KLASIFIKASI AKTIVITAS MANUSIA MENGGUNAKAN ALGORITME COMPUTED INPUT WEIGHT EXTREME LEARNING MACHINE DENGAN REDUKSI DIMENSI PRINCIPAL COMPONENT ANALYSIS

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
M. Sofyan Irwanto
NIM: 175150218113036

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2021

PENGESAHAN

KLASIFIKASI AKTIVITAS MANUSIA MENGGUNAKAN ALGORITME COMPUTED INPUT WEIGHT EXTREME LEARNING MACHINE DENGAN REDUKSI DIMENSI PRINCIPAL COMPONENT ANALYSIS

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh:
M. Sofyan Irwanto
NIM: 175150218113036

Skripsi ini telah diuji dan dinyatakan lulus pada
14 Juli 2021
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing 2
Dr. Eng. Fitra Abdurrachman Bachtiar, S.T., M.Eng.	Dr. Eng. Novanto Yudistira, S.Kom., M.Sc.
NIP: 19840628 201903 1 006	NIK: 201201 831110 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Achmad Basuki, S.T., M.MG., Ph.D.
NIP: 19741118 200312 1 002

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar referensi.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 27 Juli 2021

M. Sofyan Irwanto

NIM: 175150218113036

PRAKATA

Puji syukur selalu kita panjatkan kehadirat Tuhan Yang Maha Esa, yang telah melimpahkan rahmat, kasih sayang, dan pertolongan-Nya kepada kita semua, sehingga skripsi yang berjudul "Klasifikasi Aktivitas Manusia Menggunakan Algoritme Computed Input Weight Extreme Learning Machine dengan Reduksi Dimensi Principal Component Analysis" dapat terselesaikan dengan baik.

Dalam proses penyusunan skripsi ini, penulis sadar bahwa skripsi ini dapat terselesaikan berkat bantuan dari beberapa pihak yang turut membantu penulis. Oleh karena itu, penulis ingin menyampaikan rasa terima kasih kepada:

1. Bapak Dr. Eng. Fitra Abdurrachman Bachtiar, S.T., M.Eng. selaku dosen pembimbing 1 yang telah memberikan masukan, bimbingan, serta arahan selama proses pengerjaan skripsi.
2. Bapak Dr. Eng. Novanto Yudistira, S.Kom., M.Sc. selaku dosen pembimbing 2 yang juga telah memberikan masukan dan bimbingan dalam proses pengerjaan skripsi.
3. Bapak Achmad Basuki, S.T., M.MG., Ph.D. selaku Ketua Jurusan Teknik Informatika.
4. Bapak Adhitya Bhawiyuga, S.Kom., M.Sc. selaku Ketua Program Studi Teknik Informatika.
5. Seluruh dosen dan karyawan Fakultas Ilmu Komputer Universitas Brawijaya yang telah memberikan bantuan dan ilmu yang bermanfaat selama perkuliahan.
6. Orang tua dan seluruh keluarga yang telah memberikan kasih sayang dan nasehat kepada penulis, serta senantiasa mendo'akan yang terbaik kepada penulis.

Penulis menyadari bahwa dalam penyusunan laporan skripsi ini masih terdapat banyak kekurangan dan jauh dari kata sempurna, sehingga penulis sangat mengharapkan kritik dan saran yang membangun dari berbagai pihak. Akhir kata, penulis berhadap penelitian ini dapat memberikan manfaat bagi semua pihak.

Malang, 27 Juli 2021

M. Sofyan Irwanto

task.sofyan17@gmail.com

ABSTRAK

M. Sofyan Irwanto, Klasifikasi Aktivitas Manusia Menggunakan Algoritme Computed Input Weight Extreme Learning Machine dengan Reduksi Dimensi Principal Component Analysis

Pembimbing: Dr. Eng. Fitra Abdurrachman Bachtiar, S.T., M.Eng. dan Dr. Eng. Novanto Yudistira, S.Kom., M.Sc.

Salah satu bidang penelitian yang sangat penting yaitu pengenalan aktivitas manusia secara otomatis dikarenakan potensi penerapannya di berbagai bidang lain seperti pengawasan, lingkungan cerdas, maupun kesehatan. Dari berbagai pendekatan yang pernah dilakukan untuk mengenali aktivitas manusia, teknik berbasis sensor diketahui lebih unggul daripada teknik lain seperti teknik berbasis visi komputer. Teknik berbasis sensor juga dapat dilakukan menggunakan ponsel cerdas, namun penggunaan ponsel cerdas memiliki kekurangan dalam melakukan komputasi algoritme yang kompleks. Apalagi, data hasil ekstraksi fitur dari sinyal yang ditangkap oleh sensor memiliki dimensi yang tinggi. Oleh karena itu, diperlukan sebuah metode untuk mengurangi dimensi fitur dari data, serta melakukan klasifikasi terhadap data tersebut dengan cepat dan tepat. Salah satu metode yang dapat digunakan untuk mengurangi dimensi fitur dari sebuah data adalah Principal Component Analysis (PCA), dan salah satu metode klasifikasi yang dapat digunakan adalah Computed Input Weight Extreme Learning Machine (CIW-ELM). Oleh karena itu, penelitian ini akan menggunakan kedua metode tersebut untuk melakukan klasifikasi aktivitas manusia. Pada penelitian ini juga dilakukan pemilihan *hypeparameter* terbaik pada masing-masing metode menggunakan metode Grid Search Cross Validation. *Hyperparameter* terbaik yang didapatkan untuk algoritme PCA adalah dengan nilai *k* = 207, serta untuk algoritme CIW-ELM dengan jumlah *hidden neuron* = 600 dan fungsi aktivasi *sigmoid*. Hasil akurasi yang didapatkan pada penelitian ini adalah 0,957 dan rata-rata *f-measure* sebesar 0,958 dengan waktu pelatihan selama 0,57 detik.

Kata kunci: klasifikasi, aktivitas manusia, reduksi dimensi, *Principal Component Analysis, Computed Input Weight Extreme Learning Machine*

ABSTRACT

M. Sofyan Irwanto, *Classification of Human Activity Using Computed Input Weight Extreme Learning Machine Algorithm with Principal Component Analysis Dimensionality Reduction*

***Supervisors*: Dr. Eng. Fitra Abdurrachman Bachtiar, S.T., M.Eng. *and* Dr. Eng. Novanto Yudistira, S.Kom., M.Sc.**

One of the most important research area is automatic human activity recognition due to its potential application in various other fields such as surveillance, smart environment, and healthcare. Based on various approaches that have been used to recognize human activity, sensor-based techniques are known to be superior to other techniques such as computer vision-based techniques. Sensor-based technique can also be performed using smartphones, but smartphone has disadvantages in performing complex alghorithmic computation. Moreover, the data from feature extraction from the signal captured by the sensor has high dimensions. So, we need a methods to reduce the feature dimensions of the data, and classify the data quickly and accurately. One of the method that can be used to reduce the feature dimensions of data is Principal Component Analysis (PCA), and one of the classification methods that can be used is the Computed Input Weight Extreme Learning Machine (CIW-ELM). Therefore, this study will use both methods to classify human activities. In this study, the selection of the best hypeparameter for each method was also carried out using a Grid Search Cross Validation. The best hyperparameter obtained for the PCA algorithm is with a value of k = 207, and for the CIW-ELM algorithm with the number of hidden neurons = 600 and the sigmoid activation function. The accuracy results obtained in this study were 0.957 and the f-measure average were 0.958 with a training time of 0.57 seconds.

Keywords: classification, human activity, dimensionality reduction, Principal Component Analysis, Computed Input Weight Extreme Learning Machine

DAFTAR ISI

PENGESAHAN ii
PERNYATAAN ORISINALITAS iii
PRAKATA iv
ABSTRAK v
ABSTRACT vi
DAFTAR ISI vii
DAFTAR TABEL x
DAFTAR GAMBAR xii
DAFTAR KODE PROGRAM xiii
DAFTAR LAMPIRAN xiv
BAB 1 PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 4
1.3 Tujuan 4
1.4 Manfaat 4
1.5 Batasan Masalah 4
1.6 Sistematika Pembahasan 5
BAB 2 LANDASAN KEPUSTAKAAN 7
2.1 Tinjauan Pustaka 7
2.2 Aktivitas Manusia 9
2.3 Sensor 9
2.3.1 *Accelerometer* 10
2.3.2 *Gyroscope* 10
2.4 *Dimensionality Reduction* 11
2.4.1 Principal Component Analysis 11
2.5 One-Hot Encoding 12
2.6 Klasifikasi 13
2.6.1 Extreme Learning Machine 13
2.6.2 Computed Input Weight Extreme Learning Machine 15
2.6.3 Fungsi Aktivasi 16

2.7 K-Fold Cross Validation 17
2.8 Grid Search Cross Validation 17
2.9 Metode Evaluasi 18
BAB 3 METODOLOGI 19
3.1 Strategi Penelitian 19
3.2 Tipe Penelitian 19
3.3 Lokasi Penelitian 19
3.4 Peralatan Pendukung 20
3.4.1 Perangkat Keras 20
3.4.2 Perangkat Lunak 20
3.5 Studi Literatur 20
3.6 Pengumpulan Data 20
3.7 Implementasi Metode 21
3.7.1 Reduksi Dimensi 21
3.7.2 Klasifikasi 21
3.8 Pengujian dan Analisis 22
3.9 Kesimpulan dan Saran 22
BAB 4 PERANCANGAN 23
4.1 Identifikasi Permasalahan 23
4.2 Perancangan Algoritme 23
4.2.1 Proses One-Hot Encoding 24
4.2.2 Proses PCA 25
4.2.3 Proses CIW-ELM 26
4.2.4 Proses Evaluasi 29
4.3 Manualisasi 30
4.3.1 Pengambilan Data Sampel 30
4.3.2 Manualisasi One-Hot Encoding 31
4.3.3 Manualisasi PCA 33
4.3.4 Manualisasi CIW-ELM 39
4.3.5 Manualisasi Evaluasi 45
4.4 Perancangan Pengujian 46
BAB 5 IMPLEMENTASI 49

5.1 Implementasi Algoritme 49
5.2 Implementasi One-Hot Encoding 49
5.3 Implementasi PCA 51
5.4 Implementasi CIW-ELM 52
5.5 Implementasi K-Fold Cross Validation 56
5.6 Implementasi Metrik Evaluasi 59
5.7 Implementasi Utama 61
BAB 6 HASIL DAN PEMBAHASAN 65
6.1 Hasil 65
6.1.1 Hasil Pengujian *Hyperparameter* 65
6.1.2 Hasil Pengujian Akhir 70
6.2 Pembahasan 70
6.2.1 Pembahasan Hasil Pengujian *Hyperparameter* 70
6.2.2 Pembahasan Hasil Pengujian Akhir 72
BAB 7 PENUTUP 73
7.1 Kesimpulan 73
7.2 Saran 74
DAFTAR REFERENSI 75
LAMPIRAN A HASIL PENGUJIAN *HYPERPARAMETER* 79

DAFTAR TABEL

Tabel 4.1 Data latih untuk manualisasi 31
Tabel 4.2 Data uji untuk manualisasi 31
Tabel 4.3 Hasil One-Hot Encoding pada label data latih 32
Tabel 4.4 Hasil One-Hot Encoding pada label data uji 32
Tabel 4.5 Hasil perhitungan nilai rata-rata pada data latih 33
Tabel 4.6 Hasil perhitungan nilai data baru dari data latih 34
Tabel 4.7 Matriks kovarian 34
Tabel 4.8 *Eigenvalue* dan *eigenvector* sebelum diurutkan 37
Tabel 4.9 *Eigenvalue* dan *eigenvector* setelah diurutkan 37
Tabel 4.10 Hasil reduksi dimensi pada data latih 38
Tabel 4.11 Hasil perhitungan nilai data baru dari data uji 38
Tabel 4.12 Hasil reduksi dimensi pada data uji 39
Tabel 4.13 Data latih setelah reduksi dimensi 39
Tabel 4.14 Data uji setelah reduksi dimensi 40
Tabel 4.15 Variabel pada algoritme CIW-ELM 40
Tabel 4.16 Matriks acak biner untuk setiap blok 41
Tabel 4.17 Hasil perhitungan bobot setiap blok 41
Tabel 4.18 Magnitude vektor bobot pada setiap blok 41
Tabel 4.19 Hasil normalisasi bobot setiap blok 42
Tabel 4.20 Hasil inisialisasi bobot dan bias 42
Tabel 4.21 Hasil keluaran prediksi label data uji 45
Tabel 4.22 Hasil konversi nilai keluaran prediksi 45
Tabel 4.23 *Confusion matrix* prediksi data uji 45
Tabel 4.24 Persentase kontribusi atribut dan jumlah fitur yang diuji 46
Tabel 4.25 Nilai hyperparameter yang diuji 47
Tabel 4.26 Pasangan hyperparameter yang diuji 47
Tabel 6.1 Hasil pengujian *hyperparameter* terbaik menggunakan Grid Search Cross Validation 65
Tabel 6.2 Hasil pengujian *hyperparameter* K 66
Tabel 6.3 Hasil pengujian *hyperparameter* jumlah *hidden neuron* 67

Tabel 6.4 Hasil pengujian *hyperparameter* fungsi aktivasi 69
Tabel 6.5 Hasil *confusion matrix* pengujian akhir 70

DAFTAR GAMBAR

Gambar 2.1 Sumbu sensor *accelerometer* 10
Gambar 2.2 Sumbu sensor *gyroscope* 11
Gambar 2.3 Arsitektur ELM 13
Gambar 2.4 Ilustrasi 10-fold cross validation 17
Gambar 3.1 Diagram alir penelitian 19
Gambar 3.2 Kerangka kerja metode penelitian 22
Gambar 4.1 Diagram alir algoritme yang digunakan 24
Gambar 4.2 Diagram alir One-Hot Encoding 25
Gambar 4.3 Diagram alir algoritme PCA 26
Gambar 4.4 Diagram alir pelatihan model CIW-ELM 27
Gambar 4.5 Diagram alir inisialisasi bobot CIW-ELM 28
Gambar 4.6 Diagram alir klasifikasi menggunakan model CIW-ELM 29
Gambar 4.7 Diagram alir evaluasi klasifikasi 30
Gambar 4.8 Hasil perhitungan nilai keluaran *hidden neuron* 43
Gambar 4.9 Hasil aktivasi nilai keluaran *hidden neuron* 43
Gambar 4.10 Hasil matriks *Moore Penrose Pseudo-Inverse* 44
Gambar 4.11 Hasil perhitungan bobot keluaran 44
Gambar 4.12 Hasil perhitungan nilai keluaran *hidden neuron* pada data uji 44
Gambar 4.13 Hasil aktivasi nilai keluaran *hidden neuron* pada data uji 45
Gambar 6.1 Pengaruh pemilihan *hyperparameter k* terhadap skor skhir 66
Gambar 6.2 Pengaruh pemilihan *hyperparameter k* terhadap waktu pelatihan 67
Gambar 6.3 Pengaruh pemilihan *hyperparameter* jumlah *hidden neuron* terhadap skor skhir 68
Gambar 6.4 Pengaruh pemilihan *hyperparameter* jumlah *hidden neuron* terhadap waktu pelatihan 68
Gambar 6.5 Pengaruh pemilihan *hyperparameter* fungsi aktivasi terhadap skor skhir 69
Gambar 6.6 Pengaruh pemilihan *hyperparameter* fungsi aktivasi terhadap waktu pelatihan 69

DAFTAR KODE PROGRAM

Kode Program 5.1 Implementasi *label encoding* menggunakan One-Hot Encoding 49
Kode Program 5.2 Implementasi PCA 51
Kode Program 5.3 Implementasi CIW-ELM 52
Kode Program 5.4 Implementasi K-Fold Cross Validation 56
Kode Program 5.5 Implementasi metrik evaluasi 59
Kode Program 5.6 Implementasi *main method* 61

DAFTAR LAMPIRAN

LAMPIRAN A HASIL PENGUJIAN *HYPERPARAMETER* 79

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Pengenalan aktivitas manusia secara otomatis merupakan salah satu bidang penelitian yang sangat penting, karena potensi penerapannya di berbagai bidang lain seperti pengawasan, lingkungan cerdas, maupun kesehatan (Khan et al., 2010). Seiring dengan meningkatnya kebutuhan terhadap interpretasi perilaku manusia secara otomatis, *Human Activity Recognition* (HAR) telah menarik perhatian baik dari pihak akademis maupun industri (Jegham et al., 2020). Dari sekian banyaknya penelitian yang telah dilakukan terkait pengenalan aktivitas manusia, ada berbagai macam pendekatan yang dapat digunakan untuk mengenali aktivitas manusia. Salah satu pendekatan yang dapat dilakukan adalah dengan menggunakan teknik berbasis visi komputer, namun teknik tersebut membutuhkan dukungan infrastruktur berupa kamera pada area pemantauan yang diinginkan (Bayat et al., 2014). Alternatif lain yang dapat digunakan adalah dengan teknik berbasis sensor yang secara umum lebih unggul dibandingkan dengan teknik berbasis visi komputer karena tidak sensitif terhadap *noise*, murah, dan membutuhkan konsumsi daya yang lebih efisien (Bevilacqua et al., 2019). Penelitian terdahulu juga sudah mulai memanfaatkan ponsel cerdas untuk melakukan pengenalan aktivitas manusia, karena pada ponsel cerdas juga terdapat sensor seperti *gyroscope* dan *accelerometer*, sehingga pengguna tidak perlu memakai komponen sensor tambahan, serta tidak akan mengganggu dan lebih dapat diterima dalam kehidupan sehari-hari (Reiss et al., 2013).

Namun, penggunaan ponsel cerdas dalam melakukan pengenalan aktivitas manusia juga memiliki beberapa kekurangan. Ponsel cerdas memiliki kekurangan dalam melakukan komputasi algoritme yang kompleks, karena ponsel cerdas hanya memiliki kapasitas prosesor yang terbatas dibandingkan dengan prosesor yang terdapat pada komputer (Doewes et al., 2017). Selain itu, beberapa penelitian yang pernah dilakukan sebelumnya menunjukkan bahwa data hasil ekstraksi fitur dari sinyal yang ditangkap oleh sensor pada ponsel cerdas memiliki jumlah yang banyak serta dimensi yang tinggi (Anguita et al., 2013; Garcia-Gonzalez et al., 2020; Micucci et al., 2017). Hal tersebut tentu akan semakin membuat komputasi menjadi lebih berat. Kerugian lain dari fitur dengan dimensi yang tinggi adalah adanya kemungkinan bahwa beberapa fitur tersebut tidak relevan dan tidak memberikan informasi baru untuk meningkatkan akurasi klasifikasi, atau bahkan malah membingungkan dalam proses klasifikasi sehingga akurasi yang didapatkan akan menurun (Zhang & Sawchuk, 2012). Dengan keterbatasan prosesor, banyaknya jumlah data, serta tinggi nya dimensi fitur hasil ekstraksi, tentu akan sangat berdampak pada kecepatan komputasi serta sumber daya yang dibutuhkan dalam melakukan pengenalan aktivitas manusia.

Penelitian mengenai pengenalan aktivitas manusia menggunakan sensor yang terdapat pada ponsel cerdas sudah banyak sekali dilakukan. Penelitian pertama seperti yang dilakukan oleh Wang, *et al* (2016) yang bertujuan untuk melakukan

pengenalan aktivitas manusia. Metode yang digunakan pada penelitian ini meliputi metode seleksi fitur gabungan dari metode *filter* dan *wrapper* (FW), serta metode Naïve Bayes dan K-Nearest Neighbors (KNN) untuk melakukan klasifikasi aktivitas manusia. Naïve Bayes berhasil mendapatkan akurasi sebesar 90,1% menggunakan 74 fitur dari total 561 fitur, sedangkan KNN hanya memperoleh akurasi sebesar 87,8% dengan 66 fitur. Kemudian Doewes, *et al* (2017) melakukan penelitian pada masalah yang sama menggunakan metode minimum Redudancy Maximum Relevance (mRMR) sebagai seleksi fitur serta Support Vector Machine (SVM). Pada penelitian tersebut, akurasi yang didapatkan adalah sebesar 96,54% menggunakan 480 fitur yang dipilih dari keseluruhan fitur yang berjumlah 561. Penelitian selanjutnya dilakukan oleh Junita & Bachtiar (2019) yang menggunakan Information Gain untuk melakukan seleksi fitur dan algoritme Decision Tree C4.5 untuk melakukan klasifikasi pada *dataset* yang sama dengan penelitian sebelumnya. Akurasi yang diperoleh dari penelitian tersebut adalah sebesar 81% dengan menggunakan 90% dari jumlah fitur asli (561). Selain beberapa metode *machine learning* yang telah disebutkan, metode *deep learning* juga pernah digunakan untuk mengatasi permasalahan klasifikasi aktivitas manusia. Salah satu contoh nya adalah seperti yang dilakukan oleh Hernandez, *et al* (2019) yang menggunakan Bidirectional LSTM dan berhasil memperoleh akurasi sebesar 92,67% pada *dataset* yang sama seperti yang digunakan pada penelitian sebelumnya dengan 3 *layer* dan 175 *nodes*.

Beberapa metode yang digunakan pada penelitian yang telah disebutkan terbukti dapat digunakan untuk melakukan klasifikasi aktivitas manusia. Namun, metode-metode tersebut masih memiliki beberapa kekurangan. Penggunaan metode sederhana seperti Naïve Bayes, KNN, dan Decision Tree menyebabkan hasil akurasi yang didapatkan menjadi kurang maksimal. Selain itu, penggunaan algoritme K-Nearest Neighbors juga memiliki kekurangan lain, yaitu harus menyimpan semua data latih yang ada untuk melakukan klasifikasi pada data uji. Kemudian untuk algoritme SVM, implementasi dari metode tersebut lebih rumit daripada metode-metode sederhana yang telah dibahas sebelumnya. Algoritme SVM juga memerlukan penyesuaian beberapa *hyperparameter* yang digunakan pada algoritme tersebut untuk mendapatkan hasil yang maksimal. Beberapa metode tersebut juga kurang cocok untuk data dengan jumlah yang banyak, karena proses pelatihan model nya akan memakan waktu yang lama. Untuk Bidirectional LSTM, algoritme tersebut memiliki masalah umum yang ada pada metode *deep learning,* yaitu membutuhkan sumber daya yang besar serta waktu komputasi yang lama. Pernyataan tersebut dibuktikan dengan penelitian yang dilakukan oleh Hernandez, *et al* (2019) yang membutuhkan waktu komputasi selama 30 menit menggunakan perangkat keras yang memiliki spesifikasi yang tinggi, yaitu dengan prosesor *Intel Core i7-6700HQ* dan *GPU NVIDIA GeForce GTX 950M*.

Dengan mempertimbangkan kekurangan yang ada pada ponsel cerdas, banyaknya jumlah data, serta tingginya dimensi fitur yang ada pada data seperti yang ditunjukkan pada beberapa penelitian sebelumnya yaitu 561 fitur, maka diperlukan sebuah metode yang tidak terlalu kompleks yang dapat digunakan

untuk mengurangi dimensi fitur dari sebuah *dataset* dengan tetap mempertahankan informasi penting yang ada pada data tersebut, sehingga beban komputasi yang diperlukan menjadi berkurang. Selain itu, dibutukan juga metode klasifikasi yang dapat digunakan untuk melakukan klasifikasi aktivitas manusia dengan tepat. Metode tersebut juga harus dapat berjalan dengan cepat agar dapat melakukan klasifikasi aktivitas manusia secara langsung. Salah satu metode yang dapat digunakan untuk mengurangi dimensi fitur dari sebuah *dataset* adalah Principal Component Analysis (PCA) yang memiliki ide sederhana, yaitu mengurangi dimensi dari sekumpulan data dengan mempertahankan variabilitas sebanyak mungkin (Jollife & Cadima, 2016). Selain dapat menyederhanakan sekaligus mempercepat komputasi karena dapat mengurangi jumlah fitur yang ada, hasil reduksi dimensi dari PCA juga terbukti dapat meningkatkan akurasi dari sebuah algoritme klasifikasi (Wang et al., 2019). Kemudian untuk melakukan klasifikasi, salah satu algoritme yang dapat digunakan adalah Extreme Learning Machine (ELM) yang diketahui dapat menghasilkan generalisasi yang sangat baik dengan kecepatan pembelajaran yang sangat cepat (Huang et al., 2004). ELM juga cocok untuk digunakan pada *dataset* yang memiliki jumlah data yang besar dengan berhasil mengungguli algoritme Support Vector Machine (SVM) dan Random Forest (RF) (Ahmad et al., 2018). Pada penelitian lain, dikembangkan sebuah teknik inisialisasi bobot pada ELM yang diberi nama Computed Input Weight Extreme Learning Machine (CIW-ELM) yang membuat algoritme tersebut membutuhkan lebih sedikit hidden neuron daripada ELM konvensional, sehingga akan lebih menyederhanakan dan mempercepat komputasi ketika digunakan pada *dataset* yang besar (Tapson et al., 2015). Oleh karena itu, pada penelitian ini akan menggunakan algoritme Principal Component Analysis (PCA) dan Computed Input Weight Extreme Learning Machine (CIW-ELM) untuk melakukan klasifikasi aktivitas manusia. PCA akan digunakan untuk mengurangi dimensi fitur yang ada pada *dataset* dengan harapan dapat meningkatkan akurasi, mempercepat proses pelatihan, dan mengurangi jumlah parameter yang diperlukan. Kemudian, proses klasifikasi akan dilakukan menggunakan algoritme CIW-ELM karena algoritme tersebut lebih cocok untuk digunakan pada *dataset* yang besar daripada algoritme ELM konvensional.

Dengan demikian, klasifikasi aktivitas manusia terbukti dapat dilakukan menggunakan sensor yang ada pada ponsel cerdas. Namun, penggunaan ponsel cerdas dalam melakukan hal tersebut masih memiliki kekurangan, yaitu dalam melakukan komputasi algoritme yang kompleks. Selain itu, data hasil ekstraksi dari sensor yang ada pada ponsel cerdas diketahui memiliki jumlah yang banyak dan dimensi yang tinggi. Oleh karena itu, penelitian dengan judul "Klasifikasi Aktivitas Manusia Menggunakan Algoritme Computed Input Weight Extreme Learning Machine dengan Reduksi Dimensi Principal Component Analysis" ini akan berfokus pada penggunaan algoritme PCA dan CIW-ELM dalam mengatasi masalah klasifikasi aktvitas manusia, serta pemilihan *hyperparameter* terbaik dari kedua algoritme tersebut. Penggunaan kedua algoritme tersebut diharapkan dapat menghasilkan akurasi yang lebih unggul daripada metode yang telah

digunakan sebelumnya, serta dapat mengatasi kelemahan-kelemahan yang ada pada metode sebelumnya.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dijelaskan sebelumnya, maka dapat diambil beberapa rumusan masalah yang terdapat pada penelitian ini. Rumusan masalah pada penelitian ini adalah:

1. Bagaimanakah pengaruh pemilihan *hyperparameter* nilai *k* pada algoritme PCA untuk mengurangi dimensi data terhadap hasil pengujian klasifikasi aktivitas manusia?
2. Bagaimanakah pengaruh pemilihan *hyperparameter* jumlah *hidden neuron* dan fungsi aktivasi pada algoritme CIW-ELM terhadap hasil pengujian klasifikasi aktivitas manusia?
3. Bagaimanakah hasil akurasi dan *f-measure* pada klasifikasi aktivitas manusia menggunakan algoritme CIW-ELM dengan reduksi dimensi PCA?

1.3 Tujuan

Tujuan yang ingin dicapai dari penelitian yang dilakukan adalah:

1. Menganalisis pengaruh pemilihan *hyperparameter* nilai *k* pada algoritme PCA untuk mengurangi dimensi data terhadap hasil pengujian klasifikasi aktivitas manusia.
2. Menganalisis pengaruh pemilihan *hyperparameter* jumlah *hidden neuron* dan fungsi aktivasi pada algoritme CIW-ELM terhadap hasil pengujian klasifikasi aktivitas manusia.
3. Mengetahui hasil akurasi dan *f-measure* pada klasifikasi aktivitas manusia menggunakan algoritme CIW-ELM dengan reduksi dimensi PCA.

1.4 Manfaat

Manfaat yang dapat diambil dari penelitian ini adalah dapat mengetahui nilai *hyperparameter* terbaik dari algoritme PCA dan CIW-ELM dalam menyelesaikan masalah klasifikasi aktivitas manusia. Selain itu, penelitian ini juga memberikan informasi mengenai hasil penggunaan algoritme PCA dan CIW-ELM dalam menyelesaikan masalah klasifikasi aktivitas manusia, sehingga dapat menambah wawasan serta pengetahuan dari para pembaca. Hasil dari penelitian ini juga dapat dijadikan sebagai rujukan dan pembanding untuk penelitian-penelitian selanjutnya dengan topik yang sama.

1.5 Batasan Masalah

Batasan masalah yang ada pada penelitian ini adalah:

1. Data yang digunakan pada penelitian ini adalah data *Human Activity Recognition Using Smartphone Dataset* yang berasal dari UCI *Machine Learning Repository* (Anguita et al., 2013).
2. Data tersebut akan diolah menggunakan algoritme Principal Component Analysis (PCA) dengan tujuan untuk mengurangi jumlah dimensi fitur yang ada, untuk selanjutnya dilakukan klasifikasi menggunakan algoritme Computed Input Weight Extreme Learning Machine (CIW-ELM).
3. Untuk mendapatkan *hyperparameter* terbaik pada masing-masing algoritme, akan dilakukan validasi menggunakan Grid Search Cross Validation.
4. Hasil dari proses klasifikasi tersebut akan diukur menggunakan akurasi dan *f-measure* untuk mengetahui kinerja dari algoritme yang digunakan terhadap masalah klasifikasi aktivitas manusia.
5. Fokus pada penelitian ini adalah untuk mencari *hyperparameter* terbaik dari masing-masing algoritme yang digunakan, yaitu nilai *k* pada algoritme PCA, serta jumlah *hidden neuron* dan fungsi aktivasi pada algoritme CIW-ELM.

1.6 Sistematika Pembahasan

Sistematika atau struktur pembahasan pada penelitian ini terdiri atas bab-bab berikut:

BAB 1 PENDAHULUAN

Bab 1 Pendahuluan berisi latar belakang masalah untuk mengidentifikasi dan merumuskan permasalahan terkait klasifikasi aktivitas manusia. Pada bab ini juga terdapat tujuan penelitian, manfaat penelitian, dan batasan masalah yang mendasari dilakukannya penelitian ini. Terakhir, terdapat sistematika pembahasan yang menjelaskan struktur pembahasan yang ada pada penelitian ini.

BAB 2 LANDASAN KEPUSTAKAAN

Bab 2 Landasan Kepustakaan berisi kajian teori dari berbagai sumber pustaka sebelumnya yang terkait dengan permasalahan serta metode yang dibahas dalam penelitian ini. Pustaka yang digunakan bersumber dari buku maupun jurnal-jurnal yang dapat dipertanggung jawabkan kebenarannya.

BAB 3 METODOLOGI

Bab 3 Metodologi berisi metode-metode yang digunakan dalam menyelesaikan permasalahan pada penelitian ini. Pada bab ini juga dijelaskan mengenai informasi-informasi tambahan terkait dengan penelitian yang dilakukan.

BAB 4 PERANCANGAN

Bab 4 Perancangan berisi detail rancangan algoritme yang digunakan untuk menyelesaikan permasalahan yang ada pada penelitian ini. Perancangan yang dibuat akan disesuaikan dengan tujuan penelitian yang telah dibahas pada bab sebelumnya. Pada bab ini juga dijelaskan proses perhitungan manual dari algoritme yang digunakan.

BAB 5 IMPLEMENTASI

Bab 5 Implementasi berisi penerapan dari rancangan algoritme yang telah dibuat sebelumnya dalam bentuk kode program. Bahasa pemrograman yang digunakan pada penelitian ini adalah python. Setiap baris kode program yang dibuat akan dijelaskan secara terperinci.

BAB 6 PENGUJIAN DAN ANALISIS

Bab 6 Pengujian dan Analisis berisi hasil pengujian yang telah dilakukan, serta analisis dari hasil pengujian tersebut. Pengujian yang dilakukan bertujuan untuk menjawab pertanyaan-pertanyaan penelitian pada rumusan masalah. Pada bab ini juga dijelaskan mengenai informasi-informasi baru terkait hasil pengujian yang dilakukan.

BAB 7 PENUTUP

Bab 7 Penutup berisi kesimpulan dari hasil pengujian dan analisis yang telah dilakukan pada bab sebelumnya. Pada bab ini juga dijelaskan saran dari penulis untuk penelitian selanjutnya dengan topik yang sama.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 Tinjauan Pustaka

Pada penelitian ini, terdapat beberapa penelitian terdahulu yang digunakan sebagai acuan dalam menyelesaikan masalah penelitian. Penelitian pertama yang menjadi acuan adalah penelitian yang dilakukan oleh Wang, *et al* (2016) pada pengenalan aktivitas manusia menggunakan sensor yang ada pada ponsel cerdas. Penelitian tersebut bertujuan untuk menganalisis penggunaan sensor *accelerometer* dan *gyroscope* yang terdapat pada ponsel cerdas untuk mengenali aktivitas manusia ketika digunakan secara bersamaan maupun secara terpisah. Untuk melakukan pemilihan fitur agar menghasilkan model dengan generalisasi yang lebih baik serta mengurangi konsumsi daya yang digunakan, penelitian ini mengusulkan sebuah metode gabungan dari metode *filter* dan *wrapper* (FW). Metode tersebut akan digunakan untuk memilih fitur yang paling penting dari sebuah *dataset* sebelum dilakukan klasifikasi. Metode klasifikasi yang digunakan pada penelitian ini adalah Naïve Bayes dan K-Nearest Neighbors (KNN). Berdasarkan penelitian tersebut, didapatkan hasil akurasi tertinggi ketika menggunakan kedua sensor secara bersamaan, dengan akurasi sebesar 90,1% pada algoritme Naïve Bayes menggunakan 74 fitur, dan 87,8% pada algoritme KNN menggunakan 66 fitur.

Penelitian lain dilakukan oleh Doewes, et al (2017) pada masalah yang sama seperti pada penelitian sebelumnya. Tujuan dari penelitian tersebut adalah untuk melakukan optimasi terhadap proses komputasi dengan cara mengurangi dimensi pada fitur yang ada pada *dataset,* sehingga sistem dapat berjalan dengan baik pada ponsel cerdas yang hanya memiliki daya pemrosesan yang terbatas. Metode yang digunakan untuk mengurangi dimensi fitur adalah minimum Redundancy Maximum Relevance (mRMR), sedangkan metode klasifikasi utama yang digunakan adalah Support Vector Machine. Akurasi tertinggi yang dihasilkan adalah sebesar 96,54% dengan menggunakan fitur sebanyak 480, sedangkan akurasi terendah adalah sebesar 41,91% dengan menggunakan 1 fitur.

Penelitian selanjutnya dilakukan oleh Kishore, *et al* (2017) yang bertujuan untuk melakukan klasifikasi aktivitas manusia. Pada penelitian tersebut, algoritme yang digunakan adalah Principal Component Analysis (PCA) dan Backpropagation. PCA digunakan untuk mencari fitur-fitur penting yang ada pada *dataset* yang digunakan. Data dengam fitur yang dipilih kemudian dimasukkan ke dalam algoritme Backpropagation untuk dilakukan pelatihan model dan pengujian. Dari beberapa pengujian yang dilakukan, dipilih 381 fitur yang paling optimal dari keseluruhan fitur. Pengujian lain juga dilakukan untuk memilih *hyperparameter* terbaik yang akan digunakan pada algoritme Backpropagation. Berdasarkan pengujian yang dilakukan, dipilih *mean square error* dengan nilai 0,000001, jumlah neuron adalah 20-20-6, jumlah *hidden layer* adalah 2, dan *transfer function* secara berturut-turut untuk setiap layer adalah *tan-sig*, *tan-sig*,

dan *purelin*. Akurasi tertinggi yang didapatkan dari penelitian tersebut adalah sebesar 96,8%.

Kemudian pada penelitian yang dilakukan oleh Wang, et al (2019) untuk menyelesaikan masalah identifikasi kanker payudara, mereka mencoba menggabungkan algoritme PCA dan LVQ Neural Network. Penelitian tersebut bertujuan untuk meningkatkan performa dari algoritme LVQ dalam melakukan klasifikasi menggunakan fitur hasil reduksi dimensi dari PCA. Data yang digunakan pada penelitian tersebut adalah sebanyak 500 data untuk data latih, serta 69 data sebagai data uji yang dipilih secara acak. Hasil akurasi yang didapatkan menggunakan algoritme LVQ dari yang semula sebesar 89,9% naik menjadi 97,1% setelah dilakukan reduksi dimensi menggunakan algoritme PCA. Berdasarkan penelitian tersebut, PCA terbukti sangat efektif dalam mengurangi dimensi fitur dari *dataset* yang digunakan, sekaligus meningkatkan hasil akurasi secara signifikan.

Penelitian selanjutnya adalah penelitian yang dilakukan oleh Ahmad, *et al* (2018) mengenai deteksi gangguan pada jaringan. Tujuan dari penelitian tersebut adalah untuk membandingkan metode mana yang paling cocok digunakan untuk mengatasi masalah tersebut yang memiliki jumlah data yang besar. Algoritme yang dibandingkan pada penelitian tersebut adalah Support Vector Machine (SVM), Random Forest (RF), dan Extreme Learning Machine (ELM). Dari penelitian tersebut, diketahui banwa algoritme ELM mampu mengungguli algoritme lain dengan akurasi sebesar 99,5% menggunakan semua sampel data dengan jumlah 65535 data.

Terakhir adalah penelitian yang dilakukan oleh Tapson, et al (2015) yang bertujuan untuk mengusulkan sekaligus menguji kinerja dari algoritme Computed Input Weight Extreme Learning Machine (CIW-ELM) yang merupakan pengembangan dari algoritme ELM konvensional. Salah satu pengujian yang dilakukan adalah dengan menerapkannya pada permasalahan MNIST *handwritten recognition problem* yang merupakan masalah klasifikasi dengan 784 dimensi fitur (nilai *pixel* dari data masukan), 10 kelas keluaran, dan 60000 data latih. Pada pengujian tersebut, CIW-ELM berhasil mendapatkan akurasi sebesar 96% dengan jumlah *hidden neuron* sebanyak 700, sedangkan ELM konvensional membutuhkan 3000 *hidden neuron* untuk mendapatkan akurasi yang sama. Hal ini menunjukkan bahwa CIW-ELM lebih unggul dalam masalah kecepatan komputasi dibandingkan dengan ELM, terutama saat digunakan pada *dataset* yang besar.

Pada penelitian ini, akan dilakukan klasifikasi aktivitas manusia menggunakan data hasil ekstraksi fitur dari sensor *accelerometer* dan *gyroscope*. Algoritme yang digunakan pada penelitian ini adalah gabungan dari algoritme yang telah disebutkan pada pustaka sebelumnya, yaitu PCA untuk melakukan reduksi dimensi fitur, serta CIW-ELM yang merupakan pengembangan dari algoritme ELM konvensional untuk melakukan klasifikasi. PCA dipilih karena berdasarkan beberapa pustaka, PCA terbukti efektif dalam mengurangi jumlah fitur yang ada pada *dataset*, sekaligus meningkatkan akurasi dari klasifikasi. CIW-ELM dipilih

karena berdasarkan pustaka, CIW-ELM membutuhkan lebih sedikit *hidden neuron* untuk mendapatkan akurasi yang sama dengan ELM konvensional, sehingga waktu komputasi yang dibutuhkan akan lebih cepat. Pada penelitian ini juga akan dilakukan pengujian *hyperparameter* terbaik dari masing-masing algoritme yang digunakan. Pengujian tersebut akan dilakukan menggunakan Grid Search Cross Validation, dengan nilai rata-rata dari akurasi dan *f-measure* sebagai nilai skor dari masing-masing pasangan *hyperparameter* yang diuji. Pasangan *hyperparameter* dengan skor terbaik akan digunakan untuk membuat model akhir menggunakan keseluruhan data latih untuk selanjutnya di evaluasi menggunakan data uji.

2.2 Aktivitas Manusia

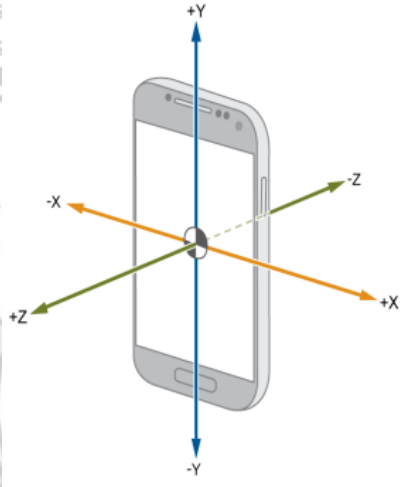
Menurut Kamus Besar Bahasa Indonesia (KBBI), kata aktivitas berarti keaktifan atau kegiatan. Aktivitas manusia bisa juga diartikan sebagai suatu kegiatan yang dilakukan oleh manusia. Berdasarkan kompleksitas nya, aktivitas manusia dibagi menjadi empat tingkatan, yaitu gerak tubuh, tindakan, interaksi, dan kegiatan kelompok (Aggarwal & Ryoo, 2011). Gerak tubuh adalah gerakan dasar bagian tubuh seseorang, seperti meregangkan lengan dan mengangkat kaki. Tindakan adalah aktivitas seseorang yang terdiri dari beberapa gerakan yang terorganisir, seperti berjalan, melambai, dan memukul. Interaksi adalah aktivitas manusia yang melibatkan dua atau lebih orang dan/atau objek. Kegiatan kelompok adalah kegiatan yang dilakukan oleh sekelompok orang dan/atau objek.

2.3 Sensor

Sensor merupakan sumber pengumpulan data mentah yang dapat digunakan pada pengenalan aktivitas manusia. Sensor diklasifikasikan menjadi tiga kategori, yaitu sensor video, sensor berbasis lingkungan, dan sensor yang dapat dikenakan (Su et al., 2014). Sensor yang dapat dikenakan (*wearable sensor*) merupakan sensor yang banyak digunakan dalam mengenali aktivitas manusia karena ukurannya yang kecil dan dapat dikenakan oleh manusia dalam melakukan aktivitas sehari-hari. Nilai yang didapatkan dari sensor tersebut kemudian di ekstrak menggunakan beberapa metode pengukuran untuk mendapatkan vektor fitur yang dapat digunakan untuk membedakan setiap aktivitas manusia. Salah satu contoh ekstraksi fitur pada sensor yang terdapat pada ponsel cerdas adalah seperti penelitian yang dilakukan oleh Anguita, et al (2013). Pada penelitian tersebut, nilai sinyal yang didapatkan dari sensor *accelerometer* dan *gyroscope* berupa nilai numerik dan diekstrak menggunakan beberapa metode pengukuran seperti rata-rata, standar deviasi, nilai maksimal, nilai minimal, dan beberapa metode lain, sehingga menghasilkan vektor fitur yang berjumlah 561 yang merepresentasikan beberapa aktivitas yang dilakukan pada tahap pengumpulan data, yaitu berjalan, berdiri, duduk, berbaring, menaiki tangga, dan menuruni tangga (Anguita et al., 2013).

2.3.1 *Accelerometer*

Accelerometer merupakan sebuah sensor yang dapat mengukur gaya percepatan arah dalam menentukan orientasi perangkat, dengan menyediakan vektor tiga dimensi untuk mengukur percepatan dalam m/s^2 di sepanjang sumbu perangkat (Vaughn et al., 2018). Sensor ini berpotensi untuk digunakan dalam menunjukkan pergerakan yang dilakukan oleh pengguna. Apalagi, data percepatannya bisa menunjukkan pola pergerakan dalam jangka waktu tertentu, sehingga dapat membantu dalam mengenali aktivitas yang kompleks (Su et al., 2014). Arah sumbu yang terdapat pada sensor *accelerometer* ditunjukkan pada Gambar 2.1.

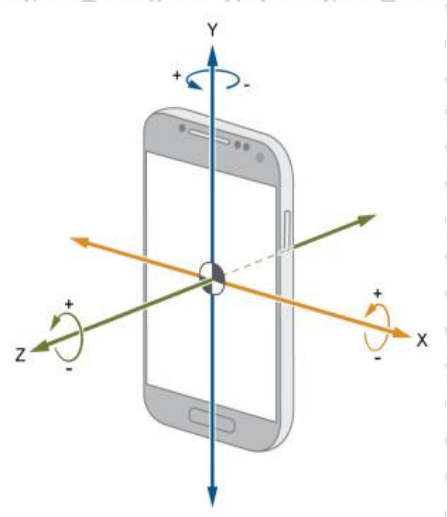


Gambar 2.1 Sumbu sensor *accelerometer*

Sumber: Vaughn, *et al* (2018)

2.3.2 *Gyroscope*

Gyroscope merupakan sebuah sensor yang digunakan untuk menghitung rotasi dalam rad/s yang akan memberikan nilai positif saat perangkat tersebut memutar searah jarum jam, serta nilai negatif saat diputar berlawanan dengan arah jarum jam pada sumbu manapun (Vaughn et al., 2018). Sama seperti *accelerometer*, *gyroscope* juga menyediakan perspektif tiga dimensi pada perangkat serta menggunakan tiga vektor yang sama. Pada pengenalan aktivitas manusia, *gyroscope* digunakan untuk membantu penggunaan sensor *accelerometer* dalam mendeteksi orientasi perangkat (Su et al., 2014). Arah sumbu yang terdapat pada sensor *gyroscope* ditunjukkan pada Gambar 2.2.



Gambar 2.2 Sumbu sensor *gyroscope*

Sumber: Vaughn, *et al* (2018)

2.4 *Dimensionality Reduction*

Reduksi dimensi adalah transformasi data yang memiliki dimensi tinggi kedalam dimensi yang lebih rendah, namun tetap merepresentasikan makna dari data tersebut (Maaten et al., 2009). Dengan kata lain, reduksi dimensi merupakan sebuah teknik untuk mengurangi dimensi dari sebuah data, dengan tetap mempertahankan informasi dari data tersebut. Reduksi dimensi sangat penting dalam menangani data dengan dimensi yang tinggi, karena dimensi data yang tinggi dapat memberatkan komputasi serta menyebabkan berbagai masalah lainnya atau yang lebih dikenal sebagai *curse of dimensionality* (Xie et al., 2017). Saat ini sudah terdapat banyak sekali metode yang dapat digunakan untuk melakukan reduksi dimensi, mulai dari metode yang menggunakan teknik *linear* seperti Principal Component Analysis (PCA), hingga yang menggunakan teknik *nonlinear* seperti Isomap dan Kernel PCA.

2.4.1 Principal Component Analysis

Principal Component Analysis (PCA) merupakan salah satu teknik untuk mengurangi dimensi fitur dari sebuah *dataset*, meningkatkan makna fitur, sekaligus meminimalkan kehilangan informasi yang ada pada *dataset* tersebut (Jollife & Cadima, 2016). PCA biasanya digunakan untuk mengurangi dimensi fitur dari sebuah *dataset* yang memiliki dimensi fitur yang tinggi dengan tujuan untuk meningkatkan hasil klasifikasi, serta mengurangi waktu komputasi. Langkah-langkah algoritme PCA adalah (Wang et al., 2019):

1. Masukkan *dataset* $(X_{m\times n})$
2. Menghitung nilai rata-rata *dataset* pada setiap kolom (X_{mean})
3. Menghitung nilai X baru dengan Persamaan 2.1.

$$X_{baru} = X_{m \times n} - X_{mean} \quad (2.1)$$

4. Menghitung matriks kovarians dengan Persamaan 2.2.

$$Cov = \frac{1}{m-1}\left(X_{baru}{}^{T} . X_{baru}\right) \quad (2.2)$$

Keterangan:

Cov = Matriks kovarian

m = Jumlah data

X_{baru} = Nilai X baru

$X_{baru}{}^{T}$ = Nilai X baru hasil transpose

5. Menghitung *eigenvalue* dan *eigenvector* dari matriks kovarian
6. Mengurutkan *eigenvalue* dari yang terbesar
7. Memilih *k eigenvalue* terbesar beserta *k eigenvector* yang berhubungan dengan *eigenvalue* tersebut, sehingga menghasilkan matriks *eigenvector* $W_{n \times k}$
8. Reduksi dimensi dilakukan dengan cara mengalikan nilai $X_{m \times n}$ baru yang didapatkan dengan Persamaan 2.1 yang dimensi fiturnya ingin di reduksi dengan matriks *eigenvector* $W_{n \times k}$, sehingga akan menghasilkan proyeksi data baru $X_{m \times k}$

2.5 One-Hot Encoding

One-Hot Encoding merupakan sebuah teknik dalam merubah bentuk data kategorik menjadi bentuk biner, dengan panjang sesuai dengan banyaknya kategori data yang berbeda (Potdar et al., 2017). Teknik ini biasa digunakan untuk melakukan perubahan pada variabel atau fitur kategorik yang terdapat pada sebuah *dataset*, namun teknik ini juga dapat digunakan untuk merubah nilai dari sebuah label atau kelas. Perubahan label atau kelas menggunakan One-Hot Encoding umumnya digunakan pada saat bekerja menggunakan algoritme jaringan syaraf tiruan, karena hasil keluaran setiap label dihitung secara terpisah. One-Hot Encoding dilakukan dengan cara merubah nilai setiap kelas menjadi nilai biner, dengan indeks kelas yang bersangkutan diberi nilai 1 dan yang lain diberi nilai 0. Contoh penggunaan One-Hot Encoding pada label atau kelas adalah:

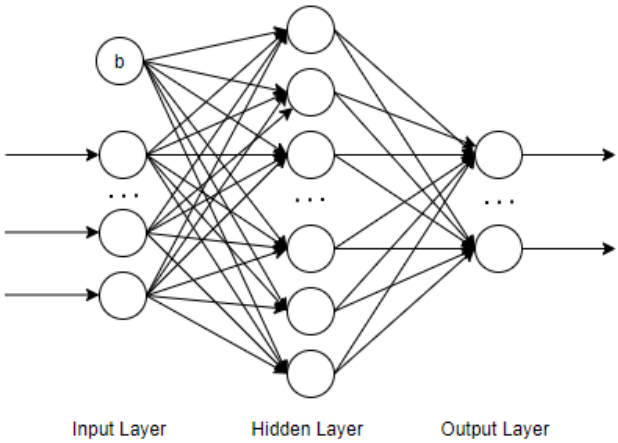
- Misal sebuah *dataset* memiliki 3 label atau kelas yaitu: berdiri, duduk, dan berbaring
- Label tersebut dapat dituliskan menjadi [berdiri, duduk, berbaring]
- Nilai pada setiap label dapat diubah menjadi seperti berikut:
  - Berdiri → [1,0,0]
  - Duduk → [0,1,0]
  - Berbaring → [0,0,1]

2.6 Klasifikasi

Klasifikasi merupakan salah satu bentuk pembelajaran mesin yang terawasi (*supervised machine learning*) yang bertujuan untuk memberikan pembelajaran terhadap mesin agar dapat membuat sebuah hipotesis atau prediksi dari suatu data (Osisanwo et al., 2017). Klasifikasi sendiri sering digunakan untuk memprediksi keanggotaan, atau mengelompokkan data baru ke dalam suatu kelas atau label berdasarkan informasi dari data yang sudah ada (Soofi & Awan, 2017). Umumnya, klasifikasi terdiri dari dua tahap, yaitu tahap pelatihan dan tahap prediksi. Tahap pelatihan dilakukan untuk belajar dari data latih sehingga menghasilkan sebuah model, sedangkan tahap prediksi digunakan untuk melakukan prediksi label atau kategori dari data uji (Kesavaraj & Sukumaran, 2013). Beberapa contoh metode klasifikasi yang dapat digunakan adalah *Naïve Bayes, K-Nearest Neighbors, Decision Tree, Support Vector Machine*, Extreme Learning Machine, dan *Backpropagation* seperti yang ada pada tinjauan pustaka sebelumnya.

2.6.1 Extreme Learning Machine

Pada tahun 2004, Huang, *et al* (2004) mengusulkan sebuah algoritme baru yang bernama Extreme Learning Machine (ELM). ELM merupakan algoritme yang dibuat untuk mengatasi masalah pelatihan *feedforward neural network* yang dinilai lebih lambat daripada yang seharusnya dibutuhkan, sehingga menghambat penerapan algoritme tersebut secara umum. Secara teori, ELM cenderung memberikan kinerja generalisasi yang lebih baik dengan kecepatan pelatihan yang sangat cepat. Algoritme ini termasuk ke dalam kelompok *single hidden layer feedforward neural networks* (SLFNs) yang memilih nilai bobot secara acak dan menentukan bobot keluaran SLFNs secara analitis. Arsitektur ELM ditunjukkan pada Gambar 2.3.



Gambar 2.3 Arsitektur ELM

Langkah-langkah pelatihan algoritme ELM adalah:

1. Masukkan pasangan data latih dan target (X, T), fungsi aktivasi *g(x)*, dan jumlah *hidden neuron N*.

2. Inisialisasi bobot w dan bias b secara acak
3. Menghitung nilai keluaran dari *hidden layer* dengan Persamaan 2.3.

$$H = X.w^T + b \quad (2.3)$$

Keterangan:

H = Matriks keluaran *hidden layer*

X = Data latih

w^T = Matriks bobot hasil transpose

b = nilai bias

4. Lakukan aktivasi pada nilai H sesuai dengan fungsi aktivasi yang digunakan
5. Menghitung matriks *Moore Penrose Pseudo-Inverse* menggunakan Persamaan 2.4.

$$H^\dagger = (H^T.H)^{-1}.H^T \quad (2.4)$$

Keterangan:

$H^\dagger$ = Matriks *Moore Penrose Pseudo-Inverse*

H = Matriks keluaran *hidden layer*

H^T = Matriks keluaran *hidden layer* hasil transpose

6. Menghitung matriks bobot keluaran menggunakan Persamaan 2.5.

$$\beta = H^\dagger.T \quad (2.5)$$

Keterangan:

β = Matriks bobot keluaran

$H^\dagger$ = Matriks *Moore Penrose Pseudo-Inverse*

T = Target

Setelah proses pelatihan selesai, tahap prediksi atau klasifikasi pada data uji dilakukan dengan langkah-langkah sebagai berikut:

1. Masukkan data uji.
2. Mengambil nilai bobot *w*, bias *b*, dan bobot keluaran β yang didapatkan pada proses pelatihan.
3. Menghitung Menghitung nilai keluaran dari *hidden layer* (*H*) menggunakan Persamaan 2.3.
4. Melakukan aktivasi pada nilai kueluaran *hidden layer* (*H*) sesuai dengan fungsi aktivasi yang digunakan.
5. Menghitung hasil prediksi target menggunakan Persamaan 2.6.

$$\check{T} = H.\beta \quad (2.6)$$

Keterangan:

$\check{T}$ = Prediksi target

β = Matriks bobot keluaran

H = Matriks keluaran hidden layer

2.6.2 Computed Input Weight Extreme Learning Machine

Computed Input Weight Extreme Learning Machine (CIW-ELM) merupakan pengembangan dari metode ELM konvensional. Pada sub bab sebelumnya, ELM diketahui dapat melakukan klasifikasi dengan hasil yang sangat baik dengan komputasi yang sangat cepat. Meski begitu, ELM masih memiliki kekurangan yaitu sangat bergantung pada bobot awal yang dipilih secara acak, sehingga hasil kinerja yang didapatkan terkadang tidak stabil atau cenderung berubah-ubah. Oleh karena itu, Tapson, *et al* (2015) mengembangkan sebuah teknik inisialisasi bobot yang dapat menghasilkan bobot awal yang dipilih secara acak menggunakan kombinasi linier dari data masukan atau data latih, sehingga bobot yang dihasilkan dapat tersusun secara sistematis. Pengembangan dari algoritme ELM berdasarkan teknik inisialisasi bobot tersebut diberi nama *Computed Input Weight* ELM (CIW-ELM). Jika pada ELM konvensional inisialisasi bobot nya dilakukan dengan memilih bilangan acak dalam rentang nilai (-1, 1), maka pada CIW-ELM inisialisasi bobot nya dihitung berdasarkan langkah-langkah berikut:

1. Normalisasi semua data latih untuk menghindari faktor skala
2. Membagi *d hidden neuron* menjadi *C* blok, satu blok untuk setiap kelas keluaran *C*. Untuk kumpulan data latih yang memiliki jumlah anggota kelas yang sama, ukuran untuk setiap blok *B = d/C*. Jika jumlah anggota kelas pada data latih tidak sama, maka ukuran blok dapat disesuaikan agar sebanding dengan jumlah data. Jumlah data latih untuk setiap kelas keluaran dinotasikan sebagai *K*
3. Untuk setiap blok, bangkitkan matriks $R_{B\times K}$ secara acak dengan nilai 1 atau -1
4. Mengalikan matriks $R_{B\times K}$ dengan data latih pada setiap kelas $X_{K\times k}$ sehingga menghasilkan matriks baru $w_{B\times k}$ yang merupakan bobot untuk setiap blok *neuron*
5. Normalisasi bobot yang didapatkan dengan Persamaan 2.7.

$$w_\alpha = \frac{R_\alpha}{|R_\alpha|} \quad (2.7)$$

Keterangan:

α = Indeks kolom

6. Menggabungkan *C* blok bobot dari setiap kelas menjadi sebuah matriks bobot $w_{d\times k}$

Langkah selanjutnya sama dengan algoritme ELM konvensional yang sudah dijelaskan.

2.6.3 Fungsi Aktivasi

Fungsi aktivasi atau juga dikenal sebagai fungsi peralihan merupakan sebuah fungsi yang digunakan untuk menentukan apakah sebuah neuron harus di aktivasi atau tidak dengan menghitung *weighted sum*, yang akan menghasilkan *non-linearity* ke dalam keluaran *neuron* sehingga hasil keluarannya berada dalam kisaran [0, 1] atau [-1, 1] (Feng & Lu, 2019). Fungsi aktivasi pada dasarnya terbagi menjadi 2 jenis, yaitu fungsi aktivasi linier dan fungsi aktivasi non-linier. Fungsi aktivasi non-linier dapat menghasilkan lebih banyak variasi nilai yang berguna dalam proses pelatihan, sedangkan fungsi aktivasi linier hanya menghasilkan nilai yang konstan, sehingga keluaran yang dihasilkan hanya akan ada satu nilai atau bisa jadi tidak terbatas (Feng & Lu, 2019). Berikut adalah beberapa fungsi aktivasi yang diketahui pernah digunakan pada algoritme ELM (Sevin, 2018):

a. *Sigmoid*

Sigmoid merupakan sebuah fungsi aktivasi non-linier yang menghasilkan keluaran dalam kisaran [0, 1]. Fungsi *sigmoid* dapat dihitung menggunakan Persamaan 2.8.

$$f(x) = \frac{1}{1+e^{-x}} \quad (2.8)$$

Keterangan:
e = nilai *exponential* (2,718281)

b. *Tanh*

Tanh merupakan sebuah fungsi aktivasi yang memiliki karakteristik yang sama seperti *sigmoid*, namun keluaran yang dihasilkan berada pada kisaran [-1, 1]. Fungsi *tanh* dapat dihitung menggunakan Persamaan 2.9.

$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad (2.9)$$

Keterangan:
e = nilai *exponential* (2,718281)

c. *Sine*

Sine adalah fungsi periodik yang naik turun, dan nilai keluarannya konvergen ke nol untuk masukan yang memiliki nilai positif dan negatif yang besar. Fungsi *sine* dapat dihitung menggunakan Persamaan 2.10.

$$f(x) = sin(x) \quad (2.10)$$

d. *Hardlim*

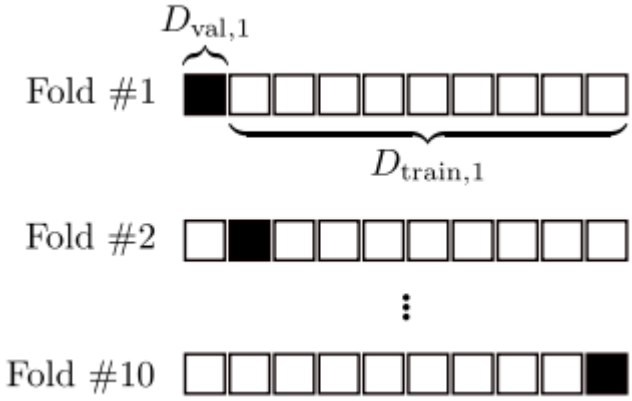
Hardlim adalah fungsi aktivasi yang memaksa neuron untuk mengeluarkan nilai "1" jika nilai masukannya mencapai batas tertentu,

jika tidak maka nilai keluarannya adalah "0". Fungsi *Hardlim* dapat dihitung menggunakan Persamaan 2.11.

$$f(x) = \begin{cases} 1, x \geq 0 \\ 0, otherwise \end{cases} \quad (2.11)$$

2.7 K-Fold Cross Validation

Cross Validation merupakan metode pengambilan sampel ulang dari sebuah data, yang paling banyak digunakan untuk memperkirakan kesalahan prediksi yang sebenarnya dari sebuah model, serta untuk menyesuaikan *hyperparameter* dari sebuah model (Berrar, 2018). Pada K-Fold Cross Validation, sebuah *dataset* dibagi menjadi beberapa *subsets (fold)* sejumlah *k* dengan ukuran yang sama (Kohavi, 1993). Ilustrasi dari K-Fold Cross ditunjukkan pada Gambar 2.4.



Gambar 2.4 Ilustrasi 10-fold cross validation

Sumber: (Berrar, 2018)

Gambar 2.4 merupakan contoh dari penggunaan 10-Fold Cross Validation. Untuk setiap *fold*, model akan dibuat menggunakan *k*-1 *subset* data latih, dan diuji menggunakan 1 *subset* validasi. Hasil evaluasi akhir dari sebuah model dihitung berdasarkan rata-rata hasil akurasi yang didapatkan pada seluruh *fold*.

2.8 Grid Search Cross Validation

Grid Search Cross Validation merupakan sebuah metode dalam memilih pasangan parameter awal yang mungkin dengan memperhatikan *grid* dan kriteria pencarian yang diberikan, serta menggunakan Cross Validation untuk memilih parameter yang paling optimal dari *grid* satu dimensi atau multi dimensi (Krstajic et al., 2014). Pencarian *grid* dirancang menggunakan sekumpulan parameter dengan nilai tetap dan memiliki peran penting dalam memberikan akurasi yang paling optimal berdasarkan K-Fold Cross Validation (Shekar & Dagnew, 2019).

2.9 Metode Evaluasi

Metode evaluasi yang digunakan pada penelitian ini adalah akurasi dan *f-measure*. Akurasi digunakan untuk mengetahui tingkat kebenaran dari prediksi yang dihasilkan oleh *classifier* terhadap label atau kelas asli dari data dari seluruh jumlah data yang di evaluasi (Hossin & Sulaiman, 2015). *F-measure* digunakan untuk mengukur keseimbangan (*harmonic mean*) dari *precision* dan *recall* (Hossin & Sulaiman, 2015). Kedua metode evaluasi tersebut bisa dihitung berdasarkan tabel *confusion matrix* yang ditunjukkan pada Tabel 2.1.

Tabel 2.1 *Confusion matrix*

	Positif (Prediksi)	**Negatif (Prediksi)**
Positif (Asli)	True Positive (TP)	False Positive (FP)
Negatif (Asli)	False Negative (FN)	True Negative (TN)

Akurasi dan *f-measure* dapat dihitung menggunakan Persamaan 2.12 dan 2.13.

$$Acc = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.12}$$

$$f-measure = 2 \times \frac{\frac{TP}{(TP+FP)} \times \frac{TP}{(TP+FN)}}{\frac{TP}{(TP+FP)} + \frac{TP}{(TP+FN)}} \tag{2.13}$$

Keterangan:

TP = True Positive (asli positif, prediksi positif)

TN = True Negative (asli negatif, prediksi negatif)

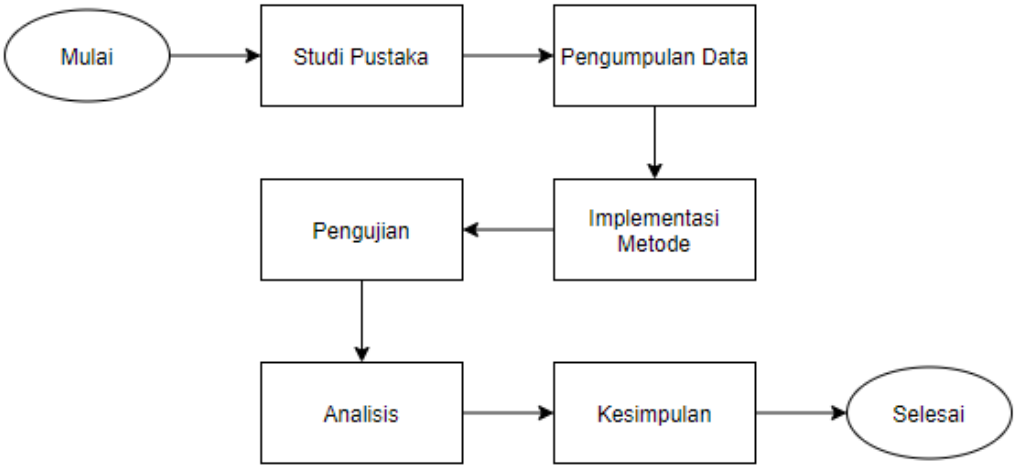
FP = False Positive (asli negatif, prediksi positif)

FN = False Negative (asli positif, prediksi negatif)

BAB 3 METODOLOGI

3.1 Strategi Penelitian

Penelitian ini dilakukan dengan beberapa tahapan. Tahap pertama yaitu mempelajari pustaka-pustaka sebelumnya yang terkait dengan topik penelitian. Tahap kedua yaitu melakukan pengumpulan data yang akan digunakan dalam penelitian. Tahap ketiga yaitu mengimplementasikan metode pada data penelitian berdasarkan teori-teori yang sudah dipelajari pada tahap sebelumnya. Tahap keempat yaitu melakukan pengujian pada data penelitian berdasarkan rumusan masalah yang telah ditentukan. Tahap kelima yaitu melakukan analisis terhadap pengujian yang dilakukan. Tahap keenam yaitu mengambil kesimpulan dari hasil penelitian yang dilakukan. Diagram alir penelitian ditunjukkan pada Gambar 3.1.



Gambar 3.1 Diagram alir penelitian

3.2 Tipe Penelitian

Tipe penelitian yang dilakukan pada penelitian ini adalah Non-Implementif – Analitik, yang bertujuan untuk menjelaskan hubungan antar elemen pada objek penelitian sesuai dengan rumusan masalah. Penelitian ini bertujuan untuk menganalisis pengaruh pemilihan nilai *hyperparameter* pada algoritme yang digunakan, serta menguji hasil klasifikasi aktivitas manusia yang dapat diperoleh menggunakan algoritme tersebut.

3.3 Lokasi Penelitian

Oleh karena kondisi pada saat penelitian ini dilakukan sedang terjadi pandemi COVID-19, maka penelitian ini dilakukan di rumah penulis.

3.4 Peralatan Pendukung

3.4.1 Perangkat Keras

Perangkat keras yang digunakan pada penelitian ini adalah *Acer Aspire E5-475G-58WK* dengan spesifikasi sebagai berikut:

a. *Processor : Intel Core i5 – 7200U*
b. *RAM : 4GB DDR4 Memory*
c. *VGA 1 : Intel® HD Graphics 620*
d. *VGA 2 : NVIDIA GeForce 940MX*

3.4.2 Perangkat Lunak

Perangkat lunak yang digunakan pada penelitian ini adalah:

a. *Windows 10 Enterprise 1511 64-bit*
b. *Python 3.8.6 64-bit*
c. *Google Chrome 88.0.4324.104 64-bit*
d. *Jupyter Notebook 6.0.3*
e. *Google Colaboratory*

3.5 Studi Literatur

Studi literatur dilakukan dengan mempelajari teori-teori yang akan digunakan dalam menyelesaikan masalah penelitian dari berbagai sumber. Beberapa teori yang dipelajari adalah mengenai aktivitas manusia, sensor, *accelerometer*, *gyroscope*, *Dimensionality Reduction*, Principal Component Analysis, klasifikasi, Extreme Learning Machine, Computed Input Weight Extreme Learning Machine, fungsi aktivasi, One-Hot Encoding, K-Fold Cross Validation, dan metode evaluasi yang digunakan, yaitu akurasi dan *f-measure*.

3.6 Pengumpulan Data

Data yang digunakan pada penelitian ini adalah data sekunder, yaitu data *Human Activity Recognition Using Smartphone Dataset* yang di ambil dari situs UCI *Machine Learning Repository*. Data tersebut dikumpulkan dari 30 sukarelawan yang berusia 19 hingga 30 tahun menggunakan sensor *accelerometer* dan *gyroscope* yang terdapat pada ponsel cerdas Samsung Galaxy S II yang diletakkan pada pinggang. Setiap orang diminta untuk melakukan enam aktivitas berupa berjalan, naik tangga, turun tangga, duduk, berdiri, dan berbaring. Selanjutnya dilakukan ekstraksi fitur pada data yang didapatkan serta diberi label secara manual, kemudian data tersebut dibagi secara acak dengan 70% dipilih sebagai data latih dan 30% dipilih secara data uji. Jumlah data yang terkumpul adalah sebanyak 10299 data dengan 561 fitur dan enam kelas, dengan 7352 data digunakan sebagai data latih dan 2947 data digunakan sebagai data uji.

3.7 Implementasi Metode

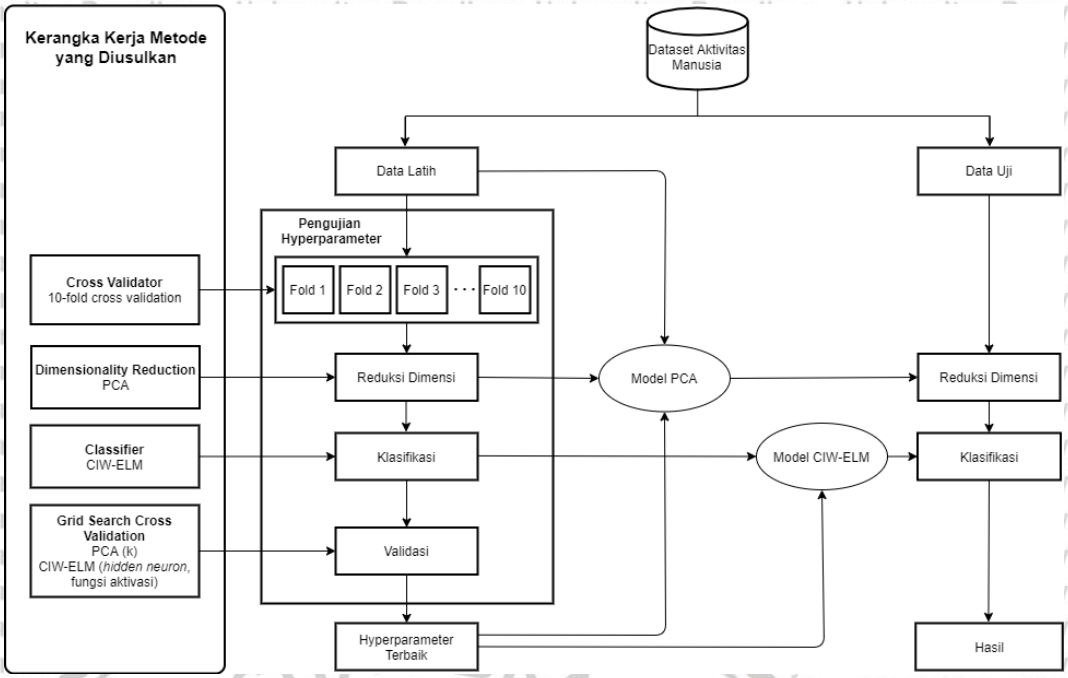
3.7.1 Reduksi Dimensi

Reduksi dimensi berguna untuk mengurangi dimensi fitur dari data yang ada, dengan tujuan untuk mempersingkat waktu komputasi pada klasifikasi. Metode yang digunakan untuk melakukan reduksi data pada penelitian ini adalah Principal Component Analysis (PCA). Ide dasar dari PCA adalah mengurangi dimensi dari sekumpulan data dengan mempertahankan informasi sebanyak mungkin (Jollife & Cadima, 2016). PCA dipilih karena sederhana dan dapat diimplementasikan dengan mudah, serta hanya membutuhkan waktu komputasi yang singkat.

3.7.2 Klasifikasi

Metode klasifikasi yang digunakan pada penelitian ini adalah *Computed Input Weight* ELM (CIW-ELM). CIW-ELM merupakan pengembangan dari Extreme Learning Machine (ELM) konvensional. Pada ELM konvensional, inisialisasi nilai parameter bobot dilakukan secara acak dalam rentang nilai (-1, 1), sedangkan pada CIW-ELM dilakukan dengan melakukan perhitungan berdasarkan kombinasi linier dari data masukan, sehingga menghasilkan bobot dengan nilai acak yang dibobotkan sejumlah sampel data latih (Tapson et al., 2015). Algoritme CIW-ELM dipilih karena terbukti mampu menghasilkan akurasi yang sangat baik, dan hanya membutuhkan waktu komputasi yang sangat singkat, serta menghasilkan akurasi yang lebih unggul daripada algoritme ELM konvensional menggunakan jumlah *hidden neuron* yang sama.

Kerangka kerja dari metode yang diusulkan pada penelitian ini ditunjukkan pada Gambar 3.2. *Dataset* yang digunakan pada penelitian ini dibagi menjadi dua, yaitu sebagai data latih dan data uji. Selanjutnya dilakukan pengujian *hyperparameter* terbaik pada data latih menggunakan Grid Search Cross Validation. *Hyperparameter* terbaik yang didapatkan kemudian digunakan untuk membuat model akhir dari algoritme PCA dan CIW-ELM menggunakan keseluruhan data latih. Model tersebut kemudian diterapkan pada data uji untuk mengurangi dimensi dari data tersebut, serta melakukan klasifikasi untuk mengetahui kinerja dari algoritme yang digunakan.



Gambar 3.2 Kerangka kerja metode penelitian

3.8 Pengujian dan Analisis

Pengujian dilakukan untuk mencari *hyperparameter* terbaik dari setiap metode yang digunakan, sehingga mampu memperoleh hasil terbaik dengan waktu komputasi yang paling efisien. Pengujian tersebut dilakukan menggunakan Grid Search Cross Validation dan di ukur menggunakan rata-rata nilai akurasi dan *f-measure* untuk menentukan *hyperparameter* terbaik dari masing-masing metode. *Hyperparameter* terbaik akan dipilih dan digunakan untuk membuat model akhir menggunakan keseluruhan data latih, yang kemudian akan digunakan untuk menguji performa model pada data uji. Selanjutnya dilakukan analisis terhadap pengaruh penggunaan *hyperparameter* terhadap hasil pengujian yang didapatkan. Pada analisis tersebut, akan dijelaskan kenapa hasil tersebut bisa terjadi untuk menjawab pertanyaan-pertanyaan pada rumusan masalah penelitian.

3.9 Kesimpulan dan Saran

Setelah semua tahapan penelitian dilakukan, langkah selanjutnya adalah mengambil kesimpulan berdasarkan penelitian yang telah dilakukan. Kesimpulan tersebut diharapkan dapat menjawab rumusan masalah yang diangkat pada penelitian ini. Setelah kesimpulan didapatkan, akan dituliskan juga saran untuk penelitian selanjutnya terkait hal-hal yang dapat dikembangkan berdasarkan kelemahan-kelemahan yang masih ada pada penelitian ini. Hasil akhir penelitian ini lebih lanjut akan dijelaskan pada bab kesimpulan dan saran.

BAB 4 PERANCANGAN

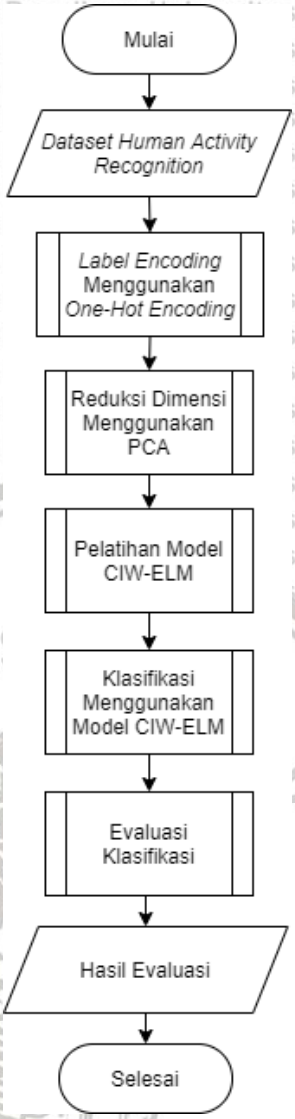
4.1 Identifikasi Permasalahan

Seperti yang telah dijelaskan pada bab sebelumnya, terdapat beberapa permasalahan yang ingin diselesaikan pada penelitian ini terkait masalah klasifikasi aktivitas manusia. Permasalahan pertama adalah keterbatasan prosesor yang ada pada ponsel cerdas dalam melakukan komputasi algoritme yang kompleks. Prosesor yang ada pada ponsel cerdas tentu saja memiliki kapasitas prosesor yang terbatas bila dibandingkan dengan prosesor yang dimiliki oleh komputer pada umumnya. Permasalahan selanjutnya yaitu besarnya dimensi yang dimiliki oleh data hasil ekstraksi dari sensor pada ponsel cerdas. Dimensi data yang besar dapat menyebabkan proses komputasi menjadi sangat lama, dan bahkan dapat membuat hasil akurasi klasifikasi menjadi menurun. Oleh karena itu, diperlukan sebuah algoritme yang dapat menyelesaikan permasalahan tersebut.

Penelitian ini akan menggunakan algoritme Principal Component Analysis *(PCA)* dan Computed Input Weight Extreme Learning Machine *(CIW-ELM)* untuk mengatasi permasalahan tersebut. PCA digunakan untuk mengurangi dimensi fitur yang ada pada data hasil ekstraksi pada sensor, sehingga dimensi fitur yang ada pada data tersebut dapat berkurang. Kemudian, proses klasifikasi akan dilakukan menggunakan algoritme CIW-ELM yang merupakan sebuah metode klasifikasi yang dapat berjalan dengan sangat cepat dan mampu menghasilkan akurasi yang sangat baik. Kedua algoritme tersebut diharapkan dapat menyelesaikan permasalahan yang terdapat pada penelitian ini, sehingga proses klasifikasi aktivitas manusia dapat dilakukan dengan cepat dan akurat.

4.2 Perancangan Algoritme

Data yang digunakan pada penelitian ini akan diproses menggunakan algoritme yang telah dijelaskan pada sub bab sebelumnya. Pertama, akan dilakukan *label encoding* menggunakan One-Hot Encoding untuk merubah bentuk label menjadi bentuk biner agar dapat digunakan pada proses pelatihan model CIW-ELM. Kemudian dilakukan reduksi dimensi pada data menggunakan algoritme PCA, sehingga dimensi fitur pada *dataset* menjadi lebih rendah. Selanjutnya dilakukan pelatihan model klasifikasi menggunakan algoritme CIW-ELM. Setelah model klasifikasi didapatkan, maka proses klasifikasi sudah dapat dilakukan. Hasil dari proses klasifikasi tersebut kemudian di evaluasi menggunakan nilai akurasi dan *f-measure* untuk mengukur performa atau kinerja dari algoritme yang digunakan. Diagram alir dari algoritme yang digunakan pada penelitian ini ditunjukkan pada Gambar 4.1.

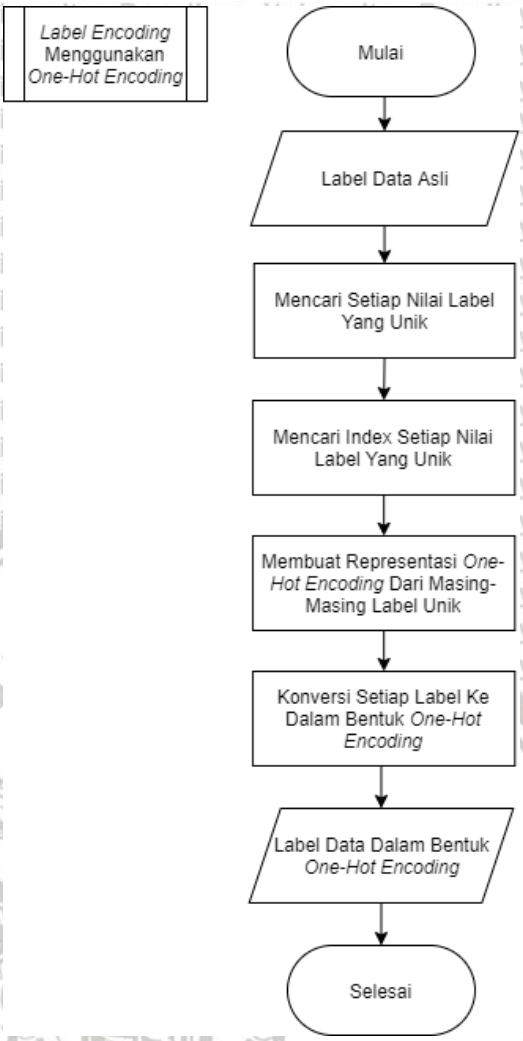


Gambar 4.1 Diagram alir algoritme yang digunakan

Dari Gambar 4.1, terlihat proses-proses yang akan dilakukan pada data yang digunakan pada penelitian ini, yaitu *Human Activity Recognition Using Smartphone Dataset. Dataset* tersebut dibagi menjadi dua, yaitu data latih dan data uji. Pada masing-masing data, akan dilakukan *label encoding* menggunakan One-Hot Encoding dengan tujuan untuk merubah bentuk label agar dapat digunakan dalam pelatihan model.

4.2.1 Proses One-Hot Encoding

One-Hot Encoding dilakukan dengan cara merubah nilai setiap kelas menjadi nilai biner, dengan indeks kelas yang bersangkutan diberi nilai 1 dan indeks kelas yang lain diberi nilai 0. Diagram alir dari proses *label encoding* ditunjukkan pada Gambar 4.2.



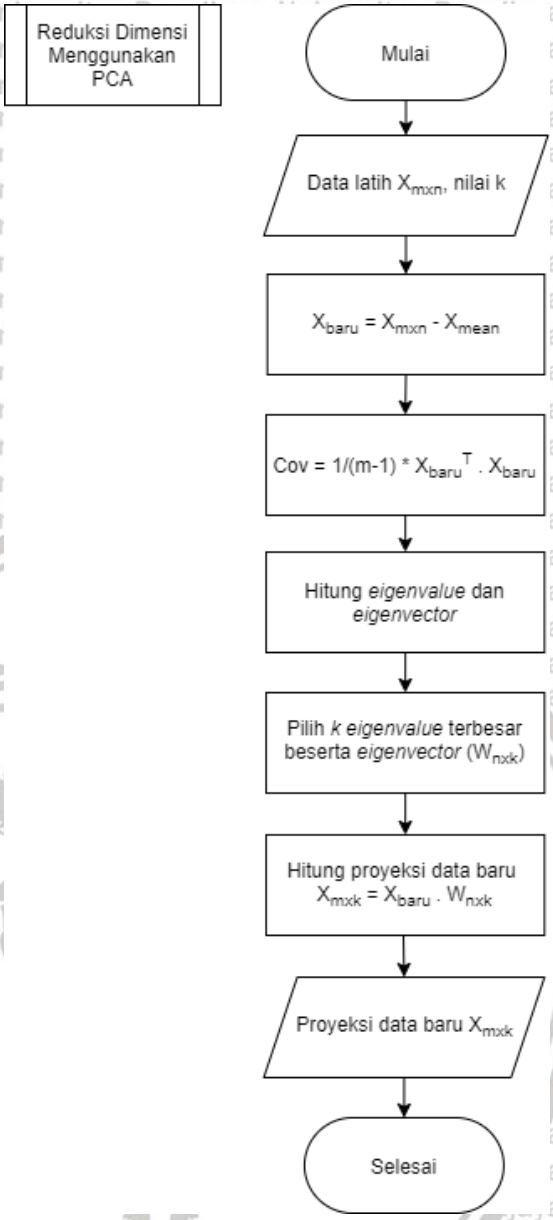
Gambar 4.2 Diagram alir One-Hot Encoding

Setelah semua label atau target pada masing-masing data diubah, langkah selanjutnya adalah melakukan reduksi dimensi untuk mengurangi jumlah fitur yang ada pada *dataset* yang digunakan.

4.2.2 Proses PCA

Pada proses ini, fitur dari data yang memiliki dimensi yang tinggi akan direduksi menjadi lebih rendah dengan cara memilih fitur yang memiliki lebih banyak informasi. Reduksi dimensi dilakukan menggunakan algoritme Principal Component Analysis (PCA). Algoritme ini membutuhkan masukan berupa data latih dan nilai *k* yang digunakan untuk merepersentasikan jumlah fitur yang ingin dipertahankan. Diagram alir dari algoritme PCA ditunjukkan pada Gambar 4.3.

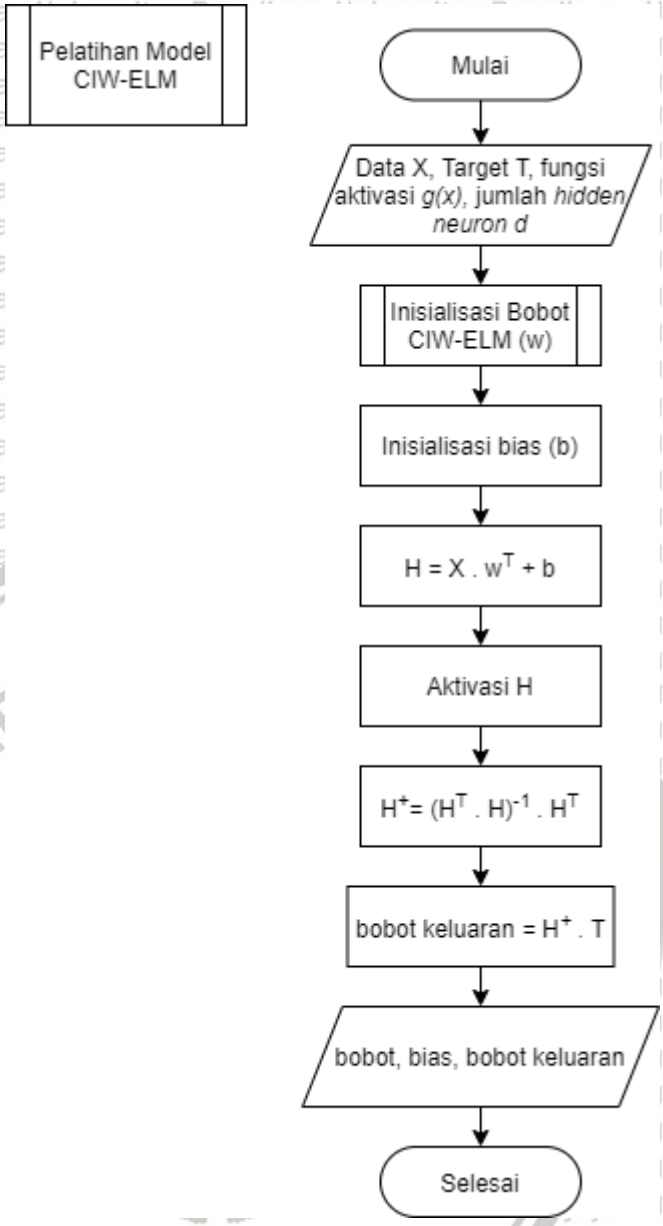
Dengan demikian, jumlah fitur dari *dataset* akan berkurang menjadi sejumlah *k*. Kemudian, dilakukan klasifikasi terhadap data yang fiturnya sudah direduksi menggunakan algoritme Computed Input Weight Extreme Learning Machine *(CIW-ELM)*.



Gambar 4.3 Diagram alir algoritme PCA

4.2.3 Proses CIW-ELM

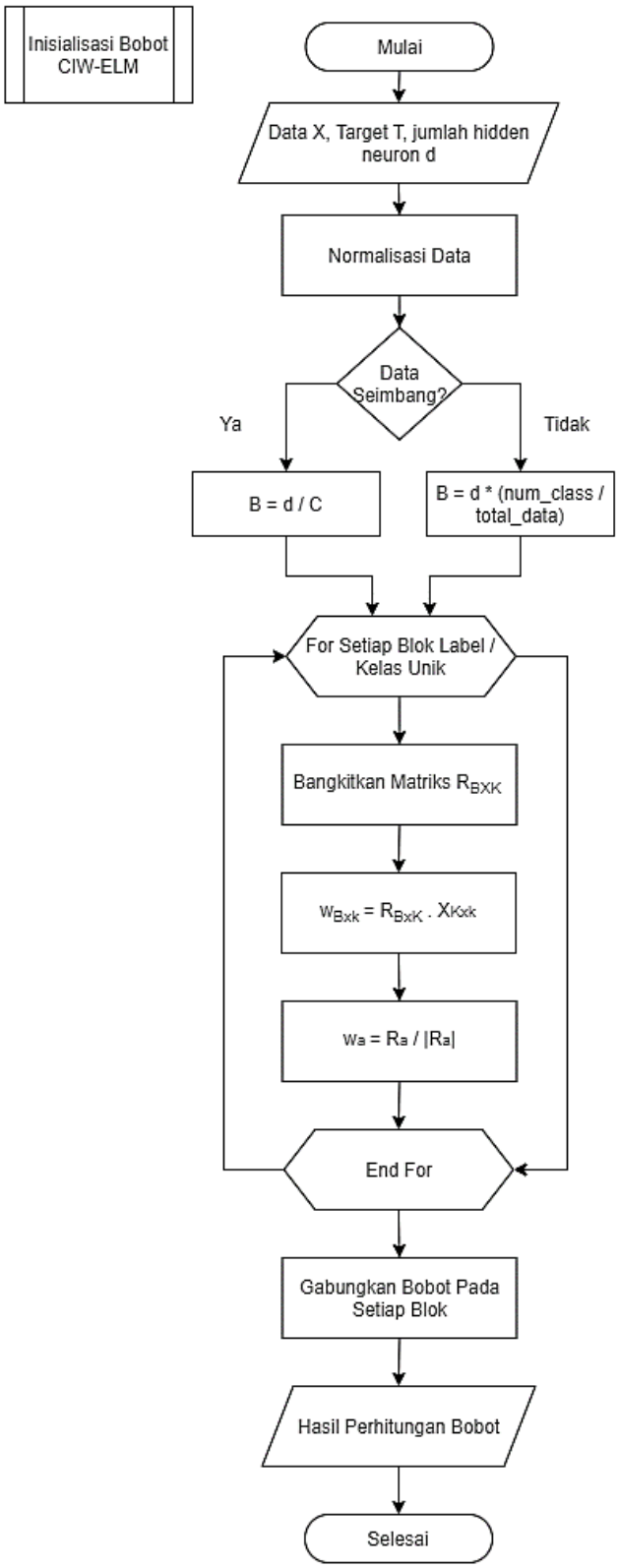
Pada algoritme CIW-ELM, terdapat dua tahapan yang dilakukan, yaitu tahap pelatihan atau pembuatan model dan tahap pengujian atau prediksi. Tahap pelatihan berguna untuk belajar dari data latih, sehingga menghasilkan sebuah model. Tahap pengujian berguna untuk menguji model yang didapatkan dari tahap pelatihan untuk melakukan klasifikasi terhadap data uji. Tahap pelatihan algotirme CIW-ELM sebagian besar masih sama seperti yang ada pada algoritme ELM konvensional, dengan diagram alir yang ditunjukkan pada Gambar 4.4.



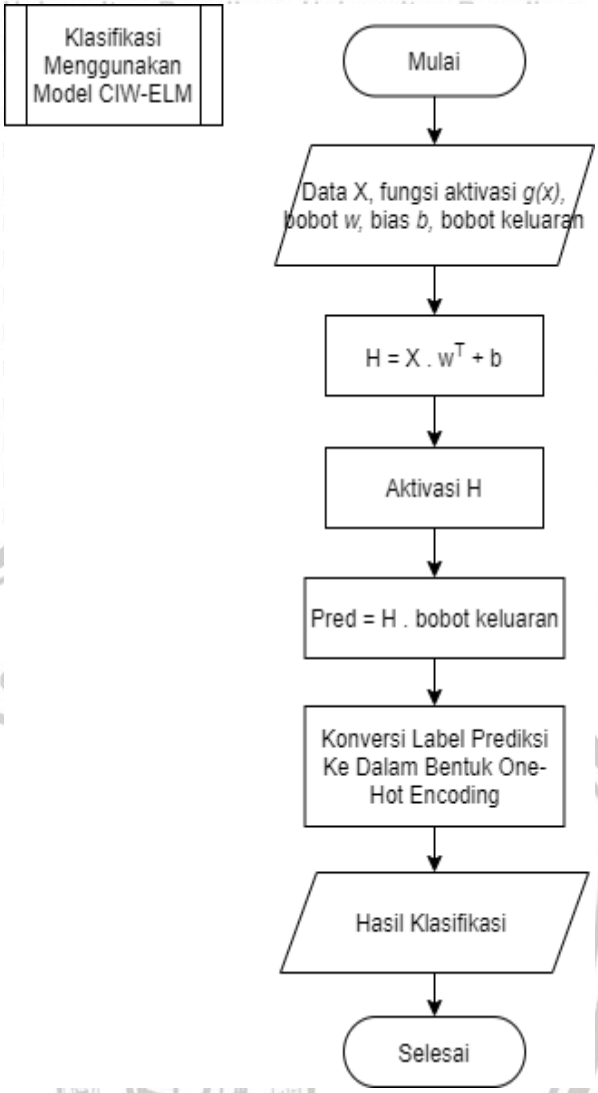
Gambar 4.4 Diagram alir pelatihan model CIW-ELM

Namun, berbeda dengan algoritme ELM konvensional, inisialisasi bobot pada algoritme CIW-ELM tidak dilakukan secara acak, melainkan dengan perhitungan sedemikian rupa terhadap data latih pada setiap kelas atau label. Diagram alir inisialisasi bobot CIW-ELM ditunjukkan pada Gambar 4.5.

Untuk bias dapat ditentukan ingin digunakan atau tidak. Jika ingin menggunakan bias, maka dapat di inisialisasi secara acak. Setelah tahap pelatihan dilakukan, selanjutnya adalah tahap pengujian atau prediksi. Diagram alir tahap pengujian atau prediksi untuk melakukan klasifikasi menggunakan model CIW-ELM ditunjukkan pada Gambar 4.6.



Gambar 4.5 Diagram alir inisialisasi bobot CIW-ELM

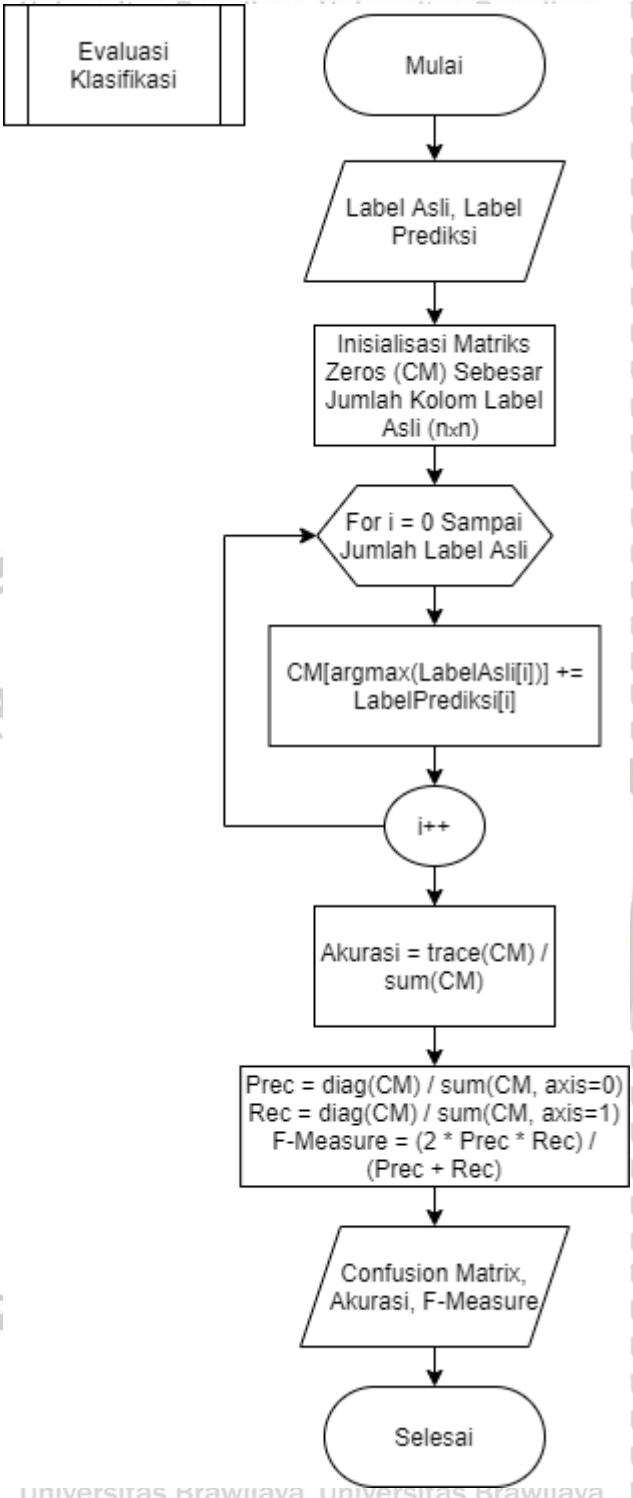


Gambar 4.6 Diagram alir klasifikasi menggunakan model CIW-ELM

4.2.4 Proses Evaluasi

Setelah semua tahapan dilakukan, maka akan didapatkan hasil klasifikasi aktivitas manusia. Seperti yang sudah dijelaskan sebelumnya, hasil tersebut kemudian akan di evaluasi menggunakan nilai akurasi dan *f-measure* untuk mengukur kinerja dari algoritme yang digunakan. Nilai akurasi dan *f-measure* dapat dihitung berdasarkan *confusion matrix* yang dibuat berdasarkan label asli dan label prediksi. Diagram alir dari proses evaluasi ditunjukkan pada Gambar 4.7.

Dari hasil akurasi dan *f-measure* tersebut, dapat diketahui performa dari algoritme yang digunakan dalam menyelesaikan permasalahan klasifikasi aktivitas manusia. Hasil tersebut akan digunakan untuk mengambil kesimpulan apakah algoritme yang digunakan cocok untuk menyelesaikan permasalahan aktivitas manusia atau tidak.



Gambar 4.7 Diagram alir evaluasi klasifikasi

4.3 Manualisasi

4.3.1 Pengambilan Data Sampel

Data latih yang digunakan pada perhitungan manual diambil dari tiga data pertama pada data latih untuk masing-masing kelas atau label, dengan hanya

menggunakan dua fitur pertama. Data latih yang digunakan ditunjukkan pada Tabel 4.1.

Tabel 4.1 Data latih untuk manualisasi

Data	Fitur 1	Fitur 2	Label
1	0,282022	-0,0377	1
2	0,255841	-0,06455	1
3	0,254867	0,003815	1
4	0,183604	-0,07656	2
5	0,279183	-0,03697	2
6	0,301162	-0,00176	2
7	0,358455	0,017975	3
8	0,301715	-0,00705	3
9	0,281552	-0,01611	3
10	0,144504	0,189263	4
11	0,287252	-0,03746	4
12	0,279998	-0,01948	4
13	0,288585	-0,02029	5
14	0,278419	-0,01641	5
15	0,279653	-0,01947	5
16	0,403474	-0,01507	6
17	0,278373	-0,02056	6
18	0,276555	-0,01787	6

Data uji yang digunakan diambil dari satu data pertama pada data uji untuk masing-masing kelas atau label, serta hanya menggunakan dua fitur pertama. Data uji yang digunakan ditunjukkan pada Tabel 4.2.

Tabel 4.2 Data uji untuk manualisasi

Data	Fitur 1	Fitur 2	Label
1	0,203962	-0,03234	1
2	0,251867	-0,03038	2
3	0,392763	-0,05917	3
4	0,296487	-0,01468	4
5	0,257178	-0,02329	5
6	0,173769	-0,02487	6

4.3.2 Manualisasi One-Hot Encoding

One-Hot Encoding dilakukan pada label data untuk merubah bentuk label dari kategorik menjadi bentuk biner, sehingga label tersebut nantinya dapat digunakan untuk proses selanjutnya. Pada kasus pengenalan aktivitas manusia yang ingin diselesaikan pada penelitian ini, *label encoding* dilakukan seperti berikut:

1. *Dataset* memiliki 6 label atau kelas yaitu: 1, 2, 3, 4, 5, dan 6, yang setiap nilai pada label tersebut mewakili aktivitas manusia, yaitu: *WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING,* dan *LAYING*.
2. Label tersebut dapat dituliskan menjadi [1, 2, 3, 4, 5, 6]
3. Index dari setiap label kemudian dicari
4. Nilai pada setiap label tersebut dapat direpresentasikan menjadi seperti berikut:
   - 1 → [1,0,0,0,0,0]
   - 2 → [0,1,0,0,0,0]
   - 3 → [0,0,1,0,0,0]
   - 4 → [0,0,0,1,0,0]
   - 5 → [0,0,0,0,1,0]
   - 6 → [0,0,0,0,0,1]

Hasil konversi One-Hot Encoding pada data latih ditunjukkan pada Tabel 4.3.

Tabel 4.3 Hasil One-Hot Encoding **pada label data latih**

Data	Fitur 1	Fitur 2	Label					
1	0,282022	-0,0377	1	0	0	0	0	0
2	0,255841	-0,06455	1	0	0	0	0	0
3	0,254867	0,003815	1	0	0	0	0	0
4	0,183604	-0,07656	0	1	0	0	0	0
5	0,279183	-0,03697	0	1	0	0	0	0
6	0,301162	-0,00176	0	1	0	0	0	0
7	0,358455	0,017975	0	0	1	0	0	0
8	0,301715	-0,00705	0	0	1	0	0	0
9	0,281552	-0,01611	0	0	1	0	0	0
10	0,144504	0,189263	0	0	0	1	0	0
11	0,287252	-0,03746	0	0	0	1	0	0
12	0,279998	-0,01948	0	0	0	1	0	0
13	0,288585	-0,02029	0	0	0	0	1	0
14	0,278419	-0,01641	0	0	0	0	1	0
15	0,279653	-0,01947	0	0	0	0	1	0
16	0,403474	-0,01507	0	0	0	0	0	1
17	0,278373	-0,02056	0	0	0	0	0	1
18	0,276555	-0,01787	0	0	0	0	0	1

Hasil konversi One-Hot Encoding pada label data uji ditunjukkan pada Tabel 4.4.

Tabel 4.4 Hasil One-Hot Encoding **pada label data uji**

Data	Fitur 1	Fitur 2	Label					
1	0,203962	-0,03234	1	0	0	0	0	0
2	0,251867	-0,03038	0	1	0	0	0	0

3	0,392763	-0,05917	0	0	1	0	0	0
4	0,296487	-0,01468	0	0	0	1	0	0
5	0,257178	-0,02329	0	0	0	0	1	0
6	0,173769	-0,02487	0	0	0	0	0	1

4.3.3 Manualisasi PCA

Setelah One-Hot Encoding dilakukan, langkah selanjutnya adalah melakukan reduksi dimensi menggunakan PCA. Reduksi dimensi menggunakan PCA dilakukan berdasarkan langkah-langkah yang sudah dijelaskan pada sub bab sebelumnya. Langkah pertama yaitu dengan memasukkan data latih yang digunakan, seperti yang ditunjukkan pada Tabel 4.1. Langkah selanjutnya yaitu menghitung nilai rata-rata dari setiap kolom pada data latih yang digunakan. Hasil dari perhitungan nilai rata-rata pada setiap kolom ditunjukkan pada Tabel 4.5.

Tabel 4.5 Hasil perhitungan nilai rata-rata pada data latih

Data	Fitur 1	Fitur 2
1	0,282022	-0,0377
2	0,255841	-0,06455
3	0,254867	0,003815
4	0,183604	-0,07656
5	0,279183	-0,03697
6	0,301162	-0,00176
7	0,358455	0,017975
8	0,301715	-0,00705
9	0,281552	-0,01611
10	0,144504	0,189263
11	0,287252	-0,03746
12	0,279998	-0,01948
13	0,288585	-0,02029
14	0,278419	-0,01641
15	0,279653	-0,01947
16	0,403474	-0,01507
17	0,278373	-0,02056
18	0,276555	-0,01787
Rata-Rata	0,278623	-0,0109

Setelah nilai rata-rata didapatkan, langkah selanjutnya adalah menghitung nilai data baru dengan cara mengurangi nilai data latih pada tiap kolom dengan nilai rata-rata yang bersangkutan seperti yang ditunjukkan pada Persamaan 2.1, sehingga akan menghasilkan nilai data baru dengan nilai rata-rata sebesar 0 atau

biasa disebut sebagai *zero mean*. Hasil perhitungan nilai data baru ditunjukkan pada Tabel 4.6.

Tabel 4.6 Hasil perhitungan nilai data baru dari data latih

Data	Fitur 1	Fitur 2
1	0,003399	-0,02679
2	-0,02278	-0,05365
3	-0,02376	0,014718
4	-0,09502	-0,06566
5	0,00056	-0,02606
6	0,022539	0,009144
7	0,079832	0,028878
8	0,023092	0,003853
9	0,002929	-0,00521
10	-0,13412	0,200166
11	0,008629	-0,02655
12	0,001375	-0,00858
13	0,009962	-0,00939
14	-0,0002	-0,00551
15	0,00103	-0,00856
16	0,124851	-0,00417
17	-0,00025	-0,00966
18	-0,00207	-0,00697
Rata-Rata	0	0

Kemudian, matriks kovarian dapat dihitung dengan cara mengalikan transpose dari matriks data baru dengan matriks data baru itu sendiri, kemudian dikalikan dengan $\frac{1}{m-1}$, dengan *m* merupakan jumlah data latih seperti yang ditunjukkan pada Persamaan 2.2. Hasil perhitungan matriks kovarian ditunjukkan pada Tabel 4.7.

Tabel 4.7 Matriks kovarian

0,003018	-0,00106
-0,00106	0,002998

Setelah itu, *eigenvalue* dan *eigenvector* dapat dihitung dari matriks kovarian. Salah satu cara untuk menghitung *eigenvalue* adalah dengan menyelesaikan Persamaan 4.1.

$$det[\lambda I - C] = 0 \quad (4.1)$$

Keterangan:

λ = Lambda

I = Matriks identitas

C = Matriks kovarian

Dari persamaan tersebut, didapatkan hasil persamaan seperti berikut:

$$det[\lambda I - C] = 0$$

$$det\left[\lambda * \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} - \begin{bmatrix}0{,}003018 & -0{,}00106\\ -0{,}00106 & 0{,}002998\end{bmatrix}\right] = 0$$

$$det\begin{bmatrix}\lambda - 0{,}003018 & 0{,}00106\\ 0{,}00106 & \lambda - 0{,}002998\end{bmatrix} = 0$$

$$(\lambda - 0{,}003018)(\lambda - 0{,}002998) - (0{,}00106 \times 0{,}00106) = 0$$

$$(\lambda - 0{,}003018)(\lambda - 0{,}002998) - (1.13E - 06\ ) = 0$$

$$\lambda(\lambda - 0{,}002998) - 0{,}003018(\lambda - 0{,}002998) - (1.13E - 06\ ) = 0$$

$$\lambda^2 - 0{,}002998\lambda - 0{,}003018\lambda + (9.05E - 05\ ) - (1.13E - 06\ ) = 0$$

$$\lambda^2 - 0{,}006015\lambda + (7.91E - 06) = 0$$

Dari hasil persamaan, didapatkan variabel yang akan digunakan pada perhitungan selanjutnya sebagai berikut:

a = 1

b = −0,06015

c = 7.91E−06

Variabel tersebut kemudian dimasukkan kedalam Persamaan 4.2 seperti yang ditunjukkan berikut.

$$\lambda_{1,2} = \frac{-b \pm \sqrt{b^2 - 4\,ac}}{2\,a} \quad (4.2)$$

Dari persamaan tersebut, akan didapatkan hasil seperti berikut.

$$\lambda_{1,2} = \frac{-b \pm \sqrt{b^2 - 4\,ac}}{2\,a}$$

$$\lambda_{1,2} = \frac{-(-0{,}006015) \pm \sqrt{(-0{,}006015)^2 - 4 \times 1 \times (7.91E - 06)}}{2 \times 1}$$

$$\lambda_{1,2} = \frac{0{,}006015 \pm \sqrt{3.62E - 05 - (3.16E - 05)}}{2}$$

$$\lambda_{1,2} = \frac{0{,}006015 \pm \sqrt{4.54E - 06}}{2}$$

$$\lambda_{1,2} = \frac{0{,}006015 \pm 0{,}00213}{2}$$

$$\lambda_1 = \frac{0{,}006015 - 0{,}00213}{2} = 0{,}001943$$

$$\lambda_2 = \frac{0{,}006015 + 0{,}00213}{2} = 0{,}004073$$

Dengan demikian, nilai *lambda* atau *eigenvalue* telah didapatkan. Langkah selanjutnya yaitu mencari *eigenvector* dari *eigenvalue* yang bersangkutan. Salah satu cara yang dapat dilakukan adalah dengan mencari persamaan seperti berikut:

$\lambda = [0{,}001943 \;\; 0{,}004073]$

$CU = \lambda U$

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \lambda \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \lambda \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = 0$$

$$\begin{bmatrix} c_{11} - \lambda & c_{12} \\ c_{21} & c_{22} - \lambda \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = 0$$

$$(c_{11} - \lambda)u_1 + c_{12}u_2 = 0$$

$$c_{21}u_1 + (c_{22} - \lambda)u_1 = 0$$

$$Matrik\ Kovarian: C = \begin{bmatrix} 0{,}003018 & -0{,}00106 \\ -0{,}00106 & 0{,}002998 \end{bmatrix}$$

Eigenvector bisa dihitung dengan persamaan:

$$(0{,}003018 - \lambda)u_1 - 0{,}00106u_2 = 0$$

$$-0{,}00106u_1 + (0{,}002998 - \lambda)u_2 = 0$$

Setelah persamaan *eigenvector* didapatkan, kita dapat memasukkan nilai matriks kovarian dan *eigenvalue* kedalam persamaan tersebut untuk mendapatkan nilai *eigenvector*. Untuk *eigenvalue* $\lambda = 0{,}001943$, maka *eigenvector* nya dapat di hitung seperti berikut:

$$(0{,}003018 - 0{,}001943)u_1 - 0{,}00106u_2 = 0$$

$$-0{,}00106u_1 + (0{,}002998 - 0{,}001943)u_2 = 0$$

$$0{,}001075u_1 - 0{,}00106u_2 = 0$$

$$-0{,}00106u_1 + 0{,}001055u_2 = 0$$

Solusi non trivial sistem persamaan:

$$0{,}001075u_1 - 0{,}00106u_2 = 0$$

$$\frac{0{,}001075u_1}{0{,}001075} + \frac{-0{,}000106u_2}{0{,}001075} = 0$$

$$u_1 - 0{,}9905u_2 = 0$$

$$u_1 = 0{,}9905u_2$$

Jadi, *eigenvector* untuk $\lambda = 0{,}001943$ adalah:

$$U = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0{,}9905u_2 \\ u_2 \end{bmatrix}$$

Nilai u_2 adalah bilangan sembarang yang bukan 0. Misalkan $u_2 = 1$, maka:

$$U = \begin{bmatrix} 0{,}9905 \\ 1 \end{bmatrix}$$

Ubah menjadi unit vektor dengan cara dibagi dengan akar dari jumlah kuadrat semua elemennya (*magnitude*)

$$U = \begin{bmatrix} \frac{0{,}9905}{\sqrt{(0{,}9905)^2+1^2}} \\ \frac{1}{\sqrt{(0{,}9905)^2+1^2}} \end{bmatrix}$$

$$U = \begin{bmatrix} \frac{0{,}9905}{1.407513} \\ \frac{1}{1.407513} \end{bmatrix}$$

$$U = \begin{bmatrix} 0{,}703725 \\ 0{,}710473 \end{bmatrix}$$

Dari perhitungan yang telah dilakukan, maka didapatkan *eigenvector* untuk *eigenvalue* $\lambda = 0{,}001943$. Untuk *eigenvalue* $\lambda = 0{,}004073$, *eigenvector* dapat dihitung menggunakan cara yang sama. Didapatkan hasil akhir dari perhitungan *eigenvalue* dan *eigenvector* seperti berikut:

Eigenvector untuk $\lambda = 0{,}001943$ adalah:

$$U = \begin{bmatrix} 0{,}703725 \\ 0{,}710473 \end{bmatrix}$$

Eigenvector untuk $\lambda = 0{,}004073$ adalah:

$$U = \begin{bmatrix} -0{,}71047 \\ 0{,}703725 \end{bmatrix}$$

Kemudian di gabungkan, sehingga akan menghasilkan matriks:

$$\lambda = [0{,}001943 \;\; 0{,}004073]$$

$$U = \begin{bmatrix} 0{,}703725 & -0{,}71047 \\ 0{,}710473 & 0{,}703725 \end{bmatrix}$$

Setelah *eigenvalue* dan *eigenvector* didapatkan, langkah selanjutnya adalah mengurutkan *eigenvalue* dari yang terbesar. Pasangan eigenvalue dan *eigenvector* sebelum diurutkan ditunjukkan pada Tabel 4.8.

Tabel 4.8 *Eigenvalue* dan *eigenvector* sebelum diurutkan

λ		0,001943	0,004073
U		0,703725	-0,71047
		0,710473	0,703725

Kemudian, pasangan *eigenvalue* dan *eigenvector* setelah diurutkan ditunjukkan pada Tabel 4.9.

Tabel 4.9 *Eigenvalue* dan *eigenvector* setelah diurutkan

λ	0,004073	0,001943
U	-0,71047	0,703725
	0,703725	0,710473

	0,703725	0,710473

Setelah *eigenvalue* diurutkan, kita perlu menentukan berapa nilai *k* atau *principal component* yang ingin digunakan. Nilai *k* yang dipilih akan menunjukkan jumlah fitur yang tersisa dari data. Misalkan disini dipilih *k = 1,* maka reduksi dimensi pada data latih dilakukan dengan mengalikan data baru atau matriks *zero mean* seperti yang ditunjukkan pada Tabel 4.7 dengan *eigenvector* yang bersangkutan dengan *k eigenvalue* yang dipilih. Hasil reduksi dimensi pada data latih ditunjukkan pada Tabel 4.10.

Tabel 4.10 Hasil reduksi dimensi pada data latih

Data	PC 1
1	-0,02127
2	-0,02157
3	0,027235
4	0,021305
5	-0,01874
6	-0,00958
7	-0,0364
8	-0,01369
9	-0,00575
10	0,23615
11	-0,02482
12	-0,00702
13	-0,01369
14	-0,00373
15	-0,00676
16	-0,09164
17	-0,00662
18	-0,00343

Untuk reduksi dimensi pada data uji, tahapan reduksi dimensi yang perlu dilakukan tetap sama, yaitu menghitung data baru atau matriks *zero mean* dari data uji terlebih dahulu. Nilai rata-rata yang digunakan untuk menghitung data baru atau matriks *zero mean* adalah nilai rata-rata yang didapatkan dari data latih sebelumnya. Hasil perhitungan data baru dari data uji ditunjukkan pada Tabel 4.11.

Tabel 4.11 Hasil perhitungan nilai data baru dari data uji

Data	Fitur 1	Fitur 2
1	-0,07466	-0,02144
2	-0,02676	-0,01948

3	0,11414	-0,04827
4	0,017864	-0,00378
5	-0,02145	-0,01238
6	-0,10485	-0,01397

Data baru tersebut kemudian dikalikan dengan eigenvector yang bersangkutan dengan *k eigenvalue* yang dipilih. Hasil reduksi dimensi pada data uji ditunjukkan pada Tabel 4.12.

Tabel 4.12 Hasil reduksi dimensi pada data uji

Data	PC 1
1	0,037955
2	0,005304
3	-0,11506
4	-0,01535
5	0,006522
6	0,064667

4.3.4 Manualisasi CIW-ELM

Data yang sudah dilakukan reduksi dimensi sebelumnya digunakan sebagai data masukan untuk perhitungan selanjutnya, yaitu perhitungan CIW-ELM. Data latih setelah dilakukan reduksi dimensi ditunjukkan pada Tabel 4.13.

Tabel 4.13 Data latih setelah reduksi dimensi

Data	PC 1	Label					
1	-0,02127	1	0	0	0	0	0
2	-0,02157	1	0	0	0	0	0
3	0,027235	1	0	0	0	0	0
4	0,021305	0	1	0	0	0	0
5	-0,01874	0	1	0	0	0	0
6	-0,00958	0	1	0	0	0	0
7	-0,0364	0	0	1	0	0	0
8	-0,01369	0	0	1	0	0	0
9	-0,00575	0	0	1	0	0	0
10	0,23615	0	0	0	1	0	0
11	-0,02482	0	0	0	1	0	0
12	-0,00702	0	0	0	1	0	0
13	-0,01369	0	0	0	0	1	0
14	-0,00373	0	0	0	0	1	0
15	-0,00676	0	0	0	0	1	0
16	-0,09164	0	0	0	0	0	1
17	-0,00662	0	0	0	0	0	1

18	-0,00343	0	0	0	0	0	1

Untuk data uji setelah reduksi dimensi ditunjukkan pada Tabel 4.14.

Tabel 4.14 Data uji setelah reduksi dimensi

Data	Fitur 1	Label					
1	0,037955	1	0	0	0	0	0
2	0,005304	0	1	0	0	0	0
3	-0,11506	0	0	1	0	0	0
4	-0,01535	0	0	0	1	0	0
5	0,006522	0	0	0	0	1	0
6	0,064667	0	0	0	0	0	1

Pada perhitungan algoritme CIW-ELM, kita juga perlu menentukan jumlah *hidden neuron* dan fungsi aktivasi yang akan digunakan. Misalkan disini jumlah *hidden neuron* yang digunakan adalah sebesar 12, dan fungsi aktivasi yang digunakan adalah *sigmoid*. Setelah itu, dilakukan inisialisasi bobot menggunakan algoritme CIW-ELM. Langkah pertama untuk melakukan inisialisasi bobot adalah dengan melakukan normalisasi data latih untuk menyamakan skala dari data tersebut. Namun pada kasus ini, data yang digunakan sudah dalam skala yang sama, yaitu dalam rentang (-1, 1) sehingga normalisasi data tidak perlu dilakukan. Langkah selanjutnya yaitu membagi *d* hidden layer menjadi *C* blok, dengan satu blok untuk setiap label keluaran. Oleh karena jumlah data latih *k* untuk setiap label keluaran berjumlah sama yaitu 3, maka ukuran untuk blok *B* dapat dihitung menggunakan Persamaan 4.3.

$$B = d/C \tag{4.3}$$

Didapatkan variabel yang dibutuhkan pada CIW-ELM seperti yang ditunjukkan pada Tabel 4.15.

Tabel 4.15 Variabel pada algoritme CIW-ELM

d	12
C	6
B	2
K	3

Keterangan:

d = jumlah *hidden neuron*

C = jumlah label/kelas untuk setiap keluaran

B = ukuran blok untuk setiap label

K = jumlah data latih untuk setiap label keluaran

Kemudian, bangkitkan matriks $R_{B\times K}$ secara acak dengan nilai biner antara 1 atau -1 untuk setiap blok. Hasil matriks $R_{B\times K}$ ditunjukkan pada Tabel 4.16.

Tabel 4.16 Matriks acak biner untuk setiap blok

Blok 1	1	1	1
	1	-1	-1
Blok 2	1	-1	-1
	-1	-1	1
Blok 3	-1	1	1
	-1	1	1
Blok 4	-1	-1	-1
	1	-1	-1
Blok 5	-1	1	1
	1	1	-1
Blok 6	1	-1	1
	-1	-1	1

Matriks tersebut akan digunakan untuk melakukan perhitungan bobot pada algoritme CIW-ELM. Perhitungan bobot dapat dilakukan dengan cara mengalikan matriks acak pada setiap blok yang telah dibuat dengan data latih pada setiap label yang bersangkutan dengan blok tersebut. Contoh perkalian yang dilakukan pada blok 1 adalah:

$$w = \begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \end{bmatrix} \cdot \begin{bmatrix} -0{,}02127 \\ -0{,}02157 \\ 0{,}027235 \end{bmatrix} = \begin{bmatrix} -0{,}0156 \\ -0{,}2694 \end{bmatrix}$$

Didapatkan bobot untuk seluruh blok seperti yang ditunjukkan pada Tabel 4.17.

Tabel 4.17 Hasil perhitungan bobot setiap blok

Blok 1	Blok 2	Blok 3	Blok 4	Blok 5	Blok 6
-0,0156	0,049623	0,016956	-0,20432	0,003197	-0,08845
-0,02694	-0,01215	0,016956	0,267981	-0,01066	0,094825

Selanjutnya, perlu dilakukan normalisasi terhadap bobot yang dibangkitkan untuk menyatukan magnitude dari bobot tersebut. Normalisasi dilakukan dengan Persamaan 2.7, *Magnitude* vektor dapat dihitung menggunakan Persamaan 4.4.

$$|R_\alpha| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \quad (4.4)$$

Contoh perhitungan *magnitude* vector untuk blok 1 adalah seperti berikut:

$$|R_1| = \sqrt{-0{,}0156^2 + 0{,}02694^2} = \sqrt{0{,}000969} = 0{,}03113$$

Didapatkan hasil perhitungan magnitude vektor dari bobot setiap blok seperti yang ditunjukkan pada Tabel 4.18.

Tabel 4.18 Magnitude vektor bobot pada setiap blok

|w|	0,03113	0,051088	0,02398	0,336986	0,011127	0,129676

Kemudian hasil normalisasi bobot ditunjukkan pada Tabel 4.19.

Tabel 4.19 Hasil normalisasi bobot setiap blok

Blok 1	Blok 2	Blok 3	Blok 4	Blok 5	Blok 6
-0,3657	-0,9289	0,539943	-0,75826	0,843155	0,682108
0,930734	-0,37033	0,841702	-0,65196	0,53767	0,731251

Bobot pada setiap blok tersebut kemudian digabungkan menjadi satu, sehingga menghasilkan bobot yang utuh sesuai dengan jumlah *hidden neuron* yang dipilih. Selain bobot, pada algoritme CIW-ELM juga terdapat parameter tambahan yang disebut bias. Penggunaan bias sebenarnya tidak diwajibkan, namun pada kasus ini penulis akan mencoba untuk menggunakannya. Bias dapat diinisialisasi dengan bilangan acak dalam rentang nilai antara -1 hingga 1. Hasil inisialisasi parameter bobot dan bias ditunjukkan pada Tabel 4.20.

Tabel 4.20 Hasil inisialisasi bobot dan bias

w1	b
-0,50118	0,101596
-0,86534	0,416296
0,97133	-0,41819
-0,23774	0,021655
0,707107	0,785894
0,707107	0,792586
-0,60631	-0,74883
0,795228	-0,58551
0,287274	-0,89707
-0,95785	-0,11838
-0,68211	-0,94025
0,731251	-0,08633

Setelah bobot dan bias didapatkan, langkah selanjutnya adalah menghitung nilai keluaran *hidden layer* berdasarkan Persamaan 2.3. Hasil perhitungan nilai keluaran *hidden neuron* ditunjukkan pada Gambar 4.8.

0.112256	0.434701	-0.43885	0.026712	0.770854	0.777546	-0.73593	-0.60243	-0.90318	-0.09801	-0.92574	-0.10189
0.112405	0.434958	-0.43914	0.026782	0.770644	0.777336	-0.73575	-0.60266	-0.90326	-0.09772	-0.92554	-0.1021
0.087946	0.392728	-0.39174	0.015181	0.805152	0.811844	-0.76534	-0.56386	-0.88924	-0.14447	-0.95882	-0.06642
0.090918	0.39786	-0.3975	0.01659	0.800959	0.807651	-0.76175	-0.56857	-0.89095	-0.13879	-0.95478	-0.07075
0.110987	0.432511	-0.43639	0.02611	0.772644	0.779336	-0.73747	-0.60042	-0.90245	-0.10043	-0.92747	-0.10004
0.106397	0.424585	-0.4275	0.023933	0.77912	0.785813	-0.74302	-0.59313	-0.89982	-0.1092	-0.93371	-0.09334
0.119837	0.447791	-0.45354	0.030308	0.760158	0.76685	-0.72676	-0.61446	-0.90752	-0.08352	-0.91542	-0.11295
0.108459	0.428146	-0.43149	0.024911	0.77621	0.782903	-0.74053	-0.5964	-0.901	-0.10526	-0.93091	-0.09635
0.104475	0.421267	-0.42377	0.023021	0.781831	0.788524	-0.74535	-0.59008	-0.89872	-0.11288	-0.93633	-0.09053
-0.01676	0.211945	-0.18881	-0.03449	0.952877	0.959569	-0.89201	-0.39772	-0.82923	-0.34458	-1.10133	0.086351
0.114033	0.43777	-0.44229	0.027555	0.768346	0.775039	-0.73378	-0.60525	-0.90419	-0.09461	-0.92332	-0.10448
0.105112	0.422366	-0.425	0.023323	0.780933	0.787626	-0.74458	-0.59109	-0.89908	-0.11166	-0.93546	-0.09146
0.108455	0.428139	-0.43148	0.024909	0.776216	0.782908	-0.74053	-0.5964	-0.901	-0.10527	-0.93091	-0.09634
0.103466	0.419524	-0.42181	0.022542	0.783256	0.789948	-0.74657	-0.58848	-0.89814	-0.11481	-0.9377	-0.08906
0.104983	0.422144	-0.42476	0.023262	0.781115	0.787807	-0.74473	-0.59089	-0.89901	-0.11191	-0.93564	-0.09128
0.147523	0.495595	-0.5072	0.043441	0.721095	0.727788	-0.69327	-0.65839	-0.92339	-0.0306	-0.87774	-0.15334
0.104913	0.422024	-0.42462	0.023229	0.781213	0.787906	-0.74482	-0.59078	-0.89897	-0.11204	-0.93573	-0.09117
0.103316	0.419266	-0.42153	0.022471	0.783466	0.790159	-0.74675	-0.58824	-0.89805	-0.11509	-0.93791	-0.08884

Gambar 4.8 Hasil perhitungan nilai keluaran *hidden neuron*

Nilai keluaran *hidden layer* tersebut kemudian di aktivasi menggunakan fungsi aktivasi yang ditentukan sebelumnya, yaitu *sigmoid*. Fungsi aktivasi *sigmoid* dapat dilakukan dengan Persamaan 2.8. Hasil nilai keluaran *hidden neuron* setelah di aktivasi ditunjukkan pada Gambar 4.9.

0.528035	0.606996	0.392015	0.506678	0.683706	0.685151	0.323894	0.353788	0.288398	0.475518	0.28379	0.47455
0.528072	0.607057	0.391946	0.506695	0.68366	0.685106	0.323933	0.353734	0.288381	0.475589	0.283831	0.474496
0.521972	0.596939	0.403299	0.503795	0.691075	0.692502	0.317488	0.362656	0.291266	0.463946	0.277114	0.483402
0.522714	0.598173	0.401914	0.504147	0.69018	0.691609	0.318267	0.361566	0.290915	0.465359	0.277925	0.482319
0.527718	0.606473	0.392601	0.506527	0.684092	0.685537	0.323558	0.354249	0.288548	0.474913	0.283439	0.475012
0.526574	0.60458	0.394725	0.505983	0.685491	0.686932	0.322344	0.355917	0.289088	0.472726	0.282172	0.476682
0.529923	0.610114	0.388519	0.507576	0.681388	0.682839	0.325906	0.351043	0.287507	0.479133	0.285892	0.471793
0.527088	0.605431	0.39377	0.506227	0.684863	0.686305	0.322889	0.355167	0.288845	0.473709	0.282741	0.475932
0.526095	0.603786	0.395615	0.505755	0.686075	0.687514	0.321836	0.356616	0.289314	0.471811	0.281643	0.477382
0.495811	0.552789	0.452937	0.491379	0.721693	0.723036	0.290695	0.40186	0.303809	0.414698	0.249491	0.521574
0.528477	0.607728	0.391194	0.506888	0.683163	0.68461	0.324365	0.353144	0.288189	0.476365	0.284282	0.473904
0.526254	0.604049	0.39532	0.50583	0.685881	0.687321	0.322004	0.356384	0.289239	0.472114	0.281818	0.47715
0.527087	0.605429	0.393772	0.506227	0.684864	0.686307	0.322888	0.355168	0.288846	0.473707	0.28274	0.475933
0.525843	0.603369	0.396083	0.505635	0.686381	0.68782	0.32157	0.356983	0.289433	0.47133	0.281365	0.477749
0.526222	0.603996	0.395379	0.505815	0.68592	0.68736	0.32197	0.356431	0.289255	0.472053	0.281782	0.477197
0.536814	0.621424	0.37585	0.510859	0.672848	0.67432	0.333307	0.341102	0.284267	0.49235	0.293646	0.461739
0.526204	0.603967	0.395412	0.505807	0.685942	0.687381	0.321952	0.356456	0.289263	0.472019	0.281763	0.477222
0.525806	0.603308	0.396152	0.505618	0.686427	0.687865	0.32153	0.357038	0.289451	0.471259	0.281324	0.477804

Gambar 4.9 Hasil aktivasi nilai keluaran *hidden neuron*

Nilai keluaran *hidden neuron* tersebut kemudian digunakan untuk menghitung matriks *Moore Penrose Pseudo-Inverse* berdasarkan Persamaan 4.4. Matriks inilah yang nantinya akan digunakan untuk melakukan perhitungan bobot keluaran pada algoritme CIW-ELM. Hasil perhitungan matriks *Moore Penrose Pseudo-Inverse* ditunjukkan pada Gambar 4.10.

-3048991	-1075320	-229717.732	650551.9	-1489292	4346186.698	-1682823	-1681748.81	-1993873	534098.6	4863278.517	304518.3
-3265690	-1151330	-245938.767	697492.3	-1595050	4654584.762	-1802283	-1801302.94	-2135138	571107.2	5209053.26	325906.5
-10079850	-3520689	-740672.255	2172269	-4930357	14399989	-5615604	-5554071.76	-6674967	1688953	16193198.22	978385.4
-3241935	-1122372	-235405.273	715358.5	-1583715	4620110.31	-1803082	-1787387.93	-2136976	520370.9	5211827.309	308568.7
-1272669	-452084	-96784.6735	266333	-622238	1817450.376	-703043	-701816.952	-834820	230339.2	2028158.468	129148.2
3823802.9	1338418	284330.8095	-828705	1867224	-5447112.15	2114628	2108769.899	2503964	-646760	-6114855.65	-374795
-15362048	-5388815	-1152962.97	3337587	-7493296	21842509.23	-8450651	-8477573.31	-9980671	2623891	24482328.3	1518518
1824200.5	635353.3	134839.9325	-400711	890095.9	-2594831	1007718	1006243.201	1191134	-301311	-2918137.96	-176868
5185943.9	1818244	386183.5199	-1117924	2533356	-7392771.6	2870392	2859544.117	3401892	-884169	-8294538.89	-510005
132834.28	44905.7	8651.339619	-29120	65586.37	-192483.089	77201.62	73254.93597	93464.67	-18937.3	-219777.236	-11587.2
-5735151	-2017252	-430855.749	1233377	-2799990	8167732.449	-3162689	-3163846.34	-3743069	992191.8	9148121.885	569570.4
4792008.4	1679348	356719.4051	-1034599	2340642	-6829730.88	2651604	2642453.446	3141745	-815208	-7663921.9	-470843
1828724.9	636943.1	135178.1396	-401683	892306.3	-2601281.48	1010220	1008738.292	1194100	-302090	-2925368.49	-177316
5687906.6	1995528	423755.8	-1223396	2779055	-8110948.49	3149618	3136091.898	3734332	-972733	-9098549.48	-560049
4876322.7	1709064	363024.8617	-1052462	2381883	-6950208.76	2698413	2688920.663	3197384	-829940	-7798872.64	-479219
3184360.2	1128010	244905.8371	-686094	1551011	-4516500.16	1733426	1760373.593	2039618	-568424	-5034875.58	-324104
4921176.3	1724875	366379.2032	-1061959	2403824	-7014306.1	2723318	2713639.582	3226991	-837783	-7870668.04	-483676
5748965.8	2017132	428330.4427	-1236137	2808959	-8198398.14	3183636	3169721.916	3774885	-983596	-9196401.9	-566155

Gambar 4.10 Hasil matriks *Moore Penrose Pseudo-Inverse*

Langkah terakhir dalam pembuatan model CIW-ELM adalah menghitung nilai bobot keluaran atau *beta* berdasarkan Persamaan 2.5. Hasil perhitungan bobot keluaran CIW-ELM ditunjukkan pada Gambar 4.11.

-16394531.44	-690801.5068	-8351903.13	-810308.2067	12392954.28	13854502.23
-5747338.4	-236037.5125	-2935217.396	-292998.5492	4341535.049	4870016.064
-1216328.754	-47859.13736	-631939.5141	-65485.00395	921958.8013	1039615.483
3520313.455	152986.4405	1818951.622	169657.6149	-2677540.856	-2984190.54
-8014698.699	-338728.8189	-4069844.196	-393761.0595	6053244.411	6763793.841
23400760.46	990448.5331	11854906.63	1145518.481	-17662438.74	-19729204.4
-9100710.772	-391496.5154	-4572541.783	-433883.6687	6858250.295	7640379.478
-9037123.501	-380434.9877	-4611785.987	-448137.956	6833750.853	7643735.092
-10803977.52	-467832.3221	-5387644.126	-507859.0572	8125816.178	9041494.58
2794158.685	103950.3072	1438411.876	158046.3168	-2104763.934	-2389803.432
26265530	1125130.13	13269651.45	1264422.753	-19822790.61	-22101945.53
1608810.281	62922.1816	831645.5096	87140.44242	-1216583.188	-1373934.772

Gambar 4.11 Hasil perhitungan bobot keluaran

Dengan demikian, semua parameter yang dibutuhkan untuk melakukan klasifikasi yaitu bobot, bias, dan bobot keluaran sudah didapatkan, sehingga tahap klasifikasi sudah bisa dilakukan. Pada tahap ini, data yang digunakan adalah data uji seperti yang ditunjukkan pada Tabel 4.12. Dari data uji tersebut, dihitung nilai keluaran *hidden layer* berdasarkan Persamaan 2.3. Hasil perhitungan nilai keluaran *hidden neuron* pada data uji ditunjukkan pada Gambar 4.12.

0.082573	0.383451	-0.38132	0.012632	0.812732	0.819425	-0.77184	-0.55533	-0.88616	-0.15474	-0.96614	-0.05858
0.098938	0.411706	-0.41304	0.020394	0.789644	0.796337	-0.75205	-0.5813	-0.89554	-0.12346	-0.94387	-0.08246
0.159263	0.515865	-0.52995	0.04901	0.704532	0.711224	-0.67907	-0.67702	-0.93012	-0.00817	-0.86176	-0.17047
0.109291	0.429582	-0.4331	0.025305	0.775037	0.78173	-0.73952	-0.59772	-0.90148	-0.10367	-0.92977	-0.09756
0.098327	0.410652	-0.41186	0.020105	0.790506	0.797198	-0.75278	-0.58033	-0.89519	-0.12463	-0.9447	-0.08156
0.069186	0.360336	-0.35538	0.006281	0.831621	0.838313	-0.78804	-0.53409	-0.87849	-0.18032	-0.98436	-0.03905

Gambar 4.12 Hasil perhitungan nilai keluaran *hidden neuron* pada data uji

Proses selanjutnya sama seperti sebelumnya, yaitu perlu dilakukan aktivasi pada nilai keluaran *hidden neuron* tersebut menggunakan fungsi aktivasi yang sama seperti yang digunakan pada tahap pelatihan model sebelumnya, yaitu

sigmoid. Hasil aktivasi nilai keluaran *hidden layer* pada data uji ditunjukkan pada Gambar 4.13.

0.520632	0.594705	0.405808	0.503158	0.692691	0.694114	0.316081	0.364628	0.291902	0.461393	0.275651	0.48536
0.524714	0.601497	0.398184	0.505098	0.687755	0.68919	0.320376	0.358634	0.289967	0.469174	0.28012	0.479398
0.539732	0.62618	0.370527	0.51225	0.669192	0.670672	0.33647	0.336928	0.2829	0.497958	0.296971	0.457485
0.527296	0.605774	0.393385	0.506326	0.68461	0.686053	0.323109	0.354865	0.288747	0.474105	0.28297	0.475629
0.524562	0.601244	0.398467	0.505026	0.68794	0.689375	0.320215	0.358857	0.29004	0.468883	0.279953	0.47962
0.51729	0.589122	0.412079	0.50157	0.696698	0.69811	0.31259	0.369564	0.293491	0.455041	0.272028	0.49024

Gambar 4.13 Hasil aktivasi nilai keluaran *hidden neuron* pada data uji

Nilai keluaran *hidden layer* yang sudah di aktivasi kemudian dikalikan dengan bobot keluaran atau *beta* yang sudah didapatkan sebelumnya sesuai dengan Persamaan 2.6, sehingga akan menghasilkan keluaran berupa prediksi dari label data uji. Hasil perhitungan nilai keluaran prediksi dari data uji ditunjukkan pada Tabel 4.21.

Tabel 4.21 Hasil keluaran prediksi label data uji

0,32169	0,545527	-0,54086	0,117961	-0,08247	0,513909
0,225741	0,349746	-0,22618	0,276363	-0,0844	0,310845
0,364669	0,183974	-0,54482	-0,07015	-0,72778	1,559994
0,125951	0,236011	-0,10362	0,348582	-0,0727	0,302978
0,230978	0,356867	-0,23547	0,271203	-0,085	0,314412
0,313839	0,69577	-0,86906	-0,01769	-0,04093	0,813237

Hasil keluaran prediksi tersebut perlu dikonversi ke dalam bentuk One-Hot Encoding berdasarkan indeks dari nilai maksimal untuk masing-masing vektor prediksi. Hasil konversi nilai keluaran tersebut ditunjukkan pada Tabel 4.22.

Tabel 4.22 Hasil konversi nilai keluaran prediksi

0	1	0	0	0	0
0	1	0	0	0	0
0	0	0	0	0	1
0	0	0	1	0	0
0	1	0	0	0	0
0	0	0	0	0	1

4.3.5 Manualisasi Evaluasi

Hasil keluaran prediksi yang sudah di konversi kedalam bentuk One-Hot Encoding tersebut kemudian dibandingkan dengan label asli pada data uji. Perbandingan tersebut dapat digunakan untuk membuat *confusion matrix* seperti yang ditunjukkan pada Tabel 4.23.

Tabel 4.23 *Confusion matrix* prediksi data uji

A\P	1	2	3	4	5	6
1	0	1	0	0	0	0

2	0	1	0	0	0	0
3	0	0	0	0	0	1
4	0	0	0	1	0	0
5	0	1	0	0	0	0
6	0	0	0	0	0	1

Dari *confusion matrix* tersebut, dapat dihitung nilai akurasi dan *f-measure* untuk mengukur kinerja dari model yang sudah dibuat. Nilai akurasi dapat dihitung menggunakan Persamaan 2.12.

$$Akurasi = \frac{3}{6} = 0{,}5$$

Nilai *f-measure* dapat dihitung menggunakan Persamaan 2.13. Contoh hasil perhitungan nilai *f-measure* untuk label kelas 6 adalah:

$$F - Measure = 2 \times \frac{\frac{1}{2} \times \frac{1}{1}}{\frac{1}{2} + \frac{1}{1}} = 0{,}667$$

4.4 Perancangan Pengujian

Pengujian dilakukan untuk mencari *hyperparameter* terbaik dari masing-masing algoritme yang digunakan. Untuk algoritme PCA, pengujian akan dilakukan untuk mencari nilai *k* atau jumlah *principal component* terbaik yang dapat merepresentasikan keseluruhan informasi dari data dengan jumlah dimensi yang paling minim. Pemilihan nilai *k* yang akan diuji merujuk pada penelitian sebelumnya yang menggunakan *dataset* yang sama dengan memperhatikan persentase kontribusi dari fitur atau atribut yang bersangkutan (Kishore et al., 2017). Persentase kontribusi atribut dan jumlah fitur yang dimaksud ditunjukkan pada Tabel 4.24.

Tabel 4.24 Persentase kontribusi atribut dan jumlah fitur yang diuji

Kontribusi atribut > x %	Jumlah fitur
10	1
1	8
0,1	97
0,01	207
0,001	339
0,0003	381
0,0001	406

Sumber: Diadaptasi dari Kishore, et al (2017)

Kemudian untuk algoritme CIW-ELM, pengujian akan dilakukan untuk mencari jumlah *hidden neuron* dan fungsi aktivasi terbaik yang dapat menghasilkan rata-rata akurasi dan *f-measure* yang paling tinggi. Jumlah *hidden neuron* yang akan diuji adalah sebesar 10, 50, 100, 200, 300, 400, 500, dan 600. Fungsi aktivasi yang akan diuji disesuaikan dengan fungsi aktivasi yang pernah digunakan pada penelitian sebelumnya, yaitu *sigmoid*, *tanh, sine,* dan *hardlim* (Sevin, 2018). Nilai *hyperparameter* yang akan diuji pada penelitian ini ditunjukkan pada Tabel 4.25.

Tabel 4.25 Nilai hyperparameter yang diuji

	Nilai *Hyperparameter*							
k	1	8	97	207	339	381	406	
hidden neuron	10	50	100	200	300	400	500	600
Fungsi aktivasi	*sigmoid*		*tanh*		*sine*		*hardlim*	

Metode pengujian yang digunakan adalah Grid Search Cross Validation, dengan masing-masing nilai *hyperparameter* yang diuji akan dipasangkan dengan nilai *hyperparameter* lain secara keseluruhan. Dengan kata lain, pengujian akan dilakukan terhadap seluruh kemungkinan pasangan nilai *hyperparameter* yang ada. Jumlah pasangan *hyperparameter* yang akan diuji pada penelitian ini adalah sebanyak 224 pasang. Contoh pasangan nilai *hyperparameter* yang akan diuji ditunjukkan pada Tabel 4.26.

Tabel 4.26 Pasangan hyperparameter yang diuji

No.	K	Jumlah Neuron	Aktivasi	Skor Akhir	Rata-Rata Waktu Pelatihan (Detik)
1	1	10	*sigmoid*		
2	1	10	*tanh*		
3	1	10	*sine*		
4	1	10	*hardlim*		
5	1	50	*sigmoid*		
…	…	…	…		
220	406	500	*hardlim*		
221	406	600	*sigmoid*		
222	406	600	*tanh*		
223	406	600	*sine*		
224	406	600	*hardlim*		

Pengujian dilakukan menggunakan 10-Fold Cross Validation, sehingga data latih akan dibagi menjadi *10-Fold*. Dari total *10-Fold* tersebut, *1-Fold* akan digunakan sebagai *validation set*, dan sisanya yaitu *9-Fold* akan digunakan sebagai *training set* secara bergantian, sehingga proses pengujian *hyperparameter* akan melibatkan keseluruhan data latih. Skor dari masing-

masing *fold* akan dihitung berdasarkan rata-rata nilai akurasi dan *f-measure*. Skor akhir untuk setiap pasangan *hyperparameter* akan dihitung berdasarkan rata-rata skor pada masing-masing fold. Dari pengujian yang dilakukan, akan diambil pasangan *hyperparameter* dengan skor akhir terbaik untuk digunakan dalam pembuatan model akhir menggunakan keseluruhan data latih.

BAB 5 IMPLEMENTASI

5.1 Implementasi Algoritme

Rancangan algoritme yang telah dibuat pada bab sebelumnya kemudian akan diimplementasikan ke dalam bentuk kode program. Bahasa pemrograman yang digunakan adalah *Python* 3.8.6, dan ditulis pada *Jupyter Notebook* 6.0.3. *Library Python* yang digunakan untuk melakukan implementasi algoritme pada penelitian ini adalah *numpy*, *pandas*, dan *time*. Setiap implementasi algoritme kecuali *main method* akan dibuat dalam bentuk *class* untuk membuat kode program menjadi lebih rapi.

5.2 Implementasi One-Hot Encoding

One-Hot Encoding digunakan untuk melakukan *encoding* terhadap label dari *dataset* menjadi bentuk biner. Hal pertama yang dilakukan adalah mencari indeks dari setiap label unik yang ada pada *dataset* yang kemudian akan digunakan untuk membuat matriks yang merepresentasikan bentuk One-Hot Encoding dari setiap label unik. Selanjutnya, setiap label yang ada pada *dataset* akan diubah ke dalam bentuk One-Hot Encoding sesuai dengan indeks dari label tersebut berdasarkan indeks label unik yang telah didapatkan sebelumnya. Implementasi *label encoding* menggunakan One-Hot Encoding dapat dilihat pada Kode Program 5.1.

Kode Program 5.1 Implementasi *label encoding* menggunakan One-Hot Encoding

Baris	Kode Program
1	`class LabelEncoder:`
2	`    def fit(self, label):`
3	`        indeks_label = self.__get_indeks(label)`
4	`        self.label_onehot = np.identity(np.max(indeks_label)+1, dtype="uint8")`
5	
6	`    def transform(self, label):`
7	`        indeks_label = self.__get_indeks(label)`
8	`        hasil = np.zeros((indeks_label.shape[0], self.label_onehot.shape[1]), dtype="uint8")`
9	`        for i, idx in enumerate(indeks_label):`
10	`            hasil[i] = self.label_onehot[idx]`
11	
12	`        return hasil`
13	
14	`    def fit_transform(self, label):`
15	`        self.fit(label)`
16	
17	`        return self.transform(label)`
18	
19	`    def __get_indeks(self, label):`
20	`        self.label_asli = np.unique(label)`
21	`        indeks_label = [np.where(self.label_asli==x)[0] for x in label]`
22	
23	`        return np.array(indeks_label)`

Penjelasan tiap baris kode program pada Kode Program 5.1 adalah:

Baris 1 : Inisialisasi *class* dengan nama `LabelEncoder`.

Baris 2 : Inisialisasi *method* dengan nama `fit()`, yang dapat menerima sebuah parameter yang bernama `label`.

Baris 3 : Memanggil *private method* yang bernama `__get_indeks()` untuk mencari indeks dari masing-masing label, kemudian menyimpan hasilnya pada variabel `indeks_label`.

Baris 4 : Membuat matriks identitas dengan ukuran sebesar nilai maksimal dari variabel `indeks_label` + 1 dengan tipe data uint8 sebagai representasi One-Hot Encoding dari masing-masing label, kemudian disimpan pada *instance* variabel yang bernama `label_onehot`.

Baris 6 : Inisialisasi *method* dengan nama `transform`, yang dapat menerima sebuah parameter yang bernama `label`.

Baris 7 : Memanggil *private method* yang bernama `__get_indeks()` untuk mencari indeks dari masing-masing label, kemudian menyimpan hasilnya pada variabel `indeks_label`.

Baris 8 : Membuat matriks nol dengan ukuran sebesar jumlah baris pada variabel `indeks_label` dikali jumlah kolom pada instance variabel `label_onehot`, dengan tipe data uint8 sebagai data awal untuk menyimpan representasi One-Hot Encoding dari masing-masing data label, kemudian disimpan pada variabel yang bernama `hasil`.

Baris 9 : Melakukan perulangan terhadap nilai pada variabel `indeks_label`.

Baris 10 : Mengganti nilai pada variabel hasil sesuai dengan representasi One-Hot Encoding dari variabel `indeks_label` yang bersangkutan.

Baris 12 : Mengembalikan nilai variabel `hasil`.

Baris 14 : Inisialisasi *method* dengan nama `fit_transform()`, yang dapat menerima sebuah parameter yang bernama `label`.

Baris 15 : Memanggil *method* `fit()`.

Baris 17 : Memanggil *method* `transform()` dan mengembalikan nilai kembalian dari method tersebut.

Baris 19 : Inisialisasi *private method* dengan nama `__get_indeks()`, yang dapat menerima sebuah parameter yang bernama `label`.

Baris 20 : Mencari nilai unik pada label, kemudian menyimpan hasilnya pada *instance* variabel yang bernama `label_asli`.

Baris 21 : Melakukan perulangan menggunakan *list comprehension* untuk mencari nilai indeks dari setiap label berdasarkan nilai unik pada

instance variabel `label_asli`, kemudian menyimpan hasilnya pada variabel yang bernama `indeks_label`.

Baris 23 : Mengembalikan nilai dari variabel `indeks_label` yang di konversi menjadi *numpy array*.

5.3 Implementasi PCA

PCA digunakan untuk melakukan reduksi dimensi fitur dari *dataset*. Reduksi dimensi dilakukan dengan cara mengalikan data yang fiturnya ingin di reduksi dengan nilai *eigenvector* sebanyak *k*. Implementasi PCA dapat dilihat pada Kode Program 5.2.

Kode Program 5.2 Implementasi PCA

Baris	Kode Program
1	`class PCA:`
2	`    def __init__(self, k):`
3	`        self.k = k`
4	
5	`    def fit(self, x_latih):`
6	`        self.x_mean = np.mean(x_latih, axis=0)`
7	`        x_baru = x_latih - self.x_mean`
8	`        cov = 1 / (x_latih.shape[0]-1) * np.matmul(x_baru.T, x_baru)`
9	
10	`        eigenvalue, eigenvector = np.linalg.eig(cov)`
11	`        indeks_terurut = np.argsort(-eigenvalue)`
12	
13	`        self.eigenvalue = eigenvalue[indeks_terurut].real`
14	`        self.eigenvector = eigenvector[:, indeks_terurut].real`
15	
16	`    def transform(self, x):`
17	`        x_baru = x - self.x_mean`
18	
19	`        return np.matmul(x_baru, self.eigenvector[:,:self.k])`
20	
21	`    def fit_transform(self, x_latih):`
22	`        self.fit(x_latih)`
23	
24	`        return self.transform(x_latih)`

Penjelasan tiap baris kode program pada Kode Program 5.2 adalah:

Baris 1 : Inisialisasi *class* dengan nama `PCA`.

Baris 2 : Inisialisasi *constructor* yang dapat menerima sebuah parameter yang bernama `k`.

Baris 3 : Memasukkan nilai parameter *k* pada instance variabel bernama `k`.

Baris 5 : Inisialisasi *method* dengan nama `fit()`, yang dapat menerima sebuah parameter yang bernama `x_latih`.

Baris 6 : Mencari nilai rata-rata setiap kolom dari `x_latih`, kemudian menyimpan hasilnya pada *instance* variabel yang bernama `x_mean`.

Baris 7 : Menghitung nilai x baru, kemudian menyimpan hasilnya pada variabel yang bernama `x_baru`.

Baris 8 : Menghitung nilai matriks kovarian, kemudian menyimpan hasilnya pada variabel yang bernama `cov`.

Baris 10 : Mencari *eigenvalue* dan *eigenvector* dari matriks kovarian, kemudian menyimpan hasilnya pada variabel `eigenvalue` dan `eigenvector`.

Baris 11 : Mencari indeks terurut dari variabel `eigenvalue` berdasarkan nilai *eigenvalue* yang paling besar, kemudian menyimpan hasilnya pada variabel `indeks_terurut`.

Baris 13 : Mengambil nilai real dari variabel `eigenvalue` dengan nilai yang sudah diurutkan berdasarkan variabel `indeks_terurut`, dan menyimpannya ke dalam instance variabel yang bernama `eigenvalue`.

Baris 14 : Mengambil nilai real dari variabel `eigenvector` dengan nilai yang sudah diurutkan berdasarkan variabel `indeks_terurut` pada masing-masing kolom, dan menyimpannya ke dalam *instance* variabel yang bernama `eigenvector`.

Baris 16 : Inisialisasi *method* dengan nama `transform()`, yang dapat menerima sebuah parameter yang bernama `x`.

Baris 17 : Menghitung nilai x baru berdasarkan *instance* variabel `x_mean`, kemudian menyimpan hasilnya pada variabel `x_baru`.

Baris 19 : Menghitung perkalian matriks dari variabel `x_baru` dengan *instance* variabel `eigenvector` dengan jumlah kolom sebesar nilai *instance* variabel `k`, kemudian mengembalikan nilai hasil perhitungan tersebut.

Baris 21 : Inisialisasi *method* dengan nama `fit_transform()`, yang dapat menerima sebuah parameter yang bernama `x_latih`.

Baris 22 : Memanggil *method* `fit()`

Baris 24 : Memanggil *method* `transform()` dan mengembalikan nilai kembalian dari *method* tersebut.

5.4 Implementasi CIW-ELM

CIW-ELM digunakan untuk melakukan klasifikasi aktivitas manusia. Algoritme tersebut memiliki dua tahapan, yaitu tahap pelatihan model dan tahap prediksi atau klasifikasi. Implementasi algoritme CIW-ELM dapat dilihat pada Kode Program 5.3.

Kode Program 5.3 Implementasi CIW-ELM

Baris	Kode Program
1	`class ELM:`
2	`    def __init__(self, jml_neuron, init_bobot="ciw", init_bias="acak", aktivasi="sigmoid", random_state=None):`

```
        self.jml_neuron = jml_neuron
        self.init_bobot = init_bobot
        self.init_bias = init_bias
        self.aktivasi = aktivasi
        self.random_state = random_state

    def latih_model(self, x_latih, y_latih, normalisasi=False):
        self.bobot = self.__get_bobot(x_latih, y_latih, normalisasi=normalisasi)
        self.bias = self.__get_bias()

        H = self.__get_H(x_latih)
        try:
            H_pinv = np.matmul(np.linalg.inv(np.matmul(H.T, H)), H.T)
        # error jika singular matriks (determinant = 0)
        except:
            H_pinv = np.linalg.pinv(H)

        self.beta = np.matmul(H_pinv, y_latih)

    def prediksi(self, x):
        H = self.__get_H(x)
        self.y_awal = np.matmul(H, self.beta)

        return self.__get_y_onehot(self.y_awal)

    def __get_bobot(self, x_latih, y_latih, normalisasi):
        np.random.seed(self.random_state)

        if self.init_bobot == "ciw":
            return self.__get_ciw(x_latih, y_latih, normalisasi=normalisasi)
        elif self.init_bobot == "acak":
            return np.random.uniform(-1, 1, size=(self.jml_neuron, x_latih.shape[1]))
        else:
            print("Inisialisasi bobot tidak tersedia")

    def __get_bias(self):
        np.random.seed(self.random_state)

        if self.init_bias == "acak":
            return np.random.uniform(-1, 1, size=self.jml_neuron)
        elif self.init_bias == "nol":
            return np.zeros(self.jml_neuron)
        else:
            print("Inisialisasi bias tidak tersedia")

    def __get_aktivasi(self, H):
        H_aktivasi = H.copy()

        if self.aktivasi == "sigmoid":
            sigmoid = lambda x: 1 / (1 + np.exp(-x))
            return sigmoid(H_aktivasi)
        elif self.aktivasi == "tanh":
            tanh = lambda x: (np.exp(x) - np.exp(-x)) / (np.exp(x) + np.exp(-x))
            return np.nan_to_num(tanh(H_aktivasi))
        elif self.aktivasi == "sine":
            sine = lambda x: np.sin(x)
            return np.nan_to_num(sine(H_aktivasi))
        elif self.aktivasi == "hardlim":
            hardlim = lambda x: np.where(x >= 0, 1, 0)
            return hardlim(H_aktivasi)
```

```
62              else:
63                  print("Fungsi aktivasi tidak tersedia")
64
65          def __get_H(self, x):
66              H = np.matmul(x, self.bobot.T) + self.bias
67
68              return self.__get_aktivasi(H)
69
70          def __get_y_onehot(self, y_awal):
71              y_onehot = np.zeros_like(y_awal, dtype="uint8")
72              for i, y in enumerate(y_awal):
73                  y_onehot[i, np.argmax(y)] = 1
74
75              return y_onehot
76
77          def __get_ciw(self, x_latih, y_latih, normalisasi):
78              jml_data_tiap_label = np.sum(y_latih, axis=0)
79              x_latih = self.__normalisasi(x_latih) if normalisasi
    else x_latih
80
81              B = np.round(self.jml_neuron * (jml_data_tiap_label /
    np.sum(jml_data_tiap_label))).astype("uint32")
82              if np.sum(B) != self.jml_neuron:
83                  B[np.argmax(B)] += self.jml_neuron - np.sum(B)
84
85              indeks_label = np.argmax(y_latih, axis=1)
86              label_unik = np.unique(indeks_label)
87
88              bobot = []
89              np.random.seed(self.random_state)
90              seed = np.random.randint(0, 1000,
    size=label_unik.shape[0])
91              for i, lbl in enumerate(label_unik):
92                  indeks_anggota = np.where(indeks_label==lbl)[0]
93                  x_anggota = x_latih[indeks_anggota]
94
95                  np.random.seed(seed[i])
96                  matriks_acak = np.random.choice([-1,1], size=(B[i],
    x_anggota.shape[0]))
97                  bobot_tiap_label = np.matmul(matriks_acak,
    x_anggota)
98                  bobot_tiap_label /=
    np.sqrt(np.sum(np.square(bobot_tiap_label), axis=0))
99                  bobot_tiap_label = np.nan_to_num(bobot_tiap_label)
100
101                 bobot = np.concatenate((bobot, bobot_tiap_label)) if
    i > 0 else bobot_tiap_label
102
103             return bobot
104
105         def __normalisasi(self, x_latih):
            return (x_latih - np.mean(x_latih, axis=0)) /
    np.std(x_latih, axis=0)
```

Penjelasan tiap baris kode program pada Kode Program 5.3 adalah:

Baris 1 : Inisialisasi *class* dengan nama `ELM`.

Baris 2-7 : Inisialisasi *constructor* yang dapat menerima beberapa parameter dengan nama `jml_neuron`, `init_bobot` (dengan nilai *default* `"ciw"`), `init_bias` (dengan nilai *default* `"acak"`), `aktivasi` (dengan nilai

default "*sigmoid*"), dan `random_state` (dengan nilai *default* `None`). Masing-masing nilai parameter pada *constructor* tersebut kemudian disimpan ke dalam *instance* variabel dengan nama yang sama.

Baris 9-19 : Inisialisasi *method* dengan nama `latih_model()`, yang dapat menerima beberapa parameter dengan nama `x_latih`, `y_latih`, dan normalisasi (dengan nilai *default* `False`). *Method* tersebut digunakan untuk melakukan pelatihan model sesuai dengan yang dijelaskan pada bab perancangan. Untuk melakukan inisialisasi bobot dan bias, program akan memanggil *private method* yang bernama `__get_bobot()` dan `__get__bias()`. Untuk mendapatkan nilai H, program akan memanggil *private method* `__get_H()`. Selanjutnya akan dihitung matriks *pseudo-inverse* dari matriks H menggunakan rumus yang sudah dijelaskan pada bab perancangan. Namun jika terjadi *error*, maka akan dihitung menggunakan bantuan *library numpy*. Terakhir, akan dihitung nilai *beta* atau bobot keluaran yang dibutuhkan.

Baris 21-25 : Inisialisasi *method* dengan nama prediksi, yang dapat menerima sebuah parameter dengan nama `x`. *Method* ini digunakan untuk melakukan prediksi label dari data yang dimasukkan, sesuai dengan perancangan yang dijelaskan pada bab sebelumnya. Nilai kembalian dari *method* ini berupa label yang sudah di konversi ke dalam bentuk One-Hot Encoding.

Baris 27-35 : Inisialisasi *private method* dengan nama `__get_bobot()`, yang dapat menerima beberapa parameter dengan nama `x_latih`, `y_latih`, dan `normalisasi`. *Method* ini digunakan untuk membangkitkan matriks bobot. Terdapat dua pilihan bobot yang dapat dibangkitkan, yaitu `"ciw"` atau `"acak"`. Nilai kembalian dari *method* ini berupa matriks bobot sesuai dengan tipe inisialisasi bobot yang dipilih. Namun jika tipe inisialisasi yang dipilih tidak tersedia, maka program akan menampilkan pesan `"Inisialisasi bobot tidak tersedia"`.

Baris 37-45 : Inisialisasi *private method* dengan nama `__get_bias()`. *Method* ini digunakan untuk membangkitkan nilai bias. Terdapat dua pilihan bobot yang dapat dibangkitkan, yaitu `"acak"` atau `"nol"`. Nilai kembalian dari *method* ini berupa nilai bias sesuai dengan tipe inisialisasi bias yang dipilih. Namun jika tipe inisialisasi yang dipilih tidak tersedia, maka program akan menampilkan pesan `"Inisialisasi bias tidak tersedia"`.

Baris 47-60 : Inisialisasi *private method* dengan nama `__get_aktivasi()`, yang dapat menerima parameter dengan nama `H`. *Method* ini digunakan untuk melakukan aktivasi terhadap nilai `H`. Terdapat empat pilihan fungsi aktivasi yang dapat dibangkitkan, yaitu `"sigmoid"`, `"tanh"`, `"sine"`, dan `"hardlim"`. Nilai kembalian dari *method* ini berupa nilai hasil aktivasi dari nilai `H` yang dimasukkan, sesuai dengan fungsi

aktivasi yang dipilih. Namun jika fungsi aktivasi yang dipilih tidak tersedia, maka program akan menampilkan pesan "`Fungsi aktivasi tidak tersedia`".

Baris 62-65 : Inisialisasi *private method* dengan nama `__get_H()`, yang dapat menerima sebuah parameter dengan nama `x`. *Method* ini digunakan untuk mendapatkan nilai keluaran *hidden neuron* `H`. Nilai `H` dihitung dengan mengalikan matriks `x` dengan *transpose* dari matriks bobot. *Method* ini akan memanggil private method `__get_aktivasi()` untuk melakukan aktivasi terhadap nilai `H` yang sudah dihitung sebelumnya, kemudian mengembalikan nilai hasil aktivasi tersebut.

Baris 67-72 : Inisialisasi *private method* dengan nama `__get_y_onehot()`, yang dapat menerima sebuah parameter dengan nama `y_awal`. *Method* ini digunakan untuk melakukan konversi label hasil prediksi ke dalam bentuk One-Hot Encoding. Pertama, akan dibangkitakan matriks nol untuk sebagai data awal untuk menyimpan representasi One-Hot Encoding dari prediksi label yang dimasukkan. Kemudian, nilai dari matriks tersebut akan diubah sesuai dengan indeks dengan nilai terbesar. Nilai kembalian dari method ini berupa label yang sudah dalam bentuk One-Hot Encoding.

Baris 74-99 : Inisialisasi *private method* dengan nama `__get_ciw()`, yang dapat menerima beberapa parameter dengan nama `x_latih`, `y_latih`, `normalisasi`. *Method* ini digunakan untuk membangkitakan bobot dengan algoritme CIW-ELM. Langkah-langkah perhitungannya sama seperti yang sudah dijelaskan pada bab perancangan. *Method* ini akan mengembalikan nilai bobot hasil perhitungan dari algoritme CIW-ELM.

Baris 101-102 : Inisialisasi *private method* dengan nama __normalisasi(), yang dapat menerima sebuah parameter dengan nama `x_latih`. *Method* ini digunakan untuk melakukan normalisasi terdahap data `x_latih` yang dimasukkan. Normalisasi pada *method* ini dilakukan menggunakan *Standard Scaler*. *Method* ini akan mengembalikan nilai berupa data hasil normalisasi dari `x_latih` yang dimasukkan.

5.5 Implementasi K-Fold Cross Validation

K-Fold Cross Validation digunakan untuk membagi data latih menjadi *k* kelompok data atau *fold*. Masing-masing *fold* tersebut kemudian digunakan sebagai data latih dan data validasi secara bergantian. Implementasi K-Fold Cross Validation dapat dilihat pada Kode Program 5.4.

Kode Program 5.4 Implementasi K-Fold Cross Validation

Baris	Kode Program
1	`class KFold:`
2	`    def __init__(self, n_fold, stratify=False, shuffle=False, random_state=None):`

```
3           self.n_fold = n_fold
4           self.stratify = stratify
5           self.shuffle = shuffle
6           self.random_state = random_state
7
8       def split(self, x_latih, y_latih):
9           fold = dict()
10
11          if self.stratify:
12              fold = self.__get_stratify(x_latih, y_latih)
13          else:
14              # Baris kode jika self.stratify = False
15
16          return fold
17
18      def __get_stratify(self, x_latih, y_latih):
19          indeks_label = np.argmax(y_latih, axis=1)
20          label_unik = np.unique(indeks_label)
21
22          fold = dict()
23          for lbl in label_unik:
24              indeks_anggota = np.where(indeks_label==lbl)[0]
25              jml_anggota_label = indeks_anggota.shape[0]
26              jml_per_fold = np.ceil(jml_anggota_label /
    self.n_fold).astype("uint32")    # selalu bulatkan keatas
27              indeks_awal, indeks_akhir = 0, jml_per_fold
28
29              for n in range(self.n_fold):
30                  try:
31                      fold[n] = np.concatenate((fold[n],
    indeks_anggota[indeks_awal:indeks_akhir]))
32                  except:
33                      fold[n] =
    indeks_anggota[indeks_awal:indeks_akhir]
34
35                  indeks_awal += jml_per_fold
36                  jml_per_fold = np.ceil((jml_anggota_label -
    indeks_awal) / (self.n_fold - n - 1)).astype("uint32")
37                  indeks_akhir += jml_per_fold
38
39          if self.shuffle:
40              np.random.seed(self.random_state)
41              for key in fold.keys():
42                  np.random.shuffle(fold[key])
43
44          return fold
```

Penjelasan tiap baris kode program pada Kode Program 5.4 adalah:

Baris 1 : Inisialisasi *class* dengan nama `KFold`.

Baris 2 : Inisialisasi *constructor* yang dapat menerima beberapa parameter dengan nama `n_fold`, `stratify` (dengan nilai *default* `False`), `shuffle` (dengan nilai *default* `False`), dan `random_state` (dengan nilai *default* `None`).

Baris 3 : Memasukkan nilai parameter `n_fold` pada *instance* variabel bernama `n_fold`.

Baris 4 : Memasukkan nilai parameter `stratify` pada *instance* variabel bernama `stratify`.

Baris 5 : Memasukkan nilai parameter `shuffle` pada *instance* variabel bernama `shuffle`.

Baris 6 : Memasukkan nilai parameter `random_state` pada *instance* variabel bernama `random_state`.

Baris 8 : Inisialisasi *method* dengan nama `split()`, yang dapat menerima dua parameter yang bernama `x_latih` dan `y_latih`.

Baris 9 : Inisialisasi variabel dengan nama `fold` yang berisi *dictionary* kosong.

Baris 11 : Seleksi kondisi jika *instance* variabel `stratify` bernilai `True`.

Baris 12 : Memanggil *private method* yang bernama `__get_stratify()`, kemudian menyimpan hasil kembaliannya ke dalam variabel `fold`.

Baris 13 : Seleksi kondisi selain kondisi yang ditentukan.

Baris 14 : Komentar yang menjelaskan isi dari baris kode (pada laporan tidak dituliskan kode program nya karena tidak digunakan).

Baris 16 : Mengembalikan nilai variabel `fold`.

Baris 18 : Inisialisasi *private method* dengan nama `__get_stratify()`, yang dapat menerima dua parameter yang bernama `x_latih` dan `y_latih`.

Baris 19 : Mencari nilai indeks maksimal dari setiap baris nilai `y_latih`, kemudian disimpan ke dalam variabel `indeks_label`.

Baris 20 : Mencari nilai unik dari variabel `indeks_label`, kemudian disimpan ke dalam variabel `label_unik`.

Baris 22 : Inisialisasi variabel dengan nama `fold` yang berisi *dictionary* kosong.

Baris 23 : Melakukan perulangan terhadap nilai pada variabel `label_unik`.

Baris 24 : Mencari indeks anggota label, kemudian menyimpan hasilnya ke dalam variabel `indeks_anggota`.

Baris 25 : Inisialisasi variabel `jml_anggota_label` dengan nilai sejumlah banyaknya baris data pada variabel `indeks_anggota`.

Baris 26 : Menghitung jumlah anggota label untuk masing-masing fold, kemudian menyimpan hasilnya ke dalam variabel `jml_per_fold`.

Baris 27 : Inisialisasi variabel dengan nama `indeks_awal` dan `indeks_akhir` dengan nilai `0` dan nilai variabel `jml_per_fold`.

Baris 29 : Melakukan perulangan sebanyak nilai *instance* variabel `n_fold`.

Baris 30 : Coba jalankan.

Baris 31 : Menggabungkan nilai variabel `fold` dengan key `n` (indeks fold), dengan nilai variabel `indeks_anggota` sesuai dengan nilai variabel `indeks_awal` dan `indeks_akhir`, kemudian menyimpan hasilnya ke dalam variabel `fold` dengan key `n`.

Baris 32 : Jika perintah sebelumnya terjadi kesalahan, maka jalankan.

Baris 33 : Inisialisasi variabel `fold` dengan key `n`, dengan nilai variabel `indeks_anggota` sesuai dengan nilai variabel `indeks_awal` dan `indeks_akhir`.

Baris 35 : Memperbarui nilai variabel `indeks_awal` dengan menambahkannya dengan nilai variabel `jml_per_fold`.

Baris 36 : Memperbarui nilai variabel berdasarkan sisa jumlah anggota label dan sisa jumlah *fold*.

Baris 37 : Memperbarui nilai variabel `indeks_akhir` dengan menambahkannya dengan nilai variabel `jml_per_fold`.

Baris 39 : Jika *instance* variabel `shuffle` bernilai `True`.

Baris 40 : Menentukan kondisi acak dari numpy berdasarkan nilai *instance* variabel `random_state`.

Baris 41 : Melakukan perulangan terhadap setiap nilai *key* pada variabel `fold`.

Baris 42 : Mengacak nilai pada variabel `fold` dengan *key* tertentu.

Baris 44 : Mengembalikan nilai variabel `fold`.

5.6 Implementasi Metrik Evaluasi

Metrik evaluasi digunakan untuk mengukur kinerja dari metode klasifikasi yang digunakan, yaitu CIW-ELM. Metrik evaluasi yang digunakan pada penelitian ini adalah akurasi dan *f-measure*. Namun, pada kode program juga dibuat metrik *precision* dan *recall* untuk memudahkan dalam perhitungan *f-measure*. Implementasi metrik evaluasi dapat dilihat pada Kode Program 5.5.

Kode Program 5.5 Implementasi metrik evaluasi

Baris	Kode Program
1	`class Metrics:`
2	`    def confusion_matrix(self, y_asli, y_pred):`
3	`        self.cm = np.zeros((y_asli.shape[1], y_asli.shape[1]), dtype="uint32")`
4	`        for i, y in enumerate(np.argmax(y_asli, axis=1)):`
5	`            self.cm[y] += y_pred[i]`
6	
7	`        return self.cm`
8	
9	`    def akurasi(self, y_asli, y_pred):`
10	`        self.confusion_matrix(y_asli, y_pred)`
	`        hasil = np.trace(self.cm) / np.sum(self.cm)     # jumlah nilai diagonal`
11	
12	`        return hasil`
13	
14	`    def precision(self, y_asli, y_pred):`
15	`        self.confusion_matrix(y_asli, y_pred)`
16	`        hasil = np.diagonal(self.cm) / np.sum(self.cm, axis=0)`
17	
18	`        return hasil`
19	
20	`    def recall(self, y_asli, y_pred):`
21	`        self.confusion_matrix(y_asli, y_pred)`

```
22          hasil = np.diagonal(self.cm) / np.sum(self.cm, axis=1)
23
24          return hasil
25
26      def f_measure(self, y_asli, y_pred):
27          prec, rec = self.precision(y_asli, y_pred),
    self.recall(y_asli, y_pred)
28          hasil = (2 * prec * rec) / (prec + rec)
29
30          return np.nan_to_num(hasil)
```

Penjelasan tiap baris kode program pada Kode Program 5.5 adalah:

Baris 1 : Inisialisasi *class* dengan nama `Metrics`.

Baris 2 : Inisialisasi *method* dengan nama `confusion_matrix()`, yang dapat menerima dua parameter yang bernama `y_asli` dan `y_pred`.

Baris 3 : Membuat matriks nol dengan ukuran sebesar jumlah kolom pada `y_asli` dikali jumlah kolom pada `y_asli`, dengan tipe data `uint32` sebagai data awal untuk menyimpan hasil *confusion matrix*, kemudian disimpan pada *instance* variabel yang bernama `cm`.

Baris 4 : Melakukan perulangan terhadap setiap nilai indeks maksimal dari setiap baris nilai `y_asli`.

Baris 5 : Memperbarui nilai *instance* variabel `cm` pada indeks ke `y` (nilai indeks maksimal dari setiap baris y_asli) dengan cara menambahkannya dengan nilai variabel `y_pred` pada indeks ke `i` (perhitungan perulangan).

Baris 7 : Mengembalikan nilai *instance* variabel `cm`.

Baris 8 : Inisialisasi *method* dengan nama `akurasi()`, yang dapat menerima dua parameter yang bernama `y_asli` dan `y_pred`.

Baris 9 : Memanggil *method* confusion_matrix().

Baris 10 : Menghitung nilai akurasi berdasarkan *instance* variabel `cm`, kemudian menyimpan hasilnya ke dalam variabel `hasil`.

Baris 12 : Mengembalikan nilai variabel `hasil`.

Baris 14 : Inisialisasi *method* dengan nama `precision()`, yang dapat menerima dua parameter yang bernama `y_asli` dan `y_pred`.

Baris 15 : Memanggil *method* `confusion_matrix()`.

Baris 16 : Menghitung nilai *precision* berdasarkan *instance* variabel `cm`, kemudian menyimpan hasilnya ke dalam variabel `hasil`.

Baris 18 : Mengembalikan nilai variabel `hasil`.

Baris 20 : Inisialisasi *method* dengan nama `recall()`, yang dapat menerima dua parameter yang bernama `y_asli` dan `y_pred`.

Baris 21 : Memanggil *method* `confusion_matrix()`.

Baris 22 : Menghitung nilai *recall* berdasarkan *instance* variabel `cm`, kemudian menyimpan hasilnya ke dalam variabel `hasil`.

Baris 24 : Mengembalikan nilai variabel `hasil`.

Baris 26 : Inisialisasi *method* dengan nama `f_measure()`, yang dapat menerima dua parameter yang bernama `y_asli` dan `y_pred`.

Baris 27 : Memanggil *method* `precision()` dan `recall()`, kemudian menyimpan nilai kembaliannya ke dalam variabel yang bernama `prec` dan `rec`.

Baris 28 : Menghitung nilai *f-measure* berdasarkan variabel `prec` dan `rec`, kemudian menyimpan hasilnya ke dalam variabel `hasil`.

Baris 30 : Mengembalikan nilai variabel `hasil`.

5.7 Implementasi Utama

Main method digunakan untuk memanggil atau menjalankan *class-class* yang telah dibuat sebelumnya. Pada *main method* ini, semua proses mulai dari pengambilan data, *label encoding*, pengujian *hyperparameter* terbaik, dan implementasi *hyperparameter* terbaik pada semua data latih dan data uji dilakukan. Method ini akan menampilkan nilai *hyperparameter* terbaik, akurasi, rata-rata *f-measure*, dan waktu komputasi pelatihan model sesuai dengan pengujian yang dilakukan. Implementasi metrik evaluasi dapat dilihat pada Kode Program 5.6.

Kode Program 5.6 Implementasi *main method*

Baris	Kode Program
1	`import pandas as pd`
2	`import numpy as np`
3	`import time`
4	
5	`def main():`
6	`    # mengambil dataset`
7	`    x_latih, x_uji, y_latih, y_uji = ambil_dataset()`
8	
9	`    # melakukan encoding terhadap label`
10	`    le = LabelEncoder()`
11	`    y_latih = le.fit_transform(y_latih)`
12	`    y_uji = le.transform(y_uji)`
13	
14	`    # membagi data latih menjadi 10 fold`
15	`    kfold = KFold(n_fold=10, shuffle=True, stratify=True)`
	`    indeks_fold = kfold.split(x_latih, y_latih)`
16	
17	`    # pengujian hyperparameter terbaik`
18	`    metrics = Metrics()`
19	`    hyperparameter = {`
20	`        "k": [1, 8, 97, 207, 339, 381, 406],`
21	`        "jml_neuron": [10, 50, 100, 200, 300, 400, 500, 600],`
22	`        "aktivasi": ["sigmoid", "tanh", "sine", "hardlim"]`
23	`    }`
24	`    hasil_keseluruhan, hasil_cv, hyperparameter_terbaik =`
25	`grid_search_cv(x_latih, y_latih, indeks_fold, hyperparameter)`

```
26      # implementasi hyperparameter terbaik pada keseluruhan data
27      pca = PCA(k=hyperparameter_terbaik["k"])
28      x_latih_pca = pca.fit_transform(x_latih)
29      x_uji_pca = pca.transform(x_uji)
30
31      elm = ELM(hyperparameter_terbaik["jml_neuron"],
32  init_bobot="ciw", init_bias="acak",
    aktivasi=hyperparameter_terbaik["aktivasi"], random_state=0)
        mulai = time.time()
33      elm.latih_model(x_latih_pca, y_latih)
34      selesai = time.time()
35      y_pred = elm.prediksi(x_uji_pca)
36
37      print(hyperparameter_terbaik)
38      print(f"Waktu Pelatihan: {round(selesai - mulai, 3)}
39  detik")
        print("Akurasi:", metrics.akurasi(y_uji, y_pred))
40      print("Avg. F-Measure:", np.average(metrics.f_measure(y_uji,
41  y_pred)))

42  if __name__=="__main__":
43      main()
44
```

Penjelasan tiap baris kode program pada Kode Program 5.6 adalah:

Baris 1 : *Import library pandas*.

Baris 2 : *Import library numpy*.

Baris 3 : *Import library time.*

Baris 5 : Inisialisasi *method* dengan nama `main()`.

Baris 6 : Komentar yang menjelaskan proses mengambil *dataset*.

Baris 7 : Memanggil *method* `ambil_dataset()`, kemudian menyimpan nilai kembaliannya ke dalam variabel `x_latih`, `x_uji`, `y_latih`, dan `y_uji`.

Baris 9 : Komentar yang menjelaskan proses *encoding* terhadap label.

Baris 10 : Membuat *object* dari *class* `LabelEncoder()` dengan nama `le`.

Baris 11 : Memanggil *method* `fit_transform()` dari *object* `le` untuk melatih sekaligus melakukan transformasi One-Hot Encoding pada `y_latih`, kemudian menyimpan hasilnya ke dalam variabel `y_latih`.

Baris 12 : Memanggil *method* `transform()` dari *object* `le` untuk melakukan transformasi One-Hot Encoding pada `y_uji`, kemudian menyimpan hasilnya ke dalam variabel `y_uji`.

Baris 14 : Komentar yang menjelaskan proses membagi data latih menjadi 10 *fold*.

Baris 15 : Membuat *object* dari class `KFold()` dengan nama `kfold`, dengan argumen `n_fold=10`, `shuffle=True`, dan `stratify=True`.

Baris 16 : Memanggil *method* `split()` dari *object* `kfold` untuk membagi data latih menjadi 10 *fold*, dengan nilai kembalian berupa indeks dari

masing-masing *fold*, kemudian menyimpan hasilnya ke dalam variabel `indeks_fold`.

Baris 18 : Komentar yang menjelaskan proses pengujian *hyperparameter* terbaik.

Baris 19 : Membuat *object* dari *class* `Metrics()` dengan nama `metrics`.

Baris 20 : Inisialisasi *dictionary* dengan nama `hyperparameter`.

Baris 21 : Inisialisasi nilai *dictionary* `hyperparameter` dengan *key* `"k"` dan *value* `[1, 8, 97, 207, 339, 381, 406]`.

Baris 22 : Inisialisasi nilai *dictionary* `hyperparameter` dengan *key* `"jml_neuron"` dan *value* `[10, 50, 100, 200, 300, 400, 500, 600]`.

Baris 23 : Inisialisasi nilai *dictionary* `hyperparameter` dengan *key* `"jml_neuron"` dan *value* `["sigmoid", "tanh", "sine", "hardlim]`.

Baris 24 : Akhir inisialisasi *dictionary* `hyperparameter`.

Baris 25 : Memanggil *method* `grid_search_cv()` untuk melakukan pengujian *hyperparameter* terbaik, kemudian menyimpan nilai keluarannya ke dalam variabel `hasil_keseluruhan`, `hasil_cv`, dan `hyperparameter_terbaik`.

Baris 27 : Komentar yang menjelaskan proses implementasi *hyperparameter* terbaik pada keseluruhan data.

Baris 28 : Membuat *object* dari *class* `PCA()` dengan nama `pca`, dengan argumen *k* berupa nilai `hyperparameter_terbaik["k"]`.

Baris 29 : Memanggil *method* `fit_transform()` dari *object* `pca` untuk melatih sekaligus melakukan reduksi dimensi dari `x_latih`, kemudian menyimpan hasilnya ke dalam variabel `x_latih_pca`.

Baris 30 : Memanggil *method* `transform()` dari object `pca` untuk melakukan reduksi dimensi dari `x_uji`, kemudian menyimpan hasilnya ke dalam variabel `x_uji_pca`.

Baris 32 : Membuat *object* dari *class* `ELM()` dengan nama `elm`, dengan argumen berupa nilai `hyperparameter_terbaik["jml_neuron"]`, `init_bobot="ciw"`, `init_bias="acak"`, dan `aktivasi= hyperparameter_terbaik["aktivasi"]`.

Baris 33 : Inisialisasi variabel yang bernama `mulai` untuk menyimpan waktu dimulai nya proses pelatihan model CIW-ELM.

Baris 34 : Memanggil *method* `latih_model()` dari *object* `elm` untuk melatih model CIW-ELM dari data `x_latih_pca` dan `y_latih`.

Baris 35 : Inisialisasi variabel yang bernama `selesai` untuk menyimpan waktu berakhir nya proses pelatihan model CIW-ELM.

Baris 36 : Memanggil *method* `prediksi()` dari *object* `elm` untuk melakukan prediksi label dari `x_uji_pca`, kemudian menyimpan hasilnya ke dalam variabel `y_pred`.

Baris 38 : Menampilkan nilai *dictionary* `hyperparameter_terbaik`.

Baris 39 : Menampilkan total waktu pelatihan model.

Baris 40 : Menghitung dan menampilkan nilai akurasi prediksi label `y_pred` terhadap `y_uji`.

Baris 41 : Menghitung dan menampilkan rata-rata nilai *f-measure* prediksi label `y_pred` terhadap `y_uji`.

Baris 42 : Jika *script* merupakan modul utama.

Baris 42 : Memanggil *method* `main()`.

BAB 6 HASIL DAN PEMBAHASAN

6.1 Hasil

Dari implementasi yang telah dilakukan, maka didapatkan hasil pengujian dari penggunaan algoritme yang diusulkan terhadap masalah klasifikasi aktivitas manusia. Pengujian tersebut dilakukan berdasarkan rancangan yang telah dibuat pada bab sebelumnya, yaitu meliputi pengujian nilai *hyperparameter* terbaik menggunakan metode Grid Search Cross Validation, serta pengujian akhir pada keseluruhan data menggunakan nilai *hyperparameter* terbaik yang telah didapatkan.

6.1.1 Hasil Pengujian *Hyperparameter*

Pada pengujian *hyperparameter* terbaik, terdapat 224 pasangan *hyperparameter* yang diuji. Hasil pengujian *hyperparameter* tersebut ditunjukkan pada Tabel 6.1.

Tabel 6.1 Hasil pengujian *hyperparameter* terbaik menggunakan Grid Search Cross Validation

No.	K	Jumlah Neuron	Fungsi Aktivasi	Skor Akhir	Rata-Rata Waktu Pelatihan (Detik)
1	1	10	*sigmoid*	0,442474	0,00838
2	1	10	*tanh*	0,425985	0,013101
3	1	10	*sine*	0,448187	0,007327
4	1	10	*hardlim*	0,230806	0,008256
5	1	50	*sigmoid*	0,314431	0,025549
…	…	…	…	…	…
125	207	600	*sigmoid*	0,985738	0,52784
…	…	…	…	…	…
220	406	500	*hardlim*	0,917095	13,37551
221	406	600	*sigmoid*	0,984714	0,584002
222	406	600	*tanh*	0,9829	0,812493
223	406	600	*sine*	0,982876	0,592092
224	406	600	*hardlim*	0,930594	19,52797

Hasil pengujian *hyperparameter* secara lengkap akan ditunjukkan pada Lampiran A. Dari hasil pengujian yang telah dilakukan, didapatkan pasangan *hyperparameter* terbaik pada pengujian ke 125 dengan nilai *k* = 207, jumlah *hidden neuron* = 600, dan fungsi aktivasi *sigmoid* dengan skor akhir sebesar 0,985738 serta rata-rata waktu pelatihan pada masing-masing *fold* selama 0,52784 detik. Untuk mengetahui pengaruh dari masing-masing *hyperparameter* yang diuji, maka dari *hasil* Grid Search Cross Validation tersebut akan diambil subset hasil dari masing-masing *hyperparameter* dengan pasangan *hyperparameter* lain dengan nilai terbaik.

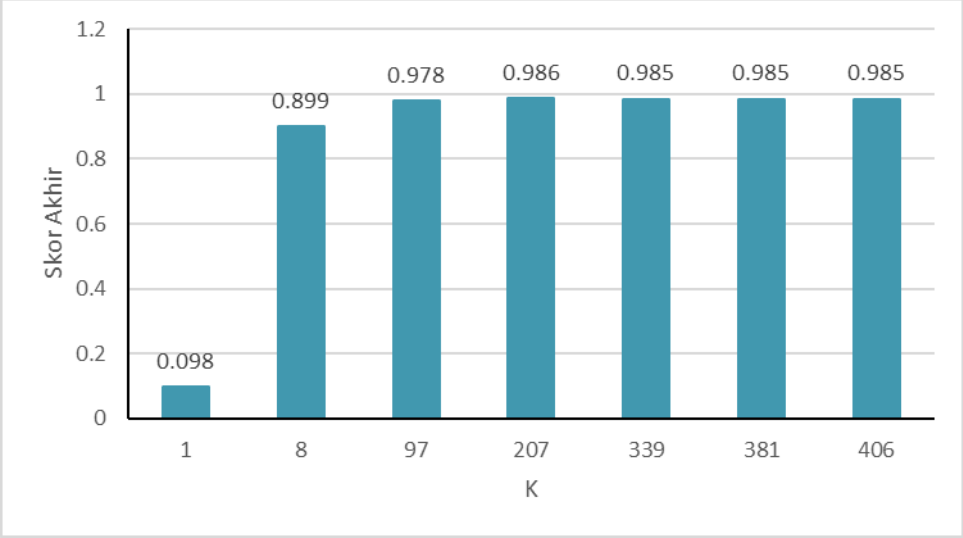
6.1.1.1 Pengaruh Pemilihan Hyperparameter K

Berdasarkan hasil pengujian sebelumnya, maka untuk menunjukkan pengaruh pemilihan nilai *k* pada pengujian *hyperparameter* terbaik, maka diambil subset dari hasil Grid Search Cross Validation dengan nilai jumlah *hidden neuron* = 600 dan fungsi aktivasi *sigmoid*. Hasil pengujian *hyperparameter k* ditunjukkan pada Tabel 6.2.

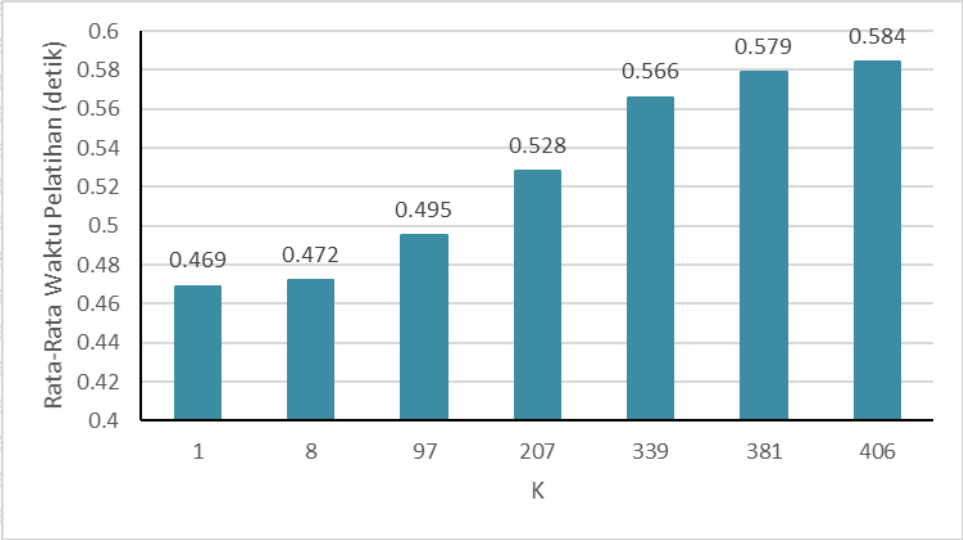
Tabel 6.2 Hasil pengujian *hyperparameter* K

K	Jumlah Neuron	Aktivasi	Skor Akhir	Rata-Rata Waktu Pelatihan (Detik)
1	600	*sigmoid*	0,098	0,469
8	600	*sigmoid*	0,899	0,472
97	600	*sigmoid*	0,978	0,495
207	600	*sigmoid*	0,986	0,528
339	600	*sigmoid*	0,985	0,566
381	600	*sigmoid*	0,985	0,579
406	600	*sigmoid*	0,985	0,584

Grafik pengaruh pemilihan *hyperparameter k* terhadap skor yang didapatkan ditunjukkan pada Gambar 6.1, sedangkan pengaruh pemilihan *hyperparameter k* terhadap waktu pelatihan ditunjukkan pada Gambar 6.2.



Gambar 6.1 Pengaruh pemilihan *hyperparameter k* terhadap skor skhir



Gambar 6.2 Pengaruh pemilihan *hyperparameter k* terhadap waktu pelatihan

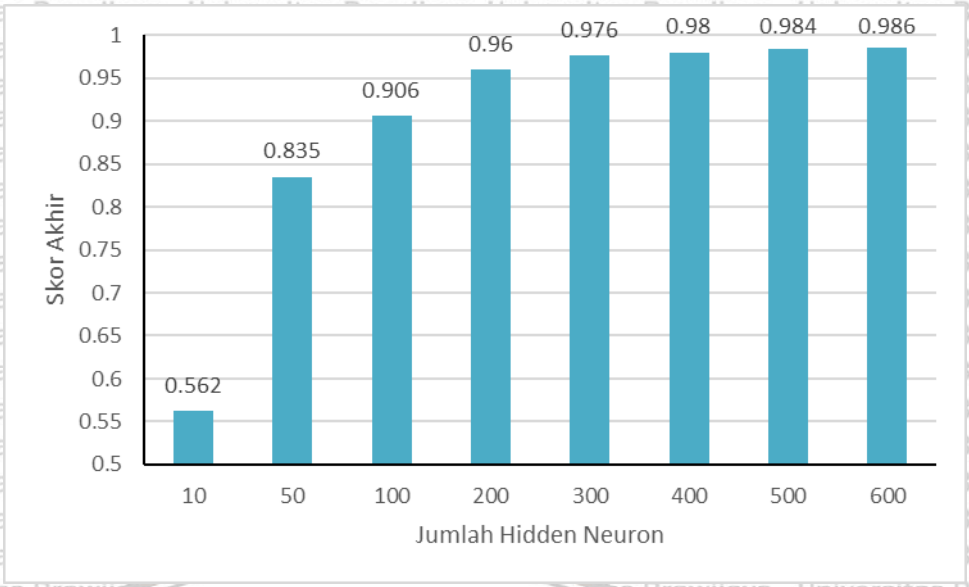
6.1.1.2 Pengaruh Pemilihan Hyperparameter Jumlah Hidden Neuron

Berdasarkan hasil pengujian sebelumnya, maka untuk menunjukkan pengaruh pemilihan nilai jumlah *hidden neuron* pada pengujian *hyperparameter* terbaik, maka diambil subset dari hasil Grid Search Cross Validation dengan nilai *k* = 207 dan fungsi aktivasi *sigmoid*. Hasil pengujian *hyperparameter* jumlah *hidden neuron* ditunjukkan pada Tabel 6.3.

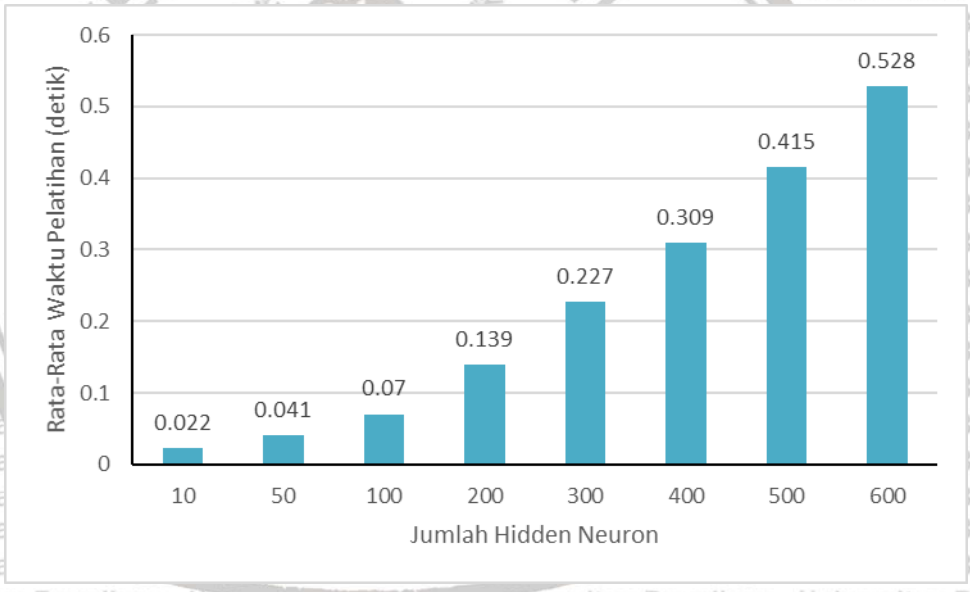
Tabel 6.3 Hasil pengujian *hyperparameter* jumlah *hidden neuron*

K	Jumlah Neuron	Aktivasi	Skor Akhir	Rata-Rata Waktu Pelatihan (Detik)
207	10	*sigmoid*	0,562	0,022
207	50	*sigmoid*	0,835	0,041
207	100	*sigmoid*	0,906	0,07
207	200	*sigmoid*	0,96	0,139
207	300	*sigmoid*	0,976	0,227
207	400	*sigmoid*	0,98	0,309
207	500	*sigmoid*	0,984	0,415
207	600	*sigmoid*	0,986	0,528

Grafik pengaruh pemilihan *hyperparameter* jumlah *hidden neuron* terhadap skor yang didapatkan ditunjukkan pada Gambar 6.3, sedangkan pengaruh pemilihan *hyperparameter* jumlah *hidden neuron* terhadap waktu pelatihan ditunjukkan pada Gambar 6.4.



Gambar 6.3 Pengaruh pemilihan *hyperparameter* jumlah *hidden neuron* terhadap skor skhir



Gambar 6.4 Pengaruh pemilihan *hyperparameter* jumlah *hidden neuron* terhadap waktu pelatihan

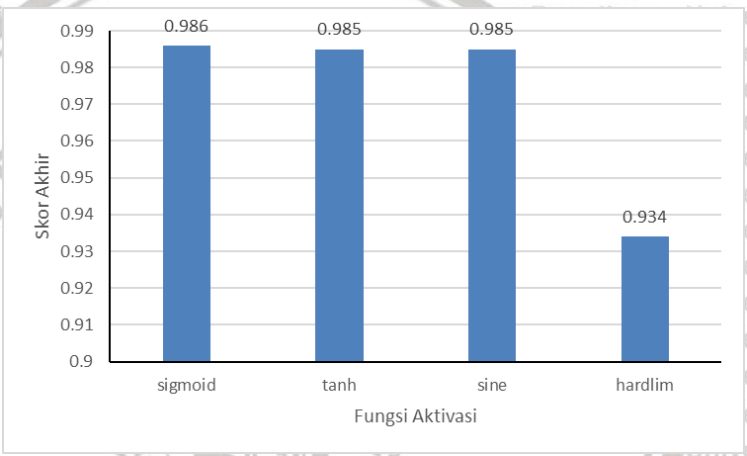
6.1.1.3 Pengaruh Pemilihan Hyperparameter Fungsi Aktivasi

Berdasarkan hasil pengujian sebelumnya, maka untuk menunjukkan pengaruh pemilihan fungsi aktivasi pada pengujian *hyperparameter* terbaik, maka diambil subset dari hasil Grid Search Cross Validation dengan nilai k = 207 dan jumlah *hidden neuron* = 600. Hasil pengujian *hyperparameter* fungsi aktivasi ditunjukkan pada Tabel 6.4.

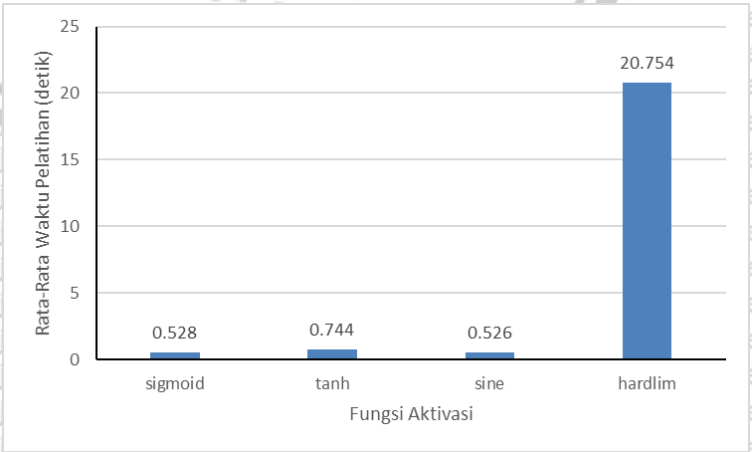
Tabel 6.4 Hasil pengujian *hyperparameter* fungsi aktivasi

K	Jumlah Neuron	Aktivasi	Skor Akhir	Rata-Rata Waktu Pelatihan (Detik)
207	600	*sigmoid*	0,986	0,528
207	600	*tanh*	0,985	0,744
207	600	*sine*	0,985	0,526
207	600	*hardlim*	0,934	20,754

Grafik Pengaruh pemilihan *hyperparameter* fungsi aktivasi terhadap skor yang didapatkan ditunjukkan pada Gambar 6.5, sedangkan pengaruh pemilihan *hyperparameter* fungsi aktivasi terhadap waktu pelatihan ditunjukkan pada Gambar 6.6.



Gambar 6.5 Pengaruh pemilihan *hyperparameter* fungsi aktivasi terhadap skor skhir



Gambar 6.6 Pengaruh pemilihan *hyperparameter* fungsi aktivasi terhadap waktu pelatihan

6.1.2 Hasil Pengujian Akhir

Hyperparameter terbaik yang didapatkan pada pengujian sebelumnya kemudian digunakan untuk menguji performa dari algoritme yang diusulkan menggunakan keseluruhan data. Hasil *confusion matrix* yang didapatkan dari pengujian akhir ditunjukkan pada Tabel 6.5.

Tabel 6.5 Hasil *confusion matrix* pengujian akhir

		Label Prediksi					
		WALKING	WALKING_UPSTAIRS	WALKING_DOWNSTAIRS	SITTING	STANDING	LAYING
Label Asli	WALKING	495	1	0	0	0	0
	WALKING_UPSTAIRS	10	461	0	0	0	0
	WALKING_DOWNSTAIRS	3	14	403	0	0	0
	SITTING	0	3	0	438	50	0
	STANDING	0	0	0	20	512	0
	LAYING	0	0	0	0	25	512

Dari hasil *confusion matrix* tersebut, dapat dihitung nilai akurasi dan *f-measure* dari hasil penggunaan algoritme PCA dan CIW-ELM pada masalah klasifikasi aktivitas manusia. Nilai akurasi yang didapatkan pada penelitian ini adalah sebesar 0,957 serta rata-rata nilai *f-measure* sebesar 0,958. Waktu komputasi yang dibutuhkan untuk melakukan pelatihan pada model CIW-ELM pada keseluruhan data latih yang terdiri dari 7352 data adalah selama 0,57 detik.

6.2 Pembahasan

Berdasarkan hasil pengujian yang telah didapatkan, langkah selanjutnya adalah melakukan analisis terhadap hasil pengujian tersebut. Pembahasan atau analisis hasil dilakukan untuk menjelaskan makna dari hasil yang diperoleh untuk menjawab pertanyaan dalam rumusan masalah. Oleh karena itu, pada pembahasan ini akan dijelaskan hasil dari implementasi algoritme yang digunakan terhadap masalah klasifikasi aktivitas manusia secara detail.

6.2.1 Pembahasan Hasil Pengujian *Hyperparameter*

Dari 224 pasangan *hyperparameter* yang telah diuji, terlihat bahwa pasangan *hyperparameter* yang memiliki skor akhir paling rendah adalah pada pasangan *hyperparameter* ketiga dengan nilai k = 1, jumlah *hidden neuron* = 10, serta fungsi aktivasi *hardlim* dengan skor akhir hanya sebesar 0,230806. Skor terbaik

didapatkan pada pasangan *hyperparameter* ke 125 dengan nilai *k* = 207, jumlah *hidden neuron* = 600, serta fungsi aktivasi *sigmoid* dengan skor akhir mencapai 0,985738.

Kemudian untuk waktu komputasi yang diperlukan untuk melatih model CIW-ELM pada masing-masing *fold*, hasil paling rendah didapatkan oleh pasangan *hyperparameter* pertama dengan nilai *k* = 1, jumlah *hidden neuron* = 10, serta fungsi aktivasi *sine* yang hanya memiliki rata-rata waktu pelatihan selama 0,007327 detik. Pasangan *hyperparameter* yang memerlukan waktu pelatihan paling tinggi adalah pasangan *hyperparameter* ke 191 dengan nilai *k* = 381, jumlah *hidden neuron* = 600, serta fungsi aktivasi *tanh* dengan rata-rata waktu pelatihan selama 21,35905 detik.

Dari hasil tersebut, terlihat bahwa tidak adanya korelasi antara skor akhir yang didapatkan dengan rata-rata waktu pelatihan yang dibutuhkan. Pernyataan tersebut dibuktikan dengan hasil skor akhir tertinggi yang didapatkan pada pasangan *hyperparameter* ke 125 yang memiliki waktu pelatihan yang lebih rendah dibandingkan dengan pasangan *hyperparameter* ke 191. Selain itu, pasangan *hyperparameter* ketiga yang memiliki rata-rata waktu pelatihan paling rendah justru memiliki skor akhir yang lebih tinggi daripada pasangan *hyperparameter* keempat.

6.2.1.1 Pembahasan Pengaruh Pemilihan Hyperparameter K

Pada pengujian pengaruh nilai *k* pada algoritme PCA yang ditunjukkan sebelumnya, terlihat bahwa skor akhir yang didapatkan cenderung meningkat sampai pada nilai *k* = 207, dan selebihnya mulai stabil bahkan cenderung menurun. Kemudian untuk waktu pelatihan modelnya, terlihat bahwa rata-rata waktu pelatihan yang diperlukan semakin bertambah seiring dengan bertambahnya nilai *k* yang digunakan. Hal tersebut menunjukkan bahwa besarnya nilai *k* yang digunakan berbanding lurus dengan waktu pelatihan yang diperlukan, meskipun tidak terlalu signifikan. Berdasarkan hasil tersebut, maka didapatkan nilai *k* paling optimal dari semua nilai yang diuji yaitu sebesar 207, dengan skor akhir yang paling baik dan waktu komputasi yang tidak terlalu tinggi.

6.2.1.2 Pembahasan Pengaruh Pemilihan Hyperparameter Jumlah *Hidden Neuron*

Selanjutnya pada pengujian *hyperparameter* jumlah *hidden neuron* pada algoritme CIW-ELM, skor akhir yang didapatkan cenderung mengalami kenaikan seiring dengan jumlah *hidden neuron* yang digunakan. Sementara itu, waktu pelatihan yang diperlukan mengalami peningkatan yang cukup signifikan terhadap penambahan jumlah *hidden neuron* yang digunakan. Meskipun begitu, waktu pelatihan tersebut bisa dibilang masih sangat cepat karena kurang dari satu detik, sehingga sangat layak untuk digunakan. Terlebih lagi, skor akhir yang didapatkan juga terbilang sangat tinggi jika dilihat dari waktu komputasi yang diperlukan. Oleh karena itu, *hidden neuron* dengan jumlah 600 sangat cocok digunakan pada penelitian ini.

6.2.1.3 Pembahasan Pengaruh Pemilihan Hyperparameter Fungsi Aktivasi

Hyperparameter terakhir yang diuji pada penelitian ini adalah empat fungsi aktivasi yang pernah digunakan pada penelitian sebelumnya. Untuk skor akhir yang didapatkan, fungsi aktivasi *sigmoid* terlihat lebih unggul dibanding fungsi aktivasi lain, meskipun perbedaan skor tersebut tidak terlalu jauh. Untuk waktu pelatihan yang dibutuhkan, fungsi aktivasi *hardlim* memiliki rata-rata waktu pelatihan yang jauh lebih tinggi daripada fungsi aktivasi lain. Oleh karena skor akhir yang didapatkan oleh fungsi aktivasi *sigmoid* paling unggul dari yang lain, serta rata-rata waktu pelatihan yang tidak terlalu berbeda jauh dengan fungsi aktivasi *sine*, maka fungsi aktivasi *sigmoid* masih menjadi pilihan terbaik untuk penelitian ini.

6.2.2 Pembahasan Hasil Pengujian Akhir

Setelah dilakukan pengujian *hyperparameter* terbaik, didapatkan nilai *hyperparameter* yang paling optimal untuk kedua algoritme yang digunakan pada penelitian ini, yaitu *k* = 207, jumlah *hidden neuron* = 600, dan fungsi aktivasi *sigmoid*. Berdasarkan hasil pengujian akhir yang telah ditunjukkan sebelumnya, terlihat pada *confusion matrix* bahwa model CIW-ELM berhasil melakukan klasifikasi terhadap aktivitas manusia dengan sangat baik. Aktivitas yang paling banyak di prediksi dengan benar adalah *WALKING*, dengan prediksi salah hanya berjumlah satu pada label *WALKING_UPSTAIRS*. Aktivitas yang paling banyak diprediksi dengan salah adalah *SITTING*, yang di klasifikasikan oleh model sebagai *WALKING_UPSTAIRS* sebanyak tiga kali, dan diprediksi sebagai *STANDING* sebanyak 50 kali. Secara keseluruhan, model CIW-ELM berhasil melakukan klasifikasi pada setiap data uji dengan benar. Hal tersebut terbukti dengan hasil akurasi yang didapatkan dari 2947 data uji mencapai 0,956 atau 95,6%, serta rata-rata *f-measure* sebesar 0,957 atau 95,7%. Apalagi, hasil tersebut didapatkan dengan waktu pelatihan model yang sangat cepat, yaitu hanya 0,57 detik.

Hasil tersebut tentu tidak lepas dari peran algoritme PCA yang mampu mereduksi dimensi fitur dari data yang digunakan, dari yang semula 561 hingga menjadi 207. Selain berguna untuk mereduksi dimensi sehingga menghasilkan fitur yang paling relevan, fitur dengan jumlah yang lebih sedikit juga dapat mempercepat sekaligus menyederhanakan komputasi yang dilakukan. Selain itu, pemilihan *hyperparameter* jumlah *hidden neuron* dan fungsi aktivasi yang dilakukan terhadap algoritme CIW-ELM juga membantu menghasilkan model yang paling optimal. Oleh karena itu, dapat dikatakan bahwa algoritme PCA dan CIW-ELM sangat cocok digunakan secara bersama-sama untuk menyelesaikan permasalahan klasifikasi aktivitas manusia. Algoritme tersebut juga terbukti dapat menghasilkan akurasi yang sangat baik dengan waktu komputasi yang sangat cepat pada masalah klasifikasi aktivitas manusia.

BAB 7 PENUTUP

7.1 Kesimpulan

Berdasarkan hasil penelitian klasifikasi aktivitas manusia menggunakan algoritme Computed Input Weight Extreme Learning Machine dengan reduksi dimensi Principal Component Analysis yang telah dilakukan, didapatkan kesimpulan untuk menjawab rumusan masalah penelitian sebagai berikut:

1. Penggunaan algoritme PCA untuk melakukan reduksi dimensi pada data penelitian yang digunakan memberikan pengaruh positif terhadap hasil klasifikasi dari algoritme CIW-ELM. Nilai *k* paling optimal yang berhasil didapatkan adalah sebesar 207, namunh selain nilai tersebut cenderung menghasilkan rata-rata akurasi dan *f-measure* yang lebih rendah. Waktu pelatihan model yang dibutuhkan juga terlihat berbanding lurus dengan seiring bertambahnya nilai k, meskipun tidak terlalu signifikan.
2. Pemilihan *hyperparameter* jumlah *hidden neuron* pada algoritme CIW-ELM sangat berpengaruh terhadap rata-rata hasil akurasi dan *f-measure* serta waktu komputasi yang dibutuhkan. Semakin banyak jumlah *hidden neuron* yang digunakan, maka rata-rata hasil akurasi dan *f-measure* yang didapatkan akan semakin tinggi, namun waktu komputasi yang dibutuhkan juga semakin lama. Meskipun demikian, waktu komputasi tersebut masih terbilang sangat cepat, yaitu kurang dari satu detik pada seluruh nilai *hidden neuron* yang diujikan pada penelitian ini. Oleh karena itu, penggunaan *hidden neuron* paling tinggi sebesar 600 tetap menjadi yang paling optimal. Selain itu, pemilihan *hyperparameter* fungsi aktivasi pada algoritme CIW-ELM juga berpengaruh terhadap rata-rata hasil akurasi dan *f-measure*, serta waktu komputasi yang dibutuhkan. Untuk hasil klasifikasi yang didapatkan, fungsi aktivasi *sigmoid* cenderung lebih unggul dibandingkan dengan fungsi aktivasi lain. Untuk waktu komputasi, fungsi aktivasi *sine* sedikit lebih unggul daripada fungsi aktivasi *sigmoid*. Namun karena hasil klasifikasi yang lebih tinggi dan perbedaan waktu komputasi yang tidak terlalu jauh, maka fungsi aktivasi *sigmoid* tetap menjadi yang paling optimal untuk digunakan pada penelitian ini.
3. Hasil akurasi yang didapatkan pada penelitian ini adalah sebesar 0,957 atau 95,7%, serta rata-rata *f-measure* sebesar 0,958 atau 95,8%. Waktu komputasi yang diperlukan untuk melakukan pelatihan model CIW-ELM hanya 0,57 detik. Hal inilah yang menjadi keunggulan dari algoritme CIW-ELM yang merupakan pengembangan dari algoritme ELM konvensional, yaitu waktu komputasinya yang sangat cepat serta hasil klasifikasinya yang sangat baik. Oleh karena itu, algoritme ini sangat cocok digunakan untuk melakukan klasifikasi aktivitas manusia, apalagi saat digunakan bersama algoritme PCA seperti yang dilakukan pada penelitian ini.

7.2 Saran

Berdasarkan hasil penelitian yang telah dilakukan, terdapat beberapa saran untuk penelitian selanjutnya yang berkaitan dengan topik penelitian ini, yaitu sebagai berikut:

1. Untuk mengurangi dimensi dari fitur yang ada pada *dataset*, pada penelitian ini menggunakan algoritme Principal Component Analysis (PCA). Meskipun PCA terbukti dapat mengurangi dimensi fitur data dari yang semula 561 menjadi 207, namun jumlah dimensi fitur tersebut masih tergolong tinggi. Untuk penelitian selanjutnya dapat menggunakan algoritme lain yang dapat mengurangi dimensi fitur menjadi lebih rendah.
2. Pada algoritme CIW-ELM masih terdapat ketidakpastian terhadap hasil klasifikasi yang dilakukan karena inisialisasi *random matrix* pada inisialisasi bobot, serta inisialisasi bias yang dilakukan secara acak. Oleh karena itu, dapat dilakukan optimasi pada proses tersebut untuk menghasilkan model klasifikasi yang lebih baik.
3. Untuk penelitian selanjutnya dapat mengimplementasikan metode yang diusulkan pada *dataset* lain selain *Human Activity Recognition Using Smartphone Dataset*.

DAFTAR REFERENSI

Aggarwal, J. K., & Ryoo, M. S. (2011). Human activity analysis: A review. *ACM Computing Surveys*, *43*(3). https://doi.org/10.1145/1922649.1922653

Ahmad, I., Basheri, M., Iqbal, M. J., & Rahim, A. (2018). Performance Comparison of Support Vector Machine, Random Forest, and Extreme Learning Machine for Intrusion Detection. *IEEE Access*, *6*(c), 33789–33795. https://doi.org/10.1109/ACCESS.2018.2841987

Anguita, D., Ghio, A., Oneto, L., Parra, X., & Reyes-Ortiz, J. L. (2013). A Public Domain *Dataset* for Human Activity Recognition Using Smartphone. *ESANN 2013 Proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 24-26 April 2013, I6doc.Com*, *April*, 437–442. https://doi.org/10.1109/SmartWorld-UIC-ATC-SCALCOM-IOP-SCI.2019.00071

Bayat, A., Pomplun, M., & Tran, D. A. (2014). A study on human activity recognition using accelerometer data from smartphones. *Procedia Computer Science*, *34*, 450–457. https://doi.org/10.1016/j.procs.2014.07.009

Berrar, D. (2018). Cross-validation. *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*, *1–3*(April), 542–545. https://doi.org/10.1016/B978-0-12-809633-8.20349-X

Bevilacqua, A., MacDonald, K., Rangarej, A., Widjaya, V., Caulfield, B., & Kechadi, T. (2019). Human activity recognition with convolutional neural networks. In *arXiv* (Vol. 1). Springer International Publishing. https://doi.org/10.1007/978-3-030-10997-4

Doewes, A., Swasono, S. E., & Harjito, B. (2017). Feature selection on Human Activity Recognition *dataset* using Minimum Redundancy Maximum Relevance. *2017 IEEE International Conference on Consumer Electronics - Taiwan, ICCE-TW 2017*, *1*, 171–172. https://doi.org/10.1109/ICCE-China.2017.7991050

Feng, J., & Lu, S. (2019). Performance Analysis of Various Activation Functions in Artificial Neural Networks. *Journal of Physics: Conference Series*, *1237*(2). https://doi.org/10.1088/1742-6596/1237/2/022030

Garcia-Gonzalez, D., Rivero, D., Fernandez-Blanco, E., & Luaces, M. R. (2020). A public domain *dataset* for real-life human activity recognition using smartphone sensors. *Sensors (Switzerland)*, *20*(8). https://doi.org/10.3390/s20082200

Hernandez, F., Suarez, L. F., Villamizar, J., & Altuve, M. (2019). *Human Activity Recognition on Smartphones Using a Bidirectional LSTM Network*.

Hossin, M., & Sulaiman, M. . (2015). A Review on Evaluation Metrics for Data

Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process*, *5*(2), 01–11. https://doi.org/10.5121/ijdkp.2015.5201

Huang, G. Bin, Zhu, Q. Y., & Siew, C. K. (2004). Extreme Learning Machine: A new learning scheme of feedforward neural networks. *IEEE International Conference on Neural Networks - Conference Proceedings*, *2*(February 2014), 985–990. https://doi.org/10.1109/IJCNN.2004.1380068

Jegham, I., Ben Khalifa, A., Alouani, I., & Mahjoub, M. A. (2020). Vision-based human action recognition: An overview and real world challenges. *Forensic Science International: Digital Investigation*, *32*, 200901. https://doi.org/10.1016/j.fsidi.2019.200901

Jollife, I. T., & Cadima, J. (2016). Principal Component Analysis: A review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *374*(2065). https://doi.org/10.1098/rsta.2015.0202

Junita, V., & Bachtiar, F. A. (2019). *Klasifikasi Aktivitas Manusia menggunakan Algoritme Decision Tree C4 . 5 dan Information Gain untuk Seleksi Fitur*. *3*(10), 9426–9433.

Kesavaraj, G., & Sukumaran, S. (2013). *A Study On Classification Techniques in Data Mining*.

Khan, A. M., Lee, Y. K., Lee, S. Y., & Kim, T. S. (2010). A triaxial accelerometer-based physical-activity recognition via augmented-signal features and a hierarchical recognizer. *IEEE Transactions on Information Technology in Biomedicine*, *14*(5), 1166–1172. https://doi.org/10.1109/TITB.2010.2051955

Kishore, S., Bhattacharjee, S., & Swetapadma, A. (2017). *A Hybrid Method for Activity Monitoring Using* Principal Component Analysis *and Back-Propagation Neural Network*. 885–889. https://ci.nii.ac.jp/naid/40021243259/

Kohavi, R. (1993). *A Study of Cross-Validation and Bootstrap for Accuracy Estimation and M o d e l Selection*. 1137–1143.

Krstajic, D., Buturovic, L. J., Leahy, D. E., & Thomas, S. (2014). Cross-validation pitfalls when selecting and assessing regression and classification models. *Journal of Cheminformatics*, *6*(1), 1–15. https://doi.org/10.1186/1758-2946-6-10

Micucci, D., Mobilio, M., & Napoletano, P. (2017). UniMiB SHAR: A *dataset* for human activity recognition using acceleration data from smartphones. *Applied Sciences (Switzerland)*, *7*(10). https://doi.org/10.3390/app7101101

Osisanwo, F. ., Akinsola, J. E. ., Awodele, O., Hinmikaiye, J. O., Olakanmi, O., & Akinjobi, J. (2017). Supervised Machine Learning Algorithms: Classification and Comparison. *International Journal of Computer Trends and Technology*, *48*(3), 128–138. https://doi.org/10.14445/22312803/ijctt-v48p126

Potdar, K., S., T., & D., C. (2017). A Comparative Study of Categorical Variable Encoding Techniques for Neural Network Classifiers. *International Journal of Computer Applications*, *175*(4), 7–9. https://doi.org/10.5120/ijca2017915495

Reiss, A., Hendeby, G., & Stricker, D. (2013). A competitive approach for human activity recognition on smartphones. *ESANN 2013 Proceedings, 21st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, *April*, 455–460.

Sevin, E. (2018). *Activation Functions in Single Hidden Layer Feed-forward Neural Networks*. *October*.

Shekar, B. H., & Dagnew, G. (2019). Grid search-based hyperparameter tuning and classification of microarray cancer data. *2019 2nd International Conference on Advanced Computational and Communication Paradigms, ICACCP 2019*, 1–8. https://doi.org/10.1109/ICACCP.2019.8882943

Soofi, A., & Awan, A. (2017). Classification Techniques in Machine Learning: Applications and Issues. *Journal of Basic & Applied Sciences*, *13*(September), 459–465. https://doi.org/10.6000/1927-5129.2017.13.76

Su, X., Tong, H., & Ji, P. (2014). Activity recognition with smartphone sensors. *Tsinghua Science and Technology*, *19*(3), 235–249. https://doi.org/10.1109/TST.2014.6838194

Tapson, J., de Chazal, P., & van Schaik, A. (2015). *Explicit Computation of Input Weights in* Extreme Learning Machine*s*. *June*, 41–49. https://doi.org/10.1007/978-3-319-14063-6

Van Der Maaten, L. J. P., Postma, E. O., & Van Den Herik, H. J. (2009). Dimensionality Reduction: A Comparative Review. *Journal of Machine Learning Research*, *10*(January), 1–41. https://doi.org/10.1080/13506280444000102

Vaughn, A., Biocco, P., Liu, Y., & Anwar, M. (2018). Activity detection and analysis using smartphone sensors. *Proceedings - 2018 IEEE 19th International Conference on Information Reuse and Integration for Data Science, IRI 2018*, 102–107. https://doi.org/10.1109/IRI.2018.00022

Wang, A., Chen, G., Yang, J., Zhao, S., & Chang, C. (2016). *A Comparative Study on Human Activity Recognition Using Inertial Sensors in a Smartphone*. *c*. https://doi.org/10.1109/JSEN.2016.2545708

Wang, B., Chen, Q., Wang, Z., & Hu, Y. (2019). The Rearch on Improved LVQ Neural Network Method. *2019 3rd International Conference on Circuits, System and Simulation, ICCSS 2019*, 206–209. https://doi.org/10.1109/CIRSYSSIM.2019.8935608

Xie, H., Li, J., & Xue, H. (2017). A survey of dimensionality reduction techniques based on random projection. *ArXiv*, 1–10.

Zhang, M., & Sawchuk, A. A. (2012). A feature selection-based framework for human activity recognition using wearable multimodal sensors. *BODYNETS 2011 - 6th International ICST Conference on Body Area Networks*, *January 2011*, 92–98. https://doi.org/10.4108/icst.bodynets.2011.247018

LAMPIRAN A HASIL PENGUJIAN *HYPERPARAMETER*

No.	K	Jumlah Neuron	Aktivasi	Skor Akhir	Rata-Rata Waktu Pelatihan (detik)
1	1	10	*sigmoid*	0.442474	0.00838
2	1	10	*tanh*	0.425985	0.013101
3	1	10	*sine*	0.448187	0.007327
4	1	10	*hardlim*	0.230806	0.008256
5	1	50	*sigmoid*	0.314431	0.025549
6	1	50	*tanh*	0.282332	0.056795
7	1	50	*sine*	0.144926	0.025552
8	1	50	*hardlim*	0.426666	0.099486
9	1	100	*sigmoid*	0.184503	0.050415
10	1	100	*tanh*	0.213817	0.111731
11	1	100	*sine*	0.124493	0.049522
12	1	100	*hardlim*	0.448498	0.265581
13	1	200	*sigmoid*	0.164491	0.111214
14	1	200	*tanh*	0.127111	0.209747
15	1	200	*sine*	0.148275	0.112332
16	1	200	*hardlim*	0.459999	1.017825
17	1	300	*sigmoid*	0.113397	0.188992
18	1	300	*tanh*	0.071637	0.310814
19	1	300	*sine*	0.116526	0.1857
20	1	300	*hardlim*	0.464713	2.988996
21	1	400	*sigmoid*	0.191401	0.269963
22	1	400	*tanh*	0.111138	0.422852
23	1	400	*sine*	0.104001	0.264561
24	1	400	*hardlim*	0.459906	7.947517
25	1	500	*sigmoid*	0.144386	0.368205
26	1	500	*tanh*	0.10846	0.552717
27	1	500	*sine*	0.113795	0.364621
28	1	500	*hardlim*	0.456118	13.19926

29	1	600	*sigmoid*	0.098397	0.468597
30	1	600	*tanh*	0.074188	0.689694
31	1	600	*sine*	0.160485	0.463851
32	1	600	*hardlim*	0.450886	20.08291
33	8	10	*sigmoid*	0.685733	0.007816
34	8	10	*tanh*	0.63611	0.013963
35	8	10	*sine*	0.439713	0.008446
36	8	10	*hardlim*	0.509868	0.006629
37	8	50	*sigmoid*	0.860681	0.024424
38	8	50	*tanh*	0.850772	0.055512
39	8	50	*sine*	0.850586	0.027188
40	8	50	*hardlim*	0.784615	0.077873
41	8	100	*sigmoid*	0.872126	0.051802
42	8	100	*tanh*	0.870961	0.11105
43	8	100	*sine*	0.872246	0.053825
44	8	100	*hardlim*	0.825934	0.202705
45	8	200	*sigmoid*	0.890426	0.111948
46	8	200	*tanh*	0.887103	0.208204
47	8	200	*sine*	0.887453	0.109938
48	8	200	*hardlim*	0.860865	1.142749
49	8	300	*sigmoid*	0.894189	0.18362
50	8	300	*tanh*	0.891305	0.314532
51	8	300	*sine*	0.89323	0.183733
52	8	300	*hardlim*	0.865775	3.030867
53	8	400	*sigmoid*	0.895748	0.268164
54	8	400	*tanh*	0.896904	0.420503
55	8	400	*sine*	0.894395	0.270728
56	8	400	*hardlim*	0.870759	8.124151
57	8	500	*sigmoid*	0.899113	0.367903
58	8	500	*tanh*	0.899334	0.550899
59	8	500	*sine*	0.895133	0.36639

60	8	500	*hardlim*	0.869982	13.04331
61	8	600	*sigmoid*	0.899081	0.472126
62	8	600	*tanh*	0.900657	0.689139
63	8	600	*sine*	0.897777	0.466994
64	8	600	*hardlim*	0.877473	19.02015
65	97	10	*sigmoid*	0.566414	0.014351
66	97	10	*tanh*	0.551487	0.019681
67	97	10	*sine*	0.204448	0.015762
68	97	10	*hardlim*	0.478552	0.012459
69	97	50	*sigmoid*	0.857237	0.033072
70	97	50	*tanh*	0.816874	0.064038
71	97	50	*sine*	0.737355	0.035885
72	97	50	*hardlim*	0.727429	0.089124
73	97	100	*sigmoid*	0.928403	0.059618
74	97	100	*tanh*	0.90051	0.119322
75	97	100	*sine*	0.880273	0.061743
76	97	100	*hardlim*	0.817784	0.200556
77	97	200	*sigmoid*	0.960883	0.121675
78	97	200	*tanh*	0.950335	0.223092
79	97	200	*sine*	0.94459	0.124689
80	97	200	*hardlim*	0.875164	0.985025
81	97	300	*sigmoid*	0.970876	0.202955
82	97	300	*tanh*	0.966226	0.328381
83	97	300	*sine*	0.966267	0.204228
84	97	300	*hardlim*	0.904766	2.758236
85	97	400	*sigmoid*	0.973013	0.285876
86	97	400	*tanh*	0.971553	0.440745
87	97	400	*sine*	0.971705	0.285331
88	97	400	*hardlim*	0.917992	7.495792
89	97	500	*sigmoid*	0.976508	0.390158
90	97	500	*tanh*	0.97633	0.572284

91	97	500	*sine*	0.977127	0.391329
92	97	500	*hardlim*	0.928599	12.17656
93	97	600	*sigmoid*	0.978134	0.495073
94	97	600	*tanh*	0.978998	0.714708
95	97	600	*sine*	0.977632	0.492318
96	97	600	*hardlim*	0.933701	18.22161
97	207	10	*sigmoid*	0.561921	0.021966
98	207	10	*tanh*	0.546158	0.02795
99	207	10	*sine*	0.189677	0.022146
100	207	10	*hardlim*	0.478882	0.020105
101	207	50	*sigmoid*	0.834607	0.040805
102	207	50	*tanh*	0.793697	0.072332
103	207	50	*sine*	0.724067	0.044829
104	207	50	*hardlim*	0.714506	0.096229
105	207	100	*sigmoid*	0.905577	0.070057
106	207	100	*tanh*	0.877241	0.129829
107	207	100	*sine*	0.855974	0.072318
108	207	100	*hardlim*	0.799463	0.217098
109	207	200	*sigmoid*	0.960025	0.139438
110	207	200	*tanh*	0.943385	0.2401
111	207	200	*sine*	0.938599	0.139479
112	207	200	*hardlim*	0.86287	0.930912
113	207	300	*sigmoid*	0.975786	0.226718
114	207	300	*tanh*	0.962844	0.349435
115	207	300	*sine*	0.963315	0.225643
116	207	300	*hardlim*	0.893248	3.538555
117	207	400	*sigmoid*	0.979861	0.308986
118	207	400	*tanh*	0.976168	0.466248
119	207	400	*sine*	0.975162	0.309267
120	207	400	*hardlim*	0.916074	7.719743
121	207	500	*sigmoid*	0.984341	0.414579

122	207	500	*tanh*	0.981223	0.606474
123	207	500	*sine*	0.981295	0.414909
124	207	500	*hardlim*	0.923373	12.17784
125	207	600	*sigmoid*	0.985738	0.52784
126	207	600	*tanh*	0.984519	0.744213
127	207	600	*sine*	0.984521	0.52564
128	207	600	*hardlim*	0.933812	20.75419
129	339	10	*sigmoid*	0.557984	0.029657
130	339	10	*tanh*	0.541249	0.034792
131	339	10	*sine*	0.188848	0.028769
132	339	10	*hardlim*	0.479342	0.026465
133	339	50	*sigmoid*	0.829318	0.053229
134	339	50	*tanh*	0.792855	0.08229
135	339	50	*sine*	0.722293	0.055921
136	339	50	*hardlim*	0.711978	0.105094
137	339	100	*sigmoid*	0.901788	0.080328
138	339	100	*tanh*	0.871796	0.142005
139	339	100	*sine*	0.852264	0.084512
140	339	100	*hardlim*	0.797419	0.228882
141	339	200	*sigmoid*	0.954865	0.155608
142	339	200	*tanh*	0.939294	0.253629
143	339	200	*sine*	0.934084	0.157931
144	339	200	*hardlim*	0.857595	0.935056
145	339	300	*sigmoid*	0.973811	0.243561
146	339	300	*tanh*	0.96215	0.371343
147	339	300	*sine*	0.959402	0.252495
148	339	300	*hardlim*	0.887912	3.369552
149	339	400	*sigmoid*	0.978611	0.338443
150	339	400	*tanh*	0.972451	0.494922
151	339	400	*sine*	0.971953	0.340842
152	339	400	*hardlim*	0.912574	8.250242

153	339	500	*sigmoid*	0.983103	0.453593
154	339	500	*tanh*	0.979036	0.634446
155	339	500	*sine*	0.978446	0.455323
156	339	500	*hardlim*	0.919987	14.0962
157	339	600	*sigmoid*	0.984565	0.565704
158	339	600	*tanh*	0.982757	0.789011
159	339	600	*sine*	0.982843	0.567584
160	339	600	*hardlim*	0.931238	20.89495
161	381	10	*sigmoid*	0.557784	0.032405
162	381	10	*tanh*	0.541988	0.03693
163	381	10	*sine*	0.189191	0.033082
164	381	10	*hardlim*	0.480552	0.030288
165	381	50	*sigmoid*	0.828545	0.054226
166	381	50	*tanh*	0.792276	0.085045
167	381	50	*sine*	0.721402	0.057545
168	381	50	*hardlim*	0.711681	0.110172
169	381	100	*sigmoid*	0.902014	0.086339
170	381	100	*tanh*	0.869647	0.146196
171	381	100	*sine*	0.851623	0.089285
172	381	100	*hardlim*	0.795405	0.235605
173	381	200	*sigmoid*	0.954585	0.160207
174	381	200	*tanh*	0.938352	0.262197
175	381	200	*sine*	0.934642	0.161955
176	381	200	*hardlim*	0.856483	0.941451
177	381	300	*sigmoid*	0.97345	0.251764
178	381	300	*tanh*	0.961648	0.377515
179	381	300	*sine*	0.95918	0.258092
180	381	300	*hardlim*	0.891085	3.322443
181	381	400	*sigmoid*	0.978935	0.35002
182	381	400	*tanh*	0.971988	0.506712
183	381	400	*sine*	0.9718	0.354125

184	381	400	*hardlim*	0.912703	8.608058
185	381	500	*sigmoid*	0.982611	0.467379
186	381	500	*tanh*	0.979561	0.65028
187	381	500	*sine*	0.979068	0.466368
188	381	500	*hardlim*	0.918448	14.58788
189	381	600	*sigmoid*	0.984697	0.579102
190	381	600	*tanh*	0.982774	0.798055
191	381	600	*sine*	0.98313	0.583734
192	381	600	*hardlim*	0.929774	21.35905
193	406	10	*sigmoid*	0.557884	0.032587
194	406	10	*tanh*	0.540529	0.037767
195	406	10	*sine*	0.190104	0.034152
196	406	10	*hardlim*	0.479136	0.031714
197	406	50	*sigmoid*	0.8282	0.058286
198	406	50	*tanh*	0.793301	0.086557
199	406	50	*sine*	0.721938	0.059284
200	406	50	*hardlim*	0.713612	0.112875
201	406	100	*sigmoid*	0.901606	0.087939
202	406	100	*tanh*	0.870172	0.147704
203	406	100	*sine*	0.851596	0.090577
204	406	100	*hardlim*	0.795801	0.230063
205	406	200	*sigmoid*	0.954352	0.163805
206	406	200	*tanh*	0.938049	0.268035
207	406	200	*sine*	0.93446	0.165761
208	406	200	*hardlim*	0.856558	1.08572
209	406	300	*sigmoid*	0.973768	0.256509
210	406	300	*tanh*	0.961097	0.385524
211	406	300	*sine*	0.959542	0.265534
212	406	300	*hardlim*	0.890489	3.440989
213	406	400	*sigmoid*	0.978336	0.356254
214	406	400	*tanh*	0.972591	0.513957
215	406	400	*sine*	0.971818	0.356319
216	406	400	*hardlim*	0.911701	8.187266

217	406	500	*sigmoid*	0.983242	0.474916
218	406	500	*tanh*	0.979305	0.657928
219	406	500	*sine*	0.978902	0.478631
220	406	500	*hardlim*	0.917095	13.37551
221	406	600	*sigmoid*	0.984714	0.584002
222	406	600	*tanh*	0.9829	0.812493
223	406	600	*sine*	0.982876	0.592092
224	406	600	*hardlim*	0.930594	19.52797