

# **Розпізнавання фізичної активності за допомогою моделі нейронної мережі**

## **Постановка задачі:**

- 1) Назбирати дата сет для декількох класів фізичної активності (збалансований), маркування даних і дата аналіз.
- 2) Використати декілька алгоритмів по feature selection і порівняти результати, які вони будуть повертати;
- 3) Поставити "правильні" активаційні функції для власних моделей;
- 4) Вивести метрики, порівняти результати метрик, зробити висновки;
- 5) Все на базі методологій;
- 6) Глянути до методів tensorflow чи pytorch.

## **Критерії валідації:**

- 1) Точність моделі нейронної мережі 80% і більше.

## Етап 1: Data Collection

Дані було зібрано за допомогою програми HyperIMU.



Налаштування параметрів запису даних наведено на рис. 1 та рис. 2.

← **HYPERIMU**

**Connection**

Protocol  
File

**Sampling rate (ms)**  
25

Server IP address  
192.168.197.1

Server Port number  
2055

**Persistent**  
Try to reconnect even if the Server is offline ☐

**Packet Configuration**

**Header**  
Configure packet's header

**Trailer**  
Configure packet's trailer

Рис. 1. Параметри запису даних акселерометра та гіроскопа

← **HYPERIMU**

File

Sampling rate (ms)  
10

Server IP address  
192.168.197.1

**Header**

☒ Timestamp

☐ MAC address

☐ Battery info

OK

Configure packet's header

**Trailer**  
Configure packet's trailer

**Prequel Packet**  
Transmit sensors names as first packet ☐

**HYPERIMU**

**Packet Preview**

**SENSOR 1**

Timestamp

**SENSOR 2**

bma4xyACCELEROMETER.x

bma4xyACCELEROMETER.y

bma4xyACCELEROMETER.z

**SENSOR 3**

oem-pseudo-gyro.x

oem-pseudo-gyro.y

oem-pseudo-gyro.z

EDIT OK

Рис. 2. Інформація про дані, які будуть записуватися

Із рис. 1 видно, що період запису даних становить 25 мс, тобто частота запису даних дорівнює  $1/25 * 10^{-3} = 40$  Гц.

Окрім даних з осей акселерометра та гіроскопа було також додано стовпець 'Timestamp', який потрібен буде для дослідження стабільності частоти забору даних.

Дані записувалися для п'яти різних видів фізичної активності:

- 1) Присідання;
- 2) Ходьба;
- 3) Легкий біг (Jogging);
- 4) Випади на кожную ногу (рис. 3);
- 5) Lateral Squat Slide (Переміщення центру маси з однієї ноги на іншу) (рис. 4).



Рис. 3. Випади на кожную ногу

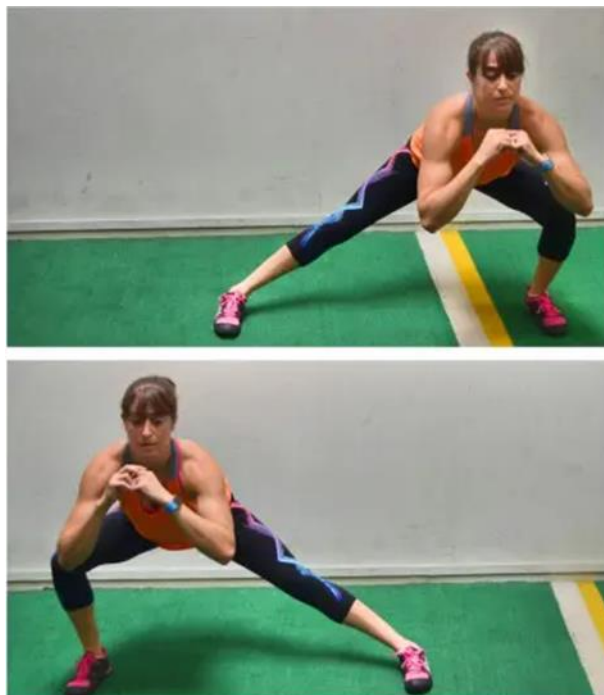


Рис. 4. Lateral squat slide

Під час запису даних смартфон знаходився у правій руці у горизонтальному положенні (вісь OZ акселерометра смартфона спрямована перпендикулярно до горизонталі).

Для кожного виду активності записувався окремий датасет/-и тривалістю більше 5 хв. Зокрема:

- 1) Присідання – 5 датасетів тривалістю понад 1 хв (приблизно 1 хв 15 с) кожен;
- 2) Ходьба – 1 датасет тривалістю 5.5 хв;
- 3) Легкий біг – 2 датасети тривалістю приблизно по 3 хв;
- 4) Випади на кожную ногу – 5 датасетів тривалістю приблизно по 1 хв 20 с кожен;
- 5) Lateral Squat Slide – 2 датасети тривалістю 1 хв 15 с + 2 датасети тривалістю приблизно 1 хв 45 с.

Усі вищезгадані датасети було об'єднано в один-єдиний тренувальний датасет.

Також для тестування було записано окремий датасет, який містить дані всіх видів активності (вправи виконувалися відразу одна після одної із невеликими перервами).

Отже, записані датасети містять сім стовпців: перший містить дані 'timestamp' у мс, наступні три відповідають даним трьох осей (OX, OY, OZ) акселерометра і виміряні у  $\text{м/с}^2$ , останні три стовпці – три осі гіроскопа (рад/с).

## Етап 2: Data Cleaning and Preprocessing

Оскільки датасети замірялися у різні дні, що було викликано неможливістю виконання вправ через втому м'язів, для дослідження стабільності частоти забору даних було створено окремий стовпець - 'time', який містить час здійснення заміру відносно початку виконання вправ (позначка 0) так, ніби згадані раніше вправи виконувалися безперервно в часі (як у випадку із тестовим датасетом). Для того, щоб об'єднати два датасети, різниця в часі між замірами яких є значною (наприклад, заміри для першого датасету присідань робилися о 13:20, а для другого – о 20:18 того ж дня), проміжок часу між двома датасетами замінявся середнім значенням періоду одного заміру першого датасету.

Також з метою запобігання проблем із маркуванням даних у Label Studio та тренування моделей нейронної мережі було здійснено декілька перевірок вмісту отриманого датафрейму:

- 1) Наявність null-values;
- 2) Наявність рядків із значенням стовпця 'time' < 0;
- 3) Час йде у правильному напрямку, тобто збільшується (перевірка наявності викидів у стовпці 'time').

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 63529 entries, 0 to 63528
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype  
---  -
 0   timestamp   63529 non-null  int64   
 1   time        63529 non-null  float64  
 2   accX        63529 non-null  float64  
 3   accY        63529 non-null  float64  
 4   accZ        63529 non-null  float64  
 5   gyrX        63529 non-null  float64  
 6   gyrY        63529 non-null  float64  
 7   gyrZ        63529 non-null  float64  
dtypes: float64(7), int64(1)
memory usage: 3.9 MB
```

Рис. 5. Інформація про тренувальний датасет

```
len(df[df['time']<0])
```

0

Рис. 6. Перевірка наявності від'ємного часу

```
# Check if the 'time' column of the dataframe is sorted in ascending order
df['time'].is_monotonic_increasing
```

True

Рис. 7. Перевірка наявності викидів у стовпці 'time'

Аналогічні перевірки було здійснено і для тестового датасету (рис. 8).

```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12557 entries, 0 to 12556
Data columns (total 8 columns):
#   Column      Non-Null Count  Dtype
---  -
0   timestamp   12557 non-null  int64
1   time        12557 non-null  float64
2   accX        12557 non-null  float64
3   accY        12557 non-null  float64
4   accZ        12557 non-null  float64
5   gyrX        12557 non-null  float64
6   gyrY        12557 non-null  float64
7   gyrZ        12557 non-null  float64
dtypes: float64(7), int64(1)
memory usage: 784.9 KB
```

```
len(test_df[test_df['time'] < 0])
```

0

```
test_df['time'].is_monotonic_increasing
```

True

Рис. 8. Перевірка вмісту датафрейму test\_df (тестового датасету)

Проведені тести не показали наявності у датасетах аномалій або викидів, тому можна переходити до маркування даних у Label Studio.

### Етап 3: Data Labeling

Тренувальний і тестовий датасети було промарковано у Label Studio (рис. 9-10).

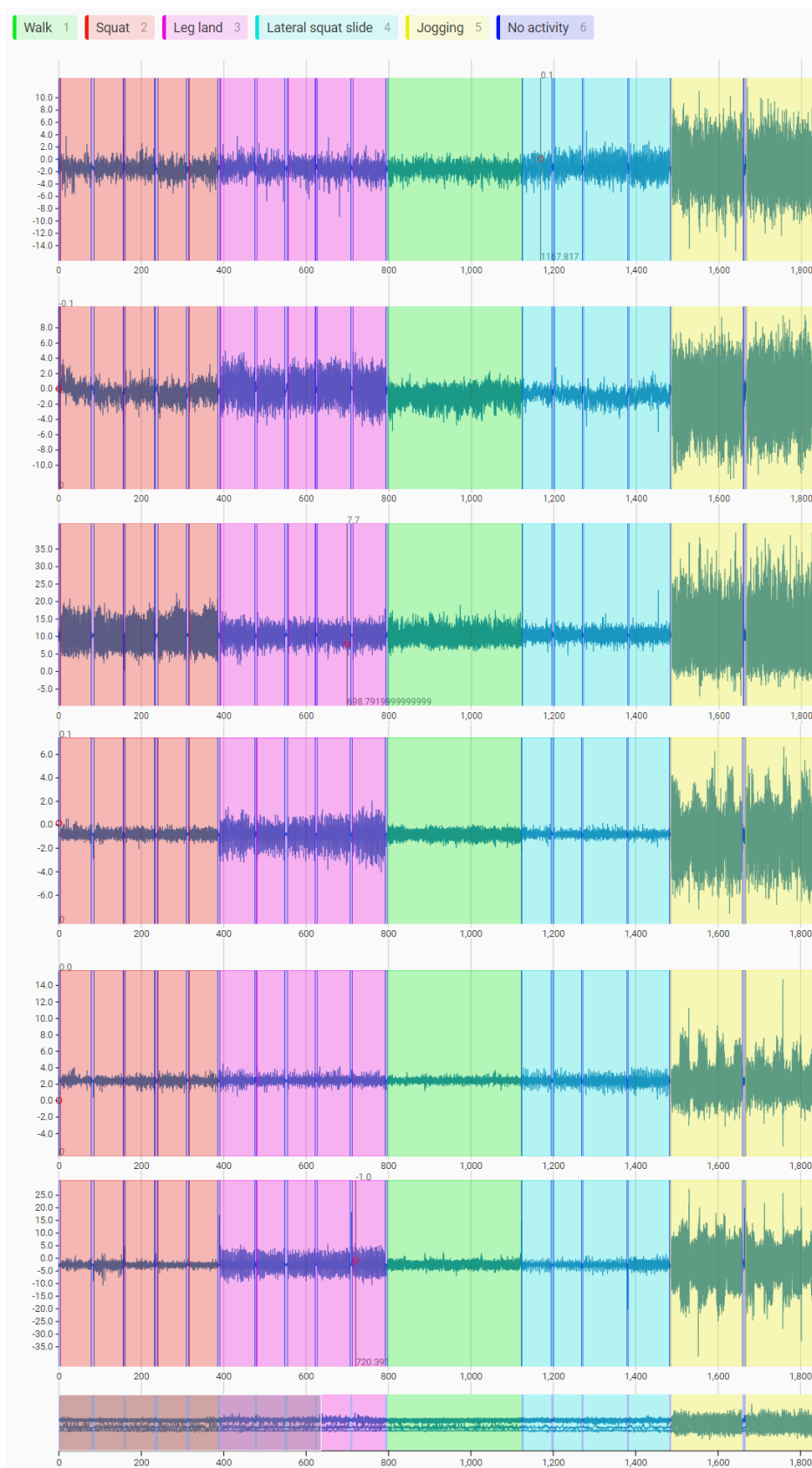


Рис. 9. Результати маркування тренувального датасету у Label Studio

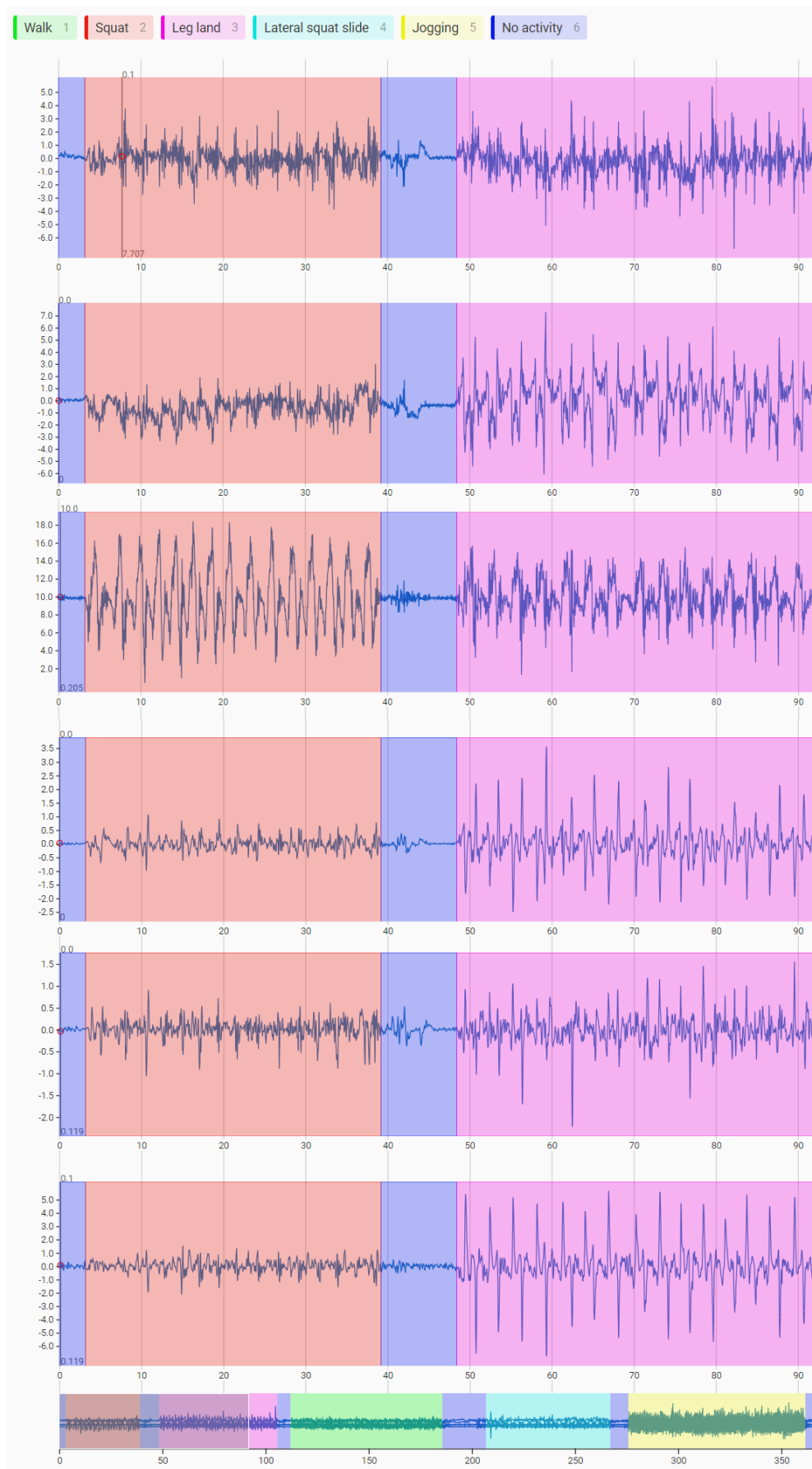


Рис. 10. Результати маркування тестового датасету у Label Studio



Варто відзначити наявність ще одного класу - 'No activity' (рис. 9-10), який використано для позначення калібрування датчиків на початку та в кінці запису даних (тренувальний датасет) та перерви між вправами (тестовий датасет), які були необхідні для відпочинку м'язів.

Після маркування даних до тренувального і тестового датасетів було додано ще один стовпець - 'activity', який містить 6 класів, наведених на рис. 9 і рис. 10.

#### Етап 4: Frequency stability analysis

Перед фільтруванням показів осей акселерометра та гіроскопа (Data Filtering), розглянемо спочатку стабільність частоти забору даних для тренувального і тестового датасетів.

```
time_diffs = df['time'].diff()
freq = 1.0 / time_diffs.mean()
print(f"Measurement time = {df.iloc[-1]['time'] - df.iloc[0]['time']} s")
print(f"Number of measurements (number of rows in the data set) = {len(df)}")
print(f"Average measurement period = {time_diffs.mean():.3f} s")
print(f"Average frequency of measurement = {freq:.3f} Hz")
```

```
Measurement time = 1852.281 s
Number of measurements (number of rows in the data set) = 63529
Average measurement period = 0.029 s
Average frequency of measurement = 34.297 Hz
```

Рис. 11. Інформація про час вимірювання, середній період та частоту забору даних тренувального датасету

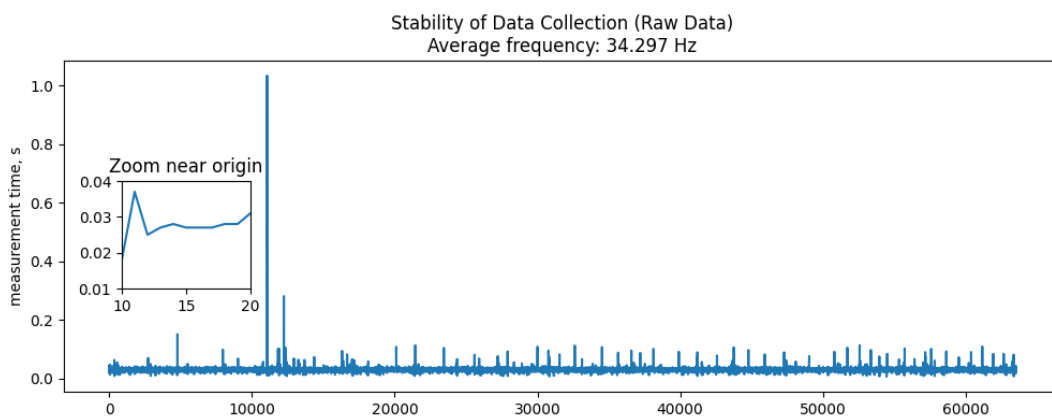


Рис. 12. Графік стабільності періоду забору даних тренувального датасету

Із рис. 12 видно, що більша частина даних замірялася із періодом приблизно 0.03 с. Однак спостерігаються і аномальні значення періоду вимірювання, серед яких яскраво вирізняється значення періоду понад 1 с.

Пояснити отримані коливання періоду забору даних можна так:

- 1) Незначні коливання відносно очікуваного значення періоду (25 мс) пояснюються наявністю обмеження апаратного забезпечення та точністю позначок часу, через що фактичні інтервали вибірки можуть дещо відрізнятися від запланованого значення. Невеликі варіації частоти дискретизації можуть призвести до запису точок даних з періодами трохи довшими за очікуване значення.
- 2) Наявність даних із періодом вимірювання понад 40 мс може бути пов'язана з перериванням процесу запису даних для виконання періодичних збережень або інших операцій. Зокрема, буферизація та очищення: для оптимізації збереження даних можна використовувати механізми буферизації для збору даних пакетами перед записом у сховище. Періодично ці буфери очищаються, що призводить до пауз у процесі запису даних.
- 3) Наявність аномалій із періодом запису понад 100 мс можна пояснити двома причинами:
  - 3.1) Фонові процеси: операційна система смартфона та інші фонові процеси можуть впливати на частоту дискретизації датчика. Якщо пристрій перебуває під великим навантаженням або працює з енергоємними програмами, це може вплинути на частоту дискретизації датчика.
  - 3.2) Помилки датчиків або збої: технічні проблеми, помилки датчиків або збої також можуть призвести до нерегулярних інтервалів вибірки в записаних даних.Обидва пояснення звучать реалістично, враховуючи, що деякі вимірювання проводилися із коротким інтервалом часу між вправами, що в свою чергу сприяло нагріванню процесора смартфона (особливо в жаркі літні дні) і тому могли відбуватися збої датчиків, а запис даних цілком міг перериватися системними процесами для охолодження пристрою.

Враховуючи майже однакову наявність аномалій та викидів (період вимірювання перевищує 40 мс) протягом усього часу вимірювання, було прийнято рішення видалити із оригінального датасету відповідні рядки, в результаті чого розмір датасету зменшився із 63529 (рис. 5) до 63049 (рис. 13).

```
df = df[df['time'].diff() <= 0.04]
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 63049 entries, 2 to 63528
Data columns (total 9 columns):
#   Column      Non-Null Count  Dtype
---  -
0   timestamp    63049 non-null  int64
1   time         63049 non-null  float64
2   accX         63049 non-null  float64
3   accY         63049 non-null  float64
4   accZ         63049 non-null  float64
5   gyrX         63049 non-null  float64
6   gyrY         63049 non-null  float64
7   gyrZ         63049 non-null  float64
8   activity     63049 non-null  object
dtypes: float64(7), int64(1), object(1)
memory usage: 4.8+ MB
```

Рис. 13. Вилучення аномалій, пов'язаних із періодом забору даних

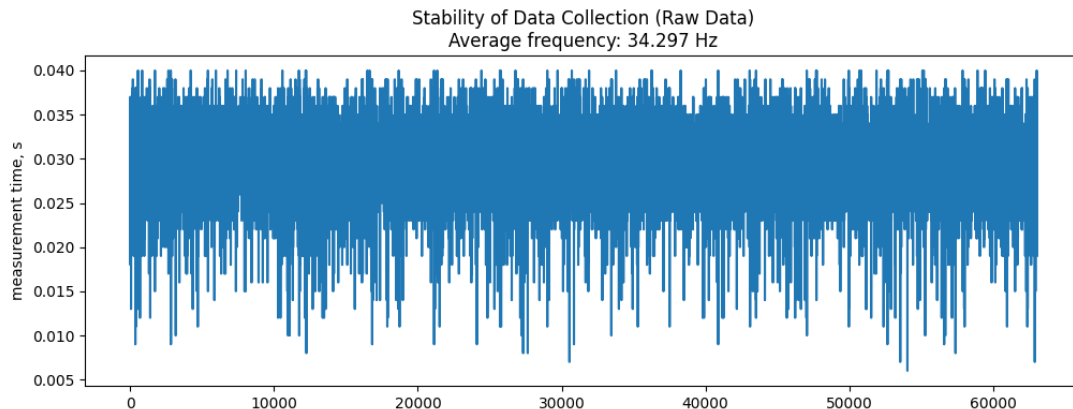


Рис. 14. Графік стабільності періоду забору даних після видалення аномалій та викидів

Розглянемо також стабільність частоти забору даних і для тестового датасету.

```
time_diffs = test_df['time'].diff()
freq = 1.0 / time_diffs.mean()
print(f"Measurement time = {test_df.iloc[-1]['time'] - test_df.iloc[0]['time']} s")
print(f"Number of measurements (number of rows in the data set) = {len(test_df)}")
print(f"Average measurement period = {time_diffs.mean():.3f} s")
print(f"Average frequency of measurement = {freq:.3f} Hz")
```

```
Measurement time = 365.243 s
Number of measurements (number of rows in the data set) = 12557
Average measurement period = 0.029 s
Average frequency of measurement = 34.377 Hz
```

Рис. 15. Інформація про час вимірювання, середній період та частоту забору даних тестового датасету

Порівнявши результати, наведені на рис. 11 і рис. 15, можна зробити висновок, що середній період і середня частота забору даних для тренувального і тестового датасету майже однакові.

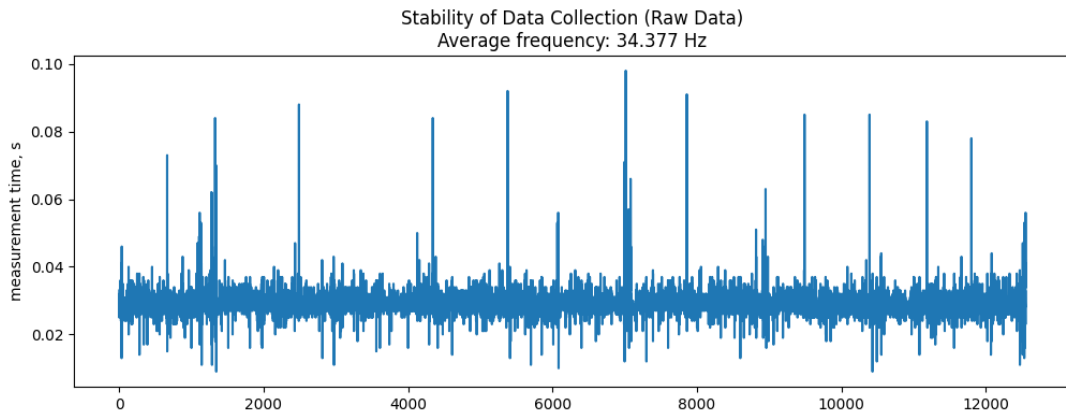


Рис. 16. Графік стабільності періоду забору даних тестового датасету

На графіку стабільності періоду забору даних (рис. 16) тестових даних все ще спостерігається наявність викидів ( $T > 0.04$  с), однак немає різких аномалій (понад 0.2 с, 1 с), як в тренувальному датасеті.

```
len(test_df[test_df['time'].diff() > 0.040])
```

84

Рис. 17. Кількість викидів у тестовому датасеті

Хоча кількість викидів, пов'язаних із стабільністю періоду забору даних, у тестовому датасеті є незначною (рис. 17) і їх можна було б видалити, для чистоти експерименту (щоб перевірити модель нейронної мережі на реальних, неопрацьованих даних), було прийнято рішення не вносити змін у тестовий датасет.

## Етап 5: Data Filtering

Перед фільтруванням даних візуалізовано покази всіх осей акселерометра та гіроскопа оригінального тренувального датасету (рис. 18).

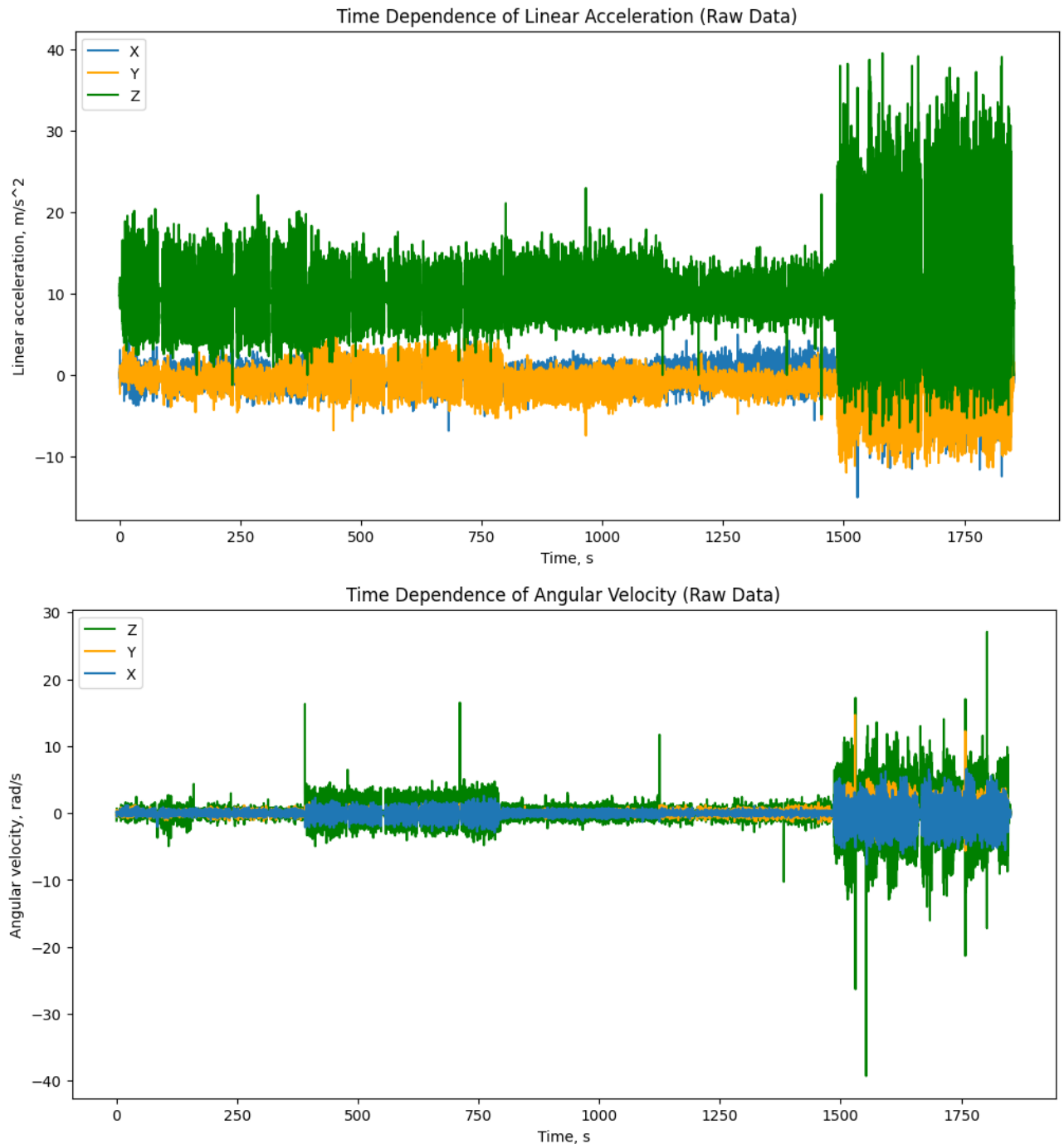


Рис. 18. Покази акселерометра та гіроскопа у тренувальному датасеті (невідфільтровані дані)

Для фільтрування даних використано медіанний фільтр із розміром вікна 10. Результати фільтрування результатів вимірювань акселерометра та гіроскопа тренувального датасету наведено на рис. 19.

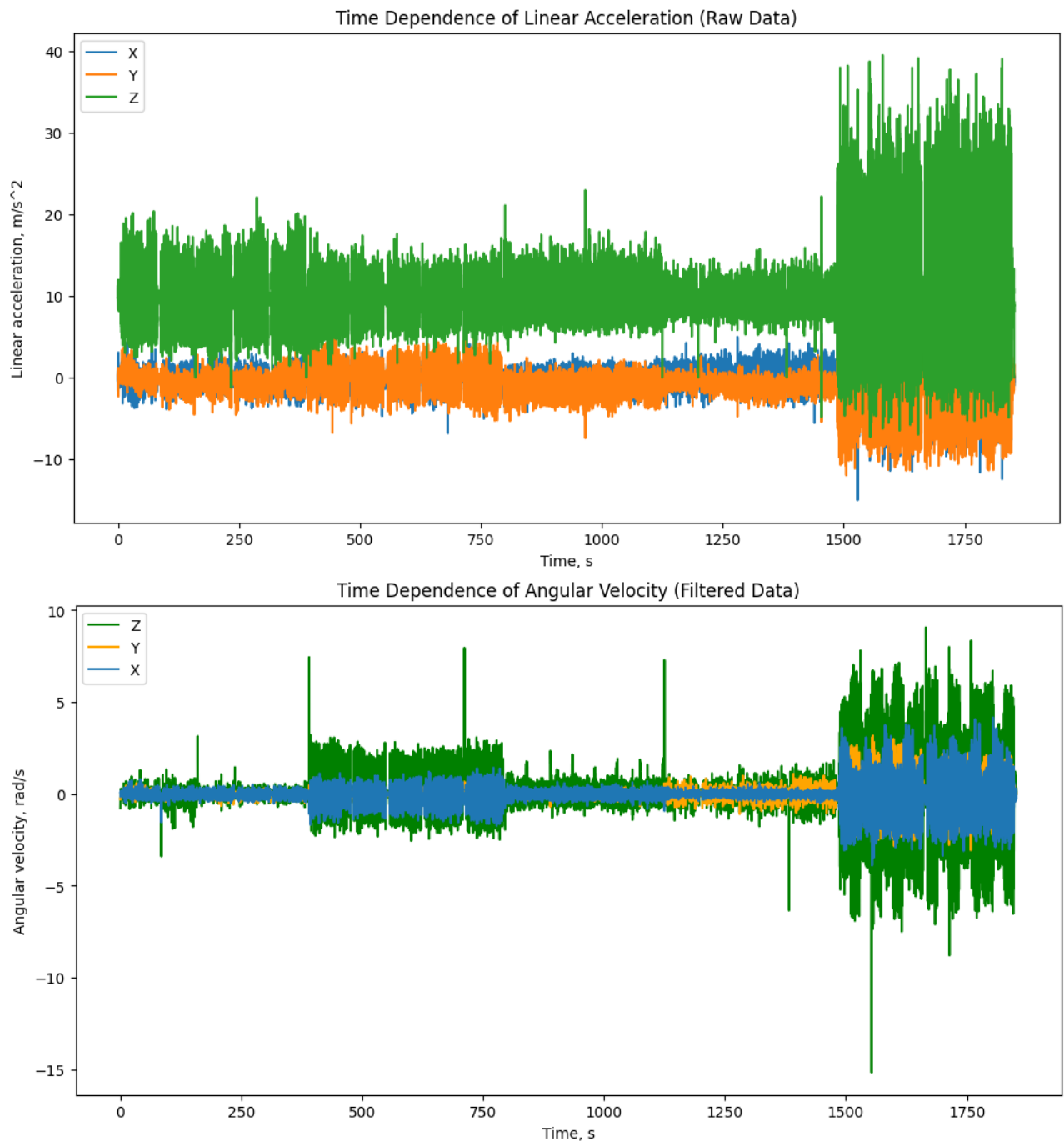


Рис. 19. Покази акселерометра та гіроскопа у тренувальному датасеті (відфільтровані дані)

Щоб наглядніше продемонструвати вплив медіанного фільтра, побудовано графіки для осі OX акселерометра (рис. 20) та осі OZ гіроскопа (рис. 21), на яких відображено невідфільтровані та відфільтровані дані.

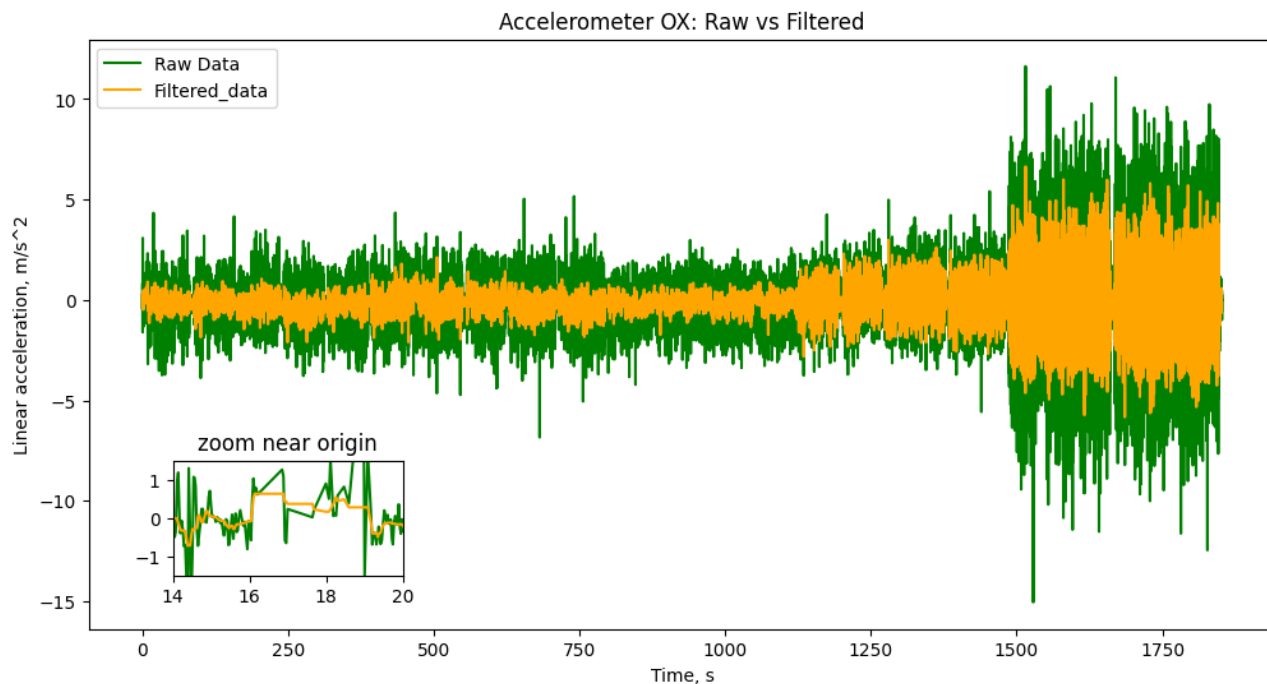


Рис. 20. Результат фільтрування осі OX акселерометра

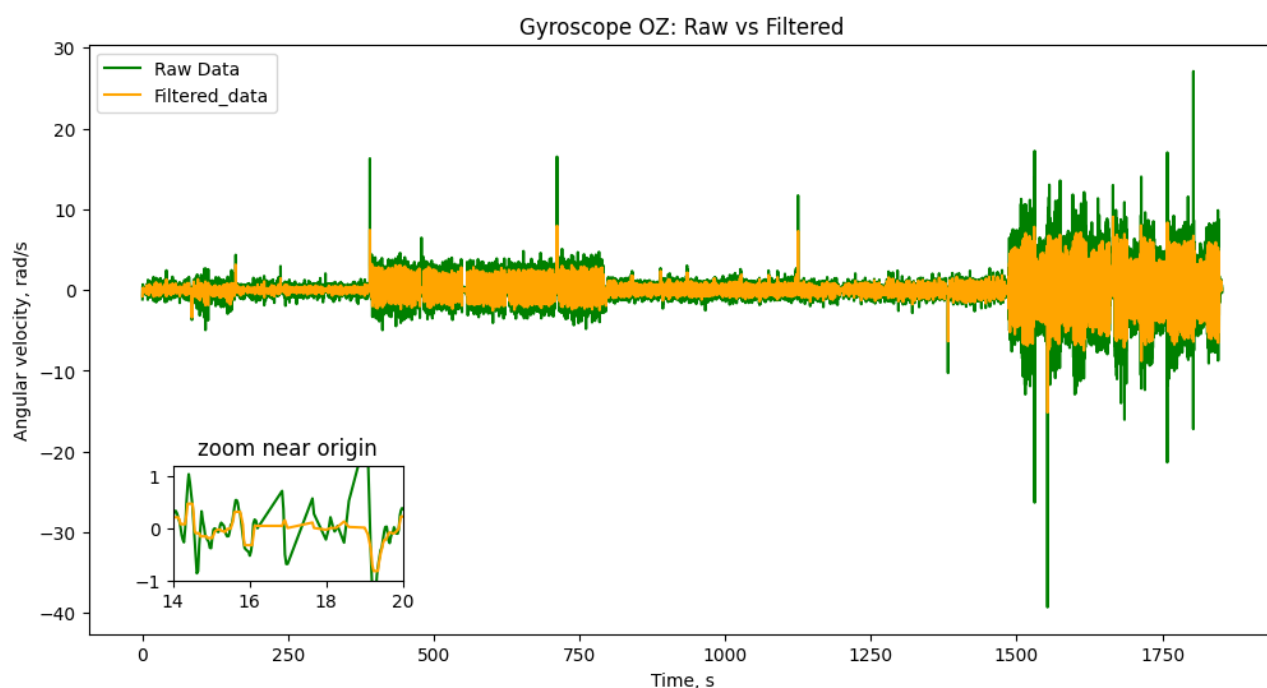


Рис. 21. Результат фільтрування осі OZ гіроскопа

Як видно із рис. 20 і рис. 21, внаслідок фільтрування відбулося згладження деяких викидів (особливо помітно на осі OZ гіроскопа), а результати загалом були усереднені. При цьому, загальні особливості для кожного виду фізичної активності збереглися.

## Етап 6: Exploratory Data Analysis

Для початку проаналізуємо розподіл міток видів активностей, тобто розподіл цільових класів.

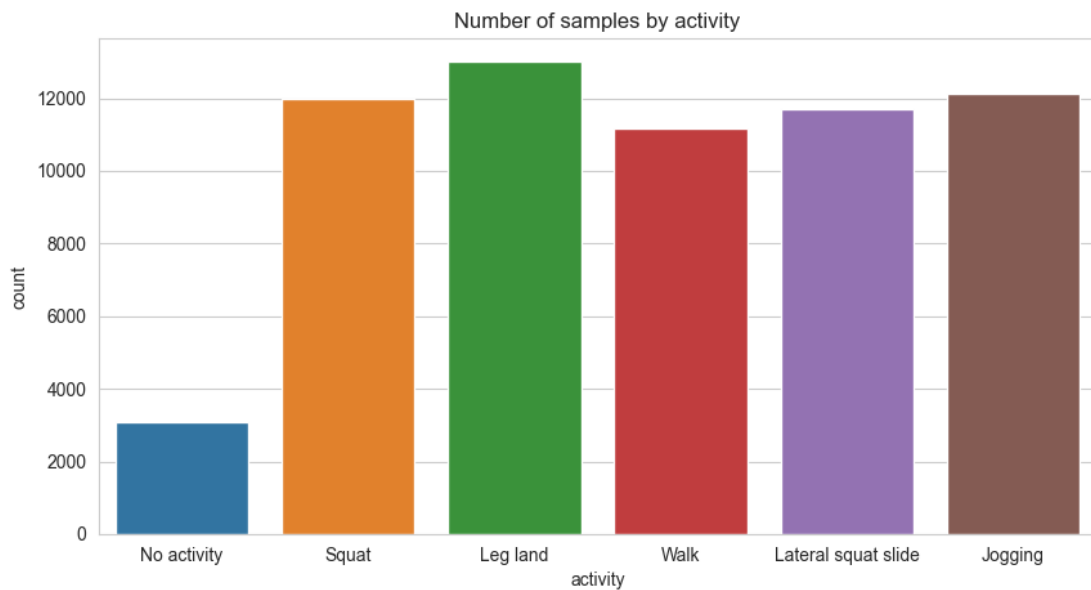


Рис. 22. Аналіз розподілу класів тренувального датасету

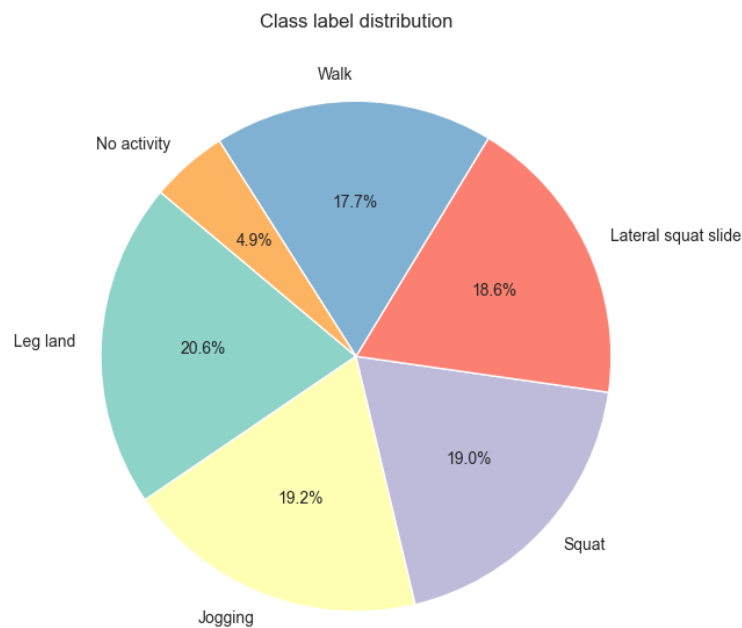


Рис. 23. Аналіз розподілу класів тренувального датасету у процентному співвідношенні

Із рис. 22 і рис. 23 видно, що 5 досліджуваних видів фізичної активності (присідання, ходьба, легкий біг, випади на ноги та lateral squat slide) представлені майже однаковою кількістю елементів. Клас 'No activity', який



становить менше 5% обсягу датасету, не становить цінності для тренування моделі нейронної мережі, тому його можна вилучити.

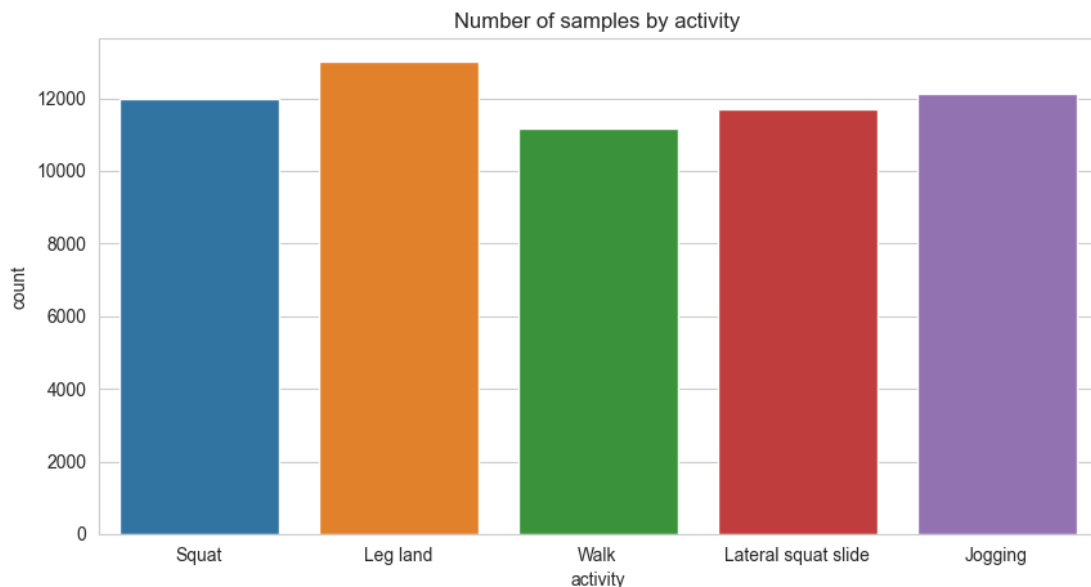


Рис. 24. Аналіз розподілу класів тренувального датасету після вилучення класу 'No activity'

Із рис. 24 видно, що кількість записів для кожного виду фізичної активності є майже однаковою. Однак, між випадками на ноги ('Leg land') та ходьбою ('Walk') спостерігається помітна різниця в кількості записів – майже 2000 (рис. 24, рис. 25).

```
activity_counts = df['activity'].value_counts()
print(activity_counts)
```

```
Leg land      13000
Jogging       12121
Squat         11985
Lateral squat slide  11710
Walk          11153
Name: activity, dtype: int64
```

Рис. 25. Кількість записів для кожного виду фізичної активності тренувального датасету

Зрівняємо кількість записів для всіх видів фізичної активності, використавши undersampling (рис. 26), відкинувши останні  $n$  записів для відповідного класу активності. Внаслідок виконання процесу undersampling, розмір тренувального датасету зменшився від 63049 (рис. 13) до 55765 записів (рис. 26). Натомість було отримано збалансований датасет, який містить 11153 записи для кожного класу фізичної активності (рис. 26-27).

```
undersampled_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55765 entries, 0 to 55764

activity_counts = undersampled_df['activity'].value_counts()
print(activity_counts)

Squat                11153
Leg land             11153
Walk                 11153
Lateral squat slide  11153
Jogging              11153
Name: activity, dtype: int64
```

Рис. 26. Результат виконання undersampling для тренувального датасету

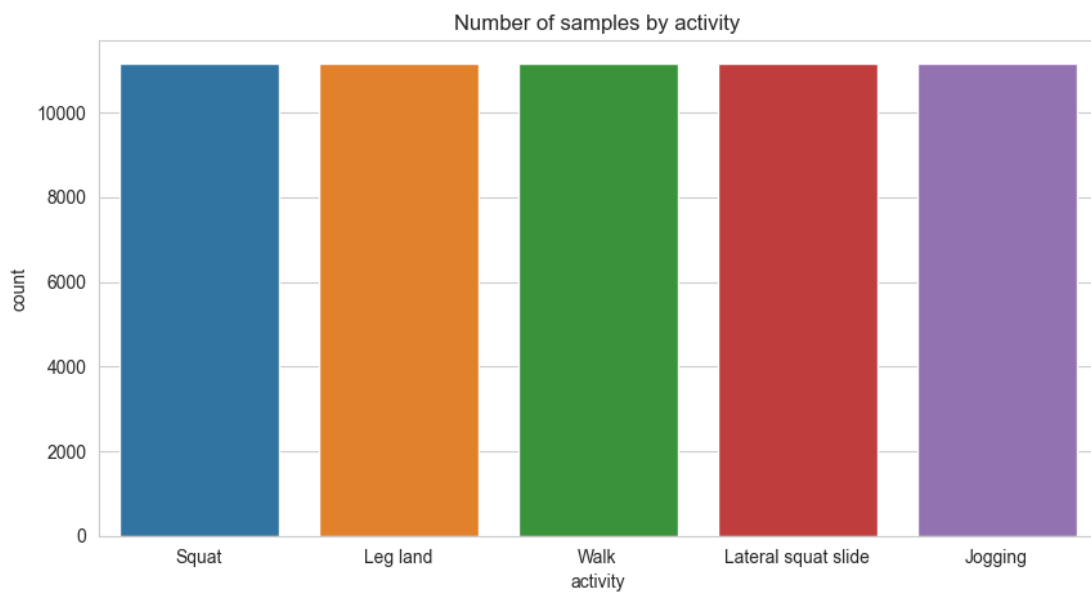


Рис. 27. Розподіл класів фізичної активності тренувального датасету після виконання undersampling

Наступним кроком побудуємо розподіл даних сигналу по осях OX, OY та OZ акселерометра та гіроскопа для того, щоб побачити, чи є якась очевидна закономірність на основі розподілу значень.

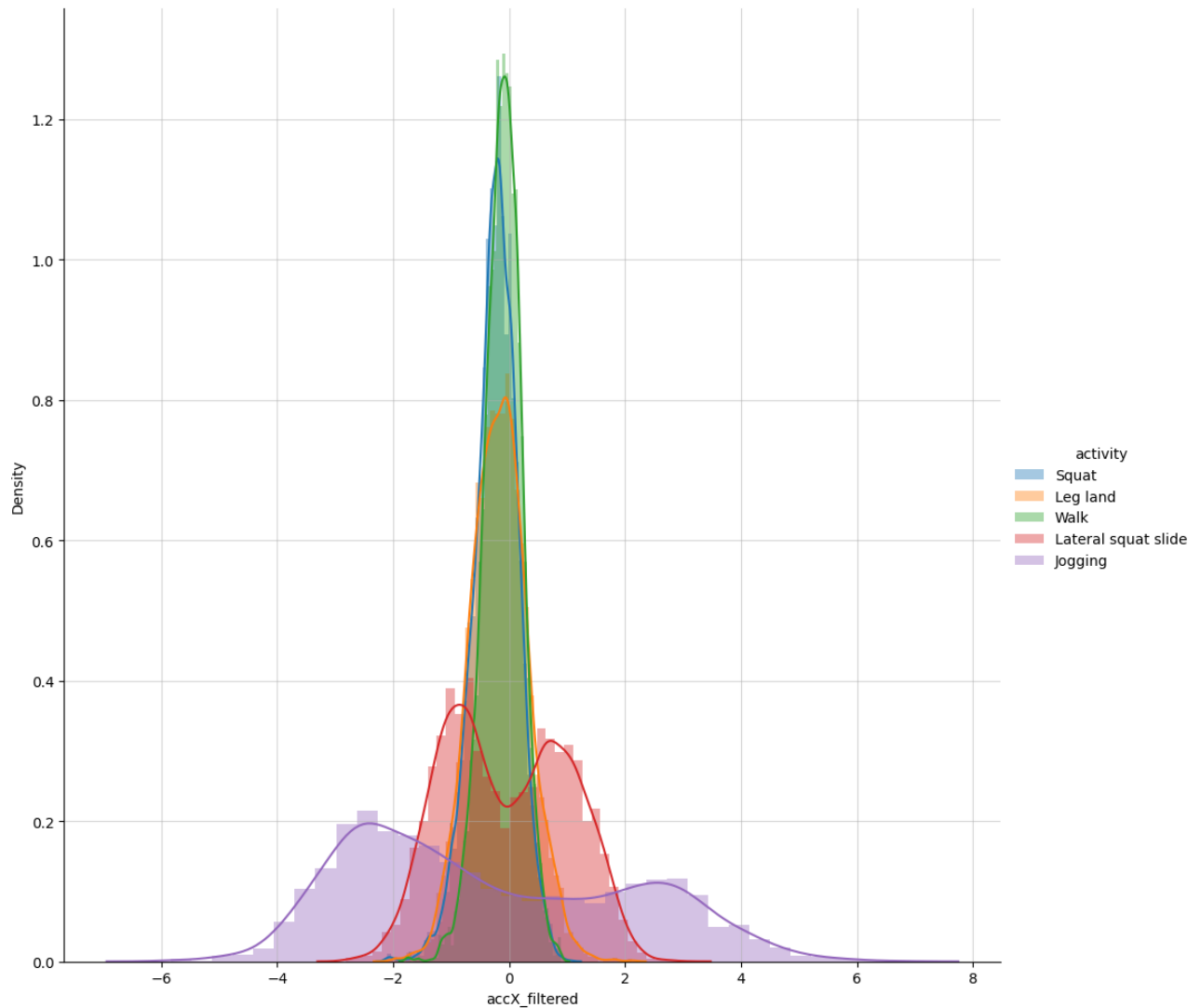


Рис. 28. Розподіл сигналів акселерометра на осі OX

Із рис. 28 можна зробити такі висновки:

- 1) Найменше розсіювання мають такі види активності: 'Walk' та 'Squat' (функції густини розподілу нагадують нормальний розподіл);
- 2) Незначне розсіювання значень має 'Leg land' (функція густини розподілу нагадує нормальний закон розподілу);
- 3) Найбільше розсіювання характерне для 'Jogging' та 'Lateral squat slide'. Це свідчить про наявність значних коливань на осі OX акселерометра для цих видів діяльності.

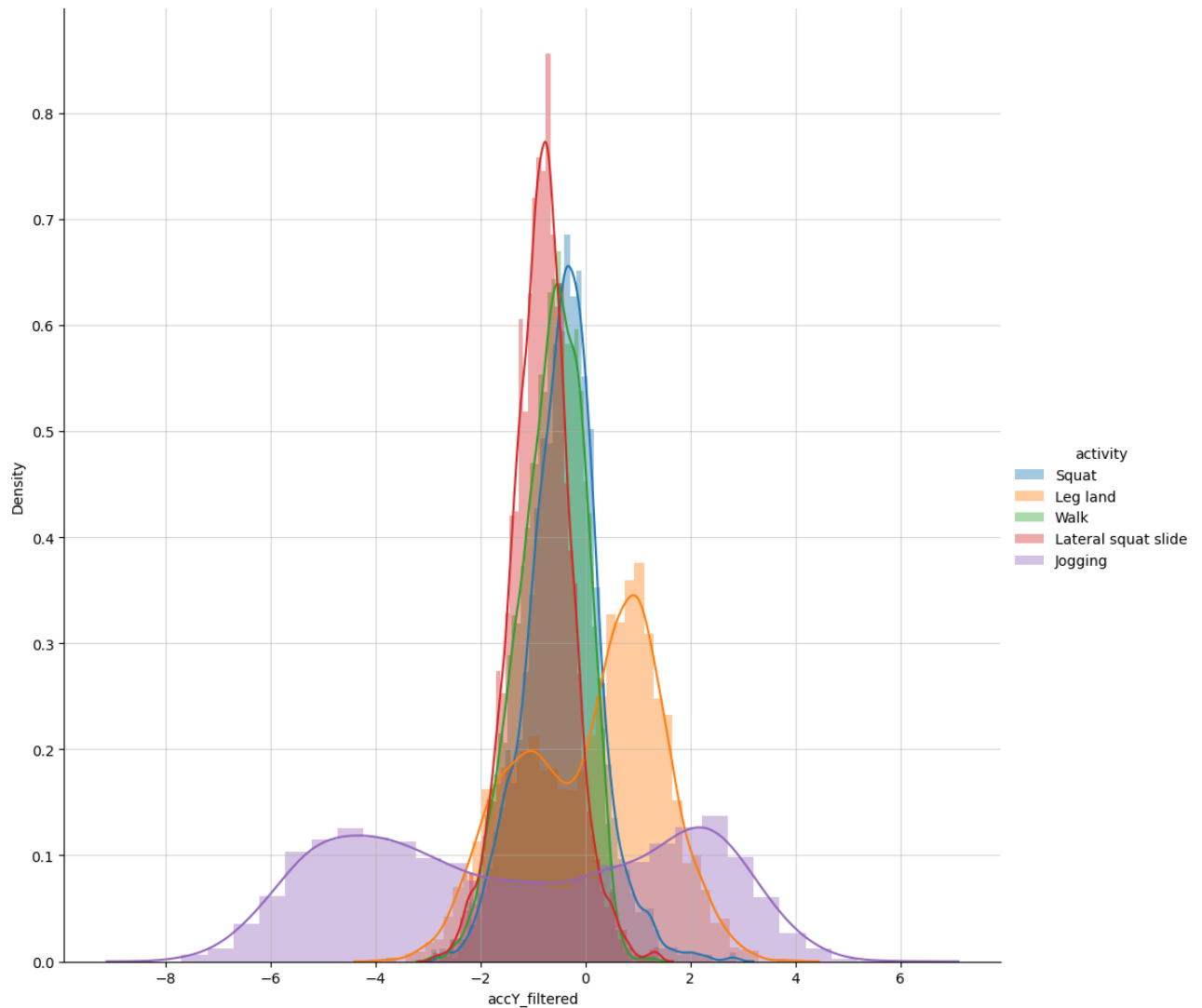


Рис. 29. Розподіл сигналів акселерометра на осі OY

Із рис. 29 можна зробити такі висновки:

- 1) Найменше розсіювання мають такі види активності: 'Lateral squat slide' 'Squat' та 'Walk' (функції густини розподілу нагадують нормальний розподіл);
- 2) Найбільше розсіювання характерне для 'Leg land' та 'Jogging'.

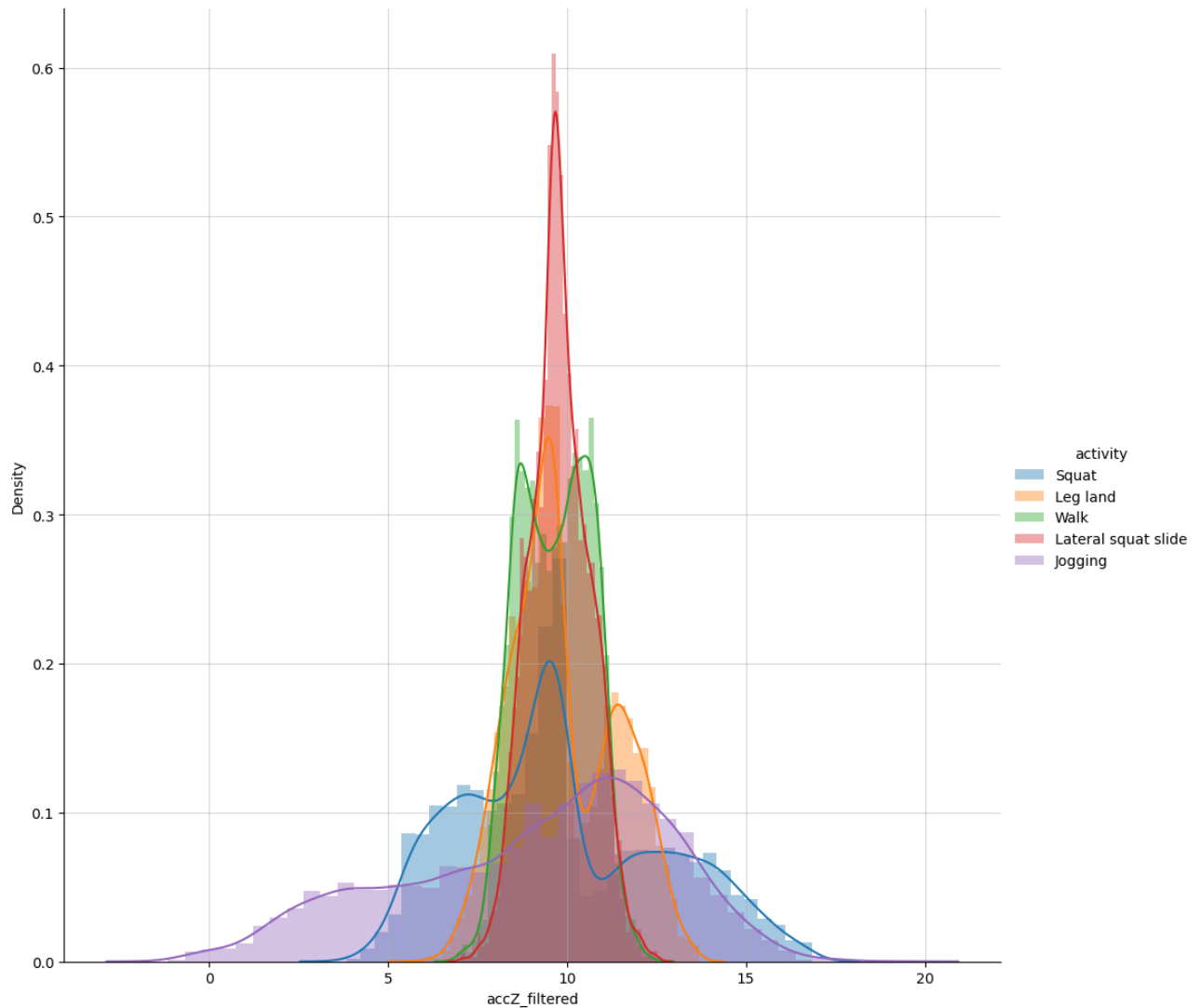


Рис. 30. Розподіл сигналів акселерометра на осі OZ

Із рис. 30 можна зробити такі висновки:

- 1) Найменше розсіювання характерне для 'Lateral squat slide' (функція густини розподілу нагадують нормальний розподіл);
- 2) Незначне розсіювання значень мають такі види активності як 'Leg land' (функція густини розподілу нагадує нормальний закон розподілу) та 'Walk';
- 3) Найбільше розсіювання характерне для 'Squat' та 'Jogging'. Це свідчить про наявність значних коливань на осі OZ акселерометра для цих видів діяльності.

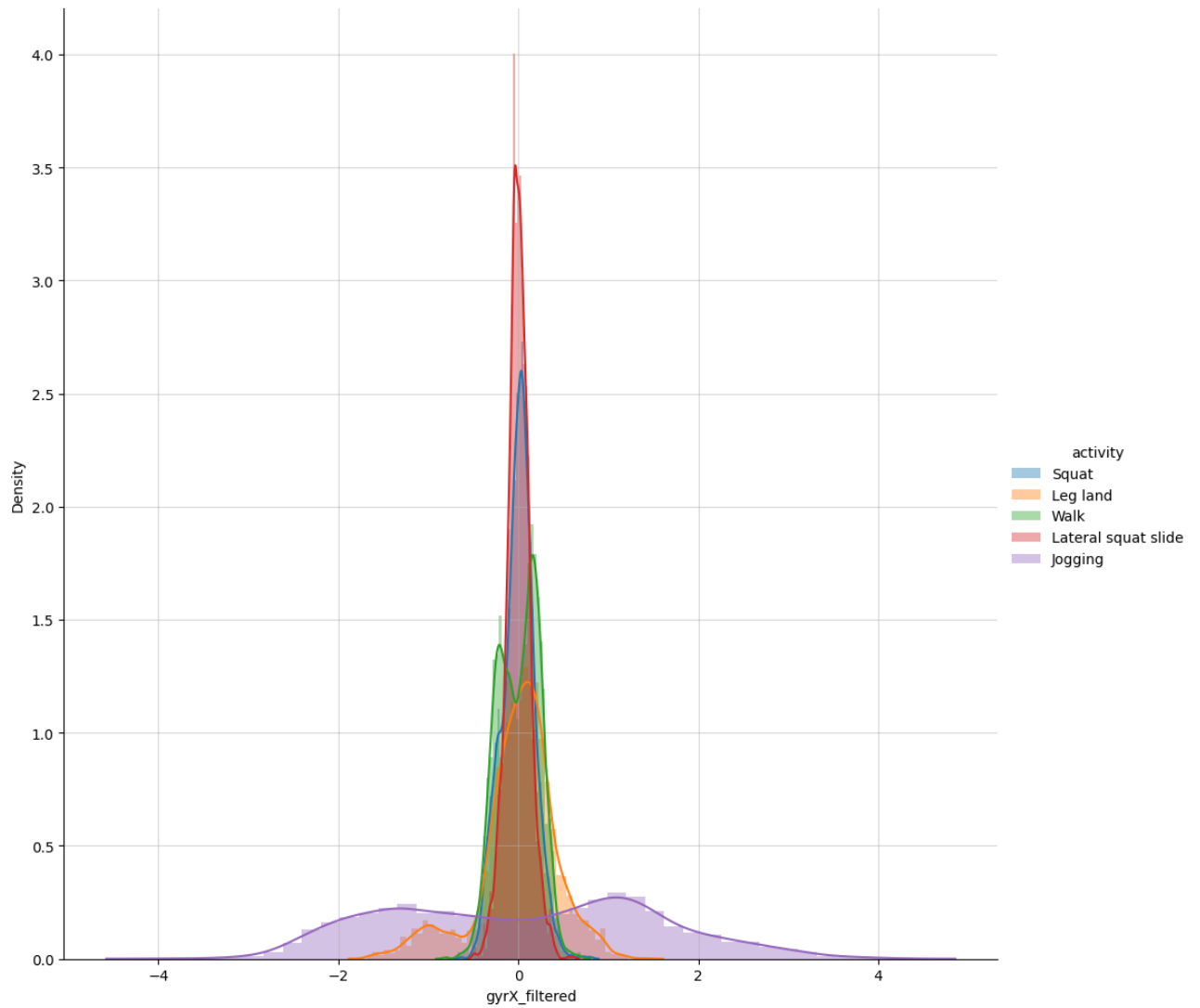


Рис. 31. Розподіл сигналів гіроскопа на осі OX

Із рис. 31 можна побачити, що:

- 1) Найменше розсіювання мають 'Lateral squat slide' та 'Squat' (функції густини розподілу нагадують нормальний розподіл);
- 2) Помітне розсіювання характерне для таких видів активності як 'Walk' і 'Leg land' (функція густини розподілу нагадує нормальний закон розподілу);
- 3) Найбільше розсіювання спостерігається для активності 'Jogging'.

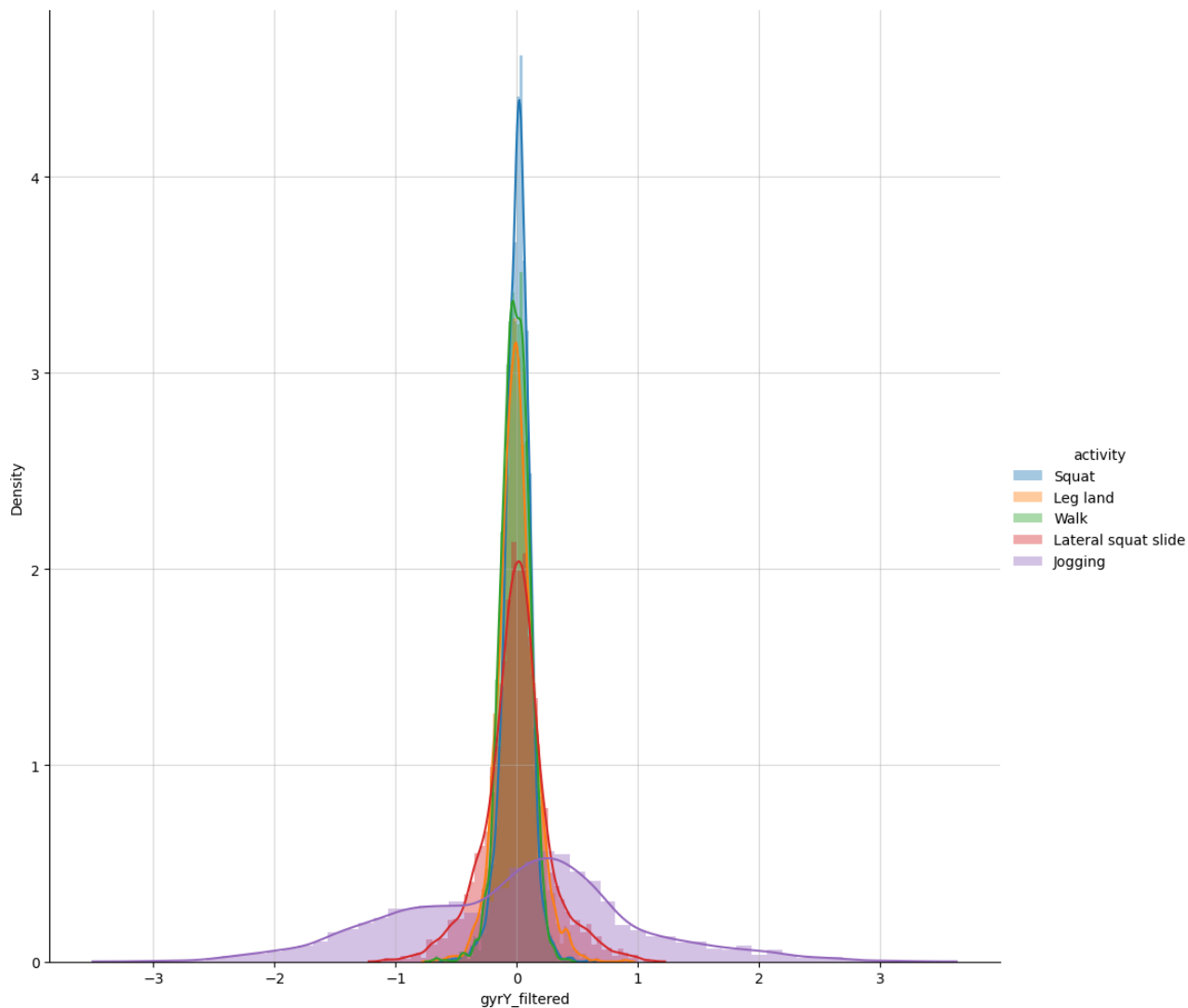


Рис. 32. Розподіл сигналів гіроскопа на осі OY

Із рис. 32 можна побачити, що:

- 1) Найменше розсіювання мають 'Squat', 'Walk' і 'Leg land' (функції густини розподілу нагадують нормальний розподіл);
- 2) Помітне розсіювання характерне для 'Lateral squat slide' (функція густини розподілу нагадує нормальний закон розподілу);
- 3) Найбільше розсіювання спостерігається для активності 'Jogging' (функція густини розподілу нагадує нормальний закон розподілу).

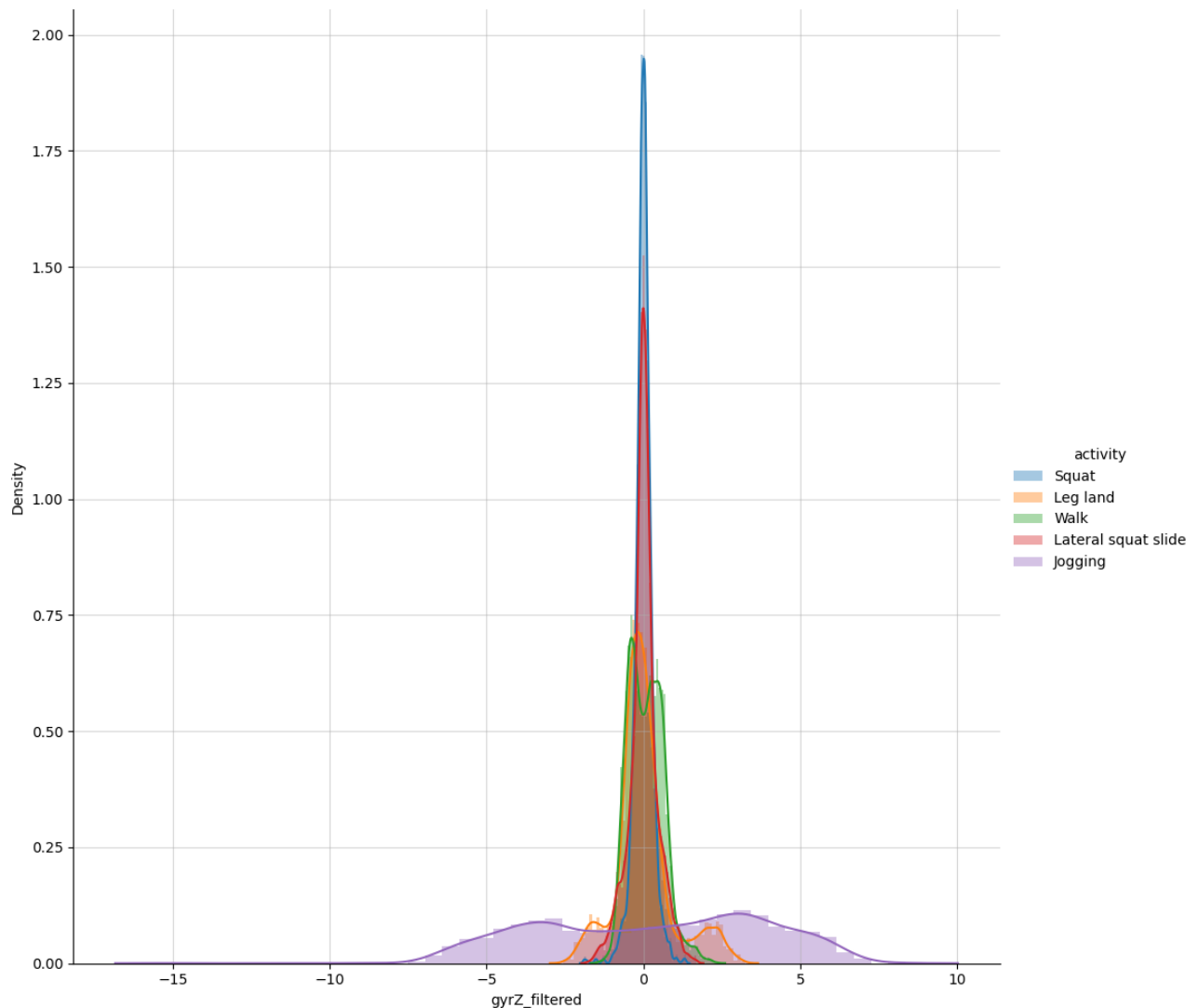


Рис. 33. Розподіл сигналів гіроскопа на осі OZ

Із рис. 33 можна зробити такі висновки:

- 1) Найменше розсіювання мають 'Squat' та 'Lateral squat slide' (функції густини розподілу нагадують нормальний розподіл);
- 2) Помітне розсіювання характерне для таких видів активності як 'Leg land' і 'Walk';
- 3) Найбільше розсіювання спостерігається для активності 'Jogging'.

Розглянувши рис. 28-33, зроблено висновки про те, що розподіли активностей трохи схожі між собою. Загалом, можна зробити такі висновки про особливості кожного досліджуваного виду фізичної активності:

- 1) Клас 'Squat' вирізняється сильним розсіюванням по осі OZ акселерометра та незначним розсіюванням на всіх інших осях акселерометра та гіроскопа.
- 2) Для класу 'Leg land' характерне значне розсіювання на осі OY акселерометра та помітне розсіювання на осях OZ акселерометра, OX та OZ гіроскопа.



- 3) 'Walk' має невелике розсіювання на осях OX і OY акселерометра, а також на осі OY гіроскопа.
- 4) Для 'Lateral squat slide' характерне велике розсіювання на осі OX акселерометра та незначне розсіювання на інших осях акселерометра та гіроскопа.
- 5) 'Jogging' вирізняється найбільшим розсіюванням на всіх осях акселерометра та гіроскопа.

Наступним кроком побудуємо матрицю кореляції для осей акселерометра та гіроскопа із метою визначення осей, які можна було б відкинути. Оскільки для обчислення статистичних параметрів (Feature Engineering) використовуватимуться відфільтровані дані акселерометра та гіроскопа, побудуємо матрицю кореляції для відповідних стовпців досліджуваного дата фрейму (рис. 34).

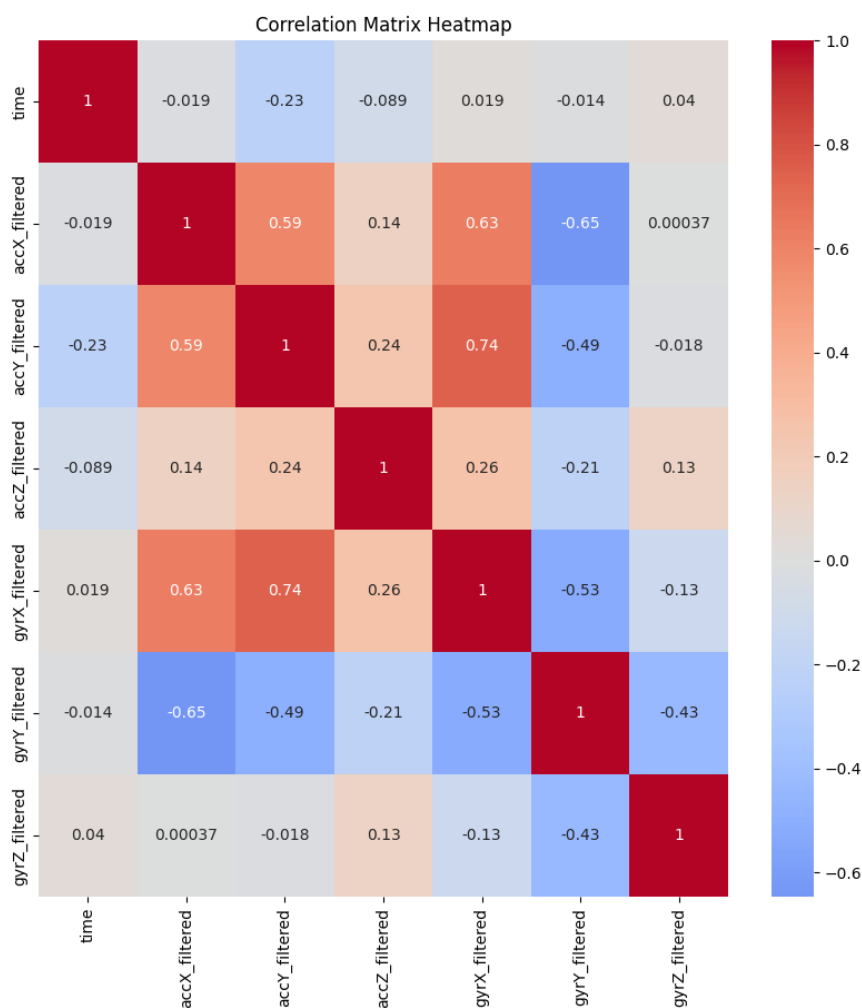


Рис. 34. Матриця кореляції відфільтрованих показів акселерометра та гіроскопа

Розглянувши рис. 34, в першу чергу можна помітити наявність кореляційного зв'язку між осями ОХ та ОУ акселерометра (коефіцієнт кореляції становить 0.59). Враховуючи величину коефіцієнта кореляції, можна було б забрати одну із згаданих осей. Однак варто зазначити, що вісь ОХ акселерометра є однією з визначальних осей для класу 'Lateral squat slide', а вісь ОУ акселерометра дозволяє відрізнити клас 'Leg land' з-поміж інших класів. Врахувавши наведені вище аргументи було прийнято рішення не видаляти жодну із згаданих осей.

Із рис. 34 також видно, що вісь ОХ гіроскопа корелює із осями ОХ та ОУ акселерометра та із віссю ОУ гіроскопа (коефіцієнти кореляції становлять 0.63, 0.74 та -0.53 відповідно). Також спостерігається наявність сильного кореляційного зв'язку між віссю ОУ гіроскопа та осями ОХ, ОУ акселерометра та віссю ОZ гіроскопа (коефіцієнти кореляції становлять -0.65, -0.49 та -0.43 відповідно). Враховуючи, що згадані осі не є визначальними для певного виду фізичної активності, їх можна вилучити для спрощення набору даних.

В результаті було отримано датафрейм, зображений на рис. 35.

```
filtered_df.head()
```

	time	accX_filtered	accY_filtered	accZ_filtered	gyrZ_filtered	activity
0	4.774	0.139050	-0.286950	9.555000	-0.111925	Squat
1	4.809	0.182025	-0.301500	9.525001	-0.098175	Squat
2	4.843	0.139050	-0.316050	9.555000	-0.084425	Squat
3	4.873	0.096000	-0.323025	9.555975	-0.084219	Squat
4	4.904	0.139050	-0.330000	9.555000	-0.084013	Squat

Рис. 35. Датафрейм із зменшеною кількістю осей гіроскопа

## Етап 7: Data Transformation

Також варто зазначити, що внаслідок вилучення даних із аномальними значеннями періоду забору даних (Етап 4: Frequency stability analysis), середня частота забору даних зменшилася (рис. 36).

```
sampling_frequency = 1.0 / filtered_df['time'].diff().mean() # Hz
print(f"sampling_frequency = {sampling_frequency: .3f} Hz")

sampling_frequency = 30.729 Hz
```

Рис. 36. Середня частота забору даних в датафреймі `filtered_df`

Здійснено поділ оригінального датафрейму `df` на вікна тривалістю 2 с. Кількість рядків, які вміщалися в одне вікно становить:

`window_size = ceil(sampling_frequency * window_duration),`

де

- `sampling_frequency` – середня частота збору даних в Гц (точне її значення наведено на рис. 36);
- `window_duration` – тривалість вікна у с (у даному випадку `window_duration = 2`);
- `ceil` – функція бібліотеки `math` у Python, яка завжди заокругляє дріб до більшого цілого (у більшу сторону).

Отже,

`window_size = ceil(30.729 * 2) = 62`

Варто також зазначити, що замість окремих вікон ми беремо вікна, що перекриваються, із 50% перекриттям (тобто `крок = window_size / 2 = 31`). Це гарантує, що кожен наступний рядок у перетвореному наборі даних також містить певну інформацію з даних у попередньому вікні.

Для призначення мітки класу активності береться найчастіша активність у цьому вікні.

В результаті виконання цієї операції отримуємо новий датафрейм – `windowed_df` (рис. 37).

windowed_df.head()							
	start_time	end_time	accX	accY	accZ	gyrZ	activity
0	4.774	6.697	[0.13905, 0.18202500500000002, 0.13905, 0.0960...	[-0.28695002, -0.30150002, -0.31605002, -0.323...	[9.555, 9.5250005, 9.555, 9.555975499999999, 9...	[-0.111925, -0.098175, -0.084425, -0.084218750...	Squat
1	5.776	7.683	[0.119025005, 0.119025005, 0.0785250065, 0.002...	[0.284475015, 0.55350001, 0.766500025, 0.76650...	[11.6035505, 11.6300255, 11.749500000000001, 1...	[0.14403125, 0.127875, 0.127875, 0.05218125, -...	Squat
2	6.728	8.639	[-0.012000000000000001, -0.012000000000000001, -...	[-0.24405001599999998, -0.45105003, -0.4510500...	[6.37605045, 6.37605045, 6.6515252, 7.04400000...	[-0.37049374999999996, -0.3551625, -0.09734999...	Squat
3	7.715	9.629	[0.40950003, 0.44550003000000005, 0.54697503, ...	[0.250500005, 0.250500005, 0.3425249999999997...	[9.1404755, 8.51602575, 7.79902555, 7.6719753,...	[0.32690625, 0.25279375, 0.109724999, -0.07198...	Squat
4	8.665	10.530	[0.1094999995, 0.148500002, 0.148500002, 0.148...	[1.2285, 0.78795003, 0.53445004, 0.43050001, 0...	[15.172500750000001, 14.9499755, 14.59950075, ...	[-0.119831249, 0.128425001, 0.2959, 0.3526875,...	Squat

Рис. 37. Вміст датафрейму `windowed_df`

Таким чином, з датафрейму `filtered_df`, який містив 55765 рядків, ми отримали `windowed_df` довжиною 1799 рядків.

```
print(f'len(filtered_df) = {len(filtered_df)}')
print(f'len(windowed_df) = {len(windowed_df)}')

len(filtered_df) = 55765
len(windowed_df) = 1799
```

Рис. 38. Довжини датафреймів `filtered_df` та `windowed_df`

Розглянемо, чи не змінилося співвідношення між класами фізичної діяльності внаслідок виконання процесу віконування (`windowing`).

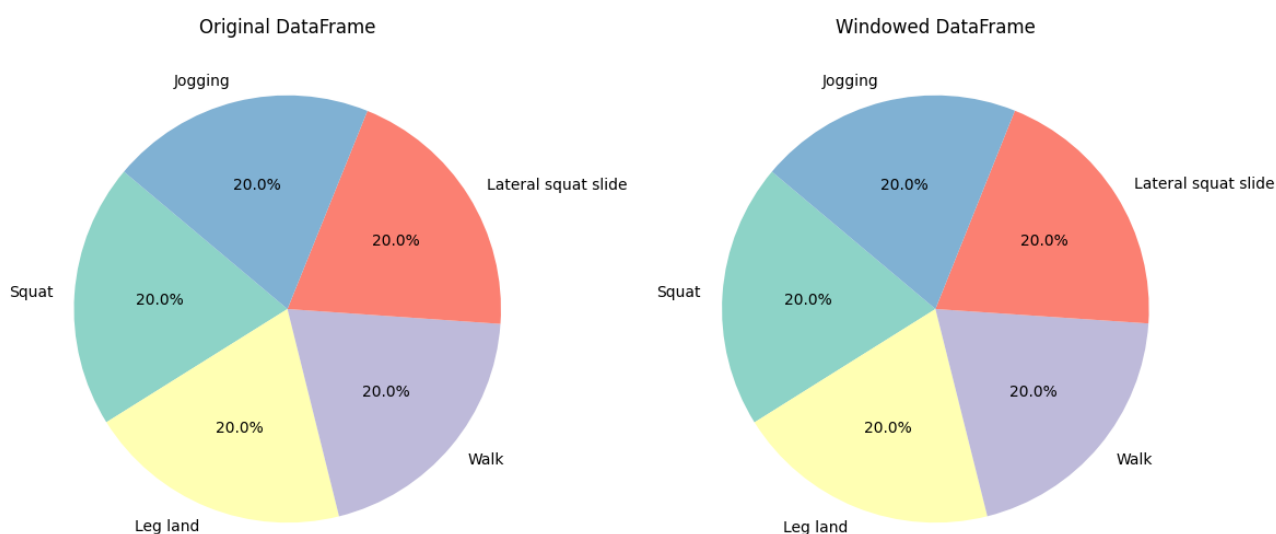


Рис. 39. Класи фізичної активності в `filtered_df` та `windowed_df`

Як видно із рис. 39, внаслідок процесу віконування (`windowing`) процентне співвідношення міток класів не змінилося.

## Етап 8: Feature Engineering

Для тренування моделі було вибрано 12 простих статистичних параметрів (features) (рис. 40):

- 1) середнє статистичне (mean);
- 2) стандартне відхилення (standard deviation);
- 3) середнє абсолютне відхилення (average absolute deviation);
- 4) мінімальне значення (min);
- 5) максимальне значення (max);
- 6) розмах = різниця максимального і мінімального значень (range);
- 7) медіана (median);
- 8) міжквартильний діапазон (interquartile range = IQR);
- 9) кількість від'ємних значень (negative values count);

- 10) кількість позитивних значень (positive values count);
- 11) асиметрія (skewness = assymetry);
- 12) ексцес (kurtosis).

```
functions_list = [
    lambda x: x.mean(), # mean
    lambda x: x.std(), # std deviation
    lambda x: np.mean(np.absolute(x - np.mean(x))), # avg absolute diff
    lambda x: x.min(), # min
    lambda x: x.max(), # max
    lambda x: x.max() - x.min(), # range = max-min diff
    lambda x: np.median(x), # median
    lambda x: np.percentile(x, 75) - np.percentile(x, 25), # interquartile range
    lambda x: np.sum(x < 0), # negative count
    lambda x: np.sum(x > 0), # positive count
    lambda x: stats.skew(x), # skewness = assymetry
    lambda x: stats.kurtosis(x) # kurtosis
]
```

Рис. 40. Обчислені статистичні параметри для датафрейму X\_train

Значення вказаних статистичних параметрів для кожного вікна зберігаються у датафреймі X\_train. В результаті обчислення всіх вищевказаних статистичних параметрів для трьох осей акселерометра та осі OZ гіроскопа, датафрейм X\_train містить 48 стовпців (рис. 41).

X\_train.head()

	accX_mean	accY_mean	accZ_mean	gyrZ_mean	accX_std	accY_std	accZ_std	gyrZ_std	accX_aad	accY_aad	...	accZ_pos_count	gyrZ_pos_count	accX_assymetry
0	0.204513	0.043750	10.181814	-0.041990	0.191702	0.446751	2.493435	0.173847	0.164112	0.352874	...	62	25	0.889454
1	0.339821	0.349124	10.100784	-0.048229	0.211233	0.441290	2.552319	0.316891	0.181807	0.378584	...	62	26	-0.752718
2	0.316942	0.815667	9.721920	-0.079716	0.216026	0.908844	2.554711	0.334650	0.187870	0.699350	...	62	25	-0.348733
3	0.193883	0.900957	9.749154	-0.026190	0.288013	0.857635	3.219180	0.307127	0.219851	0.704377	...	62	30	-0.037762
4	0.037838	0.780090	9.024925	0.038186	0.278074	0.454869	2.237759	0.241641	0.218381	0.408583	...	62	31	0.504361

5 rows × 48 columns

Рис. 41. Вміст датафрейму X\_train

Оскільки безпосередньо використовувати мітки класів у вигляді стрічок (наприклад, 'Walk', 'Jogging' і т.д.) як цільові значення для навчання та тестування моделі нейронної мережі не можна, було здійснено Label Encoding (перетворення категорійних міток (рядків) у числові значення). Числові значення для кожного виду фізичної активності було встановлено вручну за допомогою словника activity\_dict (рис. 42).

```
activity_dict = {'Squat': 0, 'Leg land': 1, 'Walk': 2, 'Lateral squat slide': 3, 'Jogging': 4}
```

Рис. 42. Числові мітки для досліджуваних видів фізичної активності

Відповідні мітки класів активностей містяться у масиві y\_train.

## Етап 9: Model Training

Безпосередньо перед тренуванням моделі підготуємо тестовий датасет (рис. 43).

```
test_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12557 entries, 0 to 12556
Data columns (total 9 columns):
#   Column      Non-Null Count  Dtype  
---  -
0   timestamp   12557 non-null  int64   
1   time        12557 non-null  float64  
2   accX        12557 non-null  float64  
3   accY        12557 non-null  float64  
4   accZ        12557 non-null  float64  
5   gyrX        12557 non-null  float64  
6   gyrY        12557 non-null  float64  
7   gyrZ        12557 non-null  float64  
8   activity    12557 non-null  object  
dtypes: float64(7), int64(1), object(1)
memory usage: 883.0+ KB
```

Рис. 43. Інформація про тестовий датасет

Аналогічно як і для тренувального датасету профільтруємо записи осей акселерометра та гіроскопа тестового датасету, використавши медіанний фільтр із вікном 10 (рис. 44). Це дозволить забезпечити узгодженість між тренувальним і тестовим датасетами і гарантує, що модель оцінюється на даних зі схожими характеристиками.

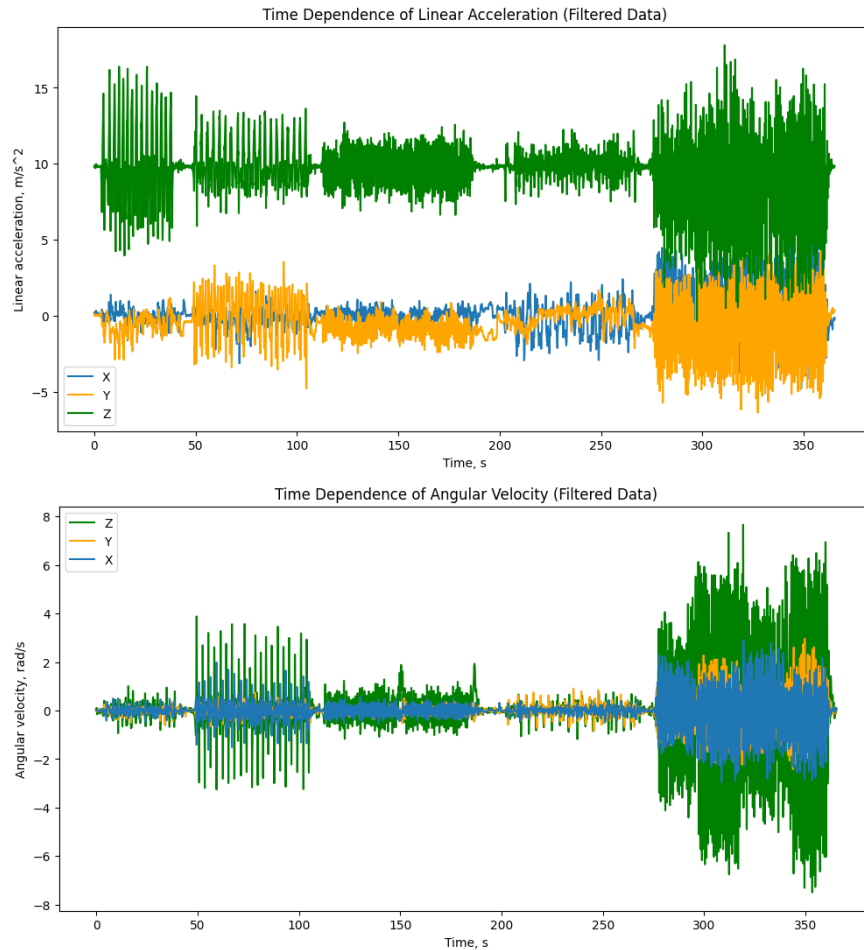


Рис. 44. Покази акселерометра та гіроскопа у тестовому датасеті (відфільтровані дані)

Наступним кроком розглянемо розподіл міток класів у тестовому датасеті (рис. 45).

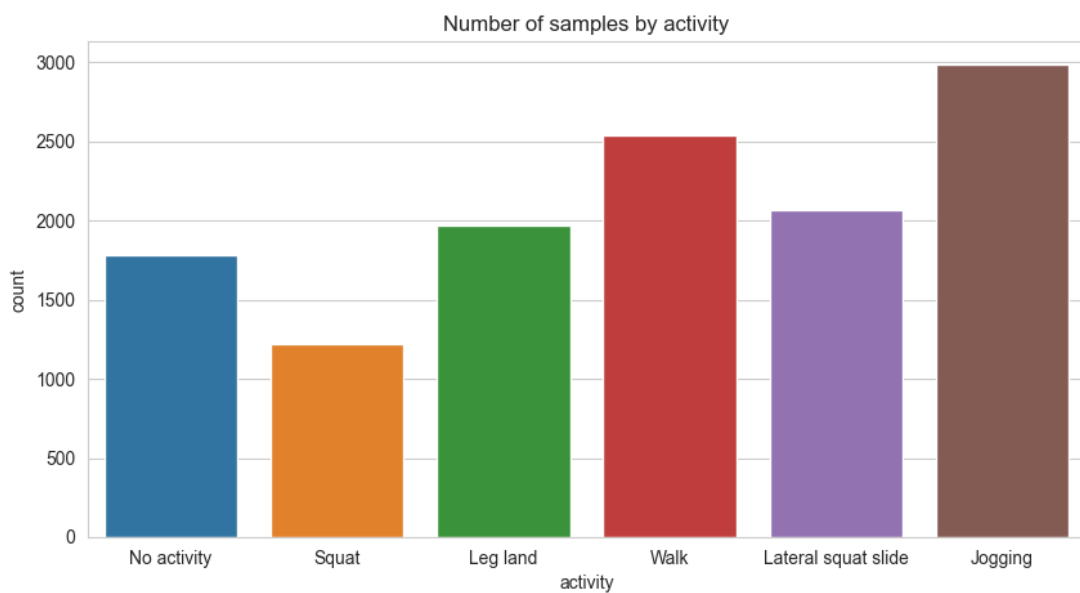


Рис. 45. Аналіз розподілу класів тестового датасету

Порівнявши результати, зображені на рис. 22 і рис. 45, можна зробити висновок, що у тестовому датасеті значно більше записів класу 'No activity' порівняно із іншими класами. Пояснюється це тим, що тестовий датасет записувався безперервно (тобто, не був утворений шляхом об'єднання декількох менших датасетів, записаних у різні дні, як тренувальний) і тому для відпочинку м'язів було виділено більше часу.

Оскільки клас 'No activity' було видалено у тренувальному датасеті, здійснимо аналогічну операцію і для тестового датасету (рис. 46-47).

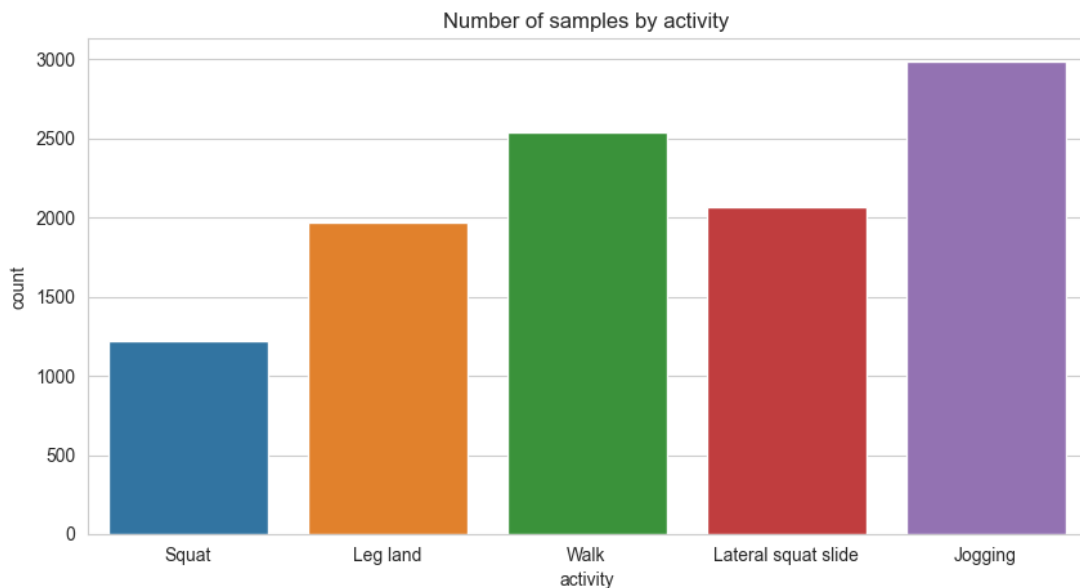


Рис. 46. Аналіз розподілу класів тестового датасету після видалення класу 'No activity'

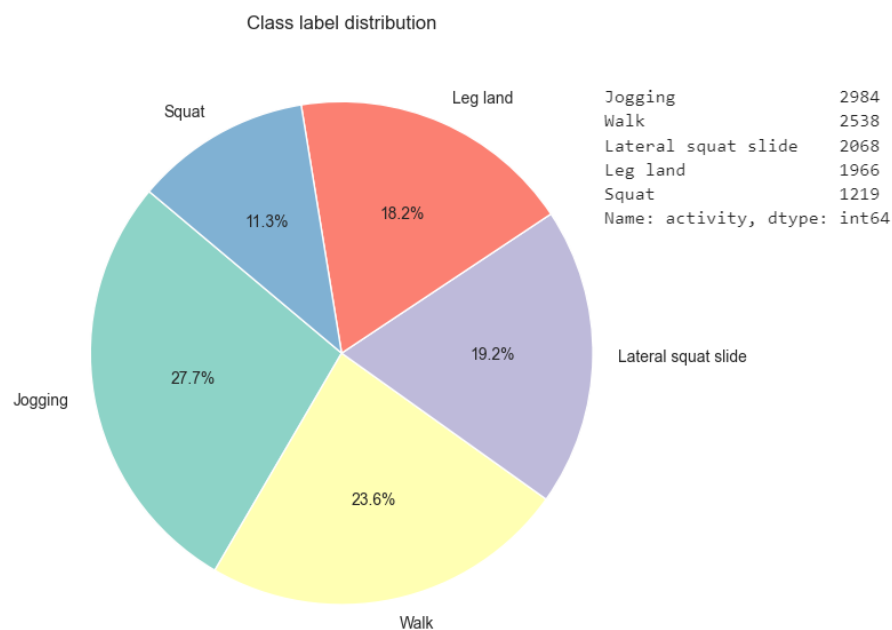


Рис. 47. Аналіз розподілу класів тестового датасету у процентному співвідношенні



Із рис. 46 видно, що тестовий датасет є незбалансованим. Балансування тренувального набору даних важливе для того, щоб модель вчилася з усіх класів і не стала упередженою до певного класу. Однак баланс тестового датасету не впливає на процес навчання моделі, оскільки модель не піддається впливу цих даних під час навчання. Саме тому не обов'язково робити тестовий набір даних збалансованим. Коли мова заходить про тестовий датасет, головне завдання полягає в тому, щоб переконаватися, що він є репрезентативним для реального розподілу класів, з якими зіткнеться модель. Накладання балансу на тестовий набір даних може призвести до нереалістичних результатів оцінки, оскільки дані реального світу навряд чи будуть збалансованими. Тому, на відміну від тренувального набору даних, тестовий не буде збалансованим.

Так само, як і для тренувального датасету, для тестового вилучимо осі ОХ і ОУ гіроскопа та виконаємо віконування із розміром вікна 2 с, в результаті чого із 10775 записів отримано 360 записів (рис. 48).

```
print(f'len(test_filtered_df) = {len(test_filtered_df)}')
print(f'len(test_windowed_df) = {len(test_windowed_df)}')

len(test_filtered_df) = 10775
len(test_windowed_df) = 360
```

Рис. 48. Довжини датафреймів test\_filtered\_df та test\_windowed\_df

Так само, як і для тренувального датасету X\_train, для X\_test обчислимо 12 тих самих простих статистичних параметрів.

Також поділимо тренувальний датасет на дані для тренування моделі (X\_train, y\_train) та дані для її валідації (X\_valid, y\_valid) у співвідношенні 80:20 (1439 записів – для тренування моделі і 360 записів для валідації). Перевіримо процентне співвідношення класів в утворених тренувальному та валідаційному датасетах (рис. 49).

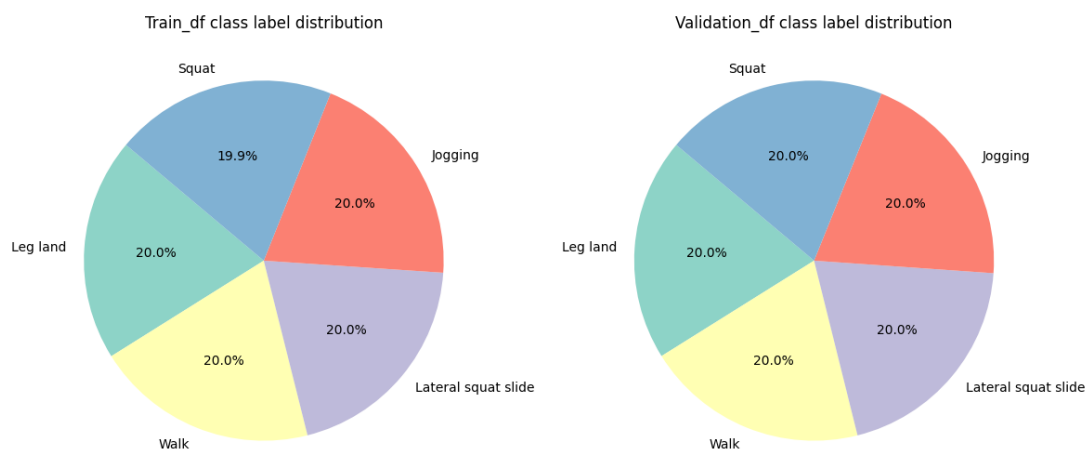


Рис. 49. Розподіл класів в train\_df та valid\_df

Із рис. 49 видно, що внаслідок поділу тренувального датасету на тренувальні та валідаційні дані, процентне співвідношення класів не змінилося (рис. 39). На рис. 50 зображено розподіл класів фізичної активності у тренувальному наборі даних.

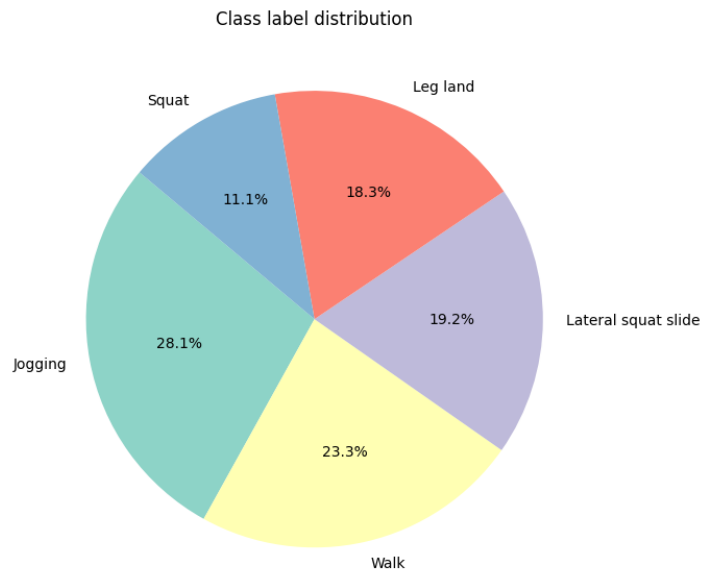


Рис. 50. Розподіл класів в test\_df

Розміри наборів даних для тренування, валідації та тестування моделі нейронної мережі відповідно дорівнюють 1439, 360 та 360 записів (рис. 51).

```
len(X_train) = 1439
len(y_train) = 1439
len(X_valid) = 360
len(y_valid) = 360
len(X_test) = 360
len(y_test) = 360
```

Рис. 51. Розміри тренувального, валідаційного та тестового датасетів

Також, оскільки завдання класифікації різних видів фізичної активності є мультикласовою задачею, потрібно перетворити перетворюєте числові мітки класів у одноразове кодоване (one-hot encoded) представлення (рис. 52).

```
# Convert Label Encoded target data to one-hot encoded format
y_train_encoded = to_categorical(y_train)
y_valid_encoded = to_categorical(y_valid)
y_test_encoded = to_categorical(y_test)
```

Рис. 52. Виконання one-hot encoding для y\_train, y\_valid, y\_test

Також було здійснено нормалізацію векторів ознак (X\_train, X\_valid, X\_test) за допомогою стандартного нормалізатора (StandardScaler) (рис. 53).

```
# Initialize the StandardScaler
scaler = StandardScaler()

# Fit the scaler on the training data and transform it
X_train_scaled = scaler.fit_transform(X_train)

# Transform the validation and test data using the same scaler
X_valid_scaled = scaler.transform(X_valid)
X_test_scaled = scaler.transform(X_test)
```

Рис. 53. Нормалізація векторів ознак

Оскільки нейронні мережі є стохастичними алгоритмами (той самий алгоритм на тих самих даних може навчати іншу модель з різними навичками кожного разу, коли виконується код), для кожної архітектури проводитимемо 5 дослідів і знаходитимемо середнє значення точності моделі. Вихідний шар моделі має 5 нейронів, активаційна функція - 'softmax'.

Також варто зазначити, що під час виконання експериментів модель із найкращою точністю зберігається в окремій змінній – best\_model.

Для тренування моделі використано різні архітектури нейронних мереж (різна кількість нейронів, прихованих шарів, функції активації, оптимізатори). Результати експериментів наведено у табл. 1.

Таблиця 1.

Тестування архітектур нейронних мереж

Архітектура моделі	Оптимізатор	Кількість епох	Середня точність на тестовому наборі, %	Найкраща точність на тестовому наборі, %
1-ий прих. шар: 40, 'relu'; 2-ий прих. шар: 20, 'relu'; 3-ій прих. шар: 10, 'sigmoid'	adam	10	98.278	98.889
1-ий прих. шар: 60, 'relu'; 2-ий прих. шар: 30, 'relu'; 3-ій прих. шар: 15, 'sigmoid'	adam	10	98.556	98.889
1-ий прих. шар: 100, 'relu'; 2-ий прих. шар: 50, 'relu'; 3-ій прих. шар: 25, 'sigmoid'	adam	10	97.833	98.611
1-ий прих. шар: 60, 'relu'; 2-ий прих. шар: 30, 'relu'; 3-ій прих. шар: 15, 'sigmoid'	SGD	10	97.778	98.889
1-ий прих. шар: 60, 'relu'; 2-ий прих. шар: 30, 'relu'; 3-ій прих. шар: 15, 'sigmoid'	RMSprop	10	<b>98.556</b>	<b>99.167</b>
1-ий прих. шар: 60, 'relu'; 2-ий прих. шар: 30, 'relu'; 3-ій прих. шар: 15, 'sigmoid'	Nadam	10	98.111	99.167

Продовження таблиці 1.

Архітектура моделі	Оптимізатор	Кількість епох	Середня точність на тестовому наборі, %	Найкраща точність на тестовому наборі, %
1-ий прих. шар: 60, 'relu'; 2-ий прих. шар: 30, 'relu'; 3-ій прих. шар: 15, 'sigmoid'	Adagrad	10	51.278	61.944
1-ий прих. шар: 60, 'relu'; 2-ий прих. шар: 30, 'relu'; 3-ій прих. шар: 15, 'LeakyReLU'	RMSprop	10	98.056	98.889
1-ий прих. шар: 60, 'relu'; 2-ий прих. шар: 30, 'relu'; 3-ій прих. шар: 15, 'tanh'	RMSprop	10	97.889	98.611
1-ий прих. шар: 60, 'swish'; 2-ий прих. шар: 30, 'swish'; 3-ій прих. шар: 15, 'sigmoid'	RMSprop	10	<b>98.667</b>	<b>99.167</b>
1-ий прих. шар: 60, 'relu'; 2-ий прих. шар: 30, 'relu'; 3-ій прих. шар: 15, 'sigmoid'	RMSprop	15	98.056 (рис. 54)	98.333
1-ий прих. шар: 60, 'relu'; 2-ий прих. шар: 30, 'relu'; 3-ій прих. шар: 15, 'sigmoid'	RMSprop	5	<b>98.556</b> (рис. 55)	<b>99.167</b>
1-ий прих. шар: 80, 'relu'; 2-ий прих. шар: 40, 'relu'; 3-ій прих. шар: 20, 'relu'; 4-ий прих. шар: 10, 'sigmoid'	RMSprop	5	98.111	98.889
1-ий прих. шар: 120, 'relu'; 2-ий прих. шар: 60, 'relu'; 3-ій прих. шар: 30, 'relu'; 4-ий прих. шар: 15, 'sigmoid'	RMSprop	5	97.444	98.889

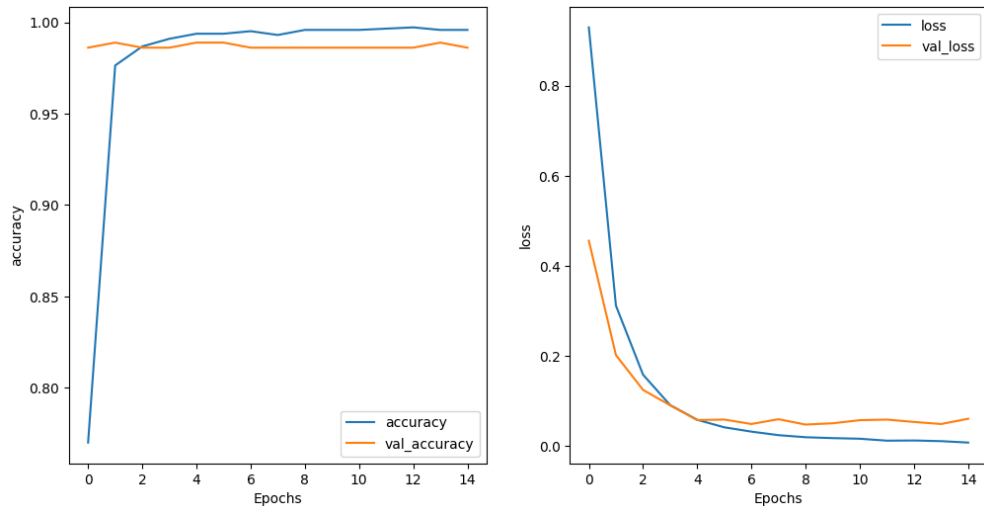


Рис. 54. Зміна тренувальних і валідаційних точності і втрат протягом 15 епох під час навчання моделі 60/30/15/5

Із рис. 54 видно, що втрати валідації приблизно після 5 епох перевищують тренувальні втрати, а з 8-ої епохи починають збільшуватися. Це означає, що модель починає перенавчатися (overfitting), тому для подальшого тренування моделей кількість епох було зменшено до 5.

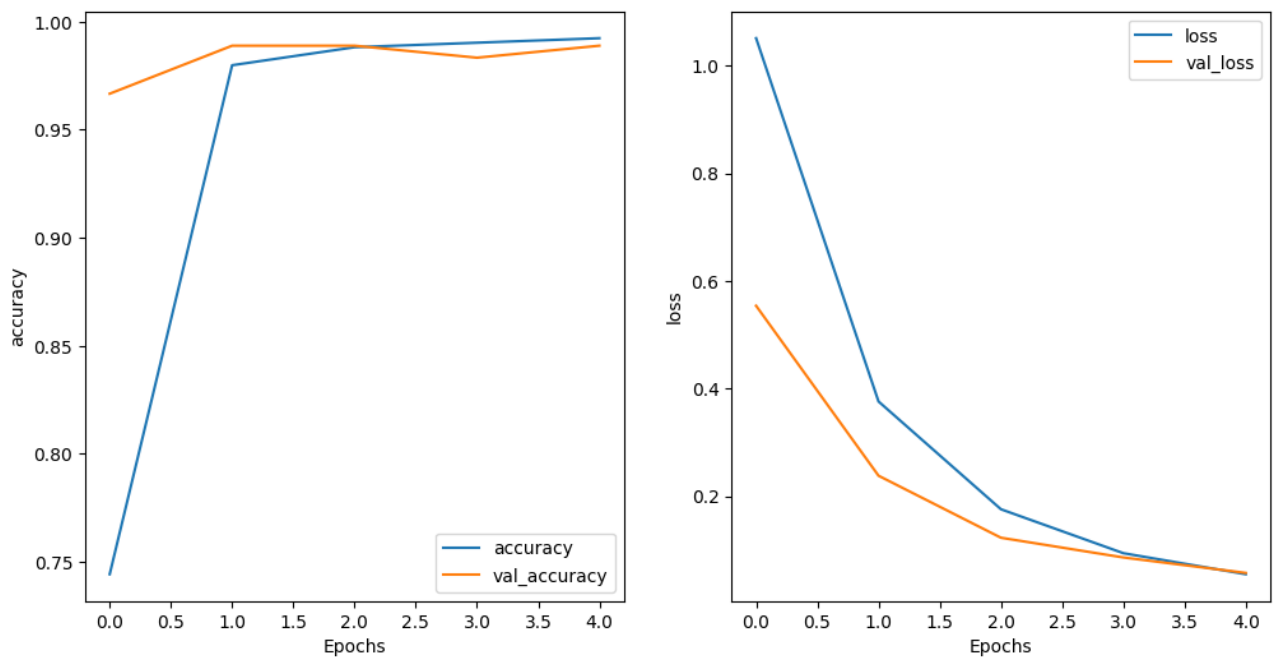


Рис. 55. Зміна тренувальних і валідаційних точності і втрат протягом 5 епох під час навчання моделі 60/30/15/5

Як бачимо із рис. 55 і табл. 1, зменшення кількості епох до 5 призвело до покращення точності моделі нейронної мережі.

Отже, найбільша точність на тренувальному датасеті була отримана для моделі 60/30/15/5 з оптимізатором RMSprop і її значення становить 99.167%.

На рис. 56 наведено зміну тренувальних і валідаційних точності і втрат під час тренування "найкращої" моделі 60/30/15/5, на рис. 57 – матрицю плутанини, а на рис. 58 – класифікаційний звіт (classification report).

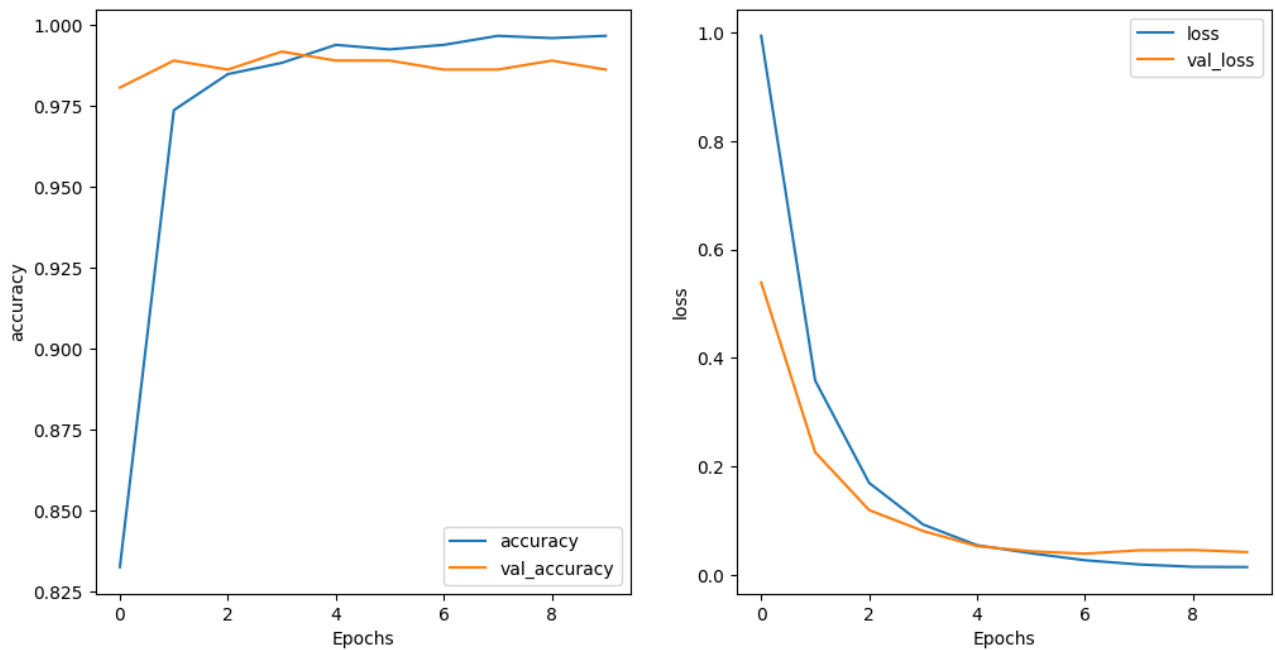


Рис. 56. Зміна тренувальних і валідаційних точності і втрат протягом 10 епох під час навчання "найкращої" моделі 60/30/15/5

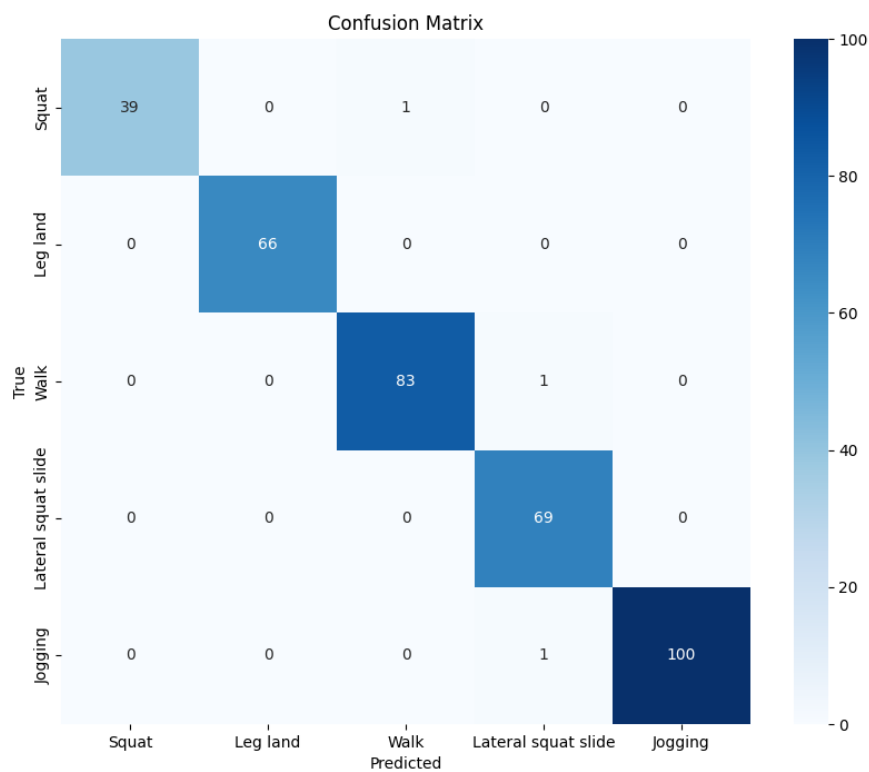


Рис. 57. Матриця плутанини "найкращої" моделі 60/30/15/5

Classification Report:				
	precision	recall	f1-score	support
Squat	1.00	0.97	0.99	40
Leg land	1.00	1.00	1.00	66
Walk	0.99	0.99	0.99	84
Lateral squat slide	0.97	1.00	0.99	69
Jogging	1.00	0.99	1.00	101
accuracy			0.99	360
macro avg	0.99	0.99	0.99	360
weighted avg	0.99	0.99	0.99	360

Рис. 58. Класифікаційний звіт "найкращої" моделі 60/30/15/5

Розглянувши класифікаційний звіт "найкращої" моделі 60/30/15/5 (рис. 58), можна зробити такі висновки:

1) 'Squat':

- Precision = 1.00 – оскільки всі присідання із тестового набору даних були класифіковані моделлю нейронної мережі правильно (False Positives = 0 – немає випадків, коли модель класифікувала певну фізичну активність як присідання, але це були не присідання);
- Recall = 0.97 – оскільки модель класифікувала неправильно один запис 'Walk' як 'Squat' (False Negatives = 1 – модель класифікувала вид фізичної діяльності як 'не присідання', але насправді це було присідання);
- F1-score = 0.99 – оскільки Recall = 0.97.

2) 'Leg land': всі показники дорівнюють 1.00, що вказує на відсутність False Positive (неправильно класифікованих записів 'Leg land') та False Negative (інші види фізичної активності помилково класифіковані як 'Leg land').

3) 'Walk':

- Precision = 0.99 – тому що 1 запис тестового датасету модель класифікувала як 'Walk', але насправді це була активність 'Squat' (FP = 1);
- Recall = 0.99 – оскільки 1 раз модель класифікувала фізичну активність як 'Lateral squat slide', але насправді це була ходьба (FN=1);
- F1-score = 0.99 – оскільки Precision = 0.99 та Recall = 0.99.

4) 'Lateral squat slide':

- Precision = 0.97 – модель двічі неправильно класифікувала фізичну активність як 'Lateral squat slide': один раз сплутала із 'Jogging', другий раз – із 'Walk' (FP = 1);

- $\text{Recall} = 1.00$  – тому що модель не сплутала жоден інший вид фізичної активності із 'Lateral squat slide' ( $\text{FN} = 0$ );
- $\text{F1-score} = 0.99$  – оскільки  $\text{Precision} = 0.97$ .

#### 5) 'Jogging':

- $\text{Precision} = 1.00$  – модель правильно класифікувала види активності 'Jogging', не присвоївши цю мітку іншим видам активності, тобто всі види активності, позначені моделлю як 'Jogging', насправді були 'Jogging' ( $\text{FP} = 0$ );
- $\text{Recall} = 0.99$  – модель нейронної мережі один раз сплутала 'Lateral squat slide' із 'Jogging', тобто класифікувала вид фізичної активності як 'не Jogging' ('Lateral squat slide'), але насправді це була активність 'Jogging' ( $\text{FN} = 1$ ).
- $\text{F1-score} = 1.00$  – точне значення становить:  $2 * 1.00 * 0.99 / (1.00 + 0.99) = 0.99497$ .

Проаналізувавши класифікаційний звіт, можна зробити висновок, що найкращі результати модель показала для випадів на ноги (клас 'Leg land').

### Етап 10: Feature Selection

Визначивши оптимальну архітектуру моделі нейронної мережі, використаємо декілька алгоритмів Feature Selection та порівняємо отримані результати.

Бажано зменшувати кількість вхідних змінних, щоб зменшити обчислювальні витрати на моделювання та, у деяких випадках, покращити продуктивність моделі. Саме для цього і потрібен етап Feature Selection.

Для зменшення кількості вхідних параметрів моделі використаємо такі алгоритми feature selection:

- 1) **Permutation Feature Importance:** цей метод працює шляхом перетасування значень кожної ознаки та вимірювання падіння продуктивності моделі. Якщо ознака важлива, перетасування її значень більше вплине на продуктивність моделі.
- 2) **MDI (Mean Decrease Impurity):** цей метод часто використовується для визначення важливості ознак в моделях на основі дерева, таких як випадкові ліси. Він розраховує зменшення домішок (наприклад, домішок Джіні), викликаних кожною ознакою, коли використовується для поділу даних у деревах рішень.
- 3) **RFE (Recursive Feature Elimination):** це техніка, яка рекурсивно видаляє найменш важливі ознаки з набору даних під час навчання та оцінки моделі. Це особливо корисно, коли є велика кількість функцій і потрібно визначити найбільш релевантну підмножину для моделі.



Результати алгоритму Permutation Feature Importance наведені на рис. 59.

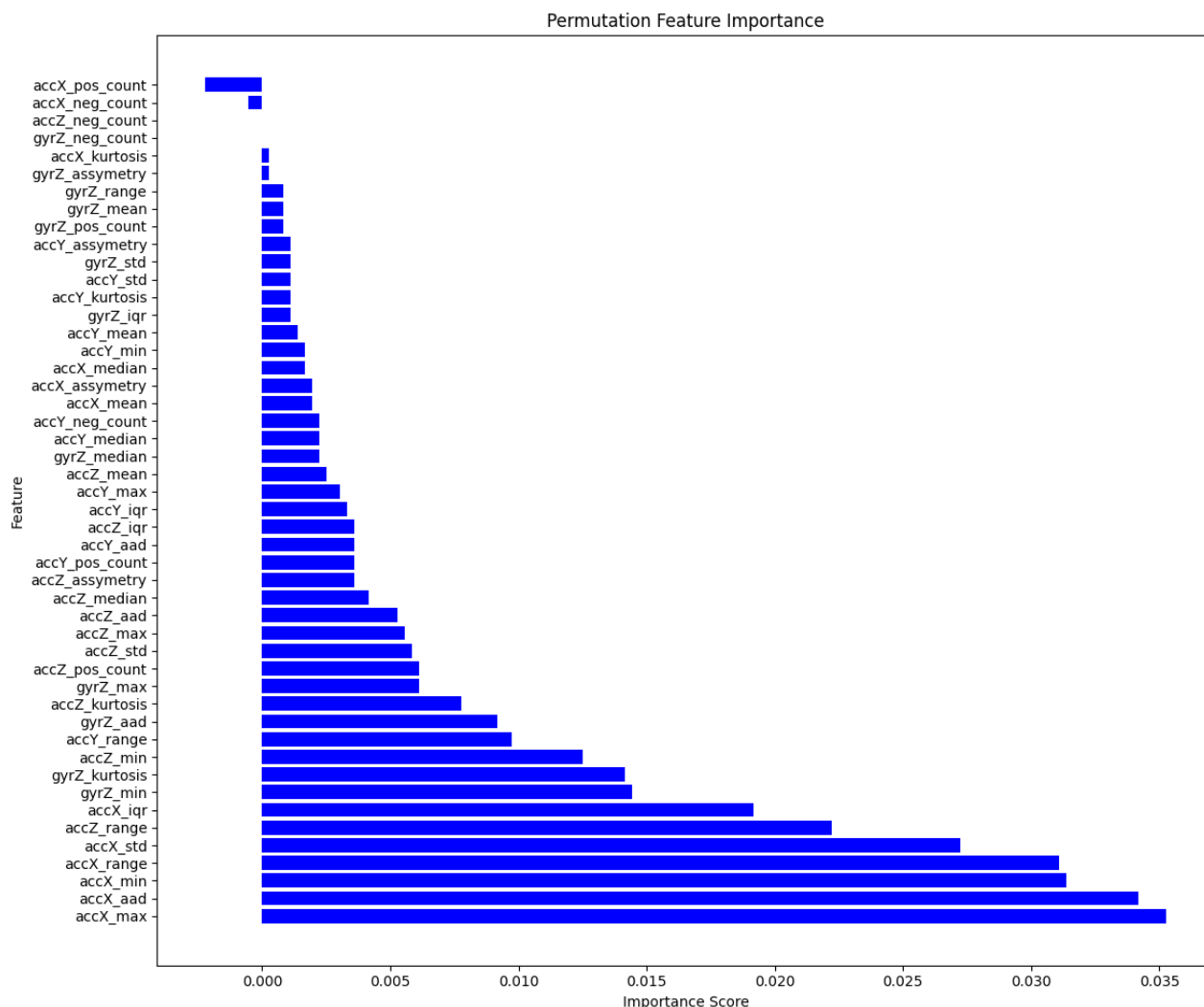


Рис. 59. Результат виконання алгоритму Permutation Feature Importance для "найкращої" моделі 60/30/15/5

Проаналізувавши рис. 59, можна зробити такі висновки:

- 1) Смуги 'accX\_pos\_count' та 'accX\_neg\_count' мають негативний напрям. Це вказує на те, що зміна значень цих характеристик призвела до зниження продуктивності моделі. Іншими словами, ці функції позитивно впливають на точність моделі. Негативні напрямні смуги, як правило, є хорошим знаком, оскільки вони представляють особливості, які позитивно впливають на здатність моделі правильно класифікувати нові дані. Тому ці ознаки видаляти не будемо.
- 2) Відсутність смуг для 'accZ\_neg\_count' та 'gyrZ\_neg\_count' означає, що важливість цих ознак для моделі нейронної мережі дорівнює 0. Тому ці ознаки можна спробувати видалити.
- 3) Всі інші ознаки мають смуги позитивного напрямку. Це вказує на те, що перестановка значень цих ознак призвела до підвищення продуктивності моделі. Чим довша смуга, тим важливіша ознака. Якщо

смуга доволі довга, це означає, що відповідна ознака має суттєвий вплив на продуктивність моделі. Тому ознаки із відносно малими позитивними смугами (наприклад, 'accX\_curtosis', 'gyrZ\_assymetry' та інші) можна також вилучити.

Результати тренування моделі нейронної мережі із "найкращою" архітектурою (60/30/15/5; 'relu'/'relu'/'sigmoid'/'softmax', оптимізатор - RMSprop) наведено у табл. 2.

Таблиця 2.

Тестування архітектури 60/30/15/5 для різної кількості ознак (Permutation Feature Importance)

Список ознак	Кількість ознак	Середня точність на тестовому наборі, %	Найкраща точність на тестовому наборі, %
Всі ознаки	48	98.556	99.167
'accX_max', 'accX_aad', 'accX_min', 'accX_range', 'accX_std', 'accZ_range', 'accX_iqr', 'gyrZ_min', 'gyrZ_kurtosis', 'accZ_min', 'accY_range', 'gyrZ_aad', 'accZ_kurtosis', 'gyrZ_max', 'accZ_pos_count', 'accZ_std', 'accZ_max', 'accZ_aad', 'accZ_median', 'accZ_assymetry', 'accY_pos_count', 'accY_aad', 'accZ_iqr', 'accY_iqr', 'accY_max', 'accZ_mean', 'gyrZ_median', 'accY_median', 'accX_neg_count', 'accY_pos_count'	30	98.167	<b>99.444</b>
'accX_max', 'accX_aad', 'accX_min', 'accX_range', 'accX_std', 'accZ_range', 'accX_iqr', 'gyrZ_min', 'gyrZ_kurtosis', 'accZ_min', 'accY_range', 'gyrZ_aad', 'accZ_kurtosis', 'gyrZ_max', 'accZ_pos_count', 'accZ_std', 'accZ_max', 'accZ_aad', 'accX_neg_count', 'accY_pos_count'	20	97.944	99.167
'accX_max', 'accX_aad', 'accX_min', 'accX_range', 'accX_std', 'accZ_range', 'accX_iqr', 'gyrZ_min', 'gyrZ_kurtosis', 'accZ_min', 'accY_range', 'gyrZ_aad', 'accZ_kurtosis', 'accX_neg_count', 'accY_pos_count'	15	97.611	98.889

Продовження таблиці 2.

Список ознак	Кількість ознак	Середня точність на тестовому наборі, %	Найкраща точність на тестовому наборі, %
'accX_max', 'accX_aad', 'accX_min', 'accX_range', 'accX_std', 'accZ_range', 'accX_iqr', 'gyrZ_min', 'accX_neg_count', 'accY_pos_count'	10	85.500	89.944

Із табл. 2 видно, що модель видає хороші результати (точність понад 95%) навіть коли кількість ознак становить 15.

Отже, завдяки використанню алгоритма Permutation Feature Importance вдалося зменшити кількість ознак від 48 до 15 без значної втрати точності (від 98.556% до 97.611%). Це дозволяє стверджувати, що алгоритм Permutation Feature Importance успішно виявив і зберіг найважливіші ознаки для моделі.

Наступним досліджуванним Feature Selection алгоритмом буде MDI (Середнє зменшення домішки), результати використання якого наведено на рис. 60.

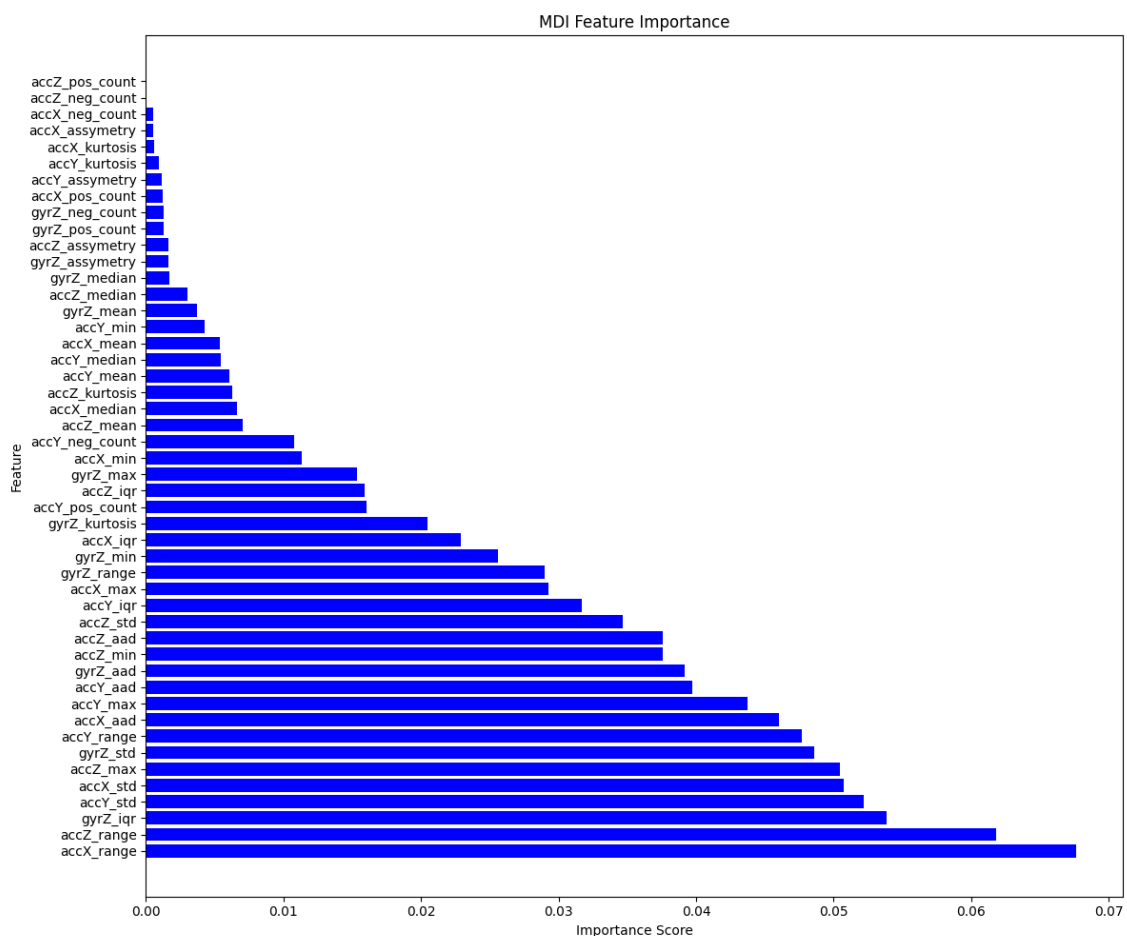


Рис. 60. Результат виконання алгоритму MDI (Mean Decrease Impurity)

Порівнявши результати алгоритмів Permutation Feature Importance (рис. 59) та MDI (рис. 60), можна помітити, що вони доволі сильно відрізняються.

Результати тренування моделі нейронної мережі із "найкращою" архітектурою для різної кількості ознак, визначених з допомогою алгоритму MDI наведено у табл. 3.

Таблиця 3.

Тестування архітектури 60/30/15/5 для різної кількості ознак (MDI - Mean Decrease Impurity)

Список ознак	Кількість ознак	Середня точність на тестовому наборі, %	Найкраща точність на тестовому наборі, %
Всі ознаки	48	98.556	99.167
'accX_range', 'accZ_range', 'gyrZ_iqr', 'accY_std', 'accX_std', 'accZ_max', 'gyrZ_std', 'accY_range', 'accX_aad', 'accY_max', 'accY_aad', 'gyrZ_aad', 'accZ_min', 'accZ_aad', 'accZ_std', 'accY_iqr', 'accX_max', 'gyrZ_range', 'gyrZ_min', 'accX_iqr', 'gyrZ_kurtosis', 'accY_pos_count', 'accZ_iqr', 'gyrZ_max', 'accX_min', 'accY_neg_count', 'accZ_mean', 'accX_median', 'accZ_kurtosis', 'accY_mean'	30	98.333	98.611
'accX_range', 'accZ_range', 'gyrZ_iqr', 'accY_std', 'accX_std', 'accZ_max', 'gyrZ_std', 'accY_range', 'accX_aad', 'accY_max', 'accY_aad', 'gyrZ_aad', 'accZ_min', 'accZ_aad', 'accZ_std', 'accY_iqr', 'accX_max', 'gyrZ_range', 'gyrZ_min', 'accX_iqr',	20	95.833	96.667
'accX_range', 'accZ_range', 'gyrZ_iqr', 'accY_std', 'accX_std', 'accZ_max', 'gyrZ_std', 'accY_range', 'accX_aad', 'accY_max', 'accY_aad', 'gyrZ_aad', 'accZ_min', 'accZ_aad', 'accZ_std'	15	96.056	96.667

Продовження таблиці 3.

Список ознак	Кількість ознак	Середня точність на тестовому наборі, %	Найкраща точність на тестовому наборі, %
'accX_range', 'accZ_range', 'gyrZ_iqr', 'accY_std', 'accX_std', 'accZ_max', 'gyrZ_std', 'accY_range', 'accX_aad', 'accY_max',	10	95.444	96.389
'accX_range', 'accZ_range', 'gyrZ_iqr', 'accY_std', 'accX_std'	5	94.111	95.278
'accX_range', 'accZ_range', 'gyrZ_iqr', 'accY_std'	4	93.556	94.444
'accX_range', 'accZ_range', 'gyrZ_iqr'	3	82.333	83.889

Результати, наведені у табл. 3 показують, що алгоритм MDI визначив ключові ознаки навіть краще, ніж Permutation Feature Importance, оскільки дозволив зменшити кількість ознак до 5 без значної втрати точності моделі нейронної мережі (точність може перевищувати 95%).

Останнім досліджуваним алгоритмом Feature Selection є RFE (Recursive Feature Elimination), результати використання якого наведено на рис. 61.

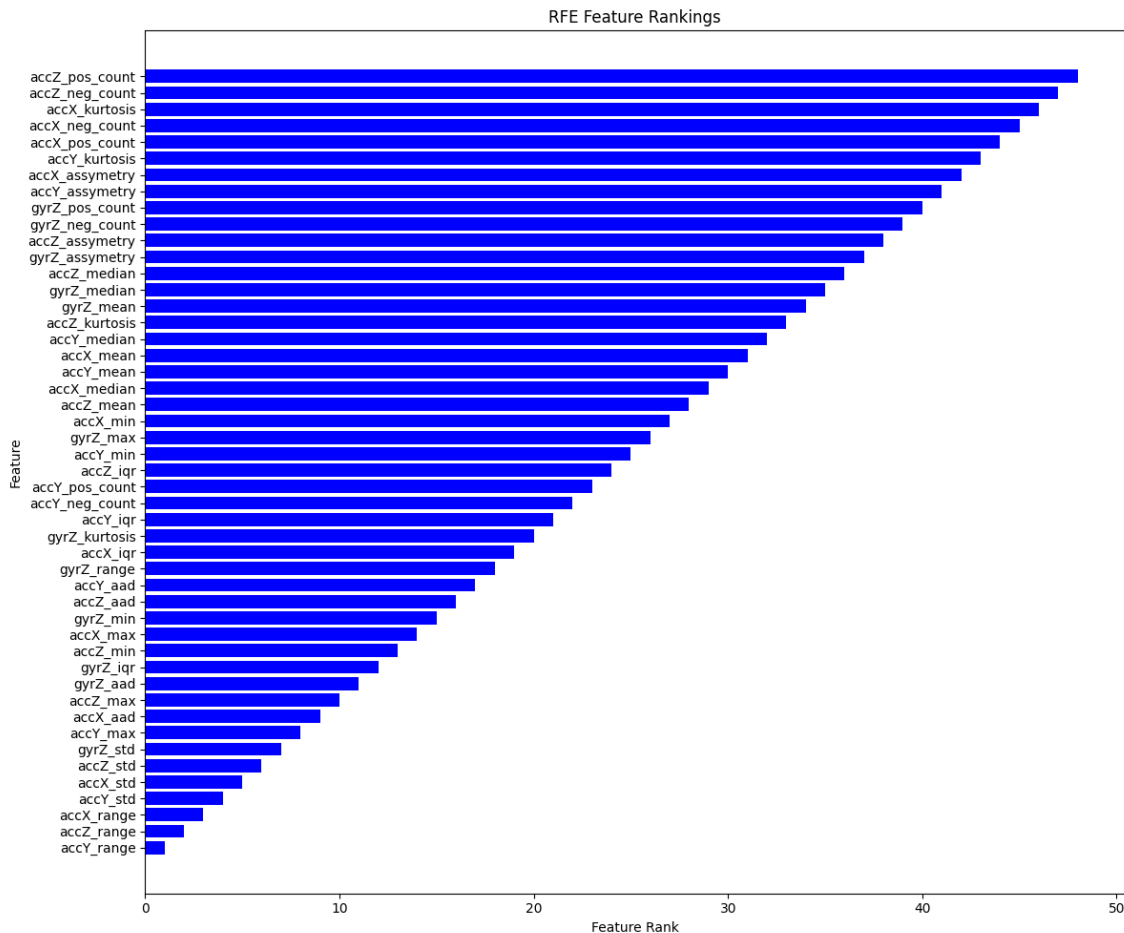


Рис. 61. Результат виконання алгоритму RFE (Recursive Feature Elimination)

RFE — це метод, який використовується для вибору ознак, і в ході експерименту він був застосований до моделі RandomForestClassifier бібліотеки Scikit-Learn.

Значення ранжування ознаки вказує на відносну важливість кожної ознаки відповідно до оцінки алгоритму RFE. Ознаки з нижчими значеннями рейтингу для алгоритму RFE вважаються більш важливими (є ознаками, які найбільше сприяють продуктивності моделі). Тобто, ознаки з нижчим рейтингом (ближче до 1) вважаються більш важливими алгоритмом RFE.

Результати тренування моделі нейронної мережі із "найкращою" архітектурою для різної кількості ознак, визначених з допомогою алгоритму RFE наведено у табл. 4.

Таблиця 4.

Тестування архітектури 60/30/15/5 для різної кількості ознак (RFE - Recursive Feature Elimination)

Список ознак	Кількість ознак	Середня точність на тестовому наборі, %	Найкраща точність на тестовому наборі, %
Всі ознаки	48	98.556	99.167
'accY_range', 'accZ_range', 'accX_range', 'accY_std', 'accX_std', 'accZ_std', 'gyrZ_std', 'accY_max', 'accX_aad', 'accZ_max', 'gyrZ_aad', 'gyrZ_iqr', 'accZ_min', 'accX_max', 'gyrZ_min', 'accZ_aad', 'accY_aad', 'gyrZ_range', 'accX_iqr', 'gyrZ_kurtosis', 'accY_iqr', 'accY_neg_count', 'accY_pos_count', 'accZ_iqr', 'accY_min', 'gyrZ_max', 'accX_min', 'accZ_mean', 'accX_median', 'accY_mean'	30	98.056	98.333
'accY_range', 'accZ_range', 'accX_range', 'accY_std', 'accX_std', 'accZ_std', 'gyrZ_std', 'accY_max', 'accX_aad', 'accZ_max', 'gyrZ_aad', 'gyrZ_iqr', 'accZ_min', 'accX_max', 'gyrZ_min', 'accZ_aad', 'accY_aad', 'gyrZ_range', 'accX_iqr', 'gyrZ_kurtosis'	20	97.222	97.500
'accY_range', 'accZ_range', 'accX_range', 'accY_std', 'accX_std', 'accZ_std', 'gyrZ_std', 'accY_max', 'accX_aad', 'accZ_max', 'gyrZ_aad', 'gyrZ_iqr', 'accZ_min', 'accX_max', 'gyrZ_min'	15	95.500	96.389
'accY_range', 'accZ_range', 'accX_range', 'accY_std', 'accX_std', 'accZ_std', 'gyrZ_std', 'accY_max', 'accX_aad', 'accZ_max'	10	95.444	96.111
'accY_range', 'accZ_range', 'accX_range', 'accY_std', 'accX_std'	5	93.944	95.000
'accY_range', 'accZ_range', 'accX_range', 'accY_std'	4	95.333	96.667
'accY_range', 'accZ_range', 'accX_range'	3	95.111	95.833
'accY_range', 'accZ_range'	2	86.444	88.889
'accY_range'	1	69.111	72.778

Результати, наведені у табл. 4 показують, що алгоритм RFE визначив ключові ознаки краще, ніж Permutation Feature Importance та MDI, оскільки дозволив зменшити кількість ознак до 3 без значної втрати точності моделі нейронної мережі (середня точність перевищує 95%).