TUGAS AKHIR - KM184801

ALGORITMA HYBRID DIFFERENTIAL EVOLUTION - SUPPORT VECTOR MACHINE (DE-SVM) UNTUK SISTEM DETEKSI INTRUSI JARINGAN

YOHANES A CRUX GOSAL
NRP. 06111640000001

Dosen Pembimbing:
Prof. Dr. Mohammad Isa Irawan, M.T

DEPARTEMEN MATEMATIKA
Fakultas Sains dan Analitika Data
Institut Teknologi Sepuluh Nopember
Surabaya 2020

TUGAS AKHIR - KM184801

ALGORITMA HYBRID DIFFERENTIAL EVOLUTION - SUPPORT VECTOR MACHINE (DE-SVM) UNTUK SISTEM DETEKSI INTRUSI JARINGAN

YOHANES A CRUX GOSAL
NRP. 06111640000001

Dosen Pembimbing:
Prof. Dr. Mohammad Isa Irawan, M.T

DEPARTEMEN MATEMATIKA
Fakultas Sains dan Analitika Data
Institut Teknologi Sepuluh Nopember
Surabaya 2020

FINAL PROJECT - KM184801

DIFFERENTIAL EVOLUTION - SUPPORT VECTOR MACHINE (DE-SVM) HYBRID ALGORITHM FOR NETWORK INTRUSION DETECTION SYSTEM

YOHANES A CRUX GOSAL
NRP. 06111640000001

Supervisor:
Prof. Dr. Mohammad Isa Irawan, M.T

DEPARTMENT OF MATHEMATICS
Faculty of Science and Data Analytics
Institut Teknologi Sepuluh Nopember
Surabaya 2020

LEMBAR PENGESAHAN

ALGORITMA HYBRID DIFFERENTIAL EVOLUTION - SUPPORT VECTOR MACHINE (DE-SVM) UNTUK SISTEM DETEKSI INTRUSI JARINGAN

DIFFERENTIAL EVOLUTION - SUPPORT VECTOR MACHINE (DE-SVM) HYBRID ALGORITHM FOR NETWORK INTRUSION DETECTION SYSTEM

TUGAS AKHIR

Diajukan untuk memenuhi salah satu syarat
Untuk memperoleh gelar Sarjana Matematika
Pada bidang studi Ilmu Komputer
Program Studi S-1 Departemen Matematika
Fakultas Sains dan Analitika Data
Institut Teknologi Sepuluh Nopember Surabaya

Oleh:

YOHANES A CRUX GOSAL
NRP. 06111640000001

Menyetujui,
Dosen Pembimbing I

Prof. Dr.techn.Drs. Mohammad Isa Irawan, M.T
NIP. 19631225 198903 1 001

Mengetahui,
Kepala Departemen
Matematika FSAD ITS

Subchan, Ph.D
NIP. 19710513 199702 1 001
Surabaya, 19 Agustus 2020

ALGORITMA HYBRID DIFFERENTIAL EVOLUTION - SUPPORT VECTOR MACHINE (DE-SVM) UNTUK SISTEM DETEKSI INTRUSI JARINGAN

Nama	:	Yohanes A Crux Gosal
NRP	:	06111640000001
Departemen	:	Matematika FSAD ITS
Dosen Pembimbing	:	Prof. Dr. Mohammad Isa Irawan, M.T

Abstrak

Banyak data penting yang disimpan dalam komputer. Sehingga, perlu suatu sistem untuk mendeteksi intrusi (aktivitas yang tidak diinginkan atau tanpa ijin) pada jaringan komputer. Seiring dengan berkembangnya teknologi, tipe dan cara serangan siber pun berkembang. Banyak peneliti mengusulkan berbagai teknik dengan menggunakan pembelajaran mesin dan kecerdasan buatan untuk membangun suatu sistem deteksi intrusi. Selain itu, deep learning juga sudah mulai digunakan untuk membangun sistem deteksi intrusi. Tujuan dari penelitian ini adalah untuk mengevaluasi kemampuan sistem deteksi intrusi berbasis support vector machine (SVM) yang sudah dioptimasi dengan differential evolution (DE). Model SVM tanpa penyetelan parameter, model hibrida PSO-SVM dan juga model deep convolutional neural network (DCNN) digunakan sebagai pembanding. Semua model ini dilatih dan diuji dengan data latih dan uji dari dataset NSL-KDD. Setiap model yang digunakan dievaluasi dengan menggunakan nilai akurasi klasifikasi pada data uji. Dari hasil eksperimen, disimpulkan bahwa model hibrida DE-SVM mampu memberikan hasil lebih baik dari model SVM.

***Kata kunci*: *support vector machine, differential evolution, deteksi intrusi*.**

DIFFERENTIAL EVOLUTION - SUPPORT VECTOR MACHINE (DE-SVM) HYBRID ALGORITHM FOR NETWORK INTRUSION DETECTION SYSTEM

Name	:	Yohanes A Crux Gosal
NRP	:	06111640000001
Department	:	Mathematics FSDA ITS
Supervisor	:	Prof. Dr. Mohammad Isa Irawan, M.T

Abstract

Lot of data are stored in computers. Hence, a system to detect an intrusion (unauthorized or unwanted activities) on a computer network is needed. As technology evolved, cyberattack types and methods are also evolving. Many researchers have proposed several techniques using machine learning and artificial intelligence to develop an intrusion detection system. Besides, deep learning also have been used to develop an intrusion detection system. The aim of this research is to evaluate the ability of intrusion detection system based on support vector machine (SVM) which have been optimized with differential evolution (DE). SVM model without parameter tuning and hybrid model PSO-SVM are also used as comparators. In this research, deep learning model namely, deep convolutional neural network (DCNN), are also used as comparator. All models are trained and evaluated using train and test set from NSL-KDD dataset. Each model are evaluated using accuracy of classification on test set. From experiment results, it is concluded that hybrid model DE-SVM yields better result than SVM model.

Keywords* : *support vector machine, differential evolution, intrusion detection

KATA PENGANTAR

Syukur dan puji dihanturkan kehadirat Tuhan Yang Maha Esa karena atas berkat dan perlindungan-Nya, Tugas Akhir ini bisa terselesaikan dengan baik dan tepat waktu.

Tugas Akhir merupakan salah satu dari mata kuliah wajib pada departemen Matematika FSAD ITS dan menjadi suatu syarat kelulusan. Mata kuliah ini ditujukan kepada mahasiswa semester 7 atau 8 yang akan menyelesaikan studi S1.

Tugas Akhir ini disusun dalam rangka untuk memenuhi salah satu syarat memperoleh gelar Sarjana Matematika pada bidang Ilmu Komputer. Dalam Tugas Akhir ini, penulis memberikan hasil penelitian tentang performa algoritma DE-SVM dalam melakukan deteksi intrusi pada jaringan komputer. Pada Tugas Akhir ini, algoritma DE-SVM akan dibandingkan juga dengan algoritma Deep CNN, PSO-SVM, dan SVM dengan tujuan untuk mengetahui seberapa baik performa algoritma ini dengan beberapa algoritma yang sudah diteliti dan diusulkan untuk melakukan deteksi intrusi. Adapun judul dari Tugas Akhir ini adalah

ALGORITMA HYBRID DIFFERENTIAL EVOLUTION - SUPPORT VECTOR MACHINE (DE-SVM) UNTUK SISTEM DETEKSI INTRUSI JARINGAN

Penyusunan Tugas Akhir ini tidak terlepas dari dukungan berbagai pihak. Oleh sebab itu, tak lupa penulis menyampaikan ucapan terima kasih kepada:

1. Orangtua penulis, Almarhum Go Ferry Gosal dan Yuliawati Sutanto, serta adik-adik penulis, Vincentius Marco Gosal dan Go Gregory Aaron Gosal, atas inspirasi, dukungan, dan bantuannya selama perjalanan hidup penulis hingga Tugas Akhir ini selesai.

2. Bapak Subchan, Ph.D, selaku kepala Departemen Matematika

FSAD ITS yang telah memberikan motivasi dan dukungan kepada penulis.

3. Ibu Dr. Dwi Ratna Sulistyaningrum, S.Si, M.T dan Bapak Dr. Budi Setiyono, S.Si, M.T, selaku Sekretaris Departemen Matematika FSAD ITS, yang telah membantu penulis dalam hal menyelesaikan kebutuhan administrasi selama periode pengerjaan Tugas Akhir.

4. Bapak Prof. Dr.techn.Drs. Mohammad Isa Irawan, M.T selaku dosen pembimbing I atas bimbingan, saran, kritik, dan motivasi selama proses pengerjaan Tugas Akhir.

5. Bapak Drs. Daryono Budi Utomo, M.Si, Bapak Dr. Darmaji, S.Si, M.T, dan Bapak Dr. Budi Setiyono, S.Si, M.T, selaku dosen penguji yang sudah memberikan masukan yang membangun.

6. Ibu Dra. Sri Suprapti H., M.Si dan Bapak Drs. Suhud Wahyudi, M.Si sebagai dosen wali yang sudah memberikan arahan selama kegiatan perkuliahan di ITS.

7. Seluruh pihak yang tidak bisa disebutkan satu-persatu, yang telah memberikan saran, dukungan dan motivasi dalam menyelesaikan Tugas Akhir ini. Penulis mengucapkan terima kasih yang sangat dalam, atas doa dan semangat yang diberikan kepada penulis.

Penulis juga mengharapkan kritik dan saran yang membangun dari berbagai pihak untuk perbaikan isi Tugas Akhir ini. Segala kritik dan saran akan penulis terima dengan senang hati.

Surabaya, 13 Mei 2020

Yohanes A Crux Gosal

DAFTAR ISI

HALAMAN JUDUL **i**

LEMBAR PENGESAHAN **v**

ABSTRAK **vii**

ABSTRACT **ix**

KATA PENGANTAR **xi**

DAFTAR ISI **xiii**

DAFTAR GAMBAR **xvii**

DAFTAR TABEL **xix**

DAFTAR SIMBOL **xxi**

1 BAB I PENDAHULUAN **1**
1.1 Latar Belakang Masalah 1
1.2 Rumusan Masalah 3
1.3 Batasan Masalah 3
1.4 Tujuan 4
1.5 Manfaat 4
1.6 Sistematika Penulisan 4

2 BAB II TINJAUAN PUSTAKA **7**
2.1 Penelitian Terdahulu 7
2.2 Sistem Deteksi Intrusi 8
2.2.1 Klasifikasi Sistem Deteksi Intrusi 8
2.3 Praproses Data 10
2.3.1 *Leave One Out Encoding* 10
2.3.2 Standardisasi Data 11

2.3.3 Seleksi Fitur 11
2.4 Support Vector Machine 12
2.4.1 SVM Linier 12
2.4.2 SVM Nonlinier 13
2.5 Kernel 14
2.6 Differential Evolution 14
2.6.1 Mutasi 15
2.6.2 *Crossover* 15
2.6.3 Seleksi 16

3 BAB III METODE PENELITIAN 17
3.1 Spesifikasi Perangkat 17
3.2 Tahapan Penelitian Tugas Akhir 17

4 BAB IV HASIL DAN PEMBAHASAN 23
4.1 Deskripsi Data 23
4.1.1 Distribusi data 29
4.2 Praproses Data 29
4.2.1 *Encoding* Data 31
4.2.2 Standardisasi Data 32
4.2.3 Transformasi Data 35
4.3 Seleksi Fitur 37
4.3.1 Seleksi Fitur dengan DE 38
4.3.2 Seleksi Fitur dengan bPSO 43
4.4 Implementasi SVM 45
4.5 Implementasi DE-SVM 47
4.5.1 Desain Algoritma DE-SVM 47
4.5.2 Hasil Implementasi Algoritma DE-SVM . . 49
4.6 Implementasi PSO-SVM 52
4.7 Implementasi DCNN 54
4.8 Evaluasi Hasil 58
4.8.1 Akurasi 58
4.8.2 Waktu Komputasi 59
4.9 Simulasi Sistem Deteksi Intrusi 59

5 BAB V KESIMPULAN DAN SARAN **63**
5.1 Kesimpulan 63
5.2 Saran 64

DAFTAR PUSTAKA **65**

LAMPIRAN **69**

DAFTAR GAMBAR

3.1 Diagram penyelesaian masalah Tugas Akhir. 21
4.1 Distribusi data pada *train set* sebelum dikelompokkan. 29
4.2 Distribusi data pada *train set* setelah dikelompokkan menjadi empat serangan. 30
4.3 Distribusi data pada *train set* setelah dikelompokkan menjadi normal dan anomali. 30
4.4 *Scatter plot* fitur "duration" dan "protocol_type". . 31
4.5 Kode untuk *leave one out encoding* pada data latih dan data uji. 32
4.6 Kode untuk standardisasi data (dengan StandardScaler). 34
4.7 Kode untuk standardisasi data (dengan RobustScaler). 35
4.8 Kode transformasi data. 37
4.9 Contoh data latih sebelum transformasi. 38
4.10 Contoh data latih sesudah transformasi. 38
4.11 Kode inisiasi parameter dan populasi DE. 39
4.12 Kode untuk menghitung nilai *fitness* populasi. . . . 40
4.13 Kode untuk melakukan mutasi. 41
4.14 Kode untuk melakukan persilangan. 41
4.15 Kode untuk melakukan seleksi. 42
4.16 Potongan keluaran proses seleksi fitur dengan DE. . 42
4.17 Kurva perubahan nilai *fitness* terbaik. 43
4.18 Kode untuk seleksi fitur dengan bPSO. 45
4.19 Kurva perubahan nilai *cost* terbaik (bPSO). 46
4.20 Kode untuk melakukan *training* dan *testing* model SVM. 47
4.21 Diagram alir desain algoritma hybrid DE-SVM yang diusulkan. 50
4.22 Kode untuk melakukan optimasi parameter SVM dengan DE. 52

4.23 Potongan keluaran proses optimasi parameter SVM dengan DE. 52
4.24 Kurva perubahan nilai *fitness* terbaik pada proses optimasi parameter SVM dengan DE. 53
4.25 Kode untuk optimasi parameter SVM dengan PSO. 54
4.26 Kurva perubahan nilai *cost* terbaik pada proses optimasi parameter SVM dengan PSO. 55
4.27 Kode inisiasi model DCNN. 56
4.28 Kurva perubahan akurasi *training* dan validasi model DCNN. 56
4.29 Desain antarmuka sistem deteksi intrusi. 60
4.30 Tampilan program bila terjadi intrusi. 60

DAFTAR TABEL

2.1 Contoh *leave one out encoding*. 10
4.1 Pengelompokkan Data Serangan pada Dataset NSL-KDD. 25
4.2 Daftar dan Deskripsi Fitur pada Dataset NSL-KDD. 25
4.3 Akurasi untuk Tiap Nilai Sigma. 33
4.4 Fitur dengan nilai kategorikal pada data latih sebelum *encoding* (lima data pertama) 33
4.5 Fitur dengan nilai kategorikal pada data latih sesudah *encoding* (lima data pertama) 34
4.6 Potongan data latih sebelum standardisasi. 35
4.7 Potongan data latih sesudah standardisasi (StandardScaler). 36
4.8 Potongan data latih sesudah standardisasi (RobustScaler). 36
4.9 Fitur yang Terpilih dengan DE. 43
4.10 Nilai Parameter yang Digunakan untuk bPSO. . . . 44
4.11 Fitur Terpilih dengan bPSO. 46
4.12 Nilai Parameter yang Digunakan untuk DE-SVM dan Akurasi. 51
4.13 Nilai Parameter yang Digunakan untuk sPSO. . . . 53
4.14 Akurasi Algoritma DCNN. 57
4.15 Akurasi Model 58
4.16 Waktu Komputasi Model 59

DAFTAR SIMBOL

x	nilai data
u	rata-rata data
s	simpangan baku data
$\mathbf{x}_i$	vektor input ke-i
y_i	target atau label dari vektor input ke-i
N	banyak data
$\mathbb{R}$	himpunan bilangan riil
D	dimensi data
$\mathbf{w}$	vektor bobot
b	bias
C	faktor penalti
ξ_i	variabel *slack* ke-i
λ	pengali Lagrange
γ	ukuran fungsi basis
NP	besar populasi
F	faktor mutasi
CR	tingkat persilangan (*crossover*)

BAB I
PENDAHULUAN

Pada bab pendahuluan ini, dipaparkan latar belakang masalah, rumusan masalah, batasan masalah, tujuan, manfaat, dan sistematika penulisan dari Tugas Akhir.

1.1 Latar Belakang Masalah

Pada era digital ini, banyak data penting dan rahasia sudah disimpan dalam komputer. Sehingga, orang luar yang menginginkan data tersebut akan berusaha untuk bisa masuk kedalam jaringan komputer untuk mengambil data tersebut. Ada cukup banyak kasus serangan siber termasuk pembobolan data (*data breach*). Pada bulan Maret 2020, terjadi 67 insiden dengan total 832 juta data yang berhasil dibobol[1]. Salah satu insiden tersebut adalah bocornya lebih dari 81,5 juta data termasuk data email perusahaan milik karyawan, nomor telefon dan sebanyak 76.000 data sidik jari di Brazil[2]. Hal ini menunjukkan, bahwa intrusi masih menjadi permasalahan hingga saat ini. Sehingga, diperlukan suatu sistem untuk mendeteksi adanya serangan yang membahayakan jaringan komputer dan data di dalamnya.

Terdapat dua jenis *intrusion detection system* (IDS) yaitu, *host* IDS (HIDS) dan *network* IDS (NIDS). HIDS berfokus pada serangan di jaringan lokal, sedangkan NIDS berfokus pada serangan dari luar melalui internet [1]. Berdasarkan cara mengidentifikasi serangan, IDS dapat dikelompokkan menjadi *anomaly detection*

[1]Irwin, L. Apr. 2020. **List of data breaches and cyber attacks in March 2020 - 832 million records breached**, <URL: www.itgovernance.co.uk/blog/list-of-data-breaches-and-cyber-attacks-in-march-2020-832-million-records-breached> (dikunjungi pada: 21 Apr. 2020).

[2]Wilson, J. Mar. 2020. **Brazil: Millions of Records Leaked, Including Biometric Data**, <URL: www.safetydetectives.com/blog/antheus-leak-report/> (dikunjungi pada: 21 Apr. 2020).

dan *misuse detection* IDS. *Misuse detection* IDS menggunakan data serangan yang sudah ada untuk mendeteksi serangan yang akan datang, sehingga akan kesulitan mendeteksi serangan dengan metode baru. Sedangkan *anomaly detection* IDS mendeteksi berdasarkan perilaku atau kejadian yang tidak biasa, sehingga dapat mendeteksi serangan dengan metode baru [1].

Serangan melalui jaringan internet sangat bervariasi dan terus berkembang. Sebagai contoh, ukuran serangan *distributed denial of servie* (DDoS) pada tahun 2012 masih berada di bawah 200 Gbps, kemudian pada tahun 2013 menjadi lebih dari 200 Gbps dan terus meningkat secara eksponensial. Akhirnya, pada tahun 2018 ukuran serangan DDoS menembus 1600 Gbps. Selain itu, pelaku serangan siber juga sudah menggunakan kecerdasan buatan untuk melakukan intrusi. Hal ini terjadi pada tahun 2018, ketika GitHub terkena serangan DDoS dengan puncak 1.35 Tbps[3].

IDS berbasis deteksi anomali cenderung lebih baik dibandingkan *misuse detection* untuk mendeteksi serangan pada jaringan komputer, karena mampu mendeteksi serangan dengan metode baru. Saat ini, algoritma dalam *machine learning* dan *deep learning* sudah banyak digunakan untuk meningkatkan performa deteksi anomali [2]. Naseer [2], mengusulkan beberapa metode *deep learning*, seperti *deep convolutional neural network* (DCNN) dan *long short term memory* (LSTM). DCNN dan LSTM ditunjukkan memiliki akurasi yang lebih tinggi dibandingkan *support vector machine* (SVM) dengan kernel Gaussian dan *decision tree*. Namun, waktu *training* dari *decision tree* jauh lebih cepat daripada DCNN dan LSTM. Sedangkan, SVM cukup lambat dibandingkan DCNN dan LSTM. Kemudian, Mahmoud [3] mengusulkan SVM dengan *particle swarm optimization* (PSO). *Binary* PSO digunakan untuk melakukan seleksi fitur dan PSO standar digunakan untuk melakukan optimisasi parameter

[3]Hao, M. Juli 2019. **DDoS in the Past Decade**, <URL: `www.nsfocusglobal.com/ddos-in-the-past-decade/`> (dikunjungi pada: 21 Apr. 2020).

dari SVM. Hasilnya, algoritma PSO-SVM memiliki akurasi dan kecepatan yang lebih baik dari SVM biasa. Akan tetapi, *false positive rate* dari PSO-SVM masih lebih tinggi dari SVM biasa.

Oleh karena itu, peneliti ingin melakukan penelitian terhadap pengembangan IDS berbasis SVM. Penelitian yang akan dilakukan mengarah pada penggunaan algoritma *Differential Evolution* (DE) untuk melakukan optimasi parameter pada SVM, dengan harapan IDS dengan algoritma hybrid DE-SVM lebih baik dibandingkan dengan algoritma yang sudah ada.

1.2 Rumusan Masalah

Berjalan dari latar belakang yang sudah dipaparkan sebelumnya, bisa diformulasikan rumusan masalah pada Tugas Akhir adalah sebagai berikut.

1. Bagaimana performa IDS berbasis algoritma DE-SVM?
2. Bagaimana performa DE-SVM dibandingkan dengan algoritma SVM, PSO-SVM dan DCNN?

1.3 Batasan Masalah

Batasan masalah yang digunakan pada Tugas Akhir ini adalah sebagai berikut.

1. Data yang digunakan adalah data NSL-KDD yang sering digunakan sebagai *benchmark* untuk menguji performa algoritma IDS.
2. Pada proses pembandingan yang dijadikan sebagai pembanding adalah akurasi dan waktu komputasi pada tahap *training* dan *testing*.
3. Analisa dilakukan secara empiris bukan secara analitik.

1.4 Tujuan

Adapun tujuan pada Tugas Akhir adalah sebagai berikut.

1. Mendapatkan hasil analisis (performa) IDS berbasis DE-SVM.
2. Mendapatkan hasil analisis perbandingan performa algoritma DE-SVM dengan algoritma SVM, PSO-SVM dan DCNN.

1.5 Manfaat

Adapun tujuan pada Tugas Akhir adalah sebagai berikut.

1. Memberikan informasi mengenai performa DE-SVM dalam melakukan deteksi intrusi.
2. Memberikan kajian berupa analisa performa dari hasil simulasi IDS berbasis DE-SVM.

1.6 Sistematika Penulisan

Sistematika penulisan Tugas Akhir ini adalah sebagai berikut.

1. BAB I PENDAHULUAN
   Pada bab ini dipaparkan latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika penulisan dari Tugas Akhir ini.
2. BAB II TINJAUAN PUSTAKA
   Pada bab ini diberikan dasar teori yang digunakan dalam pengerjaan Tugas Akhir.
3. BAB III METODE PENELITIAN
   Pada bab ini dipaparkan langkah-langkah yang dilakukan dalam pengerjaan Tugas Akhir.

4. BAB IV HASIL DAN PEMBAHASAN
   Pada bab ini dipaparkan hasil yang diperoleh selama pengerjaan Tugas Akhir beserta dengan pembahasan mengenai hasil tersebut.

5. BAB V KESIMPULAN DAN SARAN
   Pada bab ini diberikan kesimpulan berdasarkan hasil yang diperoleh selama pengerjaan Tugas Akhir dan saran untuk penelitian selanjutnya.

BAB II
TINJAUAN PUSTAKA

2.1 Penelitian Terdahulu

Penelitian untuk mengembangkan sistem deteksi intrusi dengan menggunakan *deep learning* telah dilakukan oleh Naseer [2]. Naseer, menggunakan *Deep CNN* (DCNN) dan LSTM, serta beberapa algoritma DNN yang lain. Kemudian, Naseer membandingkan performa beberapa algoritma, seperti DCNN, LSTM, Random Forest, dan RBF SVM dengan menggunakan data NSLKDD. Hasilnya, diperoleh bahwa LSTM memiliki tingkat akurasi paling tinggi (0.89) diikuti oleh DCNN (0.85). Sedangkan algoritma Nearest Neighbors, Decision Tree, dan RBF SVM berada pada urutan ketiga dengan akurasi 0.82. Akan tetapi, algoritma Decision Tree dan Random Forest memiliki waktu *training* tersingkat dari kelima algoritma yang disebutkan sebelumnya (0.28 detik dan 0.24 detik). Sedangkan, RBF SVM adalah yang paling lama dengan waktu *training* 313.9 detik. Namun, waktu *testing* dari SVM cukup cepat (0.3 detik) dibandingkan dengan LSTM (3.94 detik) dan DCNN (1.52 detik). Kemudian, Sakr [3] mengusulkan penggunaan algoritma *binary Particle Swarm Optimization* (bPSO) untuk melakukan seleksi fitur, algoritma PSO standar untuk melakukan *tuning* parameter pada SVM dan algoritma SVM untuk melakukan klasifikasi. Hasil dari penelitian ini menunjukkan bahwa algoritma bPSO+PSO+SVM berhasil mengurangi waktu *training* dari 400 detik (SVM saja) menjadi 5 detik. Selain itu, tingkat akurasi bertambah sebanyak 8%. Data yang digunakan dalam penelitian ini adalah data NSLKDD. Selain itu, Sailaja [4] pada tahun 2011 mengusulkan *intrusion detection model* berdasarkan pada algoritma Differential Evolution. Pada penelitian ini, diberikan perbandingan performa antara model yang menggunakan algoritma DE, SVM, dan RBF Network, dengan menggunakan data NSL-KDD. Hasilnya diperoleh bahwa untuk data yang tidak direduksi, model berbasis DE sedikit lebih akurat dibanding SVM (beda

0.4 %). Tetapi, untuk data yang direduksi (dilakukan seleksi fitur) diperoleh bahwa SVM cukup jauh lebih akurat dibanding DE (beda 3.2 %). Selain itu, tingkat *false positive* dari DE jauh lebih rendah daripada SVM untuk data yang direduksi dan tidak direduksi.

2.2 Sistem Deteksi Intrusi

Deteksi intrusi adalah sebuah proses memonitor kejadian-kejadian pada sistem komputer atau jaringan dan menganalisa kejadian tersebut untuk melihat tanda kemungkinan terjadinya insiden. Insiden yang dimaksud adalah pelanggaran atau ancaman terhadap keamanan komputer. Insiden memiliki berbagai macam penyebab, seperti *malware*, penyerang (*attackers*) yang berhasil mendapatkan akses ke sistem dari internet, dan pengguna lain dalam sistem yang ingin mengakses data yang tidak sesuai hak aksesnya [5]. Sistem deteksi intrusi mencoba untuk mendeteksi percobaan untuk menembus kedalam suatu sistem, bukan mencegah terjadinya intrusi, dengan memantau kejadian-kejadian yang terjadi dalam jaringan komputer dan melakukan analisa untuk menemukan tanda-tanda adanya intrusi. Sebuah sistem deteksi intrusi mampu mengeluarkan peringatan kepada operator sistem atau petugas keamanan ketika mendeteksi adanya intrusi atau suatu aktivitas yang tidak wajar. Kemudian, setelah muncul peringatan dapat dilakukan langkah-langkah pencegahan untuk mencegah penyusup (*intruder*) masuk ke dalam sistem lebih jauh [1].

2.2.1 Klasifikasi Sistem Deteksi Intrusi

Menurut Yu [1], sistem deteksi intrusi diklasifikasikan berdasarkan dua aspek: pendekatan deteksi dan sumber data. Berdasarkan pendekatan deteksi, sebuah sistem deteksi intrusi dapat berupa sistem deteksi berdasarkan anomali, penyalahgunaan (*misuse*) atau spesifikasi.

1. Sistem deteksi anomali, berfokus mengidentifikasi kejadian-

kejadian yang tidak wajar (kejadian yang tidak atau jarang dilakukan oleh seorang *user* atau komputer *host*). Profil dari sebuah entitas *user* dapat memuat informasi seperti rata-rata durasi sesi FTP, waktu *log in*, atau terminal yang biasa digunakan untuk *log in*. Profil dari sebuah komputer *host* dapat memuat informasi rata-rata penggunaan CPU, rata-rata banyak *user* yang *log in*, dll. Kemudian, sistem deteksi intrusi akan membandingkan penggunaan atau operasi komputer dengan profil yang ada. Jika sistem mendeteksi adanya simpangan yang "besar" dari perilaku normal, maka sistem akan memberi peringatan pada petugas keamanan jaringan. Ukuran "besar" ditentukan oleh petugas keamanan jaringan atau dari sistem deteksi intrusi itu sendiri. Karena model ini bekerja dengan mencari aktivitas atau perilaku yang tidak normal, maka disebut model deteksi anomali [1].

2. Sistem deteksi penyalahgunaan (*misuse*) mengkarakterisasi beberapa serangan yang sudah diketahui sebelumnya. Karakteristik dari serangan dapat direpresentasikan dalam bentuk graf, *regular expressions*, dll. Implementasi dari IDS yang demikian biasanya melibatkan sistem pakar yang melakukan pencocokan dengan *rule* yang sudah disimpan. Model deteksi ini memerlukan pembaruan *rule* setiap kali ada pola serangan yang baru. Karena model bekerja dengan mencari pola serangan yang sudah diketahui, maka model ini disebut sebagai model deteksi penyalahgunaan (*misuse*) [1].

Klasifikasi yang kedua adalah berdasarkan sumber data audit dari IDS.

1. Sistem deteksi intrusi yang menggunakan data audit pada satu mesin dan menarik kesimpulan berdasarkan data itu saja, disebut sebagai *host* IDS. Sistem deteksi yang demikian bergantung pada *log* dari sebuah sistem operasi saja, sehingga rentan terhadap serangan DOS (*Denial of Service*).

2. Sebuah solusi yang efisien diberikan oleh IDS yang memantau aktivitas mencurigakan pada sebuah jaringan. IDS yang demikian disebut *network* IDS. Karena sistem yang demikian hanya bergantung pada protokol TCP/IP, maka sistem mampu memantau jaringan yang beragam [1].

2.3 Praproses Data

2.3.1 *Leave One Out Encoding*

Leave One Out merupakan salah satu metode untuk merubah fitur kategorikal menjadi numerik (*encoding*). Metode ini menggunakan rata-rata target yang berkaitan dengan sebuah kategori. Untuk data latih, nilai target pada baris yang akan dirubah diabaikan lalu dihitung rata-rata nilai target untuk kategori yang sama dengan data pada baris tersebut. Untuk data uji, nilai pada baris yang akan dirubah tidak diabaikan. Setelah menghitung nilai rata-rata target, diberikan nilai acak dan dikalikan dengan nilai rata-rata target. Illustrasi metode ini diberikan oleh Zhang [6] seperti pada tabel 2.1. Pada

Tabel 2.1. Contoh *leave one out encoding*.

Split	User ID	Y	mean(Y)	random	Exp_UID
Training	A1	0	.667	1.05	0.70035
Training	A1	1	.333	.97	0.32301
Training	A1	1	.333	.98	0.32634
Training	A1	0	.667	1.02	0.68034
Test	A1	-	.5	1	0.5
Test	A1	-	.5	1	0.5
Training	A2	0			

contoh di tabel 2.1 yang menjadi fitur dengan nilai kategorikal adalah "User ID" dan targetnya adalah "Y". Untuk data latih pada baris

pertama, nilai "Y" diabaikan kemudian dihitung rata-rata dari tiga baris berikutnya yang juga merupakan data untuk *user* dengan ID A1. Nilai rata-rata tersebut diletakkan pada kolom mean(Y), kemudian dikalikan dengan nilai acak dan diperoleh hasil *encoding*-nya.

2.3.2 Standardisasi Data

Standardisasi data merupakan proses untuk merubah data pada tiap fitur sehingga memiliki rata-rata 0 dan simpangan baku 1. Untuk melakukan standardisasi, setiap data diskala ulang menggunakan Z-score.

$$Z = \frac{x - u}{s} \tag{2.1}$$

Dimana u merupakan rata-rata dan s merupakan simpangan baku. Apabila data mengandung *outlier*, maka melakukan standardisasi dengan rata-rata dan simpangan baku tidak akan bekerja dengan baik. Sehingga, bisa digunakan median dan jangkauan interkuartil sebagai pengganti rata-rata dan simpangan baku[4].

2.3.3 Seleksi Fitur

Seleksi fitur merupakan salah satu metode dalam praproses data. Seleksi fitur terbukti efektif dan efisien dalam mempersiapkan data (terutama data dengan dimensi tinggi) untuk berbagai masalah penambangan data dan pembelajaran mesin. Seleksi fitur terbagi dua, yaitu metode *supervised* dan metode *unsupervised*. Metode seleksi fitur *supervised* digunakan ketika terdapat cukup informasi mengenai label atau kelas data. Sedangkan metode *unsupervised*, didesain untuk masalah *clustering*, sehingga tidak memerlukan informasi label atau kelas[7].

[4]Scikit-learn developers. 2019. **Preprocessing data**, <URL: `scikit-learn.org/stable/modules/preprocessing.html#preprocessing-scaler`> (dikunjungi pada: 12 Juni 2020).

Kemudian, berdasarkan strategi pemilihan fitur, ada 3 macam metode pemilihan fitur. Metode tersebut adalah metode *wrapper*, metode *filter*, dan metode *embedded*. Metode *wrapper* bekerja dengan mencari himpunan bagian dari fitur-fitur yang diberikan, kemudian mengevaluasi fitur-fitur yang terpilih sesuai dengan algoritma pembelajaran yang digunakan. Jika suatu kondisi tercapai, maka metode *wrapper* akan berhenti melakukan seleksi fitur. Metode *filter* tidak bergantung pada algoritma pembelajaran apapun. Metode *filter* mengurutkan fitur-fitur berdasarkan kriteria tertentu, kemudian filter dengan peringkat atau urutan yang rendah akan dibuang. Metode *embedded* merupakan kombinasi dari metode *wrapper* dan *filter*. Metode ini memasukkan proses seleksi fitur dalam proses pembelajaran, tetapi tidak perlu mengevaluasi fitur secara berulang seperti pada metode *wrapper* [7].

2.4 Support Vector Machine

Support vector machine (SVM) merupakan sebuah model pembelajaran mesin yang digunakan untuk mengenali pola, seperti pada klasifikasi, regresi dan deteksi *outlier*. Untuk menyelesaikan permasalahan klasifikasi biner, SVM menggunakan formulasi seperti berikut [8], [9].

2.4.1 SVM Linier

Misalkan suatu himpunan dari data *training*: $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, dimana $\mathbf{x}_i \in \mathbb{R}^D$ merupakan vektor input ke i dan $y_i \in \{1, -1\}$ adalah label dari vektor input yang bersesuaian. Sehingga, *hyperplane* pemisah yang optimal adalah:

$$h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b \tag{2.2}$$

dimana vektor bobot $\mathbf{w}$ didefinisikan sebagai kombinasi linier dari beberapa titik data yang disebut *support vector* dan b adalah suatu skalar yang disebut bias.

2.4.2 SVM Nonlinier

Ketika data tidak dapat dipisah secara linier, *hyperplane* pemisah dapat diperoleh dengan menyelesaikan permasalahan [8]:

$$\begin{aligned} &\min \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\xi_i \\ &\text{dengan} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i > 0, \\ &\text{untuk } i = 1,2,\ldots,N \end{aligned} \tag{2.3}$$

dimana ξ_i adalah variabel *slack* dan C adalah parameter yang disebut faktor penalti. Dengan menggunakan pengali Lagrange, λ, diperoleh permasalahan *dual* dari (2.3) sebagai berikut [8], [9]:

$$\begin{aligned} &\text{maks} \quad L(\lambda) = \sum_{i=1}^{N}\lambda_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \lambda_i \lambda_j K(\mathbf{x}_i, \mathbf{x}_j) \\ &\text{dengan } C \geq \lambda_i \geq 0, \forall i \in \{1,2,\ldots,N\}, \sum_{i=1}^{N}\lambda_i y_i = 0, \\ &\text{dan } K(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(x_i), \phi(x_j) \rangle \text{ merupakan fungsi kernel.} \end{aligned} \tag{2.4}$$

Untuk melakukan klasifikasi dengan SVM, digunakan algoritma berikut [9].

1. Masukan himpunan data latih $T = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, dimana $\mathbf{x}_i \in \mathbb{R}^D$, $y_i \in \mathcal{Y} = \{1, -1\}, i = 1, \ldots, N$;
2. Tentukan kernel $K(\mathbf{x}, \mathbf{x}')$ dan parameter penalti $C > 0$;
3. Konstruksi dan selesaikan permasalahan seperti pada persamaan 2.4 sehingga diperoleh solusi $\boldsymbol{\lambda}^* = (\lambda_1^*, \ldots, \lambda_N^*)^T$;
4. Hitung b^*: Pilih sebuah komponen dari $\boldsymbol{\lambda}^*$, $\lambda_j^* \in (0, C)$, dan hitung

$$b^* = y_j - \sum_{i=1}^{N} y_i \lambda_i^* K(\mathbf{x}_i, \mathbf{x}_j); \tag{2.5}$$

5. Konstruksi fungsi keputusan (*decision function*)

$$f(x) = sgn(g(x)), \tag{2.6}$$

dimana

$$g(x) = \sum_{i=1}^{N} y_i \lambda_i^* K(\mathbf{x}_i, \mathbf{x}) + b^*. \tag{2.7}$$

2.5 Kernel

Berikut adalah definisi dari kernel [9].

Definisi 2.5.1 ***(Kernel)****. Suatu fungsi $K(\mathbf{x}, \mathbf{x}')$ terdefinisi pada $\mathbb{R}^n \times \mathbb{R}^n$ disebut kernel, jika terdapat suatu pemetaan ϕ dari ruang $\mathbb{R}^n$ ke ruang Hilbert, sedemikian hingga*

$$K(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle \tag{2.8}$$

dimana $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^n$ dan $\langle ., . \rangle$ merupakan hasil kali dalam pada ruang Hilbert.

Salah satu fungsi kernel yang umum digunakan pada SVM adalah fungsi Gaussian atau dikenal dengan istilah RBF (*Radial Basis Function*). Kernel RBF didefinisikan sebagai $K(\mathbf{x}, \mathbf{x}') = e^{-\gamma \|\mathbf{x} - \mathbf{x}'\|^2}$. Fungsi kernel dapat menambahkan parameter, seperti ukuran fungsi basis γ, ke formulasi SVM sehingga dapat memengaruhi performa dari SVM secara signifikan [8].

2.6 Differential Evolution

Differential Evolution (DE) merupakan metode pencarian paralel yang menggunakan vektor parameter berdimensi D

$$\mathbf{x}_{i,G},\ i = 1, 2, \ldots, NP \tag{2.9}$$

sebagai sebuah populasi untuk setiap generasi G [10]. DE merupakan salah satu *Evolutionary Algorithm* dan menggunakan beberapa

operasi seperti *crossover*, mutasi, dan seleksi populasi untuk meminimalkan fungsi objektif. Populasi awal dipilih secara acak sedemikian hingga populasi tersebut mewakili seluruh ruang pencarian [4].

2.6.1 Mutasi

Untuk setiap vektor target $\mathbf{x}_{i,G},\ i = 1,2,\ldots,NP$, suatu vektor hasil mutasi dibentuk berdasarkan persamaan

$$\mathbf{v}_{i,G+1} = \mathbf{x}_{r_1,G} + F \cdot (\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G}) \tag{2.10}$$

dengan indeks-indeks acak dan berbeda $r_1, r_2, r_3 \in \{1,2,\ldots,NP\}$. Pemilihan r_1, r_2, r_3 harus memenuhi $i \neq r_1 \neq r_2 \neq r_3$, sehingga NP paling sedikit bernilai 4. F merupakan bilangan riil dengan nilai 0 sampai 2 [10]. Sebagai contoh, misalkan populasi awal berisi empat vektor dua dimensi, $P = \{\,[1\ 0{,}8]\,,\,[1{,}05\ 0{,}5]\,,\,[1\ 0{,}2]\,,\,[2{,}3\ 0{,}8]\,\}$. Untuk vektor target $\mathbf{x}_1 = [1\ 0{,}8]$, dengan menggunakan tiga vektor yang tersisa dan misal $F = 0{,}8$ dapat diperoleh vektor mutan,

$$\mathbf{v}_1 = \begin{bmatrix} 1{,}05 \\ 0{,}5 \end{bmatrix} + 0{,}8 \left(\begin{bmatrix} 1 \\ 0{,}2 \end{bmatrix} - \begin{bmatrix} 2{,}3 \\ 0{,}8 \end{bmatrix} \right) = \begin{bmatrix} 0{,}01 \\ 0{,}02 \end{bmatrix}$$

2.6.2 *Crossover*

Crossover dilakukan untuk menambah variasi dalam populasi. Suatu vektor *trial* $\mathbf{u}_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1}, \ldots, u_{Di,G+1})$ terbentuk ketika

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1}, & \text{jika } rand(j) \leq CR \text{ atau } j = rnbr(i) \\ x_{ji,G} & \text{jika } rand(j) > CR \text{ dan } j \neq rnbr(i) \end{cases}$$
$$j = 1,2,\ldots,D \tag{2.11}$$

CR merupakan laju (*crossover rate*) atau kemungkinan terjadinya *crossover*. CR merupakan suatu bilangan riil dengan nilai berada

pada selang $[0,1]$ dan ditentukan oleh *user*. $rand(j)$ merupakan nilai acak yang terdistribusi uniform pada selang $[0,1]$. $rnbr(i)$ merupakan indeks yang dipilih secara acak, sehingga $\mathbf{u}_{i,G+1}$ mendapatkan paling sedikit 1 parameter dari $\mathbf{v}_{i,G+1}$ [10]. Sebagai contoh, jika $CR = 0,9$, $rand(1) = 0,5$, dan $rnbr(1) = 2$, maka komponen pertama pada vektor trial akan diambil dari vektor mutan sehingga $u_{11} = 0,01$. Kemudian, misal $rand(2) = 1$ dan $rnbr(2) = 1$, maka komponen kedua pada vektor trial akan diambil dari vektor target sehingga $u_{21} = 0,8$. Jadi, diperoleh vektor trial $\mathbf{u}_1 = [0{,}01\ 0{,}8]$.

2.6.3 Seleksi

Jika suatu vektor *trial* $\mathbf{u}_{i,G+1}$ menghasilkan nilai fungsi objektif yang lebih kecil daripada vektor target $\mathbf{x}_{i,G}$, maka $\mathbf{x}_{i,G+1} = \mathbf{u}_{i,G+1}$. Jika tidak demikian, maka $\mathbf{x}_{i,G+1} = \mathbf{x}_{i,G}$ [10]. Sebagai contoh, misalkan vektor target $\mathbf{x}_1 = [1\ 0{,}8]$ memiliki nilai *fitness* sebesar 0,85, dan vektor trial $\mathbf{u}_1 = [0{,}01\ 0{,}8]$ memiliki nilai *fitness* sebesar 0,5. Vektor target tetap bertahan dan masuk ke generasi berikutnya.

BAB III
METODE PENELITIAN

Pada bab ini dijelaskan mengenai perangkat yang digunakan dan langkah-langkah yang dilakukan dalam penyelesaian masalah Tugas Akhir ini.

3.1 Spesifikasi Perangkat

Untuk melakukan eksperimen pada penelitian tugas akhir digunakan Google Colaboratory. Spesifikasi perangkat keras yang digunakan pada Google Colaboratory adalah sebagai berikut.

- Prosesor: Intel Xeon (2.30 GHz).
- RAM: 12 GB.
- GPU: Tesla K80.

Perangkat lunak yang digunakan adalah sebagai berikut.

- Bahasa pemrograman : Python 3.
- IDE : PyCharm Community Edition.

Untuk praproses data digunakan *library* NumPy[11], pandas [12], dan scikit-learn [13]. Untuk implementasi algoritma Particle Swarm Optimization (PSO) digunakan *library* PySwarms [14]. Untuk implementasi algoritma SVM digunakan *library* ThunderSVM [15] yang di-*built* dengan CUDA 10.1 pada OS Ubuntu 20.04. Untuk implementasi algoritma DCNN digunakan Keras [16] dengan TensorFlow [17] sebagai *backend*.

3.2 Tahapan Penelitian Tugas Akhir

Berikut adalah penjelasan dari tiap langkah yang dilakukan dalam penelitian Tugas Akhir ini:

1. **Studi literatur**
   Tahap pertama adalah melakukan studi literatur mengenai algoritma *Differential Evolution*, SVM, dan sistem deteksi intrusi, serta tentang metode seleksi fitur. Studi literatur yang digunakan berupa *e-book* (untuk SVM [9] dan sistem deteksi intrusi [1]), jurnal dan paper internasional (untuk metode seleksi fitur dan DE).

2. **Pengumpulan data**
   Data NSL-KDD diambil dari (`https://www.unb.ca/cic/datasets/nsl.html`) [18]. Dataset ini berisi data berbagai macam serangan atau intrusi dengan 41 fitur seperti ditunjukkan pada tabel 4.2. Dataset ini merupakan pengembangan dari dataset KDD'99. Dalam dataset ini tidak terdapat data yang kembar dan redundan.

3. ***Preprocessing* data**
   Pada tahap ini, akan dilakukan seleksi fitur dengan algoritma Differential Evolution. Hasil dari tahap ini adalah himpunan bagian fitur yang menyebabkan model SVM mencapai akurasi paling tinggi.

4. **Desain algoritma DE-SVM**
   Setelah melakukan seleksi fitur, akan dilakukan *tuning* parameter SVM dengan menggunakan algoritma DE. *Tuning* parameter berarti mencari nilai optimal dari setiap parameter pada SVM, yaitu faktor penalti (C) dan koefisien kernel (γ). Dari tahap ini, dihasilkan sebuah model klasifikasi yang dikhususkan untuk mengklasifikasi data intrusi.

5. **Verifikasi algoritma**
   Pada tahap ini, akan diuji kebenaran dari algoritma secara empiris dengan menjalankan algoritma pada semua kemungkinan input. Kemudian, akan ditinjau keluaran dari algoritma tersebut apakah sudah sesuai atau tidak. Apabila keluaran

pada semua input yang mewakili ruang input sudah sesuai maka algoritma tersebut sudah benar.

6. **Uji performa algoritma**
   Pada tahap ini, akan dilihat akurasi dan waktu *training* dan *testing* dari DE-SVM. Apabila akurasi dan waktu *training* dan *testing* DE-SVM sudah konvergen (tidak banyak berubah), maka berhenti melakukan *tuning* parameter. Untuk melakukan uji performa, *dataset* akan dipisah menjadi 2, yaitu *training* dan *testing set* dengan rasio 80:20. Kemudian, data *training* akan di *feed* ke dalam model yang sudah dibuat. Selanjutnya, hasil klasifikasi dari model akan dibandingkan dengan data pada *test set* untuk mengukur akurasi dan waktu komputasi.

7. **Implementasi algoritma SVM**
   Pada tahap ini, akan dibuat implementasi algoritma SVM dengan kernel RBF dengan menggunakan ThunderSVM [15]. Implementasi yang dimaksud adalah membuat program untuk melakukan *training* dan validasi dengan algoritma SVM.

8. **Implementasi algoritma DCNN**
   Pada tahap ini, akan dibuat implementasi algoritma DCNN yang sudah diusulkan oleh Naseer [2]. Implementasi yang dimaksud adalah membuat program untuk melakukan *training* dan klasifikasi dengan algoritma DCNN.

9. **Implementasi algoritma PSO-SVM**
   Pada tahap ini, akan dibuat implementasi algoritma PSO-SVM yang sudah diusulkan oleh Sakr [3]. Implementasi yang dimaksud adalah membuat program untuk melakukan *training* dan klasifikasi dengan algoritma PSO-SVM.

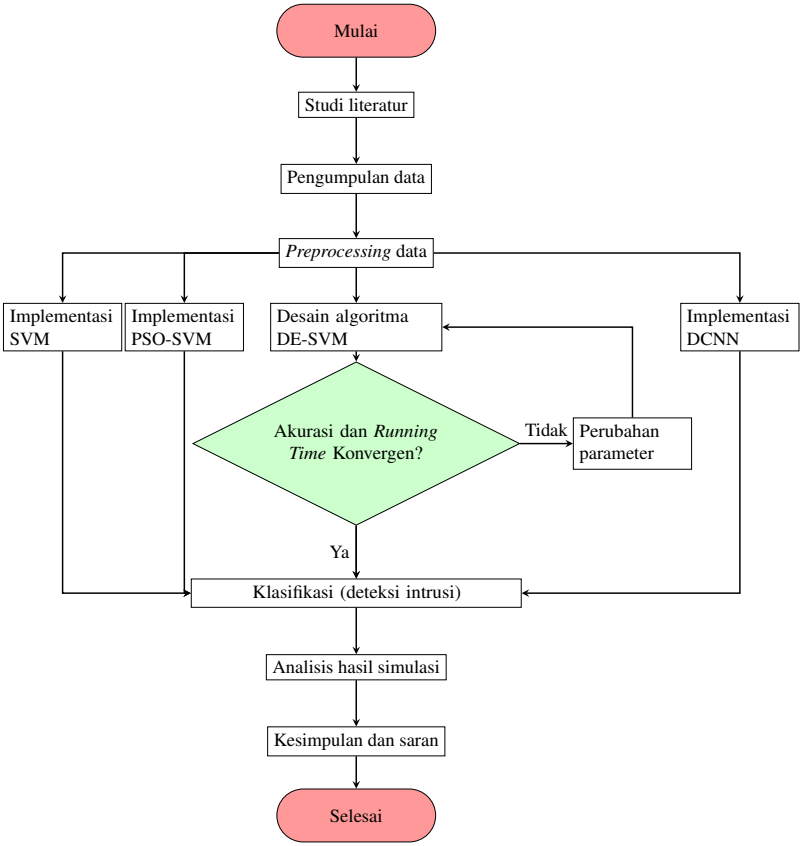
10. **Klasifikasi**
    Pada tahap ini, *dataset* akan dibagi menjadi *training* dan *testing set*, dengan rasio 80:20. Kemudian, keempat model yang

sudah dibuat akan di-*training* dengan menggunakan *training set*. Selanjutnya, dilakukan klasifikasi dengan menggunakan keempat model dengan algoritma DE-SVM, SVM, DCNN, dan PSO-SVM. Output dari proses klasifikasi adalah 1 jika merupakan anomali (aktivitas tidak wajar yang mungkin adalah intrusi) atau 0, jika merupakan aktivitas yang dianggap wajar.

11. **Analisis hasil simulasi**
    Setelah mendapatkan hasil klasifikasi, dibuat *confusion matrix* dari tiap model. Dari *confusion matrix*, kemudian diperoleh akurasi dari ketiga model tersebut. Kemudian, hasil akurasi dan waktu komputasi dari ketiga model akan dibandingkan untuk melihat bagaimana performa algoritma DE-SVM dibandingkan dengan algoritma SVM, DCNN dan PSO-SVM.

12. **Kesimpulan dan saran**
    Untuk tahap terakhir, dipaparkan kesimpulan dari hasil simulasi serta beberapa saran rencana penelitian untuk periode selanjutnya.

Berikut adalah diagram dari langkah-langkah yang sudah diuraikan di atas:



Gambar 3.1. Diagram penyelesaian masalah Tugas Akhir.

BAB IV
HASIL DAN PEMBAHASAN

Pada bab ini diberikan deskripsi data yang digunakan, praproses data, implementasi algoritma Support Vector Machine (SVM), DE-SVM, PSO-SVM, dan Deep CNN, serta analisa hasil eksperimen.

4.1 Deskripsi Data

Data yang digunakan dalam penelitian ini adalah KDDTrain+ dan KDDTrain+_20Percent untuk data latih dan KDDTest+ untuk data uji. KDDTrain+ memuat 125.973 data koneksi. KDDTrain+_20Percent merupakan subset dari KDDTrain+ dan memuat 25.192 data. KDDTest+ memuat 22.544 data koneksi. Data pada dataset NSL-KDD memiliki 41 fitur (lihat tabel 4.2) dan diberi label normal atau merupakan serangan tertentu. Setiap data serangan pada dataset tersebut dapat dikelompokkan menjadi empat kategori [18].

- DoS (*Denial of Service*) adalah suatu serangan dimana penyerang membuat sumber daya komputasi atau memori menjadi terlalu sibuk untuk menangani permintaan atau menolak akses dari pengguna sah.
- Probe merupakan suatu serangan yang mencoba untuk mendapatkan informasi dari jaringan.
- U2R (*User to Root*) merupakan suatu serangan yang mencoba untuk mendapatkan akses ke jaringan sebagai *root* dengan menggunakan akun *user* biasa.
- R2L (*Remote to Local*) merupakan suatu serangan yang mencoba untuk mendapatkan akses sebagai pengguna ke suatu mesin pada jaringan.

Pengelompokkan data serangan yang terdapat pada dataset NSL-KDD dapat dilihat pada tabel 4.1. Pada data uji (KDDTest+) terdapat 17 jenis serangan yang tidak ada di data latih yaitu, apache2, httptunnel, mailbomb, mscan, named, processtable, ps, saint, sendmail, snmpgetattack, snmpguess, sqlattack, udpstorm, worm, xlock, xsnoop, dan xterm. Deskripsi setiap fitur menurut Hassan [19] diberikan pada tabel 4.2. Fitur-fitur pada tabel 4.2 dapat dikelompokkan menjadi tiga kategori [18].

- Fitur dasar. Fitur yang dibuat dari koneksi TCP/IP tanpa inspeksi *payload*. Fitur 1 hingga 9 termasuk dalam kategori ini.

- Fitur konten. Serangan R2L dan U2R tidak memiliki pola intrusi seperti kebanyakan serangan DoS dan Probe. Karena, serangan DoS dan Probe melibatkan banyak koneksi dalam waktu singkat. Sedangkan serangan R2L dan U2R ditanamkan pada bagian data dari paket, dan biasanya menggunakan koneksi tunggal. Untuk mendeteksi serangan ini, diperlukan fitur yang mampu melihat perilaku mencurigakan pada bagian data. Fitur 10 hingga 22 merupakan fitur konten.

- Fitur lalu lintas (*traffic*). Kategori ini memuat fitur yang dihitung terhadap sebuah interval dan dibagi menjadi dua kelompok:

  – fitur "*host* yang sama": hanya memeriksa koneksi dalam dua detik terakhir yang memiliki *host* tujuan yang sama dengan koneksi sekarang, dan menghitung statistik terkait perilaku protokol, layanan, dll.
  – fitur "layanan yang sama": hanya memeriksa koneksi dalam dua detik terakhir yang memiliki layanan yang sama dengan koneksi sekarang.

  Kedua tipe dari fitur "lalu lintas" di atas disebut berbasis waktu. Fitur 23 hingga 31 termasuk dalam kelompok ini. Tetapi,

terdapat beberapa serangan yang lambat yang menggunakan selang waktu lebih lama dari dua detik. Sehingga, fitur "*host* yang sama" dan "layanan yang sama" dihitung ulang berdasarkan pada selang 100 koneksi, bukan selang waktu dua detik. Fitur ini disebut berbasis koneksi. Fitur 32 hingga 41 termasuk dalam kelompok ini.

Tabel 4.1. Pengelompokkan Data Serangan pada Dataset NSL-KDD.

Kategori	**Label Serangan**
DoS	apache2, smurf, neptune, back, teardrop, pod, land, mailbomb, processtable, udpstorm, worm
Probe	ipsweep, mscan, nmap, portsweep, saint, satan
U2R	buffer_overflow, loadmodule, perl, ps, rootkit, sqlattack, xterm
R2L	ftp_write, guess_passwd, httptunnel, imap, multihop, named, phf, sendmail, snmpgetattack, spy, snmpguess, warezclient, warezmaster, xlock, xsnoop

Tabel 4.2. Daftar dan Deskripsi Fitur pada Dataset NSL-KDD.

No.	**Fitur**	**Deskripsi**
1	duration	Lama koneksi (dalam detik)
2	protocol_type	Tipe protokol, contoh: TCP
3	service	Layanan jaringan pada komputer tujuan, contoh: http

Tabel 4.2 Daftar dan Deskripsi Fitur pada Dataset NSL-KDD. (lanjutan)

4	flag	Status koneksi normal atau error
5	src_bytes	Banyak bytes data dari sumber (*source*) ke tujuan
6	dst_bytes	Banyak bytes data dari tujuan (*destination*) ke sumber
7	land	Bernilai 1 jika koneksi menuju/dari port/host yang sama; jika tidak, bernilai 0
8	wrong_fragment	Banyak dari fragmen yang "salah"
9	urgent	Banyak dari paket yang *urgent*
10	hot	Banyak dari indikator "hot"
11	num_failed_logins	Banyak percobaan login yang gagal
12	logged_in	Bernilai 1 jika berhasil login; jika tidak, bernilai 0
13	num_compromised	Banyak dari kondisi "*compromised*"
14	root_shell	1 jika *root shell*diperoleh; jika tidak, 0
15	su_attempted	1 jika perintah "su root" dicoba, 0 jika tidak
16	num_root	Banyak akses "root"

Tabel 4.2 Daftar dan Deskripsi Fitur pada Dataset NSL-KDD. (lanjutan)

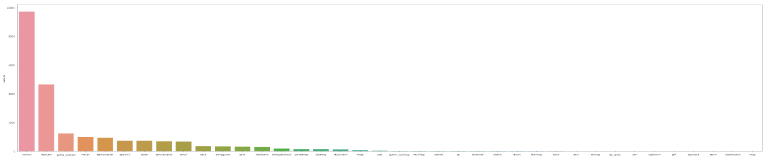
17	num_file_creations	Banyak operasi pembuatan file
18	num_shells	Banyak *shell prompts*
19	num_access_files	Banyak operasi pada berkas kontrol akses
20	num_outbound_cmds	Banyak perintah keluar dalam sebuah sesi ftp
21	is_hot_login	1 jika login termasuk dalam daftar "hot"; 0, jika tidak
22	is_guest_login	1 jika login adalah "*guest*"; 0, jika tidak
23	count	Banyak koneksi ke *host* yang sama dengan koneksi saat ini dalam dua detik terakhir
24	srv_count	Banyak koneksi ke layanan yang sama dengan koneksi saat ini dalam dua detik terakhir
25	serror_rate	Persentase koneksi yang memiliki error "SYN"
26	srv_serror_rate	Persentase koneksi yang memiliki error "SYN"
27	rerror_rate	Persentase koneksi yang memiliki error "REJ"
28	srv_rerror_rate	Persentase koneksi yang memiliki error "REJ"

Tabel 4.2 Daftar dan Deskripsi Fitur pada Dataset NSL-KDD. (lanjutan)

29	same_srv_rate	Persentase koneksi ke layanan yang sama
30	diff_srv_rate	Persentase koneksi ke layanan berbeda
31	srv_diff_host_rate	Persentase koneksi ke *host* berbeda
32	dst_host_count	count untuk *host* tujuan
33	dst_host_srv_count	srv_count untuk *host* tujuan
34	dst_host_same_srv_rate	same_srv_rate untuk *host* tujuan
35	dst_host_diff_srv_rate	diff_srv_rate untuk *host* tujuan
36	dst_host_same_src_port_rate	same_src_port_rate untuk *host* tujuan
37	dst_host_srv_diff_host_rate	srv_diff_host_rate untuk *host* tujuan
38	dst_host_serror_rate	serror_rate untuk *host* tujuan
39	dst_host_srv_serror_rate	srv_serror_rate untuk *host* tujuan
40	dst_host_rerror_rate	rerror_rate untuk *host* tujuan
41	dst_host_srv_rerror_rate	srv_rerror_rate untuk *host* tujuan

4.1.1 Distribusi data

Pada dataset NSL-KDD distribusi data tidak seimbang, karena terdapat jenis serangan yang memiliki banyak data dan beberapa jenis serangan hanya memiliki sedikit data. Untuk distribusi data pada *train set* sebelum dikelompokkan dapat dilihat pada gambar 4.1, untuk gambar yang lebih jelas dapat dilihat di lampiran A. Untuk distribusi data sesudah dikelompokkan menjadi empat kategori serangan dapat dilihat pada gambar 4.2. Setelah dikelompokkan distribusi data masih tidak seimbang, karena serangan U2R dan R2L jauh lebih sedikit dibanding DoS. Sehingga, semua data yang tidak normal dikelompokkan menjadi satu kelas, yaitu kelas anomali. Distribusi data setelah semua serangan dikelompokkan menjadi satu dapat dilihat pada gambar 4.3. Data pada dataset ini juga tidak

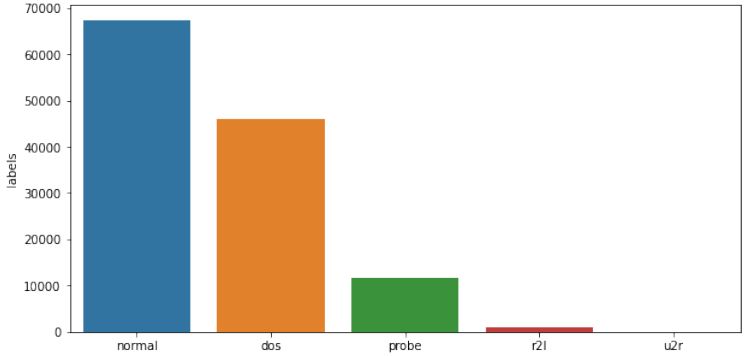


Gambar 4.1. Distribusi data pada *train set* sebelum dikelompokkan.

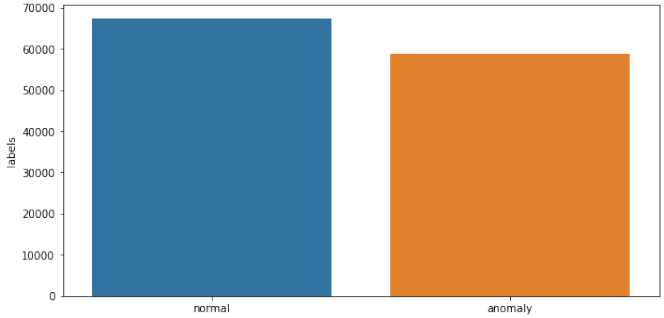
terpisah secara linier. Sebagai contoh, dapat dilihat data fitur "protocol_type" dan "duration" pada gambar 4.4 tidak dapat dipisah secara linier menjadi kelas normal atau anomali.

4.2 Praproses Data

Pertama, dilakukan praproses data NSL-KDD. Pada tahap ini, dilakukan proses *encoding* fitur-fitur yang memiliki nilai kategorikal menjadi numerik. Kemudian, dilakukan standardisasi untuk merubah data pada tiap fitur sehingga memiliki rata-rata 0 dan simpangan baku 1 untuk PSO-SVM dan DE-SVM. Digunakan median dan jangkauan interkuartil sebagai pengganti mean dan standar deviasi untuk DCNN. Selanjutnya, dilakukan seleksi atau reduksi fitur untuk

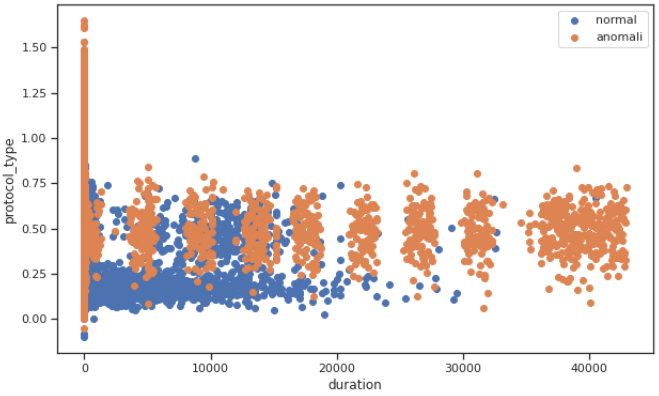


Gambar 4.2. Distribusi data pada *train set* setelah dikelompokkan menjadi empat serangan.



Gambar 4.3. Distribusi data pada *train set* setelah dikelompokkan menjadi normal dan anomali.

mengurangi dimensi dari data. Selain itu, dilakukan juga transformasi data dari vektor berdimensi 41 menjadi matriks berukuran 32 $\times$ 32 untuk digunakan sebagai input algoritma DCNN.



Gambar 4.4. *Scatter plot* fitur "duration" dan "protocol_type".

4.2.1 *Encoding* Data

Untuk merubah data kategorikal menjadi numerik digunakan LeaveOneOutEncoder dari *library* category_encoders. Metode ini menggunakan rata-rata nilai target yang bersesuaian dengan kategori tertentu, untuk penjelasan metode ini dapat dilihat pada subbab 2.3.1. Metode ini dipilih karena tidak menambah dimensi data dan data kategorikal pada *dataset* NSL-KDD merupakan data nominal bukan ordinal. Selain itu, Naseer [2] menunjukkan bahwa menggunakan LeaveOneOutEncoder menghasilkan model dengan akurasi yang lebih baik daripada OrdinalEncoder. Potongan kode untuk melakukan *encoding* pada data latih dan data uji adalah diberikan pada gambar 4.5.

Untuk menentukan nilai sigma yang sesuai, dilakukan *encoding* dengan nilai sigma berbeda. Kemudian, data hasil *encoding* distandardisasi dengan StandardScaler seperti dijelaskan pada bagian 4.2.2 dan kemudian setiap data diklasifikasi dengan SVM. Data akurasi (dalam persen) untuk tiap nilai sigma diberikan pada tabel 4.3. Akurasi yang diberikan pada tabel adalah akurasi pada data uji.

```
from category_encoders import LeaveOneOutEncoder

enc = LeaveOneOutEncoder(cols=['protocol_type', 'service', '
    flag'], return_df=True, sigma=0.25, random_state=0)
enc.fit(X=X_train, y=y_train)

X_train = enc.transform(X_train, y_train)
train_data = X_train.join(y_train)
train_data.to_csv("drive/My Drive/Datasets/KDDTrain+_Encoded.
    csv", index=False)

X_test = enc.transform(X_test)
test_data = X_test.join(y_test)
test_data.to_csv("drive/My Drive/Datasets/KDDTest+_Encoded.
    csv", index=False)
```

Gambar 4.5. Kode untuk *leave one out encoding* pada data latih dan data uji.

Dari tabel tersebut, terlihat bahwa akurasi model tertinggi diperoleh ketika sigma bernilai 0,25. Sehingga, dilakukan *encoding* dengan nilai sigma 0,25. Fitur dengan nilai kategorikal sebelum dilakukan *encoding* dapat dilihat pada tabel 4.4. Fitur dengan nilai kategorikal sesudah dilakukan *encoding* dapat dilihat pada tabel 4.5.

4.2.2 Standardisasi Data

Untuk melakukan standardisasi digunakan StandardScaler dan RobustScaler dari *library* scikit-learn. StandardScaler digunakan untuk memproses data yang akan digunakan untuk algoritma PSO-SVM dan DE-SVM, sedangkan RobustScaler digunakan untuk memproses data yang akan digunakan untuk algoritma DCNN. StandardScaler menggunakan persamaan 2.1 untuk merubah data tiap fitur. Sedangkan, RobustScaler menggunakan median dan jangkauan interkuartil. Potongan kode untuk melakukan standardisasi dapat dilihat pada gambar 4.6 dan gambar 4.7.

Data latih sebelum standardisasi (lima data pertama) dapat

Tabel 4.3. Akurasi untuk Tiap Nilai Sigma.

Sigma	Akurasi	Sigma	Akurasi
0,05	77,57	0,28	78,03
0,1	77,78	0,29	77,28
0,2	77,99	0,3	76,93
0,21	78,01	0,4	76,86
0,22	78,06	0,5	76,79
0,23	78,13	0,6	76,65
0,24	78,20	0,7	76,58
0,25	78,22	0,8	76,47
0,26	78,21	0,9	76,42
0,27	77,97		

Tabel 4.4. Fitur dengan nilai kategorikal pada data latih sebelum *encoding* (lima data pertama)

protocol_type	service	flag
tcp	ftp_data	SF
udp	other	SF
tcp	private	S0
tcp	http	SF
tcp	http	SF

dilihat pada tabel 4.6, sedangkan data latih sesudah standardisasi dapat dilihat pada tabel 4.7 dan 4.8.

Tabel 4.5. Fitur dengan nilai kategorikal pada data latih sesudah *encoding* (lima data pertama)

protocol_type	service	flag
0.688862	0.200403	0.101878
0.187767	0.329895	0.207314
0.594997	1.380124	1.330359
0.745850	0.077680	0.149149
0.701232	0.060765	0.148071

```
from sklearn.preprocessing import StandardScaler

train = pd.read_csv("drive/My Drive/Datasets/KDDTrain+
    _Encoded.csv")
X = train.drop('labels', axis=1)
y = train['labels']
test = pd.read_csv("drive/My Drive/Datasets/KDDTest+_Encoded.
    csv")
X_test = test.drop("labels", axis=1)
y_test = test["labels"]
scaler = StandardScaler()
scaler.fit(X, y)
X_scaled = pd.DataFrame(scaler.transform(X), columns=X.
    columns)
X_testscaled = pd.DataFrame(scaler.transform(X_test), columns
    =X_test.columns)
train_scaled = X_scaled.join(y)
test_scaled = X_testscaled.join(y_test)
train_scaled.to_csv("drive/My Drive/Datasets/KDDTrain+
    _StandardScaled.csv", index=False)
test_scaled.to_csv("drive/My Drive/Datasets/KDDTest+
    _StandardScaled.csv", index=False)
```

Gambar 4.6. Kode untuk standardisasi data (dengan StandardScaler).

```
from sklearn.preprocessing import RobustScaler

train = pd.read_csv('drive/My Drive/Datasets/KDDTrain+_Encoded.csv')
X = train.drop('labels', axis=1)
y = train['labels']
test = pd.read_csv("drive/My Drive/Datasets/KDDTest+_Encoded.csv")
X_test = test.drop("labels", axis=1)
y_test = test["labels"]
scaler = RobustScaler()
scaler.fit(X, y)
X_scaled = pd.DataFrame(scaler.transform(X), columns=X.columns)
X_testscaled = pd.DataFrame(scaler.transform(X_test), columns=X_test.columns)
train_scaled = X_scaled.join(y)
test_scaled = X_testscaled.join(y_test)
train_scaled.to_csv("drive/My Drive/Datasets/KDDTrain+_RobustScaled.csv", index=False)
test_scaled.to_csv("drive/My Drive/Datasets/KDDTest+_RobustScaled.csv", index=False)
```

Gambar 4.7. Kode untuk standardisasi data (dengan RobustScaler).

Tabel 4.6. Potongan data latih sebelum standardisasi.

duration	protocol_type	service	flag	src_bytes	dst_bytes	land
0	0.688862	0.200403	0.101878	491	0	0
0	0.187767	0.329895	0.207314	146	0	0
0	0.594997	1.380124	1.330359	0	0	0
0	0.745850	0.077680	0.149149	232	8153	0
0	0.701232	0.060765	0.148071	199	420	0

4.2.3 Transformasi Data

Pada tahap ini dilakukan transformasi data dari vektor berdimensi 41 menjadi matriks berukuran 32×32 dengan cara yang dilakukan Naseer [2]. Pertama, vektor akan diperbanyak sehingga menjadi tiga vektor yang identik. Kemudian, ketiga vektor terse-

Tabel 4.7. Potongan data latih sesudah standardisasi (StandardScaler).

duration	protocol_type	service	flag	src_bytes	dst_bytes	land
-0.110249	0.383088	-0.479971	-0.829797	-0.007679	-0.004919	-0.014089
-0.110249	-2.039474	-0.179530	-0.775430	-0.007737	-0.004919	-0.014089
-0.110249	0.251629	1.335787	1.527352	-0.007762	-0.004919	-0.014089
-0.110249	0.462866	-0.940088	-0.805422	-0.007723	-0.002891	-0.014089
-0.110249	0.400405	-0.947952	-0.805978	-0.007728	-0.004814	-0.014089

Tabel 4.8. Potongan data latih sesudah standardisasi (RobustScaler).

duration	protocol_type	service	flag	src_bytes	dst_bytes	land
0.0	1.080871	-0.111905	-0.127882	1.619565	0.000000	0.0
0.0	-1.364905	0.043030	0.022800	0.369565	0.000000	0.0
0.0	0.622729	1.299610	1.627787	-0.159420	0.000000	0.0
0.0	1.359018	-0.258741	-0.060326	0.681159	15.800388	0.0
0.0	1.141248	-0.278980	-0.061867	0.561594	0.813953	0.0

but digabung sehingga membentuk vektor berdimensi 123. Lalu, lima fitur pertama pada vektor diduplikat dan ditambahkan pada vektor tersebut. Sehingga diperoleh vektor berdimensi 128. Setelah itu, vektor dengan dimensi 128 diperbanyak dan dirubah menjadi matriks berukuran 32×32.

Sebagai contoh, misal $\mathbf{x}$ adalah vektor pada gambar 4.9. Kemudian, $\mathbf{x}$ diduplikat sebanyak tiga kali menjadi vektor berukuran 123 $[0\ 1{,}08087147\ \cdots\ 0\ 0\ 1{,}08087147\ \cdots\ 0\ 0\ 1{,}08087147\ \cdots\ 0]$. Kemudian, lima nilai pertama pada vektor ditambahkan, sehingga didapat vektor berukuran 128,

$$\mathbf{v} = \begin{matrix}[0\ 1{,}08087147\ \cdots\ 0\ 0\ 1{,}08087147 & \cdots & 0 \\ 0\ 1{,}08087147\ \cdots\ 0\ 0\ 1{,}08087147 & -0{,}11190536 & -0{,}12788232\ 1{,}61956522].\end{matrix}$$

Setelah itu, $\mathbf{v}$ diduplikat sebanyak delapan kali, sehingga diperoleh vektor berukuran 1024. Karena $32 \times 32 = 1024$, vektor tersebut dapat dirubah menjadi matriks berukuran 32×32. Tiap komponen pada vektor dituliskan secara urut, hingga pada kolom ke 32,

kemudian dilanjutkan pada baris berikutnya hingga baris ke 32. Sebagai contoh, berikut adalah matriks hasil transformasi (tanda titik tiga digunakan untuk menyingkat penulisan dan tidak menandakan perulangan nilai),

$$M = \begin{bmatrix} 0 & 1{,}08087147 & -0{,}11190536 & \cdots & 0 & -0{,}60693642 \\ -0{,}15510204 & -0{,}35789474 & 0{,}14285714 & \cdots & 0 & -0{,}08510638 \\ -0{,}375 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -0{,}12788232 & 1{,}61956522 \end{bmatrix}.$$

Untuk melakukan transformasi vektor menjadi matriks (image) digunakan fungsi toimage seperti pada gambar 4.8.

```
1  import numpy as np
2
3  def toimage(data):
4          images = []
5          for x in data:
6                  v_bar = x * 3 + x[0:5]
7                  v_arr = v_bar * 8
8                  img = np.reshape(v_arr, (32, 32))
9                  images.append(img)
10         return np.asarray(images)
```

Gambar 4.8. Kode transformasi data.

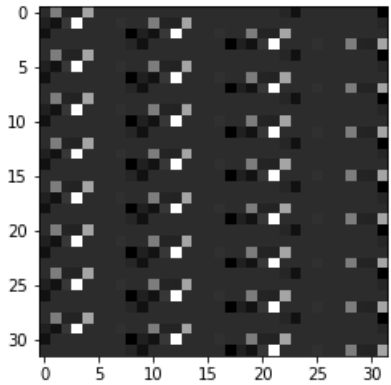
Contoh vektor pada data latih sebelum di transformasi dapat dilihat pada gambar 4.9 dan hasil transformasi vektor tersebut dapat dilihat pada gambar 4.10. Data hasil transformasi hanya digunakan untuk algoritma DCNN.

4.3 Seleksi Fitur

Ada dua metode seleksi fitur yang digunakan dalam penelitian ini. Pertama, seleksi fitur dengan menggunakan algoritma Differential Evolution (DE). Kedua, seleksi fitur dengan menggunakan algoritma binary Particle Swarm Optimization (bPSO) seperti diusulkan oleh Sakr [3]. Untuk metode yang pertama digunakan

```
array([ 0.        ,  1.08087147, -0.11190536, -0.12788232,  1.61956522,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        ,  0.        , -0.08510638, -0.375     ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
        0.        , -0.60693642, -0.15510204, -0.35789474,  0.14285714,
        2.83333333,  0.        ,  0.        ,  0.        ,  0.05      ,
        0.        ,  0.        ])
```

Gambar 4.9. Contoh data latih sebelum transformasi.



Gambar 4.10. Contoh data latih sesudah transformasi.

sebagai bagian tahap praproses untuk sistem deteksi intrusi dengan DE-SVM, sedangkan yang kedua digunakan sebagai bagian tahap praproses untuk PSO-SVM.

4.3.1 Seleksi Fitur dengan DE

Untuk melakukan seleksi fitur digunakan input vektor bernilai biner berdimensi 41. Setiap komponen pada vektor mewakili fitur yang ada pada data. Jika fitur tersebut digunakan, maka komponen vektor bernilai 1. Jika tidak digunakan, maka komponen vektor bernilai 0. Sebagai contoh, sebuah vektor $[1\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0 \cdots 0\ 0\ 1]$ merepresentasikan subhimpunan fitur yang memuat fitur pertama

hingga ke-4, ke-6, ke-8 hingga ke-12, dan ke-41. Terdapat 10 individu dalam populasi dan inisiasi populasi dilakukan secara acak. Faktor mutasi sebesar 0,8 dan tingkat *crossover* 0,7. Iterasi dilakukan sebanyak 100 kali. Data latih dibagi menjadi data latih dan data validasi dengan rasio 80:20. Potongan kode untuk melakukan inisiasi parameter dan populasi untuk seleksi fitur diberikan pada gambar 4.11.

```
import de
class BinaryDE(object):
        def __init__(self, X, y, validation_size, pop_size,
    dimensions, f, cr):
                self.pop_size = max(pop_size, 4)
                self.mutation_factor = f
                self.cross_rate = cr
                self.dimensions = dimensions
                self.validation_size = validation_size
                self.X = X
                self.y = y
                self.population = self.init_pop()

        def init_pop(self):
                pop = numpy.random.randint(low=0, high=2,
    size=(self.pop_size, self.dimensions))
                return pop
bde = de.BinaryDE(X.to_numpy(), y.to_numpy(), validation_size
    =0.2, pop_size=10, dimensions=41, f=0.8, cr=0.7)
```

Gambar 4.11. Kode inisiasi parameter dan populasi DE.

Kemudian, dihitung nilai *fitness* dari setiap individu pada populasi awal. Untuk fungsi *fitness* digunakan akurasi validasi dari model SVM. Untuk menghitung *fitness* tiap individu dalam populasi, digunakan fungsi seperti pada gambar 4.12.

Selanjutnya, untuk setiap iterasi dilakukan mutasi dengan memilih tiga vektor selain vektor target. Kemudian, vektor mutan dibentuk dengan menggunakan persamaan 2.10. Karena vektor input bernilai biner, maka digunakan *threshold* bernilai 0,5 untuk merubah nilai riil yang lebih dari 0,5 menjadi 1 dan yang lain

```
def f_individual(self, x):
        classifier = SVC(kernel='rbf', gamma='auto')
        selected_elements_indices = numpy.where(numpy.asarray
    (x) == 1)[0]
        X_subset = self.X[:, selected_elements_indices]
        if 0 < self.validation_size < 1:
                X_train, X_val, y_train, y_val =
    train_test_split(X_subset, self.y, test_size=self.
    validation_size, random_state=0)
                classifier.fit(X_train, y_train)
                acc = accuracy_score(y_val, classifier.
    predict(X_val))
        f_val = 1.0 - acc
        return f_val

def f(self, pop):
        j = [self.f_individual(pop[i]) for i in range(self.
    pop_size)]

        return numpy.array(j)
```

Gambar 4.12. Kode untuk menghitung nilai *fitness* populasi.

dirubah menjadi 0. Sebagai contoh, misalkan vektor target $\mathbf{x} = [1\ 1\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0 \cdots 0\ 0\ 1]$, dan tiga vektor lainnya adalah $\mathbf{u}_1 = [0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0 \cdots 0\ 0\ 1]$, $\mathbf{u}_2 = [0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0 \cdots 0\ 0\ 1]$, dan $\mathbf{u}_3 = [1\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0 \cdots 0\ 0\ 1]$. Dengan menggunakan persamaan 2.10, diperoleh vektor mutan $\mathbf{v} = \mathbf{u}_1 + 0.8(\mathbf{u}_2 - \mathbf{u}_3) = [-0{,}8\ 0\ 1\ 0{,}2\ 0\ 1\ 0\ 1{,}8\ 1{,}8\ 1{,}8\ 1\ 1\ 0\ 0 \cdots 0\ 0\ 2]$. Karena vektor harus bernilai biner, dengan memetakan semua nilai yang lebih dari 0,5 ke 1 dan memetakan semua nilai selain itu ke 0, diperoleh $\mathbf{v} = [0\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0 \cdots 0\ 0\ 1]$. Potongan kode untuk melakukan mutasi diberikan pada kode 4.13. Setelah didapat vektor mutan, kemudian dilakukan persilangan (*crossover*) antara vektor mutan dengan vektor target dengan menggunakan persamaan 2.11. Hasil dari persilangan ini adalah satu vektor trial. Persilangan dimulai dengan membangkitkan satu bilangan acak untuk setiap komponen vektor. Apabila nilai bilangan acak tersebut kurang dari tingkat

```
def mutation(self, target_index):
        indices = [index for index in range(self.pop_size) if
     index != target_index]
        v1, v2, v3 = self.population[numpy.random.choice(
    indices, 3, replace=False)]
        mutant = v1 + self.mutation_factor * (v2 - v3)
        mutant = mutant > 0.5
        return mutant
```

Gambar 4.13. Kode untuk melakukan mutasi.

persilangan (*crossover rate*), maka nilai pada vektor mutan yang akan digunakan. Jika tidak, maka nilai vektor target pada komponen tersebut tetap. Sebagai contoh, untuk vektor target dan vektor mutan yang diberikan sebelumnya, misal bilangan acak yang dibangkitkan adalah 0,5 untuk indeks pertama dan 0,6 untuk indeks kedua, selain itu semua bilangan acak yang dibangkitkan bernilai 1. Diperoleh, vektor trial $\mathbf{u} = [0\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ \cdots\ 0\ 0\ 1]$. Potongan kode untuk melakukan persilangan diberikan pada gambar 4.14.

```
def crossover(self, target, mutant):
        trial = numpy.empty(self.dimensions)

        for j in range(self.dimensions):
                if numpy.random.rand() < self.cross_rate or j
     == numpy.random.randint(0, self.dimensions):
                        trial[j] = mutant[j]
                else:
                        trial[j] = target[j]
        return trial
```

Gambar 4.14. Kode untuk melakukan persilangan.

Kemudian, dilakukan seleksi untuk menentukan vektor target atau vektor trial yang akan digunakan untuk iterasi selanjutnya. Jika nilai *fitness* vektor target lebih kecil daripada nilai *fitness* vektor trial, maka vektor trial yang akan lanjut ke iterasi berikutnya. Sebagai

contoh, misalkan nilai *fitness* vektor target adalah 0.75, sedangkan nilai *fitness* vektor trial adalah 0.83. Berarti vektor yang akan lanjut ke generasi berikutnya adalah vektor trial, menggantikan vektor target. Potongan kode untuk melakukan seleksi diberikan pada gambar 4.15.

```
1 trial_fit = self.f_individual(trial)
2 if trial_fit <= f_vals[target_index]:
3         self.population[target_index] = trial
4         f_vals[target_index] = trial_fit
```

Gambar 4.15. Kode untuk melakukan seleksi.

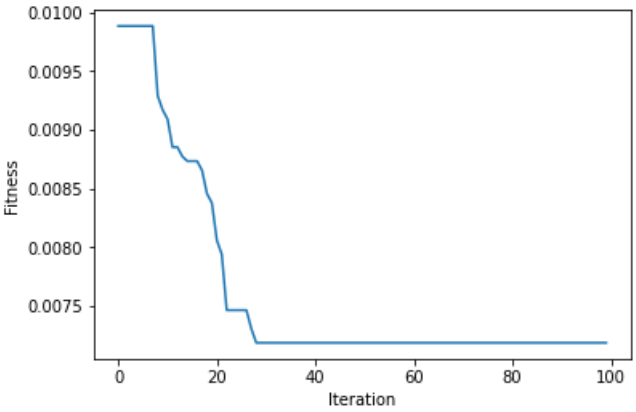
Setelah semua iterasi selesai, dihitung nilai *fitness* pada generasi terakhir dan dicari yang terbaik. Potongan keluaran dari proses seleksi fitur ditunjukkan pada gambar 4.16. Banyak fitur yang terpilih dengan menggunakan DE sebanyak 23 fitur dan diberikan pada tabel 4.9. Kurva perubahan nilai *fitness* terbaik pada setiap generasi ditunjukkan pada gambar 4.17. Untuk skrip secara lengkap diberikan di lampiran B

```
Optimization started  2020-06-23 05:33:46.959529
Generation  1
Best Solution:  [1 1 1 1 0 1 0 1 1 1 1 1 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1
 1 0 1 1]
Best Fitness Value:  0.00988291327644375
Generation  2
Best Solution:  [1 1 1 1 0 1 0 1 1 1 1 1 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1
 1 0 1 1]
Best Fitness Value:  0.00988291327644375
Generation  3
Best Solution:  [1 1 1 1 0 1 0 1 1 1 1 1 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1
 1 0 1 1]
Best Fitness Value:  0.00988291327644375
Generation  4
Best Solution:  [1 1 1 1 0 1 0 1 1 1 1 1 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1
 1 0 1 1]
Best Fitness Value:  0.00988291327644375
Generation  5
Best Solution:  [1 1 1 1 0 1 0 1 1 1 1 1 0 1 1 1 0 0 0 0 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 1
 1 0 1 1]
Best Fitness Value:  0.00988291327644375
```

Gambar 4.16. Potongan keluaran proses seleksi fitur dengan DE.

Tabel 4.9. Fitur yang Terpilih dengan DE.

Fitur Terpilih
duration, protocol_type, service, wrong_fragment, urgent, hot, logged_in, su_attempted, num_file_creations, num_shells, is_host_login, count, srv_serror_rate, srv_rerror_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_srv_serror_rate, dst_host_rerror_rate



Gambar 4.17. Kurva perubahan nilai *fitness* terbaik.

4.3.2 Seleksi Fitur dengan bPSO

Algoritma bPSO digunakan untuk seleksi fitur seperti dilakukan oleh Sakr [3] dan diimplementasikan menggunakan *library* pyswarms [14]. Parameter yang digunakan diberikan pada tabel

4.10. Untuk parameter banyak tetangga yang diperhitungkan dan

Tabel 4.10. Nilai Parameter yang Digunakan untuk bPSO.

Parameter	Nilai
Banyak partikel	40
Dimensi	41
Banyak iterasi	30
Inersia	0,7
Kognitif	0,8
Sosial	0,9
Banyak tetangga	40
Minkowski p-norm	2

norm yang digunakan tidak diberikan pada paper milik Sakr. Norm yang digunakan adalah norm L2 untuk menghitung jarak Euclid. Dipilih norm ini karena input berada pada ruang vektor Euclid, $\mathbb{R}^n$ dengan $n = 41$. Untuk mencapai solusi global terbaik banyak tetangga yang diperhitungkan sebanyak partikel yang ada. Fungsi *cost* yang digunakan adalah akurasi klasifikasi. Kurva pergerakan nilai *cost* terbaik diberikan pada gambar 4.19. Dalam implementasi seleksi fitur dengan bPSO digunakan 80% data untuk melatih model dan 20% data untuk evaluasi model. Nilai akurasi yang digunakan adalah akurasi pada data validasi. Potongan kode untuk melakukan seleksi fitur dengan bPSO diberikan pada gambar 4.18.

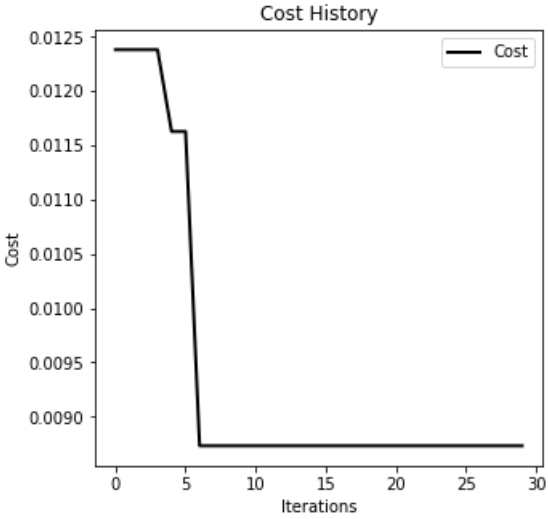
Dengan menggunakan algoritma bPSO diperoleh 24 fitur. Fitur yang terpilih dengan menggunakan algoritma ini diberikan pada tabel 4.11.

```
classifier = SVC(kernel='rbf', gamma='auto')

def f_per_particle(m, alpha=0.5):
  total_features = 41

  if np.count_nonzero(m) == 0:
    X_subset = X
  else:
    selected_elements_indices = np.where(m == 1)[0]
    X_subset = X[:, selected_elements_indices]

  classifier.fit(X_subset, y)
  P = accuracy_score(y, classifier.predict(X_subset))

  return 1.0 - P

def f(x):
  n_particles = x.shape[0]
  j = [f_per_particle(x[i]) for i in range(n_particles)]
  return np.array(j)

bpso = BinaryPSO(n_particles=40, dimensions=41, options={'w'
    :0.7, 'c1':0.8, 'c2':0.9, 'k':40, 'p':2})
cost, pos = bpso.optimize(f, iters=30)
```

Gambar 4.18. Kode untuk seleksi fitur dengan bPSO.

4.4 Implementasi SVM

Model SVM yang digunakan adalah SVC (*Support Vector Classifier*) dari *library* ThunderSVM [15]. Model SVM yang digunakan adalah SVM non-linier. Dipilih SVM non-linear karena dari gambar 4.4 terlihat bahwa data tidak dapat terpisah secara linier. Parameter yang digunakan adalah parameter *default* (C = 1 dan gamma = 'auto') dengan kernel RBF (*Radial Basis Function*). Model dilatih pada data latih hasil *encoding* dan standardisasi dengan StandardScaler tanpa reduksi fitur. Potongan kode untuk melatih dan menguji model SVM diberikan pada gambar 4.20.



Gambar 4.19. Kurva perubahan nilai *cost* terbaik (bPSO).

Tabel 4.11. Fitur Terpilih dengan bPSO.

Fitur Terpilih
duration, protocol_type, service, src_bytes, wrong_fragment, hot, logged_in, num_compromised, num_root, num_outbound_cmds, is_guest_login, count, serror_rate, rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate, dst_host_count, dst_host_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate

```
train_data = pd.read_csv("/content/drive/My Drive/Datasets/
    KDDTrain+_StandardScaled.csv")
test_data = pd.read_csv("/content/drive/My Drive/Datasets/
    KDDTest+_StandardScaled.csv")

X = train_data.drop('labels', axis=1)
y = train_data['labels']

X_test = test_data.drop("labels", axis=1)
y_test = test_data['labels']

clf = SVC(kernel='rbf', gamma='auto')
clf.fit(X, y)
y_pred = clf.predict(X_test)
```

Gambar 4.20. Kode untuk melakukan *training* dan *testing* model SVM.

4.5 Implementasi DE-SVM

4.5.1 Desain Algoritma DE-SVM

Desain dari algoritma hybrid yang diusulkan adalah sebagai berikut.

1. **Input**
   Data input yang digunakan adalah data latih hasil standardisasi yang kemudian dibagi menjadi data latih dan validasi. Digunakan pasangan terurut (C, γ) sebagai individu dalam populasi. Parameter C dan γ merupakan parameter dari SVM. Input dari algoritma DE, selain data, adalah besar populasi, faktor mutasi, tingkat persilangan, dan batas-batas dari parameter C dan γ

2. **Mutasi**
   Kemudian, dimulai proses pencarian nilai optimal dari C dan γ dengan menggunakan DE. DE dimulai dengan menginisiasi populasi awal secara acak. Kemudian untuk setiap vektor

target dalam populasi, dipilih tiga vektor lain untuk mutasi. Pada tahap ini, akan dihasilkan satu vektor mutan. Proses mutasi sama seperti pada subbab 4.3.1.

3. ***Persilangan***
   Pada tahap ini, vektor mutan dan satu vektor target yang dipilih dari populasi akan disilangkan. Kemudian, dari tahap ini akan dihasilkan satu vektor hasil persilangan, yang disebut vektor *trial*. Proses persilangan juga sama seperti pada subbab 4.3.1.

4. ***Training* SVM**
   Setelah mendapatkan vektor *trial*, kemudian akan dilakukan *training* model SVM menggunakan nilai pada vektor target dan vektor *trial*.

5. **Validasi**
   Kemudian, model yang sudah di *training* akan digunakan untuk melakukan klasifikasi pada data validasi.

6. **Evaluasi nilai *fitness* dan seleksi**
   Pada tahap ini, akan dibandingkan hasil prediksi (klasifikasi) dari SVM dengan label pada data validasi. Kemudian, akan dihitung akurasi dari model yang dibuat. Akurasi ini yang akan dijadikan nilai *fitness* dari vektor target dan vektor *trial*. Lalu, diambil satu vektor yang memiliki nilai *fitness* lebih kecil untuk dimasukkan ke generasi selanjutnya. Proses mutasi, *crossover*, hingga seleksi akan dilakukan berulang kali hingga mencapai batas iterasi.

7. **Output Differential Evolution**
   Kemudian akan didapat output dari DE, yaitu vektor yang memiliki nilai *fitness* paling baik. Sehingga, pada tahap ini diperoleh nilai optimal untuk parameter C dan γ.

8. **Melatih model SVM**
   Lalu, dilakukan *training* model SVM dengan menggunakan nilai optimal untuk parameter C dan γ yang diperoleh sebelumnya. Data yang digunakan merupakan gabungan dari data latih dan validasi. Dari tahap ini, akan dihasilkan satu model yang memiliki performa paling baik. Selanjutnya, model ini yang akan digunakan untuk melakukan deteksi intrusi.

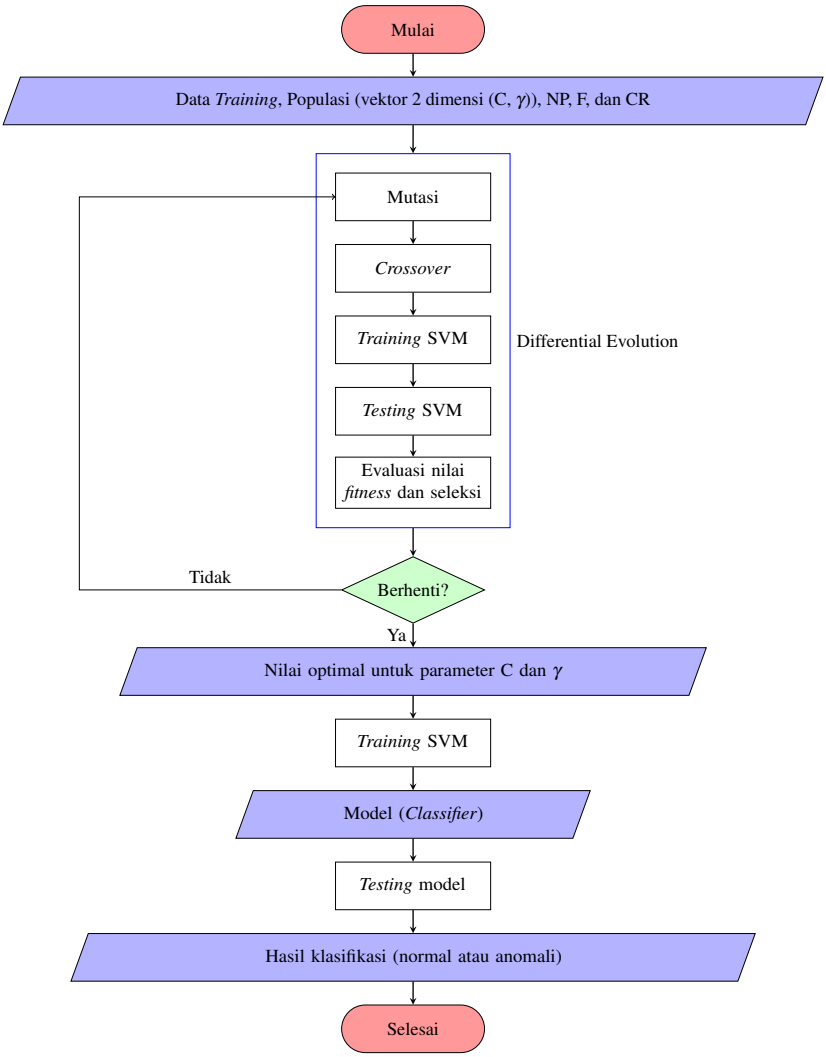
9. **Uji model**
   Kemudian, model yang sudah dilatih tersebut akan digunakan untuk melakukan klasifikasi pada data uji.

10. **Hasil Klasifikasi**
    Dari tahap pengujian, diperoleh hasil apakah data yang diberikan menandakan aktivitas yang normal atau anomali (intrusi).

Diagram alir dari desain algoritma yang diusulkan diberikan pada gambar 4.21.

4.5.2 Hasil Implementasi Algoritma DE-SVM

Kode yang digunakan untuk mengimplementasikan DE seperti desain yang diusulkan diberikan pada lampiran B pada bagian *class* DifferentialEvolution. Implementasi algoritma ini mirip dengan saat melakukan seleksi fitur, hanya tidak perlu merubah komponen vektor menjadi biner. Dalam penelitian ini, dilakukan implementasi DE-SVM dengan beberapa parameter berbeda. Untuk nilai parameter besar populasi diambil tetap 10 dan nilai parameter tingkat persilangan diambil tetap 0,9. Banyak iterasi yang dilakukan juga tetap 100 iterasi. Parameter batas pencarian diberikan sebagai pasangan terurut (batas parameter C, batas parameter gamma). Batas parameter diberikan sebagai selang tertutup [batas minimum, batas maksimum]. Hasil akurasi (dalam persen) dari tiap percobaan diberikan pada tabel 4.12.



Gambar 4.21. Diagram alir desain algoritma hybrid DE-SVM yang diusulkan.

Tabel 4.12. Nilai Parameter yang Digunakan untuk DE-SVM dan Akurasi.

Faktor Mutasi	Batas Pencarian	Akurasi Latih	Akurasi Uji
0,5	([0,01, 100], [0,01, 100])	99,45	79,42
0,7	([0,01, 100], [0,01, 100])	99,72	75,91
0,5	([0,001, 1000], [0,001, 1000])	99,74	76,84
0,5	([1, 4000], [0, 30])	99,89	80,26

Dari tabel 4.12, diperoleh bahwa dengan meningkatkan faktor mutasi, hasil parameter yang diperoleh menyebabkan model SVM mengalami *overfit*. Sebab nilai akurasi latih berhasil meningkat, tetapi menyebabkan akurasi uji menurun. Sehingga untuk percobaan ketiga dan keempat digunakan nilai faktor mutasi 0,5. Kemudian, batas pencarian diperluas dan diperoleh bahwa akurasi model pada data uji tetap menurun dibandingkan percobaan pertama. Lalu, batas atas parameter C diperbesar menjadi 4000 dan batas bawah diperbesar jadi 1. Sedangkan, batas parameter gamma dirubah menjadi [0, 30]. Hasilnya, diperoleh bahwa akurasi model SVM berhasil meningkat baik pada data latih maupun data uji. Sehingga, dipilih nilai parameter pada percobaan keempat karena hasil akurasi ujinya paling baik. Potongan kode untuk melakukan optimasi parameter SVM dengan DE diberikan pada gambar 4.22.

Potongan keluaran dari proses optimasi parameter dengan DE diberikan pada gambar 4.23. Kurva perubahan nilai *fitness* terbaik

```
import de

de_optimizer = de.DifferentialEvolution(X, y, validation_size
    =0.2, pop_size=10, dimensions=2, bounds=[(1, 4000), (0,
    30)], f=0.5, cr=0.9)
best_sol, best_fit = de_optimizer.optimize(iteration=100)
```

Gambar 4.22. Kode untuk melakukan optimasi parameter SVM dengan DE.

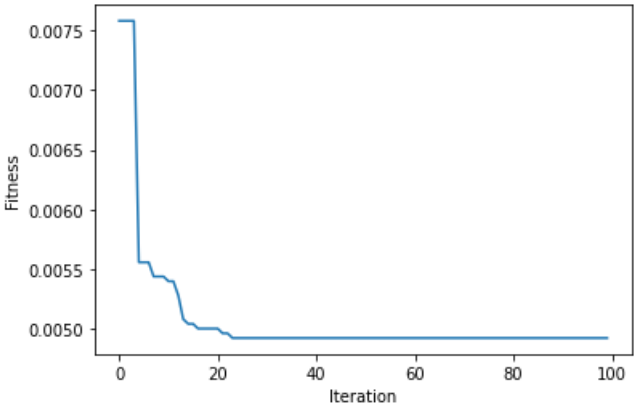
pada setiap generasi diberikan pada gambar 4.24. Dengan menggunakan DE untuk melakukan optimasi parameter SVM, diperoleh bahwa nilai parameter C yang optimal adalah 3686,31834 dan nilai parameter gamma yang optimal adalah 0,121845318.

```
Optimization started  2020-06-25 12:04:47.864256
Generation  1
Best Solution:  [2.87051879e+03 1.36485843e+00]
Best Fitness Value:  0.007580869220083386
Generation  2
Best Solution:  [2.87051879e+03 1.36485843e+00]
Best Fitness Value:  0.007580869220083386
Generation  3
Best Solution:  [2.87051879e+03 1.36485843e+00]
Best Fitness Value:  0.007580869220083386
Generation  4
Best Solution:  [2.87051879e+03 1.36485843e+00]
Best Fitness Value:  0.007580869220083386
Generation  5
Best Solution:  [3.8492857e+03 2.1527543e-01]
Best Fitness Value:  0.005556658067076836
```

Gambar 4.23. Potongan keluaran proses optimasi parameter SVM dengan DE.

4.6 Implementasi PSO-SVM

Data yang digunakan untuk melatih model adalah data KDDTrain+ yang sudah dirubah menjadi data numerik (*encoding*), distandardisasi dengan StandardScaler, dan sudah direduksi menggunakan



Gambar 4.24. Kurva perubahan nilai *fitness* terbaik pada proses optimasi parameter SVM dengan DE.

bPSO seperti pada bagian 4.3.2. Untuk melakukan optimasi parameter SVM digunakan *standard* PSO (sPSO) dengan domain bilangan riil pada selang [1, 4000] untuk parameter C dan pada selang [0, 30] untuk parameter gamma. Untuk parameter PSO yang digunakan diberikan pada tabel 4.13. Fungsi objektif yang digunakan adalah 1 − akurasi klasifikasi. Potongan kode untuk melakukan optimasi pa-

Tabel 4.13. Nilai Parameter yang Digunakan untuk sPSO.

Parameter	**Nilai**
Banyak partikel	30
Dimensi	2
Banyak iterasi	40
Inersia	0,7
Kognitif	0,8
Sosial	0,9

rameter SVM dengan PSO menggunakan pyswarms [14] diberikan pada gambar 4.25.

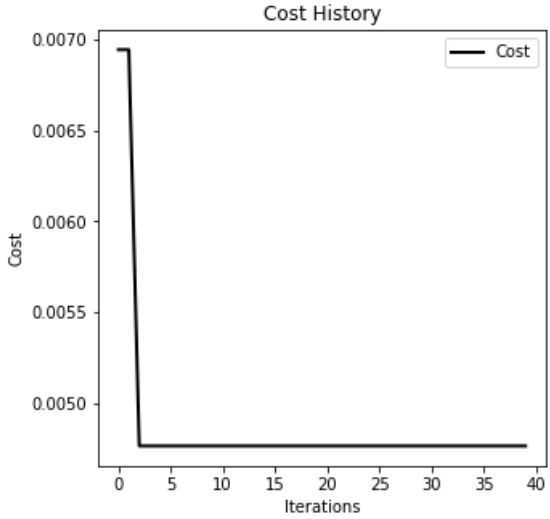
```
from pyswarms.single.global_best import GlobalBestPSO

X_train, X_val, y_train, y_val = train_test_split(X_subset, y
    , test_size=0.2, random_state=0)

def obj_func(x):
  n_particles = x.shape[0]
  fitness = np.empty(n_particles)
  for i in range(n_particles):
        clf = SVC(kernel='rbf', C=x[i][0], gamma=x[i][1])
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_val)
        fitness[i] = 1-accuracy_score(y_val, y_pred)
  return fitness

spso = GlobalBestPSO(n_particles=30, dimensions=2, options={'
    w':0.7, 'c1':0.8, 'c2':0.9}, bounds=([1, 0], [4000, 30])
    )
cost, pos = spso.optimize(obj_func, iters=40)
```

Gambar 4.25. Kode untuk optimasi parameter SVM dengan PSO.

Dengan menggunakan PSO untuk melakukan optimasi parameter SVM, diperoleh nilai optimal untuk parameter C adalah 355,126680 dan nilai optimal untuk parameter gamma adalah 0,0945191944. Kurva perubahan nilai *cost* terbaik pada setiap iterasi diberikan pada gambar 4.26.

4.7 Implementasi DCNN

Input dari algoritma ini adalah data latih yang sudah di standardisasi dan transformasi seperti ditunjukkan pada subbab 4.2.2 dan 4.2.3. Untuk mengimplementasikan algoritma ini digunakan *library* Keras dengan TensorFlow sebagai *backend*. Arsitektur dari model DCNN yang digunakan mengikuti arsitektur yang diusulkan oleh Naseer [2]. Semua parameter dan tipe serta ukuran *layer* dan



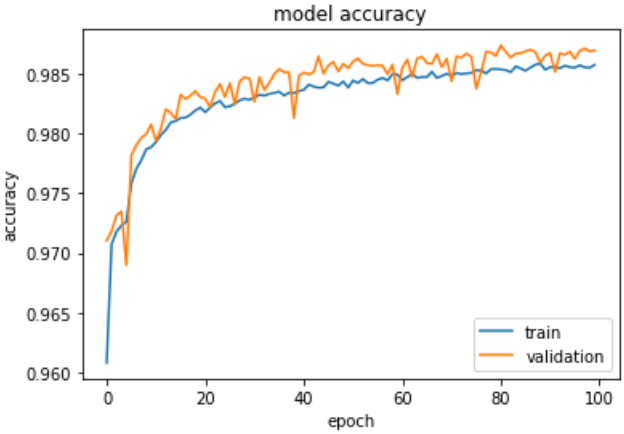
Gambar 4.26. Kurva perubahan nilai *cost* terbaik pada proses optimasi parameter SVM dengan PSO.

kernel mengikuti arsitektur milik Naseer. Tetapi, fungsi *loss* tidak diberikan, sehingga digunakan fungsi *mean squared error*. Supaya hasil yang diperoleh tetap sama dalam tiap percobaan, digunakan *seed* untuk pembangkit bilangan acak pada NumPy, Python, dan TensorFlow. Untuk NumPy dan Python digunakan angka 123 dan untuk TensorFlow digunakan angka 1234 sebagai *seed*. Potongan kode untuk membuat model DCNN dengan Keras diberikan pada gambar 4.27.

Kemudian, model dilatih dengan 80% data dari data latih (gabungan KDDTrain+ dan KDDTrain+_20Percent) dan divalidasi dengan 20% data dari data latih. Pertama, model dilatih selama 100 *epoch* dan diperoleh kurva akurasi pada data latih dan validasi seperti pada gambar 4.28. Kemudian, karena pada beberapa *epoch* akurasi validasi turun dibawah akurasi latih, model dilatih ulang

```
model = Sequential()
model.add(Dense(1, activation='softsign', kernel_initializer=
    'he_normal', input_shape=(32, 32, 1)))
model.add(Conv2D(16, kernel_size=1, kernel_initializer='
    he_normal', padding='valid', activation='softsign'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=1, kernel_initializer='
    he_normal', padding='valid', activation='softsign'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(4096))
model.add(Dense(128))
model.add(Dense(16))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='mean_squared_error', optimizer='adadelta'
    , metrics=['accuracy'])
model.summary()
```

Gambar 4.27. Kode inisiasi model DCNN.



Gambar 4.28. Kurva perubahan akurasi *training* dan validasi model DCNN.

selama 60 dan 49 *epoch*. Hasilnya, model berhasil mencapai akurasi yang lebih tinggi pada data uji. Sehingga, muncul dugaan bahwa model mengalami *overfitting*. Kemudian, dicoba untuk melatih ulang model selama 20 *epoch*. Setelah dilatih, model digunakan untuk memprediksi data uji (KDDTest+). Akurasi model (dalam persen) setelah 20, 49, 60, dan 100 *epoch* dapat dilihat pada tabel 4.14. Dari

Tabel 4.14. Akurasi Algoritma DCNN.

Iterasi	Latih	Validasi	Uji
20	98,21	98,32	78,12
49	98,45	98,37	79,77
60	98,50	98,28	77,02
100	98,58	98,70	77,16

tabel 4.14 dapat dilihat bahwa model mengalami penurunan akurasi pada data uji setelah iterasi ke-49, padahal akurasi data latih meningkat. Hal ini berarti model mengalami *overfitting* setelah iterasi ke-49 dan proses *training* perlu dihentikan pada iterasi ke-49 untuk menghindari *overfitting*. Tetapi, bila dilihat dari kurva pada gambar 4.28 pada sekitar *epoch* ke-5 tampak ada penurunan akurasi validasi. Sehingga, kemudian dicoba untuk menghentikan proses latih pada *epoch* ke-5 dan menambah 1 *epoch* secara bertahap. Hasilnya, pada *epoch* ke-12 diperoleh akurasi uji 81,99% dan kemudian akurasi model mulai menurun lagi setelah *epoch* ke-12. Sehingga, diputuskan bahwa proses latih model dihentikan pada *epoch* ke-12. Untuk waktu komputasi pada proses latih selama 12 iterasi adalah 324 detik dan waktu komputasi pada proses *testing* adalah 2 detik. Akurasi model yang diperoleh sedikit berbeda dengan paper milik Naseer [2], karena parameter pada proses *encoding* bisa jadi berbeda (Naseer tidak memberikan parameter untuk *encoding* dengan LeaveOneOutEncoder).

4.8 Evaluasi Hasil

Pada bagian ini, dibahas hasil dari implementasi algoritma SVM, DE-SVM, PSO-SVM, dan DCNN yang sudah dijelaskan pada subbab-subbab sebelumnya.

4.8.1 Akurasi

Akurasi tiap model (dalam persen) dalam melakukan deteksi intrusi diberikan pada tabel 4.15. Hasil akurasi model pada data uji lebih rendah dibandingkan pada data latih dan validasi. Hal ini dikarenakan tipe serangan ps, sqlattack, mailbomb, httptunnel, processtable, snmpgetattack, xsnoop, xterm, worm, snmpguess, mscan, apache2, saint, named, udpstorm, sendmail, dan xlock tidak terdapat pada data latih. Sehingga, model tidak pernah mengenal dan mempelajari karakteristik dari data serangan tersebut. Dengan menggunakan data uji ini, dapat dinilai performa model dalam melakukan deteksi intrusi yang belum pernah terjadi sebelumnya (belum diketahui model). Dari tabel 4.15, model SVM yang dioptimasi dengan menggunakan DE memiliki akurasi lebih tinggi pada data uji dibandingkan model SVM dan sedikit lebih rendah dibandingkan model PSO-SVM. Model DCNN memiliki akurasi terbaik pada data uji, meskipun pada data latih akurasinya paling rendah.

Tabel 4.15. Akurasi Model

Nama Model	Akurasi Latih	Akurasi Uji
SVM	99,24	78,22
SVM + seleksi fitur (DE)	99,86	78,29
SVM + seleksi fitur (PSO)	99,17	77,95
DE-SVM	99,90	80,26
PSO-SVM	99,77	77,09
DCNN	98,06	81,99

4.8.2 Waktu Komputasi

Setiap model menggunakan GPU baik untuk proses *training* maupun *testing*. Waktu komputasi (dalam detik) dari tiap model diberikan pada tabel 4.16. Dari tabel tersebut, model DCNN memiliki waktu paling lama untuk proses latih dan proses uji. Untuk model DE-SVM dan PSO-SVM waktu komputasi yang diberikan merupakan waktu komputasi pada saat proses latih dan uji model setelah dioptimasi. Setelah dilakukan seleksi fitur, baik dengan DE maupun PSO, waktu komputasi dari kedua model SVM turun dibandingkan dengan sebelum seleksi fitur. Akan tetapi, dengan parameter hasil optimasi, waktu komputasi kedua model SVM kembali naik. Bahkan, untuk model SVM yang dioptimasi dengan DE dan PSO, waktu komputasinya lebih lambat daripada model SVM sebelum seleksi fitur.

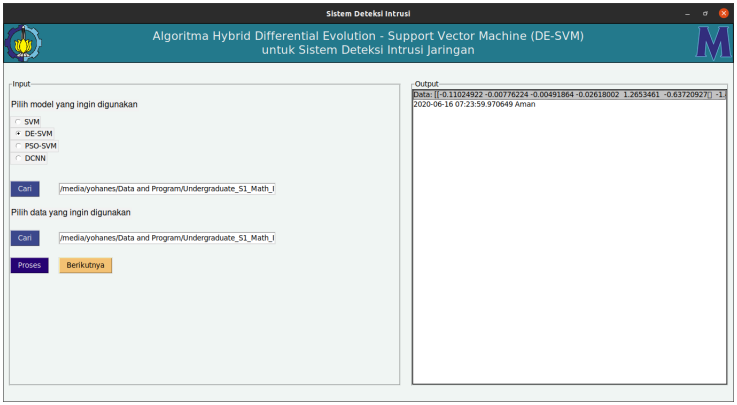
Tabel 4.16. Waktu Komputasi Model

Nama Model	Waktu Latih	Waktu Uji
SVM	2,40	0,28
SVM + seleksi fitur (DE)	1,00	0,10
SVM + seleksi fitur (PSO)	1,56	0,18
DE-SVM	11,41	0,07
PSO-SVM	3,59	0,08
DCNN	324	2

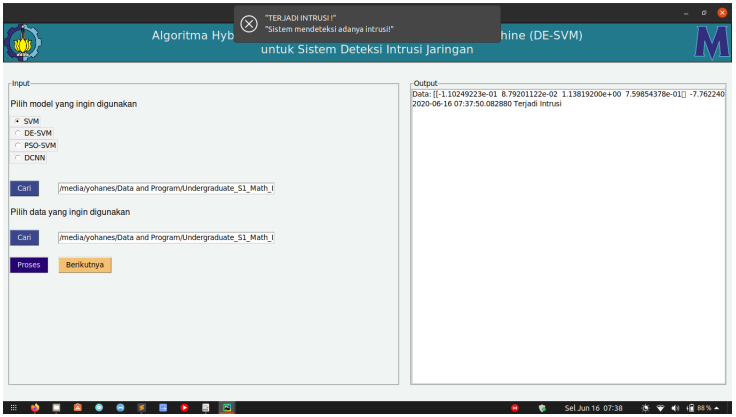
4.9 Simulasi Sistem Deteksi Intrusi

Setelah mendapatkan model yang sudah dilatih (SVM, DE-SVM, PSO-SVM, dan DCNN), model tersebut akan disimpan dan digunakkan untuk melakukan simulasi deteksi intrusi. Data yang digunakan untuk melakukan simulasi adalah data uji. Program untuk

melakukan simulasi dibuat dengan menggunakan *framework* tkinter pada Python. Untuk desain antarmuka dari program simulasi dapat dilihat pada gambar 4.29.



Gambar 4.29. Desain antarmuka sistem deteksi intrusi.



Gambar 4.30. Tampilan program bila terjadi intrusi.

Pada program ini, pengguna dapat memilih empat jenis mo-

del yang ingin digunakan untuk melakukan deteksi intrusi (SVM, PSO-SVM, DE-SVM, dan DCNN). Setelah memilih jenis model, pengguna memilih sebuah model yang sudah disimpan. Untuk model SVM, PSO-SVM, dan DE-SVM harus merupakan data model yang disimpan menggunakan modul pickle pada Python. Sedangkan, untuk model DCNN harus merupakan data model yang disimpan menggunakan fungsi *save* dari Keras. Setelah memilih model, pengguna harus memilih data yang akan dideteksi. Data tersebut harus dalam format CSV (*Comma Separated Value*) dan memiliki fitur yang sama seperti pada data NSL-KDD. Apabila pengguna sudah memasukkan model dan data yang ingin digunakan, pengguna dapat menekan tombol "Proses" untuk memulai proses simulasi. Program akan membaca data pertama dari berkas data yang dipilih dan menentukan apakah data tersebut merupakan intrusi atau bukan. Jika merupakan intrusi, maka program akan mengeluarkan pemberitahuan seperti pada gambar 4.30. Apabila pengguna ingin melanjutkan mendeteksi data berikutnya, maka pengguna dapat menekan tombol "Berikutnya" untuk memproses data selanjutnya.

BAB V
KESIMPULAN DAN SARAN

Pada bab ini, diberikan kesimpulan berdasarkan hasil implementasi pada bab IV. Selain itu, juga diberikan saran untuk penelitian selanjutnya.

5.1 Kesimpulan

Berdasarkan hasil implementasi dari SVM, DE-SVM, PSO-SVM, dan DCNN yang dibahas pada bab IV, diperoleh beberapa kesimpulan sebagai berikut.

1. Sistem deteksi intrusi dengan menggunakan DE-SVM memiliki akurasi 99,90% pada data latih dan memiliki akurasi 80,26% pada data uji. Waktu latih model DE-SVM adalah 11,41 detik dengan menggunakan GPU dan waktu uji model adalah 0,07 detik.

2. Model DE-SVM mampu mencapai akurasi uji yang lebih tinggi dibandingkan model SVM yang tidak dioptimasi. Model DE-SVM memiliki akurasi lebih rendah dari model DCNN pada data uji. Model DE-SVM memiliki waktu latih yang paling lama dibandingkan SVM dan PSO-SVM. Tetapi, waktu uji dari DE-SVM lebih cepat dari SVM. Model DE-SVM mampu melakukan proses latih dan uji dengan jauh lebih cepat dibandingkan dengan model DCNN. Seleksi fitur dengan menggunakan DE mampu mencapai konvergensi lebih cepat dari bPSO. Dengan 10 individu dalam populasi, DE mampu mencapai konvergensi pada iterasi ke-30. Artinya, setelah melakukan evaluasi *fitness* sebanyak 300 kali dapat diperoleh fitur yang optimal. Sedangkan dengan menggunakan bPSO diperlukan 400 kali evaluasi *fitness* untuk mencapai konvergensi.

5.2 Saran

Penelitian Tugas Akhir ini masih memiliki kekurangan. Oleh karena itu, disarankan beberapa hal yang dapat dilakukan untuk penelitian selanjutnya.

1. Memperluas daerah pencarian dalam proses optimasi parameter C dan gamma.
2. Menggunakan varian lain dari algoritma Differential Evolution untuk proses optimasi atau *tuning* parameter dari SVM.
3. Meninjau pengaruh setiap parameter dari algoritma DE terhadap hasil optimasi parameter SVM dan hasil seleksi fitur.

DAFTAR PUSTAKA

[1] Yu, Z., dan Jeffrey J. P. T. 2011. **Intrusion Detection: A Machine Learning Approach**. London, UK: Imperial College Press, vol. 3.

[2] Naseer, S., 2018. "Enhanced Network Anomaly Detection Based on Deep Neural Networks". **IEEE Access**, vol. 6, hal. 48 231–48 246. DOI: 10.1109/ACCESS.2018.2863036.

[3] Sakr, M., Medhat T., dan Ashraf E. S., Mar. 2019. "Network Intrusion Detection System based PSO-SVM for Cloud Computing". **International Journal of Computer Network and Information Security**, vol. 11, hal. 22–29. DOI: 10.5815/ijcnis.2019.03.04.

[4] Sailaja, M., R. Kiran K., dan P. Sita Rama M., 2011. "Intrusion Detection Model based on Differential Evolution". **International Journal of Computer Applications**, vol. 36, no. 6, hal. 10–13. DOI: 10.5120/4494-6328.

[5] Scarfone, K., dan Peter M. Feb. 2007. **NIST Special Publication 800-94, Guide to Intrusion Detection and Prevention Systems (IDPS)**.

[6] Zhang, O. 2015. **Tips for data science competitions**. <URL: https://www.slideshare.net/OwenZhang2/tips-for-data-science-competitions>.

[7] Jundong, L., Jan. 2016. "Feature Selection: A Data Perspective". **ACM Computing Surveys**, vol. 50. DOI: 10.1145/3136625.

[8] Rojas-Domínguez, A., 2018. "Optimal Hyper-Parameter Tuning of SVM Classifiers With Application to Medical Diagnosis". **IEEE Access**, vol. 6, hal. 7164–7176, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2779794.

[9] Deng, N., Yingjie T., dan Chunhua Z. 2012. **Support Vector Machines: Optimization Based Theory, Algorithms, and Extensions**. Boca Raton, FL: CRC Press, ISBN: 978-1-4398-5793-9.

[10] Storn, R., dan Kenneth P., Des. 1997. "Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces". **Journal of Global Optimization**, vol. 11, no. 4, hal. 341–359, ISSN: 1573-2916. DOI: 10.1023/A:1008202821328.

[11] Oliphant, T. E. 2006. **A guide to NumPy**. Trelgol Publishing USA, vol. 1.

[12] McKinney, W. 2010. "Data Structures for Statistical Computing in Python". **Proceedings of the 9th Python in Science Conference**, hal. 56–61. Diedit oleh S. van der Walt dan J. Millman. DOI: 10.25080/Majora-92bf1922-00a.

[13] Pedregosa, F., Gaél V., Alexandre G., Vincent M., Bertrand T., Olivier G., Mathieu B., Peter P., Ron W., Vincent D., Jake V., Alexandre P., David C., Matthieu B., Matthieu P., dan Édouard D., 2011. "Scikit-learn: Machine Learning in Python". **Journal of Machine Learning Research**, vol. 12, no. 85, hal. 2825–2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

[14] Miranda, L. J. V., 21 2018. "PySwarms, a research-toolkit for Particle Swarm Optimization in Python". **Journal of Open Source Software**, vol. 3. DOI: 10.21105/joss.00433.

[15] Wen, Z., Jiashuai S., Qinbin L., Bingsheng H., dan Jian C., 2018. "ThunderSVM: A Fast SVM Library on GPUs and CPUs". **Journal of Machine Learning Research**, vol. 19, hal. 797–801.

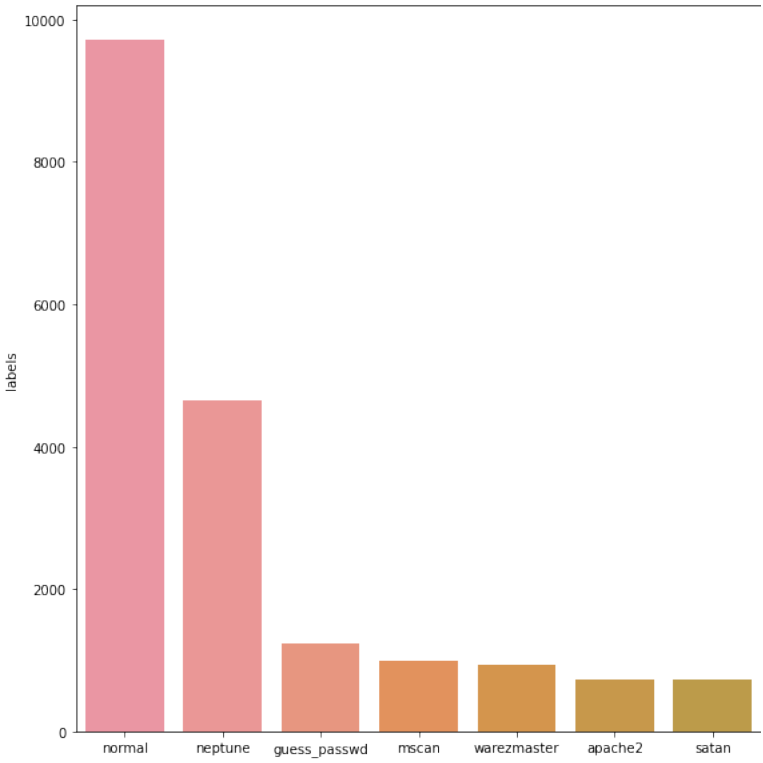
[16] Chollet, F. 2015. **Keras**. <URL: https://keras.io/>.

[17] Martín Abadi. 2015. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. Perangkat lunak tersedia di tensorflow.org. <URL: https://www.tensorflow.org/>.

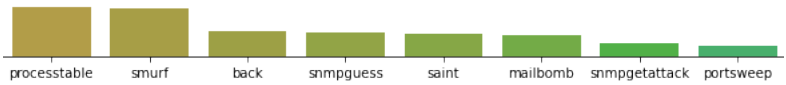
[18] Tavallaee, M., Ebrahim B., Wei L., dan Ali G., Juli 2009. "A detailed analysis of the KDD CUP 99 data set". **IEEE Symposium. Computational Intelligence for Security and Defense Applications, CISDA**, vol. 2. DOI: 10.1109/CISDA.2009.5356528.

[19] Hassan, A. A., Alaa F. S., dan Talaat M. W., 2017. "Intrusion Detection Using Neural Network : A Literature Review". **International Journal of Science and Research (IJSR)**, vol. 6. URL: https://www.ijsr.net/search_index_results_paperid.php?id=4091703.

Lampiran A
Gambar Distribusi Label Data Latih



Gambar A-1. Distribusi data pada *train set* sebelum dikelompokkan.



Gambar A-2. Distribusi data pada *train set* sebelum dikelompokkan (lanjutan).

Gambar A-3. Distribusi data pada *train set* sebelum dikelompokkan (lanjutan).

Gambar A-4. Distribusi data pada *train set* sebelum dikelompokkan (lanjutan).

Gambar A-5. Distribusi data pada *train set* sebelum dikelompokkan (lanjutan).

Lampiran B
Skrip de.py

```
import datetime
import numpy
import time
import matplotlib
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from thundersvm import SVC
import pandas


class DifferentialEvolution(object):

        def __init__(self, X, y, validation_size, pop_size, dimensions, bounds, f, cr):
                self.pop_size = max(pop_size, 4)
                self.mutation_factor = f
                self.cross_rate = cr
                self.dimensions = dimensions
                self.validation_size = validation_size
                self.X = X
                self.y = y
                self.low_bounds, self.up_bounds = numpy.asarray(bounds).T
                self.population = self.init_pop()

        def init_pop(self):
                pop = numpy.random.uniform(low=self.low_bounds, high=self.up_bounds, size=(self.pop_size, self.dimensions))
                return pop

        def mutation(self, target_index):
                indices = [index for index in range(self.pop_size) if index != target_index]
                v1, v2, v3 = self.population[numpy.random.choice(indices, 3, replace=False)]
                mutant = v1 + self.mutation_factor * (v2 - v3)

                return numpy.clip(mutant, self.low_bounds, self.up_bounds)

        def crossover(self, target, mutant):
```

```
36                 trial = numpy.empty(self.dimensions)
37
38                 for j in range(self.dimensions):
39                         if numpy.random.rand() < self.
    cross_rate or j == numpy.random.randint(0, self.
    dimensions):
40                                 trial[j] = mutant[j]
41                         else:
42                                 trial[j] = target[j]
43                 return trial
44
45         def optimize(self, iteration):
46                 print("Optimization started ", datetime.
    datetime.now())
47
48                 f_vals = self.f(self.population)
49                 best_fvals = numpy.empty(iteration)
50                 for i in range(iteration):
51                         print("Generation ", i+1)
52                         for target_index in range(self.
    pop_size):
53                                 mutant = self.mutation(
    target_index)
54                                 trial = self.crossover(self.
    population[target_index], mutant)
55                                 trial_fit = self.f_individual
    (trial)
56
57                                 if trial_fit <= f_vals[
    target_index]:
58                                         self.population[
    target_index] = trial
59                                         f_vals[target_index]
    = trial_fit
60
61                         best_index = numpy.argmin(f_vals)
62                         best_fvals[i] = f_vals[best_index]
63                         print("Best Solution: ", self.
    population[best_index])
64                         print("Best Fitness Value: ", f_vals[
    best_index])
65                 print("Optimization finished ", datetime.
    datetime.now())
66
67                 matplotlib.pyplot.plot(best_fvals)
68                 matplotlib.pyplot.xlabel("Iteration")
69                 matplotlib.pyplot.ylabel("Fitness")
```

```
70                 matplotlib.pyplot.show()
71                 return self.population[best_index], f_vals[
    best_index]
72
73         def f_individual(self, x):
74                 classifier = SVC(kernel='rbf', C=x[0], gamma=
    x[1])
75                 if 0 < self.validation_size < 1:
76                         X_train, X_val, y_train, y_val =
    train_test_split(self.X, self.y, test_size=self.
    validation_size,
77
78                 random_state=0)
79                         classifier.fit(X_train, y_train)
80                         acc = accuracy_score(y_val,
    classifier.predict(X_val))
81                 return 1.0 - acc
82
83         def f(self, pop):
84                 j = [self.f_individual(pop[i]) for i in range
    (self.pop_size)]
85
86                 return numpy.array(j)
87
88 class BinaryDE(object):
89
90         def __init__(self, X, y, validation_size, pop_size,
    dimensions, f, cr):
91                 self.pop_size = max(pop_size, 4)
92                 self.mutation_factor = f
93                 self.cross_rate = cr
94                 self.dimensions = dimensions
95                 self.validation_size = validation_size
96                 self.X = X
97                 self.y = y
98                 self.population = self.init_pop()
99
100        def init_pop(self):
101                pop = numpy.random.randint(low=0, high=2,
    size=(self.pop_size, self.dimensions))
102                return pop
103
104        def mutation(self, target_index):
105                indices = [index for index in range(self.
    pop_size) if index != target_index]
```

```
106                 v1, v2, v3 = self.population[numpy.random.
    choice(indices, 3, replace=False)]
107                 mutant = v1 + self.mutation_factor * (v2 - v3
    )
108                 mutant = mutant > 0.5
109                 return mutant
110 
111         def crossover(self, target, mutant):
112                 trial = numpy.empty(self.dimensions)
113 
114                 for j in range(self.dimensions):
115                         if numpy.random.rand() < self.
    cross_rate or j == numpy.random.randint(0, self.
    dimensions):
116                                 trial[j] = mutant[j]
117                         else:
118                                 trial[j] = target[j]
119                 return trial
120 
121         def optimize(self, iteration):
122                 print("Optimization started ", datetime.
    datetime.now())
123                 f_vals = self.f(self.population)
124                 best_fvals = numpy.empty(iteration)
125                 for i in range(iteration):
126                         print("Generation ", i+1)
127                         for target_index in range(self.
    pop_size):
128                                 mutant = self.mutation(
    target_index)
129                                 trial = self.crossover(self.
    population[target_index], mutant)
130                                 trial_fit = self.f_individual
    (trial)
131 
132                                 if trial_fit <= f_vals[
    target_index]:
133                                         self.population[
    target_index] = trial
134                                         f_vals[target_index]
    = trial_fit
135 
136                         best_index = numpy.argmin(f_vals)
137                         best_fvals[i] = f_vals[best_index]
138                         print("Best Solution: ", self.
    population[best_index])
139                         print("Best Fitness Value: ", f_vals[
```

```
    best_index])
140             print("Optimization finished ", datetime.
    datetime.now())
141
142             matplotlib.pyplot.plot(best_fvals)
143             matplotlib.pyplot.xlabel("Iteration")
144             matplotlib.pyplot.ylabel("Fitness")
145             matplotlib.pyplot.show()
146
147             return self.population[best_index], f_vals[
    best_index]
148
149         def f_individual(self, x):
150             classifier = SVC(kernel='rbf', gamma='auto')
151             selected_elements_indices = numpy.where(numpy
    .asarray(x) == 1)[0]
152             X_subset = self.X[:,
    selected_elements_indices]
153             if 0 < self.validation_size < 1:
154                 X_train, X_val, y_train, y_val =
    train_test_split(X_subset, self.y, test_size=self.
    validation_size,
155
                                                 random_state=0)
156                 classifier.fit(X_train, y_train)
157                 acc = accuracy_score(y_val,
    classifier.predict(X_val))
158             f_val = 1.0 - acc
159             return f_val
160
161         def f(self, pop):
162             j = [self.f_individual(pop[i]) for i in range
    (self.pop_size)]
163
164             return numpy.array(j)
```

Kode B.1. Skrip de.py

BIODATA PENULIS

Penulis lahir di Surabaya, 14 Desember 1998. Pendidikan penulis bermula di SDK Karitas III Surabaya, SMP St. Yosef Surabaya, dan SMA Katolik St. Louis 1 Surabaya. Setelah lulus SMA pada tahun 2016, penulis menempuh pendidikan tinggi di Departemen Matematika, Fakultas Sains dan Analitika Data (FSAD) Institut Teknologi Sepuluh Nopember (ITS) melalui jalur SNMPTN. Selama proses perkuliahan, penulis menekuni bidang minat Ilmu Komputer (*Computer Science*). Selain berkuliah, penulis juga aktif mengikuti kompetisi matematika, seperti Fun Mathematics Competition (FMC), Calculus Cup UNJ, MaG-D ITB, dan ONMIPA-PT. Dibalik kesibukan di dunia akademik, penulis juga mengikuti organisasi intra kampus. Penulis pernah menjabat sebagai Konseptor Soal OMITS 13th dan pernah menjadi staff Departemen ASCI di Himpunan Mahasiswa Matematika ITS (HIMATIKA ITS). Apabila ada pertanyaan dan saran, bisa menghubungi penulis via email *yohanesacgosal@gmail.com*.