1.233 sampel. Ini menunjukkan performa yang cukup baik.

2. **Macro Avg (Precision, Recall, F1-Score = 0.88)**
   Rata-rata performa antar kelas relatif tinggi, tetapi sedikit di bawah akurasi global. Hal ini menunjukkan adanya ketidakseimbangan performa antar kelas.

3. **Weighted Avg (Precision, Recall, F1-Score = 0.90)**
   Rata-rata berbobot ini konsisten dengan akurasi global karena mempertimbangkan jumlah sampel per kelas. Kelas LAYING yang memiliki performa sempurna mendominasi kontribusi ke rata-rata berbobot.

```
from sklearn.metrics import roc_curve, auc

# Plot ROC Curve
n_classes = y_train_onehot.shape[1]
plt.figure(figsize=(10, 8))
for i in range(n_classes):
    fpr, tpr, _ = roc_curve(y_test_onehot[:, i], y_pred[:, i])
    roc_auc = auc(fpr, tpr)
     plt.plot(fpr, tpr, label=f'ROC curve (class {le.classes_[i]})
area = {roc_auc:.2f}')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Each Class')
plt.legend(loc="lower right")
plt.show()
```



Langkah selanjutnya bertujuan untuk menghitung dan memvisualisasikan ROC (Receiver Operating Characteristic) Curve serta nilai AUC (Area Under the Curve) untuk setiap kelas dalam model klasifikasi. ROC Curve digunakan untuk mengevaluasi kemampuan model dalam membedakan kelas dengan berbagai ambang batas (threshold).

1. **Menginisialisasi Variabel**
   - n_classes: Jumlah kelas dalam dataset dihitung dari dimensi label satu-hot (y_train_onehot.shape[1]).
   - Grafik ROC akan dibuat untuk masing-masing kelas berdasarkan nilai probabilitas prediksi dari model.
2. **Menghitung *False Positive Rate* (FPR) dan *True Positive Rate* (TPR)**
   - Untuk setiap kelas, fungsi roc_curve menghitung FPR dan TPR berdasarkan probabilitas prediksi model (y_pred[:, i]) dibandingkan dengan label sebenarnya (y_test_onehot[:, i]).
   - FPR adalah rasio *false positives* terhadap semua sampel negatif.
   - TPR adalah rasio *true positives* terhadap semua sampel positif.

3. **Menghitung AUC (Area Under the Curve)**
    - Fungsi auc digunakan untuk menghitung luas di bawah kurva ROC untuk setiap kelas. AUC menggambarkan performa model: nilai AUC mendekati 1 menunjukkan performa baik, sedangkan AUC mendekati 0.5 menunjukkan model yang tidak lebih baik dari tebakkan acak.
4. **Memplot ROC Curve**
    - Setiap ROC Curve untuk tiap kelas digambarkan dengan warna berbeda.
    - Garis diagonal putus-putus ([0, 1], [0, 1]) adalah garis referensi yang menunjukkan prediksi acak.

Deskripsi Visualisasi:

ROC curve pada visualisasi menggambarkan kemampuan model dalam membedakan antara tiga kelas: **LAYING**, **WALKING**, dan **WALKING_DOWNSTAIRS:**

1. **Kelas "LAYING"**
   ROC curve untuk kelas "LAYING" menunjukkan performa sempurna, dengan nilai **AUC = 1.00**. Hal ini mengindikasikan bahwa model memiliki kemampuan 100% untuk memisahkan kelas "LAYING" dari kelas lainnya tanpa adanya kesalahan prediksi pada berbagai ambang batas (threshold). ROC curve ini mendekati sumbu kiri dan atas, mencerminkan akurasi yang sangat tinggi.
2. **Kelas "WALKING"**
   ROC curve untuk kelas "WALKING" memiliki nilai **AUC = 0.96**, yang mencerminkan performa model yang sangat baik, meskipun tidak sempurna. Grafik menunjukkan bahwa model cukup konsisten dalam membedakan kelas "WALKING" dari dua kelas lainnya, tetapi ada sedikit tumpang tindih dalam prediksi, yang dapat menyebabkan kesalahan pada ambang batas tertentu.
3. **Kelas "WALKING_DOWNSTAIRS"**
   ROC curve untuk kelas "WALKING_DOWNSTAIRS" juga memiliki nilai **AUC = 0.96**, yang mirip dengan kelas "WALKING". Ini menunjukkan performa model yang sangat baik dalam memprediksi kelas ini, tetapi ada beberapa tantangan dalam membedakannya dari kelas lain, khususnya "WALKING".
4. **Performa Keseluruhan**
   Garis diagonal putus-putus menunjukkan performa tebakkan acak (AUC = 0.5). Semua ROC curve berada jauh di atas garis ini, mengindikasikan bahwa model memiliki performa prediksi yang jauh lebih baik daripada prediksi acak. Nilai AUC yang tinggi untuk ketiga kelas mengindikasikan bahwa model secara keseluruhan mampu membedakan kelas dengan baik.

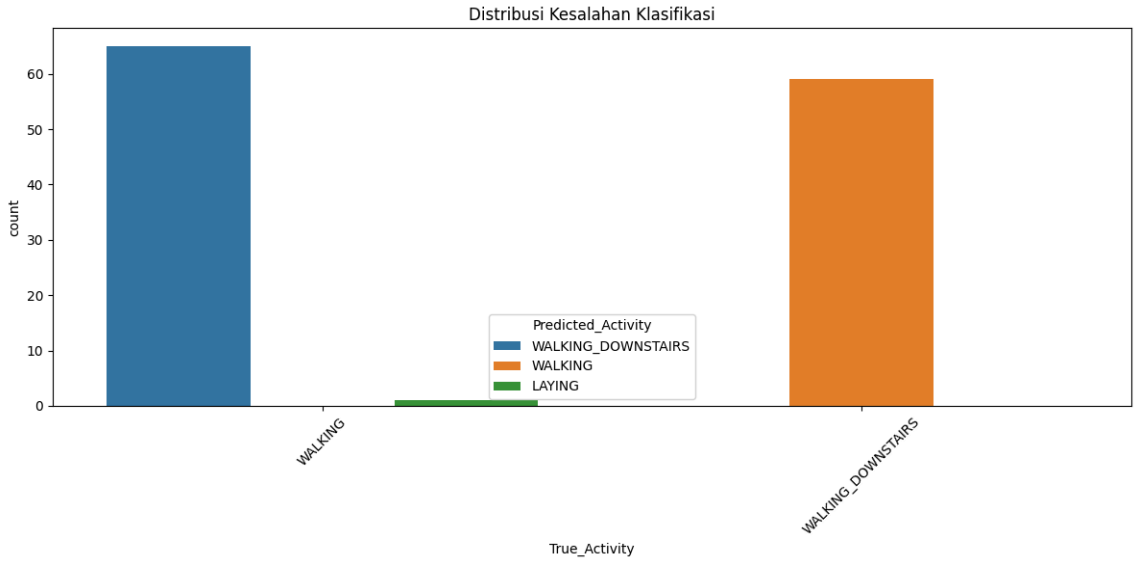
Kesimpulannya, model menunjukkan performa yang sangat baik untuk semua kelas, terutama pada kelas "LAYING" yang sempurna. Namun, model masih dapat ditingkatkan untuk mengurangi tumpang tindih antara kelas "WALKING" dan

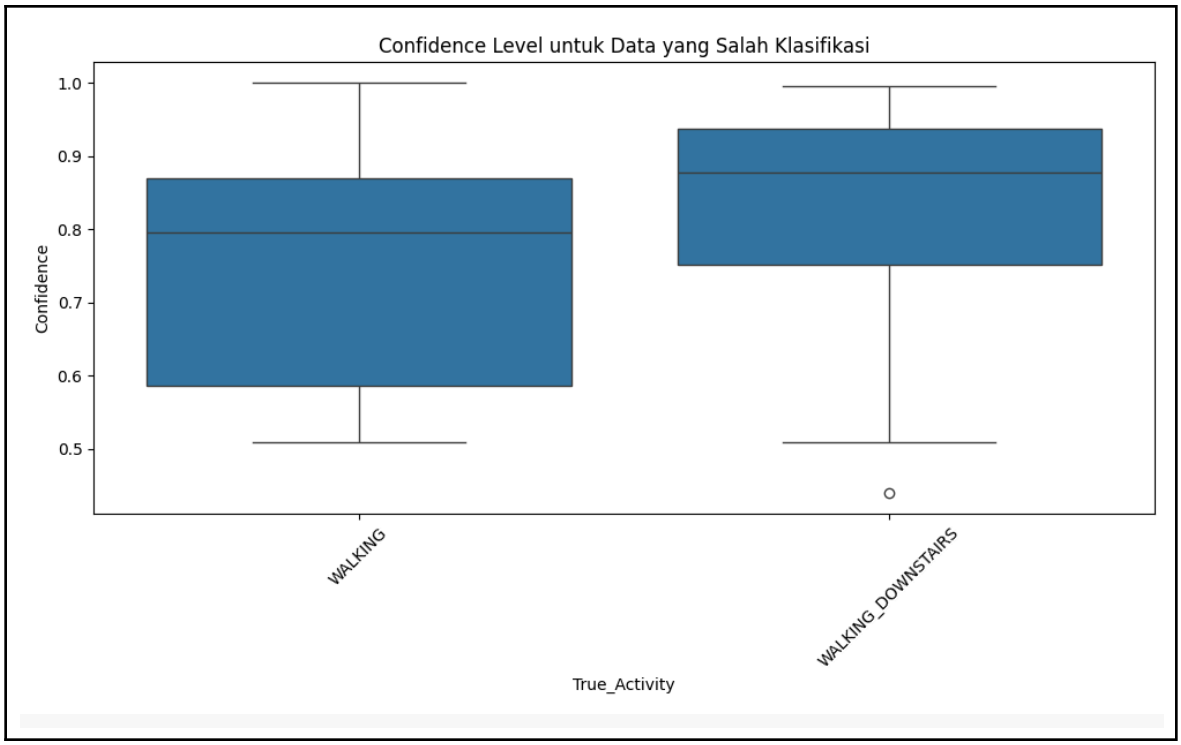
"WALKING_DOWNSTAIRS", yang terlihat dari AUC yang tidak mencapai 1.00 untuk kedua kelas tersebut.

```
# 11. Analisis Misclassification
print("Analisis Misclassification...")
misclassified_idx = np.where(y_pred_classes != y_true_classes)[0]
print(f"Jumlah data yang salah klasifikasi:
{len(misclassified_idx)}")

if len(misclassified_idx) > 0:
    print("\nDetail Misclassification")
    misclassified_data = pd.DataFrame({
                                                'True_Activity':
le.classes_[y_true_classes[misclassified_idx]],
                                              'Predicted_Activity':
le.classes_[y_pred_classes[misclassified_idx]],
        'Confidence': np.max(y_pred[misclassified_idx], axis=1)
    })

    print("Distribusi kesalahan klasifikasi:")
                print(misclassified_data.groupby(['True_Activity',
'Predicted_Activity']).size())

    # Visualisasi misclassification
    plt.figure(figsize=(12, 6))
         sns.countplot(data=misclassified_data,  x='True_Activity',
hue='Predicted_Activity')
    plt.title('Distribusi Kesalahan Klasifikasi')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
    print()

    # Analisis confidence level untuk misclassification
    plt.figure(figsize=(10, 6))
           sns.boxplot(data=misclassified_data,  x='True_Activity',
y='Confidence')
    plt.title('Confidence Level untuk Data yang Salah Klasifikasi')
    plt.xticks(rotation=45)
```

```
    plt.tight_layout()
    plt.show()
```

```
Analisis Misclassification...
Jumlah data yang salah klasifikasi: 125

Detail Misclassification
Distribusi kesalahan klasifikasi:
True_Activity        Predicted_Activity
WALKING              LAYING                1
                     WALKING_DOWNSTAIRS    65
WALKING_DOWNSTAIRS   WALKING               59
```





Langkah selanjutnya bertujuan untuk mengevaluasi data yang salah diklasifikasikan (misclassification) oleh model prediksi.

1. **Identifikasi Indeks Kesalahan:**
   - misclassified_idx menghitung indeks data di mana label sebenarnya (y_true_classes) berbeda dengan label prediksi model (y_pred_classes).
   - Program mencetak jumlah data yang salah klasifikasi, yaitu **125 data**.
2. **Detail Kesalahan Klasifikasi:**
   - Data yang salah klasifikasi dirangkum dalam sebuah *DataFrame* bernama misclassified_data, berisi:
     - **True_Activity**: Aktivitas sebenarnya.
     - **Predicted_Activity**: Aktivitas yang diprediksi model.
     - **Confidence**: Tingkat keyakinan model terhadap prediksinya (nilai probabilitas tertinggi).
3. **Distribusi Kesalahan:**
   - Distribusi pasangan kesalahan dicetak:
     - *WALKING* salah diklasifikasikan sebagai *LAYING*: 1 kasus.
     - *WALKING* salah diklasifikasikan sebagai *WALKING_DOWNSTAIRS*: 65 kasus.
     - *WALKING_DOWNSTAIRS* salah diklasifikasikan sebagai *WALKING*: 59 kasus.

Visualisasi Kesalahan Klasifikasi:

Grafik 1: Distribusi Kesalahan Klasifikasi

- Grafik batang (bar plot) menunjukkan jumlah kesalahan klasifikasi berdasarkan aktivitas sebenarnya (*True_Activity*) dan aktivitas yang diprediksi (*Predicted_Activity*).
- Sumbu X: Aktivitas sebenarnya.
- Sumbu Y: Jumlah kesalahan.
- *Legend*: Warna batang menunjukkan aktivitas prediksi.
- **Penjelasan:**
    - Grafik ini menunjukkan bahwa sebagian besar kesalahan klasifikasi terjadi antara aktivitas *WALKING* dan *WALKING_DOWNSTAIRS*.
    - Aktivitas *LAYING* salah diklasifikasikan sebagai *WALKING* hanya pada satu kasus, yang menunjukkan bahwa aktivitas ini relatif lebih mudah dibedakan.
    - Sebaliknya, aktivitas dinamis seperti *WALKING* dan *WALKING_DOWNSTAIRS* memiliki lebih banyak tumpang tindih, sehingga lebih sering terjadi kesalahan.

Grafik 2: Confidence Level pada Data Salah Klasifikasi

- Grafik kotak (*box plot*) menggambarkan tingkat keyakinan model terhadap prediksi pada data yang salah klasifikasi.
- Sumbu X: Aktivitas sebenarnya (*True_Activity*).
- Sumbu Y: Confidence level (probabilitas tertinggi dari prediksi model).
- **Penjelasan:**
    - Grafik menunjukkan tingkat kepercayaan model pada data yang salah diklasifikasikan.
    - Aktivitas dengan confidence tinggi pada kesalahan klasifikasi menunjukkan bahwa model sangat yakin terhadap prediksi yang ternyata salah. Ini mungkin menunjukkan adanya masalah pada data fitur atau model.
    - Aktivitas seperti *WALKING_DOWNSTAIRS* cenderung memiliki rentang confidence yang lebih besar, yang menunjukkan prediksi yang tidak konsisten.

8. KESIMPULAN

Berdasarkan evaluasi yang dilakukan, berikut kesimpulan performa model neural network yang telah dibangun:

a. Performa Keseluruhan

- Model mencapai akurasi global yang baik sebesar 90% dari total 1,233 sampel
- ROC curve menunjukkan performa yang sangat baik dengan nilai AUC di atas 0.96 untuk semua kelas
- Model berhasil konvergen dengan baik, ditunjukkan oleh learning curve yang stabil

b. Performa Per Kelas

- LAYING: Performa sempurna (100%) dengan precision, recall, dan F1-score = 1.0
- WALKING: Performa baik dengan F1-score 0.84
- WALKING_DOWNSTAIRS: Performa terendah dengan F1-score 0.79

c. Analisis Kesalahan

- Kesalahan klasifikasi paling sering terjadi antara aktivitas WALKING dan WALKING_DOWNSTAIRS:
  - 65 kasus WALKING salah diklasifikasi sebagai WALKING_DOWNSTAIRS
  - 59 kasus WALKING_DOWNSTAIRS salah diklasifikasi sebagai WALKING
- Hanya 1 kasus WALKING salah diklasifikasi sebagai LAYING
- Kesalahan klasifikasi ini menunjukkan bahwa model kesulitan membedakan aktivitas yang memiliki pola gerakan mirip

d. Rekomendasi Perbaikan

- Model dapat ditingkatkan terutama untuk membedakan antara aktivitas WALKING dan WALKING_DOWNSTAIRS
- Penambahan fitur yang lebih diskriminatif atau peningkatan arsitektur model mungkin diperlukan
- Penambahan data training untuk kelas WALKING_DOWNSTAIRS yang memiliki jumlah sampel paling sedikit (297) bisa membantu meningkatkan performa

Secara keseluruhan, model menunjukkan performa yang memuaskan untuk klasifikasi aktivitas manusia, dengan keunggulan khusus dalam mengenali aktivitas LAYING dan area yang perlu ditingkatkan dalam membedakan aktivitas berjalan yang mirip.