ANALISIS PERFORMA TEKNIK RESAMPLING UNTUK MENGATASI KETIDAKSEIMBANGAN DATA LATIH PADA MODEL DETEKSI INTRUSI JARINGAN

Disusun Oleh:

N a m a : Mahesa Cadi Rajasa

NIM : 19523122

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2024

HALAMAN PENGESAHAN DOSEN PEMBIMBING

ANALISIS PERFORMA TEKNIK RESAMPLING UNTUK MENGATASI KETIDAKSEIMBANGAN DATA LATIH PADA MODEL DETEKSI INTRUSI JARINGAN

TUGAS AKHIR

Disusun Oleh:

N a m a : Mahesa Cadi Rajasa

NIM : 19523122

Yogyakarta, 15 Januari 2024

Pembimbing,

(Fayruz Rahma, S.T., M.Eng.)

HALAMAN PENGESAHAN DOSEN PENGUJI

ANALISIS PERFORMA TEKNIK RESAMPLING UNTUK MENGATASI KETIDAKSEIMBANGAN DATA LATIH PADA MODEL DETEKSI INTRUSI JARINGAN

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 15 Januari 2024

Tim Penguji

Fayruz Rahma, S.T., M.Eng. _____

Anggota 1

Dr. Ahmad Luthfi, S.Kom., M.Kom. _____

Anggota 2

Kurniawan Dwi Irianto, S.T., M.Sc. _____

Mengetahui,

Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri

Universitas Islam Indonesia

(Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Mahesa Cadi Rajasa

NIM : 19523122

Tugas akhir dengan judul:

ANALISIS PERFORMA TEKNIK RESAMPLING UNTUK MENGATASI KETIDAKSEIMBANGAN DATA LATIH PADA MODEL DETEKSI INTRUSI JARINGAN

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 27 Desember 2023

(Mahesa Cadi Rajasa)

HALAMAN PERSEMBAHAN

Alhamdulillahirobbil'alamin atas segala nikmat yang telah diberikan kepada kita. Shalawat serta salam kita haturkan kepada junjungan kita Nabi Muhammad saw. yang kita nantikan safa'atnya di yaumul akhir nanti.

Terima kasih yang amat besar saya haturkan kepada orang tua saya yang telah mengasuh dan mendidik saya sejak di dalam kandungan sampai pada saat ini juga dan kedua adik saya yang telah memberikan dukungan baik moril maupun materil. Semoga kita senantiasa diberikan kesehatan, kebahagiaan, dan panjang umur.

Terima kasih kepada Dosen Pembimbing saya, Ibu Fayruz Rahma yang selalu melatih dan membimbing saya dengan sabar dan selalu memperhatikan saya di setiap pertemuan bimbingan.

Terima kasih kepada semua pihak yang tidak bisa disebutkan satu per satu atas dukungan dan bantuannya baik secara langsung maupun tidak langsung.

HALAMAN MOTO

“Kegagalan adalah peluang belajar”

“Dan mintalah pertolongan dengan sabar dan shalat” (Q.S Al-Baqarah: 45)

“Allah tidak membebani seseorang melainkan sesuai dengan kesanggupannya...” (Q.S Al-Baqarah: 286)

KATA PENGANTAR

Assalamu'alaikum warahmatullahi wabarakatuh.

Puji dan syukur atas ke hadirat Allah Swt. atas segala nikmat, rahmat, dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir ini. Tugas Akhir yang berjudul "Analisis Performa Teknik Resampling untuk Mengatasi Ketidakseimbangan Data Latih pada Model Deteksi Intrusi Jaringan" disusun untuk memenuhi salah satu syarat untuk mencapai gelar sarjana (S1) pada Jurusan Informatika, Fakultas Teknologi Industri, Universitas Islam Indonesisa.

Pada kesempatan ini, penulis mengucapkan banyak terima kasih kepada berbagai pihak yang telah memberikan bantuan dan dukungan baik secara langsung maupun secara tidak langsung dalam penyelesaian Tugas Akhir ini, yaitu:

1. Orang tua dan kedua adik penulis yang telah memberikan berbagai macam dukungan baik moril maupun materil kepada penulis.
2. Prof. Fathul Wahid, S.T., M.Sc., Ph.D. selaku Rektor Universitas Islam Indonesia.
3. Dr. Raden Teduh Dirgahayu, S.T., M.Sc. selaku Ketua Jurusan Informatika UII
4. Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D. selaku Ketua Prodi Informatika – Program Sarjana UII.
5. Fayruz Rahma, S.T., M.Eng. selaku Dosen Pembimbing yang telah memberikan pengarahan, bimbingan, bantuan, serta masukan kepada penulis sehingga Tugas Akhir ini dapat diselesaikan dengan baik.
6. Kepada teman-teman SMA yang selalu ada sebagai tempat berbagi cerita dan dukungan kepada peneliti.
7. Teman seperjuangan penulis seluruh mahasiswa Informatika angkatan 2019.
8. Seluruh pihak yang telah membantu yang tidak dapat disebutkan satu per satu.

Penulis sadar banyak terdapat kekurangan dalam pembuatan tugas akhir ini. Namun, penulis selalu berharap tugas akhir ini dapat bermanfaat atau mungkin bisa dikembangkan menjadi hal yang lebih besar lagi.

Yogyakarta, 27 Desember 2023

(Mahesa Cadi Rajasa)

SARI

Pesatnya perkembangan koneksi jaringan menyebabkan peningkatan aktivitas pada lalu lintas jaringan. Lonjakan tersebut menimbulkan tantangan baru di dunia maya sehingga rentan terhadap serangan siber. Untuk mengatasi tantangan ini, para peneliti telah beralih ke teknik cerdas seperti *machine learning* untuk meningkatkan kualitas deteksi serangan pada lalu lintas jaringan. Namun, muncul fenomena yang dikenal sebagai ketidakseimbangan data. Salah satu penyebab terjadinya fenomena ini adalah distribusi kelas yang tidak merata, di mana dalam kasus deteksi intrusi jaringan terjadi ketidakseimbangan antarkelas serangan, yang membuat performa klasifikasi dari model *machine learning* menjadi kurang optimal. Untuk mengatasi ketidakseimbangan kelas pada data, dapat digunakan teknik *resampling*. Terdapat berbagai teknik resampling yang dapat digunakan, sehingga diperlukan analisis terhadap teknik resampling yang digunakan. Pada penelitian ini, digunakan berbagai teknik *resampling* seperti ADASYN, *Borderline* SMOTE, *Random Oversampling*, *Random Undersampling*, SMOTE, SMOTE-*Tomek*, SVM-SMOTE, dan *Tomek Links*. Penelitian ini menggunakan dataset UNSW-NB15 untuk melatih dan menguji beberapa macam model *Network Intrusion Detection*, di antaranya adalah model dengan algoritma *Decision Tree*, *Random Forest*, *Gradient Boosting*, *XGBoost*, dan 1D-CNN. Berdasarkan hasil analisis yang telah dilakukan, ditemukan bahwa penerapan teknik *resampling* terbukti memengaruhi performa dari model *machine learning* dan *deep learning*. Performa terbaik didapatkan dari penerapan teknik *Tomek Links* pada model 1D-CNN dengan skor akurasi 75.27%, skor presisi 87.58%, dan F1-*score* 76.22%. Sementara itu, skor *recall* terbaik didapatkan dari model 1D-CNN tanpa *resampling*, yaitu dengan skor *recall* sebesar 67.57%. Hasil penelitian ini diharapkan dapat memberikan pengetahuan yang dapat membantu peneliti dan *engineer* untuk mengetahui kelebihan dan kekurangan teknik *resampling* yang akan digunakan dalam pengembangan model deteksi jenis serangan pada lalu lintas jaringan.

Kata kunci: Ketidakseimbangan Data, *Network Intrusion Detection*, Teknik *Resampling*.

GLOSARIUM

Dataset	Data yang digunakan pada proses pembentukan dan pengujian model.
Hybrid Sampling	Pendekatan teknik *resampling* dengan menggabungkan pendekatan *Oversampling* dan *Undersampling*.
Layer	Lapisan pada arsitektur suatu model *deep learning*
Model	Hasil dari *training* digunakan untuk klasifikasi serangan.
Oversampling	Pendekatan teknik *resampling* untuk meningkatkan distribusi pada kelas minoritas di dataset.
Resampling	Teknik untuk mengatasi ketidakseimbangan pada data dengan cara meningkatkan jumlah sampel pada kelas minoritas atau mengurangi sampel pada kelas mayoritas.
Training	Pelatihan model
Testing	Pengujian model
Undersampling	Pendekatan teknik *resampling* untuk mengurangi distribusi pada kelas mayoritas di dataset.
UNSW-NB15	Dataset yang berisi data trafik lalu lintas jaringan

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Di seluruh belahan dunia, internet terus menjadi bagian dari kehidupan sehari-hari. Sekitar 40% dari populasi dunia memiliki akses untuk menggunakan internet (Darina, 2023). Tidak hanya itu, sekitar 5 miliar orang di dunia mempunyai perangkat *mobile* (Darina, 2023). Semua perangkat itu terhubung ke internet bersama dengan perangkat IoT lain yang saling terhubung, membentuk yang dinamakan jaringan. Jaringan seperti *smart cars*, *smartwatch*, sistem keamanan, telepon, dan bahkan sinyal lalu lintas berbagi sejumlah besar informasi. Diperkirakan akan ada 38,6 miliar perangkat yang terhubung dengan IoT di seluruh dunia pada tahun 2025 dan 50 miliar pada tahun 2030 (McCain, 2023).

Selain itu, ada teknologi seperti *Artificial Intelligence* (AI) dan *Machine Learning* yang dengan kehadirannya dapat membantu di kehidupan sehari-hari. Teknologi internet dan AI telah mengubah dunia konsumen, perusahaan, dan pemerintah di seluruh dunia. Setiap harinya, semakin banyak orang yang mengandalkan teknologi modern dan semakin banyak bisnis yang menggunakan perangkat digital serta perangkat lunak berbasis web. Hal ini dapat meningkatkan ancaman siber yang lebih besar daripada tahun-tahun sebelumnya. Kerugian akibat ancaman siber ini meningkat sebesar 15% dalam lima tahun ke depan (Boskamp, 2023). Diperkirakan pada tahun 2025, ancaman siber akan merugikan dunia sekitar 10.5 triliun dollar setiap tahunnya (Boskamp, 2023). Sekitar 43% serangan siber menargetkan perusahaan kecil.

Pesatnya pertumbuhan konektivitas antarperangkat digital berdampak besar pada peningkatan aktivitas serangan siber. Kerentanan pada sistem perangkat dieksploitasi untuk mencuri informasi berharga atau memperlambat layanan jaringan (Chapaneri & Shah, 2022). Adapun salah satu cara untuk mendeteksi terjadinya akses, penggunaan, modifikasi, dan interupsi yang tidak sah, baik pada suatu jaringan itu sendiri maupun sumber daya komputasi lainnya yang terhubung ke jaringan tersebut, yaitu dengan *Network Intrusion Detection*.

Deteksi intrusi jaringan berfungsi melakukan pemantauan terus menerus pada semua aktivitas di jaringan dan memberikan alarm ketika terjadi pelanggaran kebijakan atau aktivitas yang berbahaya lainnya (Rahma & Pratama, 2020). Mengingat banyaknya ancaman siber yang muncul setiap hari dengan lalu lintas jaringan yang besar, ada kebutuhan mendesak untuk mengembangkan teknik cerdas untuk mengatasi tantangan ini. Dalam kasus lalu lintas jaringan, terdapat fenomena ketidakseimbangan data, dimana salah satu penyebabnya adalah distribusi

kelas yang tidak merata. Ketidakseimbangan data serangan dapat menurunkan kinerja klasifikasi dari model (Chapaneri & Shah, 2022). Maka dari itu, dibutuhkan cara untuk mengatasi ketidakseimbangan pada data lalu lintas jaringan tersebut.

Untuk mengatasi ketidakseimbangan data, dapat digunakan teknik *resampling* (Ghorbani & Ghousi, 2020). Terdapat berbagai teknik *resampling* yang dapat digunakan, sehingga diperlukan analisis terhadap teknik *resampling* yang digunakan. Teknik ini berfungsi dengan cara membuat versi baru atau melakukan transformasi pada dataset pelatihan yang digunakan. Secara umum, ada tiga pendekatan utama dari teknik ini, yaitu *oversampling*, *undersampling*, dan *hybrid sampling*. Pada tiap macam pendekatan, terdapat beberapa strategi yang bisa digunakan. Pada penelitian ini, diterapkan teknik *resampling* dengan berbagai macam pendekatan agar distribusi antar kelas serangan bisa seimbang serta dilihat pengaruhnya pada performa model deteksi jenis intrusi jaringan. Hasil penelitian ini diharapkan dapat memberikan pengetahuan terkait kelebihan dan kekurangan teknik *resampling* serta dapat membantu peneliti dan *engineer* dalam memilih teknik *resampling* yang akan digunakan untuk pengembangan model deteksi jenis serangan pada lalu lintas jaringan.

1.2 Rumusan Masalah

Masalah yang akan dibahas pada penelitian ini adalah:

a. Bagaimana cara penggunaan teknik *resampling* pada dataset intrusi jaringan?
b. Bagaimana penerapan teknik *resampling* pada data pelatihan dapat memengaruhi performa model deteksi jenis intrusi jaringan?

1.3 Lingkup Masalah

Penelitian ini akan berfokus pada penggunaan teknik *resampling* pada dataset yang tidak seimbang dengan melihat performa F1-*score* pada model deteksi jenis intrusi jaringan. Untuk membatasi ruang lingkup pada penelitian ini, diberikan batasan masalah sebagai berikut:

a. Dataset yang digunakan adalah dataset tidak seimbang yang berkaitan dengan *Network Intrusion Detection*, yaitu UNSW-NB15.
b. Teknik *resampling* yang digunakan, yaitu: ADASYN, *Borderline* SMOTE, *Random Oversampling*, *Random Undersampling*, SMOTE, SMOTE-*Tomek*, SVM-SMOTE, dan *Tomek Links*.
c. Algoritma model yang digunakan, yaitu: *Decision Tree*, *Random Forest*, *Gradient Boosting*, *XGBoost*, dan 1D-CNN.

1.4 Tujuan Penelitian

Tujuan penelitian yang diharapkan pada penelitian ini adalah sebagai berikut:

a. Dapat mengetahui cara penggunaan teknik *resampling* pada dataset intrusi jaringan.
b. Dapat mengetahui pengaruh dari penerapan teknik *resampling* pada data pelatihan terhadap performa model deteksi intrusi jenis jaringan.

1.5 Manfaat Penelitian

Manfaat penelitian yang diharapkan antara lain:

a. Bagi Mahasiswa sebagai peneliti, penelitian ini diharapkan dapat menjadi referensi ilmiah dalam menangani dataset yang tidak seimbang dan meningkatkan performa dari model *Network Intrusion Detection*.
b. Bagi *Network Security Engineer*, penelitian ini diharapkan dapat membantu dalam pengembangan sistem keamanan untuk mendeteksi serangan pada lalu lintas jaringan.

1.6 Sistematika Penulisan

Untuk memberikan gambaran yang singkat tentang penelitian ini, penelitian ini dibagi menjadi lima bab yang saling berhubungan. Penulisan penelitian ini dibuat berdasarkan kaidah penulisan ilmiah dengan sistematika sebagai berikut:

a. BAB I PENDAHULUAN

   Bab ini berisi penguraian tentang latar belakang, rumusan masalah, lingkup masalah, tujuan penelitian, manfaat penelitian, dan sistematika penulisan.

b. BAB II LANDASAN TEORI

   Bab ini berisi teori-teori dasar pada umumnya yang digunakan sebagai acuan dalam pembuatan penelitian ini. Teori yang dijelaskan adalah pengertian tentang *network intrusion detection*, *imbalance dataset*, teknik *resampling*, *machine learning*, *deep learning*, dan penelitian terkait.

c. BAB III METODOLOGI PENELITIAN

   Bab ini berisi alur pengerjaan tugas akhir dari awal sampai akhir dan uraian terkait metodologi yang digunakan.

d. BAB IV HASIL DAN PEMBAHASAN

Bab ini berisi pembahasan dari setiap aktivitas yang dilakukan dalam pelaksanaan eksperimen. Bab ini juga berisikan hasil evaluasi untuk mengetahui kelebihan dan kekurangan dari teknik yang sudah diimplementasi.

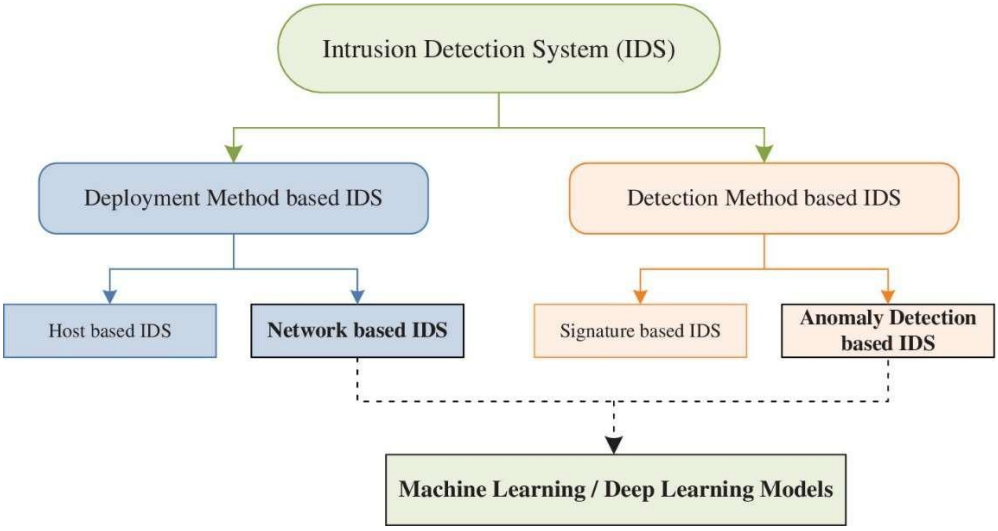
e. BAB V KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan yang diperoleh dari hasil penelitian dan saran-saran yang membangun sebagai pemecahan masalah yang lebih baik di masa depan.

BAB II
LANDASAN TEORI

Bab ini berisi teori-teori dasar yang digunakan sebagai acuan dalam penelitian ini. Teori yang dijelaskan adalah pengertian tentang *network intrusion detection*, *imbalanced dataset*, teknik *resampling*, *machine learning*, *deep learning*, dan penelitian terkait.

2.1 Network Intrusion Detection System



Gambar 2.1 Klasifikasi *Intrusion Detection System* (Ahmad dkk., 2021)

Network Intrusion Detection System adalah sistem, atau sekelompok alat, yang dibuat untuk mengawasi aktivitas sistem atau jaringan jika ada indikasi aktivitas jahat atau pelanggaran kebijakan keamanan, serta merupakan komponen yang krusial dalam infrastruktur keamanan jaringan. Untuk mengidentifikasi potensi ancaman keamanan, NIDS menganalisis data jaringan dan mencari pola atau perilaku (Ahmad dkk., 2021). Hal tersebut menggunakan metode *detection-based* NIDS seperti yang terlihat pada Gambar 2.1. Pola atau perilaku dapat dianalisis menggunakan *machine learning* dan *deep learning*. *Detection-based* NIDS hadir dalam dua jenis: *Anomaly-based* dan *Signature-based* (Sharma dkk., 2022).

a. *Anomaly-based Intrusion Detection*: Sistem ini memberikan garis dasar dari perilaku jaringan yang umum daripada bergantung pada tanda atau pola yang telah ditentukan. Peringatan akan dipicu ketika ada penyimpangan dari garis dasar, yang mungkin menunjukkan adanya serangan atau aktivitas yang tidak biasa.

b. *Signature-based Intrusion Detection*: Jenis ini menggunakan basis data dengan tanda atau pola serangan yang dikenal. Peringatan akan dipicu dan tindakan yang telah ditentukan diambil oleh sistem untuk menghentikan ancaman ketika lalu lintas jaringan sesuai dengan tanda atau pola yang diketahui.

Berdasarkan Gambar 2.1 dapat diketahui bahwa model *machine learning* dan *deep learning* dapat digunakan pada *Anomaly-based Intrusion Detection*.

2.2 Imbalanced Dataset pada Deteksi Intrusi Jaringan

Imbalanced dataset atau dataset yang tidak seimbang adalah salah satu masalah dalam *machine learning*, terutama dalam hal mendeteksi intrusi pada suatu jaringan. Ketika distribusi kelas suatu dataset tidak seimbang, jumlah satu kelas jauh lebih banyak daripada kelas lainnya. Hal tersebut dikatakan tidak seimbang. Dalam hal *network intrusion detection*, hal ini biasanya berarti bahwa jumlah suatu kelas serangan melebihi kelas serangan lainnya (Tanha dkk., 2020).

Mengatasi dataset yang tidak seimbang sangat penting untuk mengembangkan model *network intrusion detection* yang andal dan tepat. Beberapa contoh dataset tidak seimbang yang sering digunakan peneliti untuk pengembangan model *network intrusion detection* di antaranya: NSL-KDD, CIC-IDS2017, UNSW-NB15, dan KDD-Cup 1999 (Rajasa dkk., 2023). Dataset tersebut terbuka untuk umum (*open source*) dan bisa diunduh secara gratis melalui web masing-masing.

Pada penelitian ini, digunakan dataset UNSW-NB15. Dataset ini dibuat di *Cyber Range Lab, Australian Centre for Cyber Security* (ACCS) (Moustafa & Slay, 2015). Mereka menghasilkan data langsung dari lalu lintas jaringan dua *server* menggunakan alat IXIA *PerfectsStorm*. *Raw* data pada dataset ini diolah menggunakan berbagai *tools* seperti Argus, Bro-IDS, dan dua belas algoritma yang dikembangkan menggunakan bahasa pemrograman C#. Alat Argus memproses *raw network packets* (misalnya file .pcap) dan menghasilkan atribut atau fitur. Alat Bro-IDS digunakan untuk mengekstrak fitur dari lalu lintas jaringan dan dua belas algoritma dikembangkan untuk menganalisis aliran paket koneksi.

Dataset ini memiliki jumlah total sekitar 2.5 juta *records*. Adapun partisi (subset) yang dibuat sebagai *Training Set* dan *Testing Set*, berjumlah total sekitar 250 ribu *records*. Dataset UNSW-NB15 terdiri dari 44 fitur dan satu label kelas yang menunjukkan apakah itu normal atau sebuah serangan. Dataset ini terdistribusi menjadi sepuluh kategori, yaitu: Normal, Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode, dan

Worms. Proporsi dari setiap kategori dapat dilihat pada Tabel 2.1. Deskripsi setiap fitur dapat dilihat pada Tabel 2.2.

Tabel 2.1 Distribusi dataset UNSW-NB15 pada *Training & Testing Sets*

Kategori	**Training Set**		**Testing Set**	
	Records	**%**	**Records**	**%**
Normal	56 000	31.94	37 000	44.94
Fuzzers	18 184	10.37	6062	7.36
Analysis	2000	1.14	677	0.82
Backdoor	1746	1.00	583	0.71
DoS	12 264	6.99	4089	4.97
Exploits	33 393	19.04	11 132	13.52
Generic	40 000	22.81	18 871	22.92
Reconnaissance	10 491	5.98	3496	4.25
Shellcode	1133	0.65	378	0.46
Worms	130	0.07	44	0.05
Total	175 341	100	82 332	100

Tabel 2.2 Deskripsi setiap fitur pada dataset UNSW-NB15 (Moustafa & Slay, 2015)

#	**Nama Fitur**	**Tipe**	**Deskripsi**
0	*id*	Integer	Penomoran sampel
1	*dur*	Float	*Record* durasi total
2	*proto*	Nominal	*Transactional protocol*
3	*service*	Nominal	http, ftp, ssh, dns, dan lainnya
4	*state*	Nominal	*State* dan protokol yang bergantung padanya, misal: ACC, CLO, dan lainnya
5	*spkts*	Integer	Jumlah paket dari sumber ke tujuan
6	*dpkts*	Integer	Jumlah paket dari tujuan ke sumber
7	*sbytes*	Integer	*Bytes* dari sumber ke tujuan
8	*dbytes*	Integer	*Bytes* dari tujuan ke sumber
9	*rate*	Float	Laju data ethernet yang dikirim dan diterima
10	*sttl*	Integer	Waktu untuk hidup dari sumber ke tujuan
11	*dttl*	Integer	Waktu untuk hidup dari tujuan ke sumber

12	*sload*	Float	*Bits* dari sumber per detiknya
13	*dload*	Float	*Bits* dari tujuan per detiknya
14	*sloss*	Integer	Paket dari sumber yang ditransmisikan ulang atau dijatuhkan
15	*dloss*	Integer	Paket dari tujuan yang ditransmisikan ulang atau dijatuhkan
16	*sintpkt*	Float	Waktu kedatangan antar paket (mSec) sumber
17	*dintpkt*	Float	Waktu kedatangan antar paket (mSec) tujuan
18	*sjit*	Float	*Jitter* (mSec) sumber
19	*djit*	Float	*Jitter* (mSec) tujuan
20	*swin*	Integer	Iklan TCP *window* sumber
21	*stcpb*	Integer	Nomor urut TCP sumber
22	*dtcpb*	Integer	Nomor urut TCP tujuan
23	*dwin*	Integer	Iklan TCP *window* tujuan
24	*tcprtt*	Float	Jumlah ‘synack’ dan ‘ackdat’ dari TCP
25	*synack*	Float	Waktu antara SYN dan SYN_ACK dari paket TCP
26	*ackdat*	Float	Waktu antara SYN_ACK dan SYN dari paket TCP
27	*smean*	Integer	Rata-rata ukuran *flow packet* yang dikirimkan oleh sumber
28	*dmean*	Integer	Rata-rata ukuran *flow packet* yang dikirimkan oleh tujuan
29	*trans_depth*	Integer	Kedalaman koneksi *http request* atau *response transaction*
30	*response_body_len*	Integer	Ukuran konten data yang ditransfer dari layanan *http server*
31	*ct_srv_src*	Integer	Jumlah koneksi yang mengandung *sevice* yang sama (14) dan alamat sumber (1) di 100 koneksi sesuai dengan waktu terakhir (26)

32	*ct_state_ttl*	Integer	Nomor untuk setiap *state* (6) menurut rentang nilai tertentu untuk sumber/tujuan waktu hidup (10) (11)
33	*ct_dst_ltm*	Integer	Jumlah koneksi dengan alamat tujuan sama (3) dalam 100 koneksi terakhir kali (26)
34	*ct_src_dport_ltm*	Integer	Jumlah koneksi dari alamat sumber (1) dan *port* tujuan yang sama (4) dalam 100 koneksi sesuai dengan waktu terakhir (26)
35	*ct_dst_sport_ltm*	Integer	Jumlah koneksi dari alamat tujuan (1) dan *port* sumber yang sama (4) dalam 100 koneksi sesuai dengan waktu terakhir (26)
36	*ct_dst_src_ltm*	Integer	Jumlah koneksi dari sumber (1) dan alamat tujuan yang sama (3) dalam 100 koneksi sesuai dengan waktu terakhir (26)
37	*is_ftp_login*	Binary	Jika *ftp session* diakses oleh pengguna dan *password* maka 1 jika tidak 0
38	*ct_ftp_cmd*	Integer	Jumlah aliran yang memiliki perintah di *ftp session*
39	*ct_flw_http_mthd*	Integer	Jumlah aliran yang memiliki metode seperti *Get* dan *Post* di *http service*.
40	*ct_src_ltm*	Integer	Jumlah koneksi dengan alamat sumber sama (3) dalam 100 koneksi terakhir kali (26)
41	*ct_srv_dst*	Integer	Jumlah koneksi yang mengandung *sevice* yang sama (14) dan alamat tujuan (1) di 100 koneksi sesuai dengan waktu terakhir (26)
42	*is_sm_ips_ports*	Binary	Jika sumber (1) sama dengan tujuan (3) alamat IP dan nomor port (2) (4) adalah sama, variabel ini mengambil nilai 1 jika tidak 0
43	*attack_cat*	Nominal	Nama setiap kategori serangan. Di dalam dataset, sembilan kategori: Fuzzers, Analysis, Backdoors, DoS, Exploits, Generic, Reconnaissance, Shellcode dan Worms
44	*label*	Binary	0 untuk normal dan 1 untuk *attack*

2.3 Teknik Pre-Processing

Pre-processing adalah langkah penting dalam tahapan sains data. Langkah ini melibatkan manipulasi, penyaringan, atau augmentasi data sebelum dianalisis (Suripto dkk., 2022). Tujuan utama dari *pre-processing* adalah untuk meningkatkan kualitas data dan membuatnya lebih cocok untuk diproses dan dianalisis (deepak_jain, 2019). Adapun langkah *pre-processing* yang dilakukan pada penelitian ini, yaitu *cleaning*, *encoding*, *normalization*, dan *feature selection*.

a. *Cleaning* adalah proses pembersihan data. Proses ini melibatkan penanganan seperti *missing value, noisy data*, dan lainnya (deepak_jain, 2019). Pada penelitian ini, data yang tidak digunakan dibersihkan untuk mempermudah proses pada algoritma *machine learning*.
b. *Encoding* adalah proses transformasi data kategorik menjadi numerik. Proses ini melibatkan perubahan tipe data agar lebih mudah diproses oleh algoritma *machine learning*. Hal ini dilakukan karena mesin hanya mengerti input numerik (SagarDhandare, 2022). Pada penelitian ini, metode *encoding* yang digunakan adalah *One-hot Encoding* dan *Label Encoding*. Metode *One-hot Encoding* membuat sebuah kolom baru untuk setiap *unique categorical value* pada dataset (Cavin, 2022). Sementara itu, pada metode *Label Encoding*, *categorical value* diganti dengan label bertipe bilangan bulat, biasanya bilangan yang diganti berurutan (Cavin, 2022).
c. *Normalization* adalah proses transformasi *value* pada dataset ke skala yang lebih umum seperti -1.0 ke 1.0 atau 0.0 ke 1.0 (deepak_jain, 2019). Alasan dilakukan *normalization* karena banyak algoritma *machine learning* yang sensitif pada skala dari input data (deepak_jain, 2019). Pada penelitian ini, *normalization* dilakukan menggunakan *Min-Max Scaler*. Pada persamaan (2.1), (x) adalah nilai asli, dan (x *scaled*) adalah nilai yang telah dinormalisasi (Loukas, 2020).

$$x_{\text{scaled}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{2.1}$$

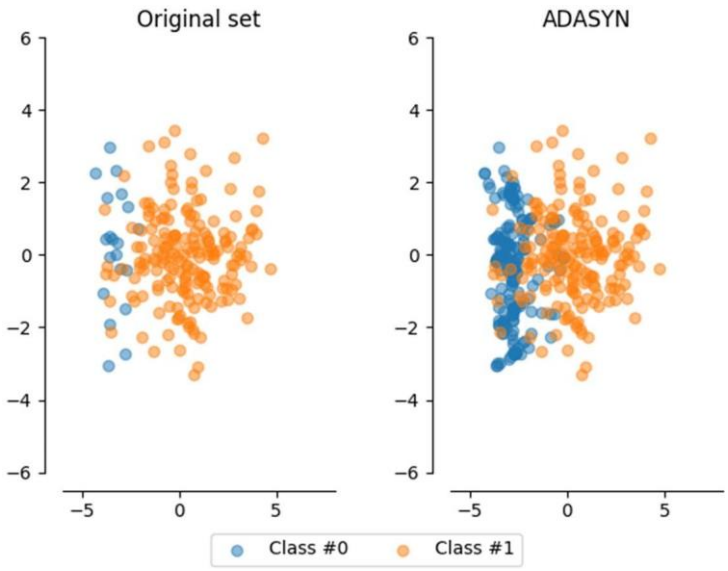
d. *Feature Selection* adalah proses menyeleksi fitur yang relevan dan penting dari dataset. Tujuan dari proses ini adalah untuk mengurangi ukuran dataset, menghilangkan fitur yang tidak relevan atau berlebihan, dan meningkatkan performa model (javaTpoint, 2021). Selain itu, seleksi fitur dapat mengurangi waktu dan sumber daya komputasi (Brownlee, 2019). Teknik seleksi fitur yang digunakan pada penelitian ini adalah *filter-based selection* (Brownlee, 2019). Kemudian karena input data pada penelitian ini berupa data numerik, dipilih *pearson correlation* sebagai salah satu yang paling umum digunakan untuk variabel numerik (Brownlee, 2019). Teknik ini memberikan nilai

antara -1 dan 1, di mana 0 adalah tidak ada korelasi, 1 adalah total korelasi positif, dan -1 adalah total korelasi negatif (Nettleton, 2014).

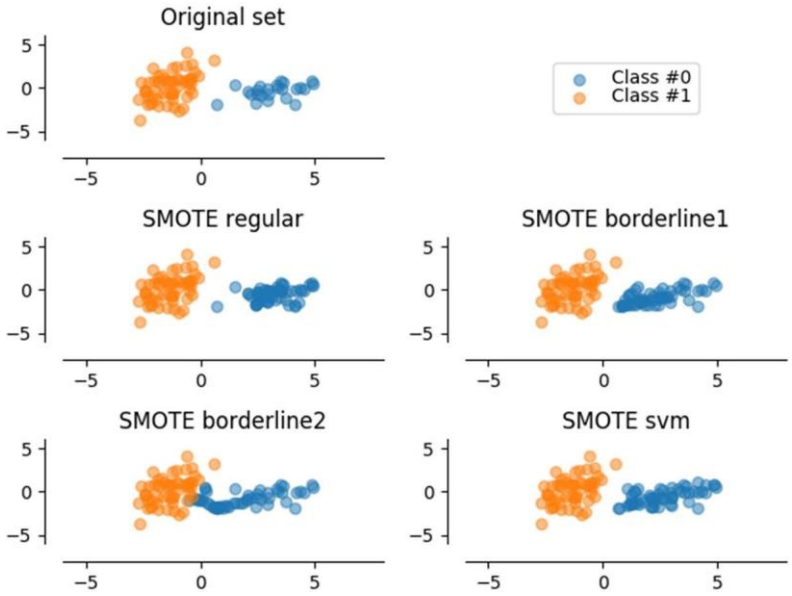
2.4 Teknik Resampling

Tindakan strategis dan perencanaan yang matang diperlukan untuk mengatasi dataset yang tidak seimbang. Salah satu cara dalam menyeimbangkan dataset untuk menjamin representasi yang adil dari kelas mayoritas dan minoritas adalah dengan teknik *resampling*. Teknik *resampling* melibatkan modifikasi dataset untuk menyeimbangkan kelas, baik dengan melakukan *oversampling* pada kelas minoritas, *undersampling* pada kelas mayoritas, atau kombinasi keduanya (*hybrid sampling*) (Ghorbani & Ghousi, 2020).

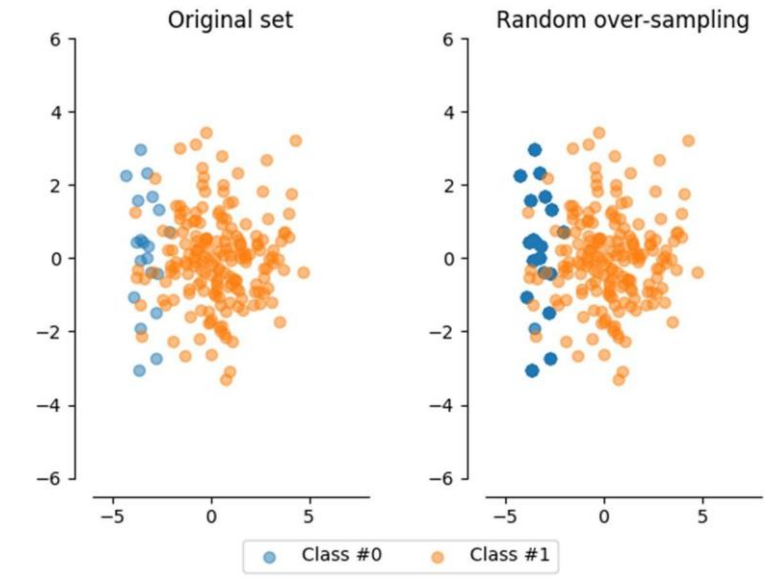
a. *Oversampling* adalah teknik yang bekerja meningkatkan jumlah sampel pada kelas minoritas dengan menghasilkan data sintetis. Salah satu cara yang paling simpel adalah dengan menggunakan *Random Oversampling*. Adapun cara lain yang bisa juga digunakan yaitu ADASYN, *Borderline* SMOTE, SMOTE, dan SVM-SMOTE.
b. *Undersampling* adalah teknik yang bekerja mengurangi jumlah sampel pada kelas mayoritas dengan menghapus sampel data. Hal ini dapat menimbulkan masalah jika informasi penting hilang dalam proses tersebut. Salah satu cara yang paling simpel adalah dengan menggunakan *Random Undersampling*. Cara lainnya yang dapat digunakan, yaitu *Tomek Links*.
c. *Hybrid Sampling* merupakan kombinasi dari teknik *oversampling* dan *undersampling*. Strategi ini bekerja dengan meningkatkan jumlah sampel pada kelas minoritas terlebih dahulu, kemudian mengurangi jumlah sampel pada kelas mayoritas ataupun sebaliknya. Salah satu kombinasi yang dapat digunakan adalah SMOTE-*Tomek*.



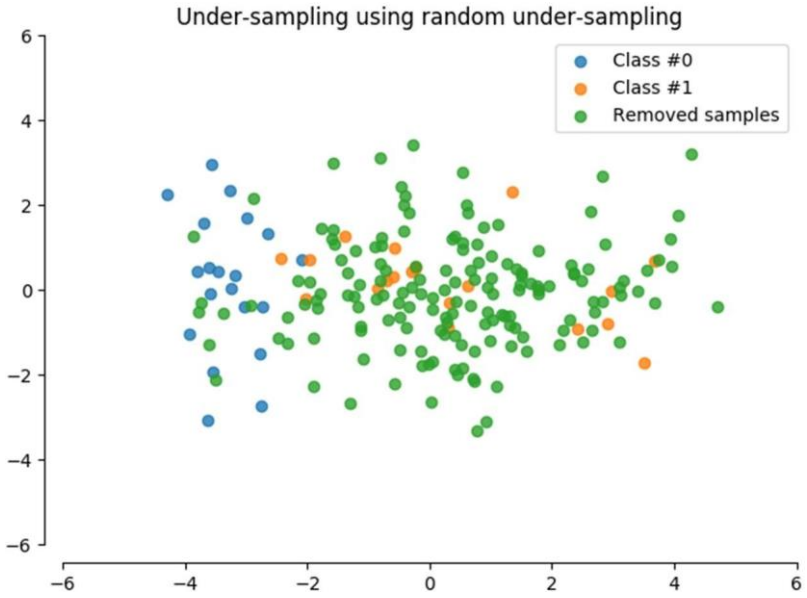
Gambar 2.2 Ilustrasi ADASYN (Lemaitre, 2016a)



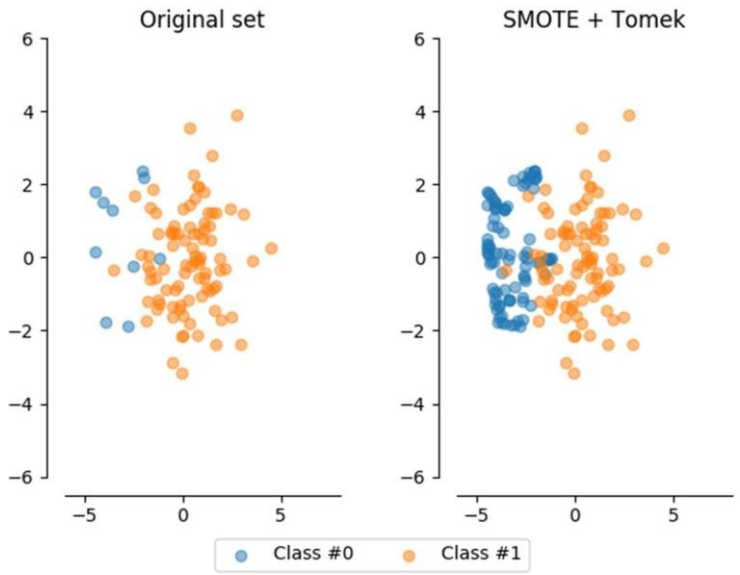
Gambar 2.3 Ilustrasi *Borderline* SMOTE, SMOTE, dan SVM-SMOTE (Lemaitre, 2016e)



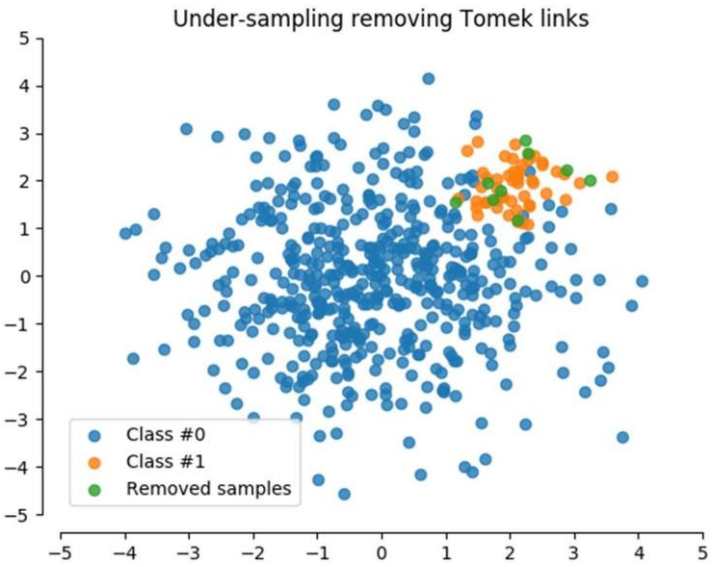
Gambar 2.4 Ilustrasi Random Oversampling (Lemaitre, 2016b)



Gambar 2.5 Ilustrasi Random Undersampling (Lemaitre, 2016c)



Gambar 2.6 Ilustrasi SMOTE-Tomek (Lemaitre, 2016d)



Gambar 2.7 Ilustrasi Tomek Links (Lemaitre, 2016f)

2.4.1 ADASYN

Adaptive Synthetic Sampling (ADASYN) dapat secara adaptif mensintesis sampel data baru pada kelas minoritas sesuai dengan densitas data pada dataset. Lebih sedikit sampel kelas minoritas yang dihasilkan di area klasifikasi yang mudah, dan lebih banyak sampel kelas minoritas yang disintesis di area klasifikasi yang sulit (He dkk., 2008). Teknik ini dapat mengatasi masalah distribusi kelas yang tidak seimbang pada dataset (Chen dkk., 2022).

Kemudian terdapat penelitian yang menerapkan teknik ADASYN, disimpulkan bahwa teknik ini dapat meningkatkan performa klasifikasi dari model (Ding dkk., 2022). Ilustrasi penggunaan ADASYN dapat dilihat pada Gambar 2.2.

2.4.2 Borderline SMOTE

Pada penelitian yang dilakukan oleh Ghorbani & Ghousi. (2020), digunakan teknik *Borderline* SMOTE untuk meningkatkan jumlah sampel pada kelas minoritas pada dataset yang mereka gunakan. Sebelum diterapkan *Borderline* SMOTE, sampel-sampel yang berdekatan lebih kemungkinan untuk salah diklasifikasikan daripada yang jauh dari garis batas. Teknik ini berfokus pada pembuatan sampel baru pada kelas minoritas secara sintetis yang berada di dekat *decision boundary* antara kelas minoritas dan mayoritas (Han dkk., 2005). Ilustrasi penggunaan *Borderline* SMOTE dapat dilihat pada Gambar 2.3.

2.4.3 SMOTE

Synthetic Minority Oversampling Technique, disingkat SMOTE, adalah teknik *resampling* dalam *machine learning* yang bekerja dengan membuat sampel buatan dari kelas minoritas. Dengan menghasilkan sampel sintetis di sepanjang segmen garis yang menghubungkan sampel kelas minoritas yang sudah ada, teknik ini secara efektif menambah dataset dan menawarkan representasi kelas yang lebih seimbang (Chawla dkk., 2002). SMOTE meningkatkan kinerja generalisasi dan klasifikasi untuk algoritma *machine learning* dengan memasukkan contoh sintetik. Pada penelitian yang dilakukan oleh Li, dkk. (2022), SMOTE terbukti meningkatkan performa dari model yang menggunakan *XGBoost*. Kemudian pada penelitian yang dilakukan oleh Abdulkareem, dkk. (2022), terbukti bahwa SMOTE memerlukan waktu pelatihan tambahan. Namun, performa dalam hal skor matrik klasifikasi mengungguli pengklasifikasi sebelumnya. Ilustrasi penggunaan SMOTE dapat dilihat pada Gambar 2.3.

2.4.4 SVM-SMOTE

SVM-SMOTE adalah teknik augmentasi data khusus yang menggabungkan kekuatan *Support Vector Machine* (SVM) dan *Synthetic Minority Oversampling Technique* (SMOTE) untuk mengatasi dataset yang tidak seimbang. Teknik ini berfokus pada pembuatan sampel baru pada kelas minoritas di dekat garis batas menggunakan model SVM untuk membantu menetapkan batasan antar kelas (Ghorbani & Ghousi, 2020). Teknik ini tidak hanya menyeimbangkan distribusi kelas tetapi juga meningkatkan pemisahan antar kelas, serta

performa generalisasi dan klasifikasi menjadi lebih baik. Terdapat penelitian yang menerapkan SVM-SMOTE, penelitian ini membuktikan bahwa teknik ini meningkatkan efisiensi dan performa dari model klasifikasi IDS, tetapi dengan konsumsi waktu sekitar 9 menit (Mahalakshmi dkk., 2022). Pada penelitian yang dilakukan oleh Ndichu, dkk. (2021), terbukti bahwa menggunakan SVM-SMOTE dapat mengatasi masalah dataset yang tidak seimbang serta meningkatkan performa dari model klasifikasi. Ilustrasi penggunaan SVM-SMOTE dapat dilihat pada Gambar 2.3.

2.4.5 Random Oversampling

Random oversampling merupakan teknik yang simpel untuk digunakan. Teknik ini meningkatkan ukuran dataset dengan pengulangan sampel asli atau menduplikasi sampel asli. Dengan kata lain, teknik ini tidak membuat sampel baru dan variasi sampel tidak berubah (Ghorbani & Ghousi, 2020). Tanpa mensintesis sampel baru, sampel dibuat dengan menyalin sampel kelas minoritas secara acak. Kemudian, jumlah kelas minoritas dan mayoritas sama sehingga diperoleh dataset yang seimbang (Ding dkk., 2022). Ilustrasi penggunaan *Random Oversampling* dapat dilihat pada Gambar 2.4.

2.4.6 Random Undersampling

Random undersampling adalah teknik yang digunakan untuk memecahkan masalah dataset yang tidak seimbang. Untuk menghasilkan distribusi yang lebih seimbang, *random undersampling* melibatkan penghapusan sampel secara acak dari kelas mayoritas seperti yang dapat dilihat pada Gambar 2.5. Tujuannya adalah untuk meningkatkan kapasitas model dalam mengidentifikasi pola-pola di kelas minoritas sekaligus mencegah agar model tersebut tidak terlalu terpengaruh oleh kelas mayoritas. Pada penelitian yang dilakukan oleh Zuech, dkk. (2021), dibuktikan bahwa penerapan teknik ini dapat meningkatkan performa klasifikasi untuk mendeteksi serangan web pada dataset CSE-CIC-IDS2018.

2.4.7 SMOTE-Tomek

SMOTE-*Tomek* adalah *hybrid sampling*, kombinasi dari SMOTE dan *Tomek Links* yang dirancang untuk mengatasi distribusi kelas yang tidak seimbang pada dataset. SMOTE digunakan untuk menghasilkan sampel sintetik dari kelas minoritas, sehingga menyeimbangkan distribusi kelas, sementara *Tomek Links* digunakan untuk mengidentifikasi dan menghapus sampel yang berpotensi menimbulkan gangguan atau garis batas yang dapat

menyebabkan kesalahan klasifikasi (Jiajia dkk., 2021). Dengan menggabungkan teknik ini, SMOTE-*Tomek* bertujuan untuk meningkatkan performa generalisasi secara keseluruhan pada model *machine learning* sehingga mendapatkan hasil yang lebih akurat dan kuat. Pada penelitian yang dilakukan oleh Mbow, dkk. (2021), terbukti bahwa penggunaan SMOTE-*Tomek* jelas dapat meningkatkan performa dari IDS dan membuat model menjadi sangat sederhana. Ilustrasi penggunaan SMOTE-*Tomek* dapat dilihat pada Gambar 2.6.

2.4.8 Tomek Links

Tomek links adalah teknik *undersampling* yang bekerja dengan cara mengidentifikasi dan menghapus pasangan sampel, yang dikenal sebagai *Tomek Links*, di mana dua sampel dari kelas yang berbeda merupakan tetangga terdekat satu sama lain. Tujuan penghapusan ini adalah untuk menyeimbangkan distribusi kelas minoritas dan mayoritas. Berdasarkan penelitian yang dilakukan oleh (Thiyam & Dey, 2023) dan (Batchu & Seetha, 2021), *Tomek Links* dapat mengatasi masalah dataset yang tidak seimbang dan mampu meningkatkan performa dari model klasifikasi. Ilustrasi penggunaan *Tomek Links* dapat dilihat pada Gambar 2.7.

2.5 Machine Learning

Machine learning adalah cabang ilmu komputer yang melibatkan penggunaan algoritma dan model statistik untuk memungkinkan sistem komputer belajar dari data, tanpa diprogram secara eksplisit. Tujuan *machine learning* adalah untuk mengembangkan sistem yang secara otomatis dapat meningkatkan kinerjanya pada tugas tertentu dari waktu ke waktu, berdasarkan pengalaman. Ada berbagai jenis algoritma machine learning, yaitu: *supervised learning*, *unsupervised learning*, dan *reinforcement learning*. *Supervised learning* melibatkan pelatihan model pada data berlabel, sedangkan *unsupervised learning* melibatkan pelatihan model pada data tidak berlabel. *Reinforcement learning* melibatkan pelatihan model untuk membuat keputusan berdasarkan umpan balik dari lingkungannya (Panesar, 2019).

Pada penelitian ini, digunakan algoritma *supervised learning*. Algoritma yang akan digunakan adalah *Decision Tree*, *Random Forest*, *Gradient Boosting*, *XGBoost* dan 1D-CNN. Algoritma tersebut diuraikan sebagai berikut.

2.5.1 Decision Tree

Decision tree adalah algoritma *supervised machine learning* yang digunakan untuk tugas klasifikasi dan regresi. Algoritma ini memiliki struktur seperti pohon di mana setiap *node*

internal mewakili pengujian pada suatu atribut, cabang mewakili hasil pengujian, dan *node* daun mewakili label atau keputusan kelas (Ndichu dkk., 2022). *Decision tree* telah diterapkan di berbagai bidang, termasuk *network intrusion detection*, diagnosis medis, penilaian kredit, dan deteksi penipuan, dan masih banyak lagi. Pada penelitian yang dilakukan oleh Ndichu, dkk. (2022) *decision tree* digunakan untuk membedakan kelas serangan pada dataset dan secara efisien dapat menangani data renggang berdimensi tinggi untuk membedakan kelas *security alert* dari kelas *non-security alert*.

2.5.2 Random Forest

Random forest adalah algoritma *machine learning* yang digunakan untuk tugas klasifikasi dan regresi. Ini adalah metode *ensemble learning* yang beroperasi dengan membangun banyak *decision tree* pada waktu pelatihan dan melakukan klasifikasi kelas atau prediksi rata-rata (regresi) dari masing-masing pohon (Li dkk., 2022). *Random forest* menyempurnakan algoritma *decision tree*, yang rentan terhadap *overfitting*, dengan menggunakan prediksi rata-rata dari masing-masing pohon. Algoritma ini dikenal karena akurasinya yang tinggi, fleksibilitas, dan ketahanannya terhadap *overfitting*. Pada penelitian yang dilakukan oleh Li, dkk. (2022), terbukti bahwa metode *ensemble learning* khusunya *random forest* merupakan algoritma yang sangat kuat pada *machine learning*. Karena itu, mereka menggunakan *random forest*.

2.5.3 Gradient Boosting

Gradient boosting adalah algoritma *machine learning* yang menggabungkan beberapa *weak learner* untuk menciptakan *strong learner*. *Weak learner* adalah model yang memiliki daya prediksi yang rendah, tetapi jika digabungkan dengan *weak learner* lainnya, model tersebut dapat menghasilkan prediksi yang akurat. Algoritma *gradient boosting* bekerja dengan menambahkan *weak learner* ke model secara berulang. Setiap *weak learner* dilatih untuk meminimalkan gradien *loss function* sehubungan dengan prediksi *weak learner* sebelumnya (Sabari & Shrivastava, 2022). *Loss function* adalah ukuran seberapa cocok prediksi model dengan nilai sebenarnya. Pada penelitian yang dilakukan oleh Sabari & Shrivastava. (2022), terbukti bahwa *gradient boosting* mampu mengungguli performa dari *random forest*.

2.5.4 XGBoost

XGBoost, kependekan dari *eXtreme Gradient Boosting*, adalah algoritma *machine learning* yang sangat populer dan kuat yang digunakan untuk tugas regresi dan klasifikasi. *XGBoost* adalah implementasi *gradient boosting*, model ini secara berurutan membangun serangkaian *weak learner*, biasanya *decision tree*, untuk memperbaiki kesalahan yang dibuat oleh model sebelumnya. Algoritma ini menggunakan *regularization*, mengatasi *overfitting* dengan memberikan penalti pada model yang kompleks. Algoritma ini memiliki fitur pemangkasan pohon yang efisien, memungkinkan kontrol atas kedalaman pohon, dan mendukung pemrosesan paralel untuk pelatihan model yang dipercepat pada dataset berukuran besar. Memiliki *cross-validation* bawaan, mempermudah menangani *missing values* pada dataset. Telah terbukti kerangka kerja (Li dkk., 2022) yang menggunakan *XGBoost* mampu mengungguli performa dari *classifier* dasar lainnya yaitu, *LightGBM* dan *CatBoost*.

2.5.5 1D-CNN

Deep learning adalah jenis *machine learning* yang menggunakan jaringan syaraf tiruan untuk memungkinkan komputer belajar dari data. Disebut "*deep*" karena algoritma ini menggunakan arsitektur kompleks dengan banyak lapisan pemrosesan. *Deep learning* telah diterapkan ke berbagai bidang, seperti *computer vision*, *natural language processing*, dan *voice recognition*. Teknologi ini telah menunjukkan keberhasilan besar dalam tugas-tugas seperti pengenalan gambar dan ucapan, dan merupakan teknologi kunci di balik banyak kemajuan terkini dalam *artificial intelligence* (Paluszek & Thomas, 2020). Salah satu contoh dari *deep learning* adalah 1D-CNN yang digunakan dalam penelitian ini.

1D-CNN adalah singkatan dari *one-dimensional Convolutional Neural Network*. Ini adalah jenis jaringan saraf yang biasa digunakan untuk *supervised learning* pada *time-series data*. Dalam konteks *network intrusion detection*, 1D-CNN dapat digunakan untuk mengekstrak representasi fitur tingkat tinggi yang mewakili bentuk abstrak kumpulan fitur tingkat rendah dari koneksi lalu lintas jaringan. Pada penelitian ini, digunakan arsitektur 1D-CNN dari penelitian yang dilakukan oleh (Azizjon dkk., 2020).

2.6 Teknik Evaluasi

Evaluasi adalah proses menganalisis performa model *machine learning* atau *deep learning* menggunakan matrik. Secara umum, matriks yang biasa digunakan untuk mengukur performa didasarkan pada atribut berbeda yang digunakan dalam *Confusion Matrix* seperti terlihat pada

Tabel 2.3. Matriks ini memiliki ukuran dua dimensi dan memberikan informasi tentang kelas Aktual dan Prediksi (Ahmad dkk., 2021).

Tabel 2.3 *Multi-Class Confusion Matrix* (Bharathi, 2021)

		Predicted Class		
		Class 1	*Class 2*	*Class 3*
Actual Class	*Class 1*	*True Positive (cell 1)*	*False Negative (cell 2)*	*False Negative (cell 3)*
	Class 2	*False Positive (cell 4)*	*True Negative (cell 5)*	*True Negative (cell 6)*
	Class 3	*False Positive (cell 7)*	*True Negative (cell 8)*	*True Negative (cell 9)*

Adapun uraian setiap atribut di dalam *Multi-Class Confusion Matrix* pada Tabel 2.3 sebagai berikut:

a. *True Positive* (TP): Sampel diprediksi sebagai *class 1* dan faktanya sampel tersebut benar *class 1*. Nilai TP diperoleh dari *cell 1*.
b. *False Negative* (FN): Sampel diprediksi sebagai *class 2* atau *class 3* dan faktanya sampel tersebut adalah *class 1*. Nilai FN diperoleh dari jumlah semua nilai pada baris kelas berada (*cell 2* dan *cell 3*) kecuali nilai TP.
c. *False Positive* (FP): Sampel diprediksi sebagai *class 2* atau *class 3* dan faktanya sampel tersebut adalah *class 1*. Nilai FP diperoleh dari jumlah semua nilai pada kolom tempat kelas berada (*cell 4* dan *cell 7*) kecuali nili TP.
d. *True Negative* (TN): Sampel diprediksi sebagai *class 2* atau *class 3* dan faktanya sampel tersebut adalah *class 2* atau *class 3*. Nilai TN diperoleh dari jumlah semua nilai pada kolom dan baris (*cell 5, cell 6, cell 7,* dan *cell 8*) kecuali target *class*.

Informasi tersebut kemudian digunakan untuk menghitung skor akurasi, presisi, *recall*, dan F1-*score* seperti pada persamaan (2.2) sampai (2.5). Pada kasus *network intrusion detection*, pengertian dari akurasi, presisi, *recall*, dan F1-*score* diuraikan sebagai berikut:

a. Akurasi mengukur proporsi sampel yang diklasifikasikan dengan benar terhadap jumlah total sampel (EvidentlyAITeam, 2023).

$$Akurasi = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}} \quad (2.2)$$

b. Presisi adalah sampel yang diklasifikasikan dengan benar sebagai milik kelas tertentu dari semua sampel yang diprediksi oleh model untuk menjadi bagian dari kelas tersebut (EvidentlyAITeam, 2023).

$$Presisi = \frac{\text{TP}}{\text{TP+FP}} \quad (2.3)$$

c. *Recall* adalah sampel pada suatu kelas yang diklasifikasikan dengan benar oleh model dari semua sampel dalam kelas tersebut (EvidentlyAITeam, 2023).

$$Recall = \frac{\text{TP}}{\text{TP+FN}} \quad (2.4)$$

d. F1-*score* diartikan sebagai *harmonic mean* dari Presisi dan *Recall*. Dengan kata lain, ini adalah teknik statistik untuk memeriksa keakuratan suatu sistem dengan mempertimbangkan presisi dan *recall* (Ahmad dkk., 2021).

$$F1_{\text{score}} = 2\left(\frac{\text{Presisi .Recall}}{\text{Presisi+Recal}}\right) \quad (2.5)$$

2.7 Penelitian Terkait

Penelitian ini bertujuan agar dapat mengetahui cara penggunaan teknik *resampling* pada dataset intrusi jaringan dan mengetahui pengaruh dari penerapan teknik *resampling* pada data pelatihan terhadap performa model deteksi intrusi jenis jaringan. Adapun beberapa penelitian terdahulu telah dilakukan oleh peneliti lain yang mirip dan digunakan sebagai acuan pada penelitian ini. Berikut beberapa penelitian terdahulu yang dapat dilihat pada Tabel 2.4.

Tabel 2.4 Penelitian Terkait

Referensi	Dataset	Teknik Resampling	Algoritma Model	Hasil
(Bagui & Li, 2021)	KDD Cup 1999, UNSW-NB15, UNSW-NB17, dan UNSW-NB18	*Random Undersampling*, *Random Oversampling*, RU-RO, RU-SMOTE, RU-ADASYN	ANN	Skor *macro precision*, *macro recall*, *macro* F1-*score* dibandingkan dengan skor sebelum dilakukan *resampling*. Terdapat hasil yang membuktikan bahwa secara keseluruhan teknik *Random Undersampling* dan *Random Oversampling* dapat meningkatkan performa model dan mengungguli teknik *hybrid sampling* yang digunakan pada penelitian ini, yaitu RU-RO, RU-SMOTE, dan RU-ADASYN. Terdapat keuntungan lain, yaitu waktu untuk melakukan *training* model yang menggunakan teknik *resampling* umumnya lebih rendah dibandingkan tanpa menggunakan teknik *resampling*.
(Li dkk., 2022)	CIC-IDS2017	SMOTE, ADASYN, *Borderline* SMOTE	*Random Forest*, *XGBoost*, *LightGBM*, *CatBoost*	Hasil eksperimen membuktikan bahwa penerapan teknik *resampling* mampu meningkatkan performa dari model. Adapun performa terbaik didapatkan dari model *XGBoost* yang diterapkan teknik *resampling* SMOTE. Kombinasi ini mengungguli model lainnya dengan *Macro Average Precision* 93.2%, *Macro Average* F1-*score* 95.5%, *Macro Average AUC* 99.4%, dan *Macro Average Recall* 98.9%.
(Lee dkk., 2022)	CIC-IDS2017	*Random Oversampling*, SMOTE, *Borderline* SMOTE, ADASYN, OSS-ROS, OSS-SMOTE, OSS-*Borderline* SMOTE, OSS-ADASYN	CNN	Skor akurasi, F1-*Score (micro)*, dan F1-*Score (macro)* dibandingkan dengan sebelum dilakukan teknik *resampling*. Terakhir didapatkan kesimpulan bahwa kombinasi dari *One Side Selection* sebagai *undersampling* dan *Borderline* SMOTE sebagai *oversampling* pada dataset CIC-IDS2017 dapat meningkatkan performa model dengan algoritma CNN dengan skor akurasi 94.58%, F1-*score (micro)* 94.58%, dan F1-*score (macro)* 91.36% dibandingkan dengan teknik *resampling* lainnya dan sebelum dilakukan *resampling*.

(Rahma dkk., 2023)	UNSW-NB15	*Random Oversampling*, SMOTE, ADASYN, *Random Undersampling*, AIIKNN, *Tomek Links*, SMOTE-ENN, SMOTE-*Tomek*	CNN-BiLSTM	Hasil eksperimen yang dilakukan pada dataset UNSW-NB15 mendapatkan akurasi dan presisi tertinggi dengan tanpa dilakukan *resampling*, yaitu sebesar 0.965 dan 0.777. Sementara itu, *recall* dan F1-*score* tertinggi didapatkan dari penerapan teknik *Random Oversampling*, yaitu sebesar 0.677 dan 0.589.
(Khan dkk., 2023)	UNSW-NB15 dan CIC-IDS2017	*Random Oversampling*, ADASYN, SMOTE-*Tomek*, SMOTE	*Random Forest*, *Gradient Boosting*, *Extra Tree*, *Multi Layer Perceptron*, OE-IDS	Hasil eksperimen secara keseluruhan dapat disimpulkan bahwa menggunakan data hasil *resampling* SMOTE ditinjau dari akurasi, presisi, *recall*, dan F1-*score*, lebih baik daripada teknik *resampling* lainnya yang digunakan dalam eksperimen.
(Abdelkhalek & Mashaly, 2023)	NSL-KDD	ADASYN, ADASYN-*Tomek Links*, *Random Oversampling*, *Random Undersampling*	*Multi Layer Perceptron*, DNN, CNN, CNN-BLSTM	Berdasarkan hasil eksperimen, kombinasi dari ADASYN-*Tomek Links* dapat meningkatkan *detection rate* dari kelas minoritas dibandingkan dengan tidak menggunakan teknik *resampling*. Pada klasifikasi *binary*, model CNN dengan penerapan teknik *resampling* ADASYN-*Tomek Links* mendapatkan akurasi 99.8% dan *detection rate* 99%. Sedangkan pada klasifikasi *multi-class* model MLP dengan penerapan teknik *resampling* ADASYN-*Tomek Links* mendapatkan akurasi 99.9% dan *detection rate* 99%.
(Mahalakshmi dkk., 2022)	SCADA (*Supervisory Control And Data Acquisition*)	SVM-SMOTE	*Logistic Regression*, *Cost Sensitive Learning*	Akurasi tertinggi didapatkan dari model *Logistic Regression* tanpa *resampling* 78.4%. Sedangkan untuk presisi, *recall*, dan F1-*score* tertinggi didapatkan dari model *Logistic Regression* yang dibantu *Cost Sensitive Learning* dengan diterapkan juga teknik SVM-SMOTE dan mendapatkan skor presisi 69.1%, *recall* 66.6%, dan F1-*score* 66.2%. Terdapat juga penurunan dri FPR, FNR, dan FAR secara signifikan ketika diterapkan teknik SVM-SMOTE.
Penelitian ini	UNSW-NB15	ADASYN, *Borderline* SMOTE, *Random*	*Decision Tree*, *Random Forest*,	Hasil penelitian dibahas di Bab IV

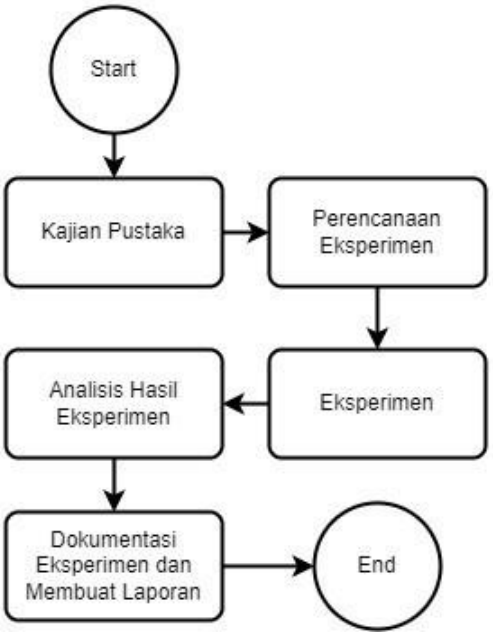
		Oversampling, *Random Undersampling*, SMOTE, SMOTE-*Tomek*, SVM-SMOTE, *Tomek Links*	*Gradient Boosting*, *XGBoost*, 1D-CNN	

BAB III
METODOLOGI PENELITIAN

Bab ini berisi alur pengerjaan tugas akhir dari awal sampai akhir, yaitu: kajian pustaka, perencanaan eksperimen, eksperimen, analisis hasil eksperimen, dan membuat laporan. Setiap tahapan eksperimen yang diuraikan pada bab ini, yaitu: dataset, *pre-processing*, *resampling* data, pelatihan model, pengujian, dan evluasi.

3.1 Alur Pengerjaan Tugas Akhir

Proses pengerjaan tugas akhir ini tergambar dalam alur yang memberikan ikhtisar mengenai penelitian yang dilakukan mulai dari tahap awal hingga penyelesaian akhir. Alur pengerjaan tersebut dapat dilihat pada Gambar 3.1.



Gambar 3.1 Alur Pengerjaan Tugas Akhir

Berdasarkan diagram alur pengerjaan, terdapat beberapa langkah-langkah pada penelitian ini, yaitu:

a. Kajian Pustaka

Langkah pertama dalam pengerjaan tugas akhir ini adalah melakukan kajian pustaka dengan cara mencari dan membaca jurnal dan artikel penelitian terkait dengan "*imbalanced data*", "*network intrusion detection*", dan "*resampling technique*".

b. Perencanaan Eksperimen

Langkah kedua dalam pengerjaan tugas akhir ini adalah merencanakan eksperimen yang akan dilakukan, dalam hal ini direncanakan menggunakan dataset UNSW-NB15, selanjutnya dipilih teknik *resampling*, yaitu: ADASYN, *Borderline* SMOTE, *Random Oversampling*, *Random Undersampling*, SMOTE, SMOTE-Tomek, SVM-SMOTE, dan *Tomek Links*. Kemudian dipilih algoritma model, yaitu: *Decision Tree*, *Random Forest*, *Gradient Boosting*, *XGBoost*, dan 1D-CNN.

c. Eksperimen

Langkah ketiga dalam pengerjaan tugas akhir ini adalah melaksanakan eksperimen berdasarkan yang sudah direncanakan pada langkah kedua. Eksperimen dilaksanakan mulai dari mendapatkan dataset, melakukan *pre-processing* pada data, menerapkan teknik *resampling* pada data pelatihan, melakukan *training* pada model menggunakan data yang sudah diproses, melakukan *testing*, dan terakhir melakukan evaluasi.

d. Analisis Hasil Eksperimen

Langkah keempat dalam pengerjaan tugas akhir adalah dengan menganalisis hasil eksperimen, yaitu menganalisis hasil evaluasi pada langkah eksperimen dan membuat kesimpulan dari analisis tersebut.

e. Dokumentasi Eksperimen dan Membuat Laporan

Langkah kelima dalam pengerjaan tugas akhir adalah menyimpan hasil eksperimen seperti model dan hasil prediksi, dan melaporkan penelitian dengan membuat laporan tugas akhir ini.

3.2 Uraian Pengerjaan Tugas Akhir

3.2.1 Kajian Pustaka

Pada langkah ini, dilakukan kajian pustaka dengan mencari artikel ilmiah dan jurnal dari IEEE Xplore dan ScienceDirect menggunakan kata kunci "*imbalanced data*", "*network intrusion detection*", dan "*resampling technique*". Kemudian menyeleksi jurnal dan artikel penelitian yang diperoleh berdasarkan kriteria sebagai berikut:

a. Publikasi makalah penelitian diterbitkan pada atau setelah 2021;

b. Makalah merupakan artikel konferensi dan jurnal. Literature review akan dikecualikan;

c. Makalah penelitian terkait tentang *network intrusion detection*;

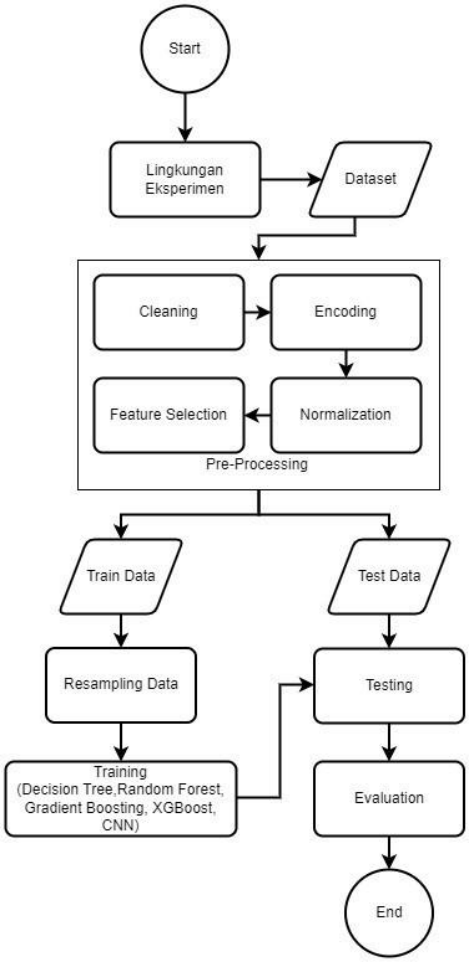
d. Dataset yang digunakan terkait dengan *network intrusion detection*;

e. Adanya nama dan penjelasan penggunaan teknik *resampling* yang digunakan;

f. Makalah penelitian dapat diakses atau *open acces*.

Setelah melakukan seleksi berdasarkan kriteria tersebut, selanjutnya adalah membaca dan mengekstrak informasi dengan dibantu sebuah alat *spreadsheet*. Hasil ekstraksi informasi ini telah dipresentasikan pada konferensi ilmiah internasional (Rajasa dkk., 2023) dan digunakn dalam langkah perencanaan eksperimen.

3.2.2 Perencanaan Eksperimen



Gambar 3.2 Alur Eksperimen

Setelah melakukan kajian pustaka langkah selanjutnya adalah perencanaan eksperimen. Penelitian ini mengikuti diagram alur eksperimen yang sudah dibuat yang dapat dilihat pada Gambar 3.2. Berdasarkan diagram tersebut dapat diketahui langkah-langkah eksperimen yang akan dilakukan. Pertama, menentukan lingkungan eksperimen; kedua, menentukan dataset yang akan digunakan; ketiga, melakukan *pre-processing* seperti membersihkan data, melakukan *encoding* pada data, normalisasi data, dan seleksi fitur yang akan digunakan pada data; keempat, mengatasi ketidakseimbangan data dengan menerapkan teknik *resampling*; kelima, melakukan *training* menggunakan algoritma model yang sudah direncanakan dan data pelatihan yang sudah diolah; keenam, melakukan *testing* menggunakan model yang sudah dilatih dengan data pengujian yang sudah diolah; ketujuh, melakukan evaluasi skor dari model yang sudah dilatih dan diuji.

3.2.3 Eksperimen

a. Lingkungan Eksperimen

Pada langkah ini, ditentukan lingkungan yang dapat mendukung kelancaran proses eksperimen, mulai dari *hardware* sampai *software* yang digunakan. *Hardware* yang digunakan berupa laptop Acer Predator Triton 500 dengan spesifikasi seperti pada Tabel 3.1. Sementara itu, *software* yang digunakan adalah *Miniconda3* sebagai *environment* dan *Visual Studio Code* sebagai *code editor*. Pada *Miniconda3*, dibuat dua *environment* dengan beberapa *library* berbeda, yaitu *myenv* dan *tf_gpu*. *Environtment myenv* digunakan untuk menjalankan kode menggunakan CPU. Sementara itu, *tf_gpu* digunakan untuk menjalankan kode dengan algoritma model 1D-CNN menggunakan *TensorFlow* GPU. *Library* yang digunakan pada dua *environment* ini dapat dilihat pada Tabel 3.2.

b. Dataset

Pada langkah ini, ditentukan dataset yang akan digunakan. Dataset yang digunakan adalah UNSW-NB15 karena dataset ini memiliki masalah ketidakseimbangan data. Secara spesifik yang digunakan dalam eksperimen adalah subset UNSW-NB15 yaitu, *Training Set* dan *Testing Set*. Subset tersebut akan digunakan sebagai input pada eksperimen di penelitian ini. Proporsi dari dataset ini dapat dilihat pada Tabel 2.1.

Tabel 3.1 Spesifikasi *hardware*

Spesifikasi	
Laptop	Acer Predator Triton 500
Processor	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz
GPU	Nvidia GeForce RTX 2060 6GB
Storage	SSD NVMe PCIex3 512GB
RAM	16 GB DDR4
Sistem Operasi	Windows 10
Tipe Sistem	64-bit

Tabel 3.2 *Library* yang digunakan pada *environment*

Environment	
myenv	**tf_gpu**
python v3.10 pandas matplotlib seaborn imbalanced-learn scikit-learn xgboost joblib	python v3.10 pandas matplotlib seaborn numpy imbalanced-learn scikit-learn tensorflow v2.10.0 tensorflow_addons keras joblib

c. Pre-Processing

Pada langkah ini dilakukan *pre-processing* pada data untuk meningkatkan kualitasnya, mengatasi masalah yang mungkin timbul, dan membuat data lebih sesuai untuk digunakan dalam algoritma model *machine learning* dan *deep learning*. Pada langkah ini dilakukan *cleaning*, *encoding*, *normalization*, dan *feature selection*. Secara detail akan diuraikan sebagai berikut.

Cleaning

Pada proses ini, dilakukan pembersihan data dengan label “0” dari data karena penelitian ini hanya menggunakan data dengan label “1”. Pembersihan ini dilakukan pada *Training Set* dan *Testing Set*. Hal ini dilakukan karena hasil klasifikasi yang

diharapkan adalah berupa deteksi jenis serangan saja. Pembersihan data dengan *missing value* dan NaN tidak dilakukan karena dataset sudah bersih dari hal tersebut.

Encoding

Pada proses ini, diubah tipe data untuk kolom yang berisi string atau berupa *object* menjadi *numerical*. Kolom tersebut adalah kolom *attack_cat*, *state*, *service*, dan *proto*. Proses ini dilakukan pada *Training Set* dan *Testing Set*. Hal ini dilakukan karena pelatihan model memerlukan input dalam bentuk angka. Teknik *encoding* yang dipilih adalah *One-hot Encoding* dan *Label Encoding*. Pada penelitian ini, *One-hot Encoding* dilakukan pada model dengan algoritma *deep learning*, yaitu 1D-CNN. Hal ini dilakukan agar input data yang berupa *numeric* bisa dilatih untuk menghitung *categorical cross-entropy loss* yang didesain untuk bekerja dengan *one-hot encoded vector*. Kemudian *Label Encoding* digunakan karena mudah untuk diterapkan. Pada penelitian ini, *Label Encoding* digunakan pada model dengan algoritma *machine learning*, yaitu *Decision Tree*, *Random Forest*, *Gradient Boosting*, dan *XGBoost*.

Normalization

Pada proses ini, dilakukan normalisasi pada semua kolom pada *Training Set* dan *Testing Set*. Normalisasi bekerja dengan mentransformasi rentang skala pada data menjadi rentang tertentu, misalnya: -1.0 ke 1.0 atau 0.0 ke 1.0. Selain itu, tujuan dilakukan normalisasi adalah untuk memastikan bahwa skala atau rentang nilai dari berbagai variabel seragam, sehingga mencegah pengaruh dominan dari variabel dengan skala besar. Pada penelitian ini, dipilih *Min-Max Scaler* karena pada teknik ini dapat memilih rentang spesifik untuk normalisasi dataset.

Feature Selection

Proses terakhir, dilakukan pemilihan fitur yang difokuskan untuk menghilangkan prediktor yang redundan dari dataset. Pada penelitian ini, dipilih *Pearson Correlation* untuk pemilihan fitur. Hal ini dilakukan karena *Pearson Correlation* mudah untuk diterapkan dan hasil yang ditampilkan juga mudah untuk dipahami.

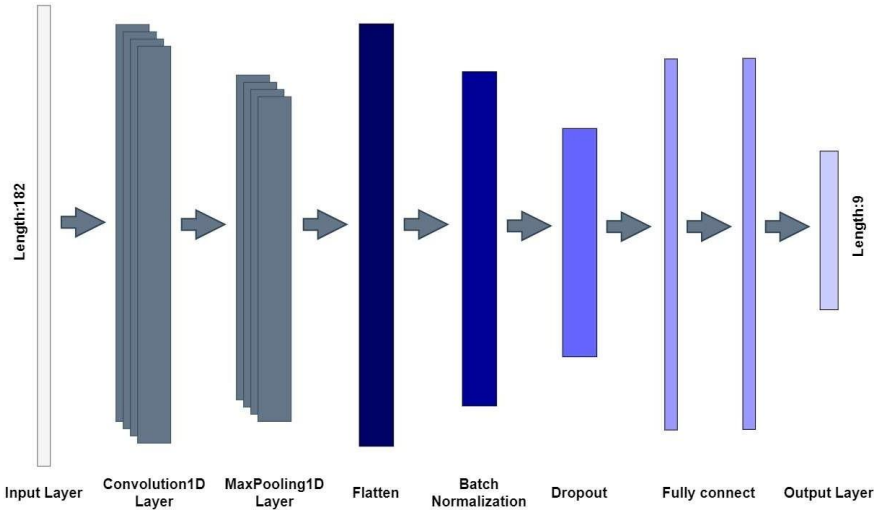
d. *Resampling* Data

Pada langkah ini, diterapkan berbagai macam teknik *resampling* pada *Training Set*. Adapun teknik *oversampling* yang digunakan pada penelitian ini, yaitu: ADASYN, *Borderline* SMOTE, *Random Oversampling*, SMOTE dan SVM-SMOTE. Kemudian teknik *undersampling* yang digunakan adalah *Random Undersampling* dan

Tomek Links. Teknik *hybrid sampling* yang digunakan adalah SMOTE-*Tomek*. Ini dilakukan untuk meningkatkan kualitas data dan mengatasi masalah dataset yang tidak seimbang. Semua teknik *resampling* tersebut dipilih karena terbukti dapat memengaruhi performa model *machine learning* ataupun *deep learning* berdasarkan temuan pada penelitian terkait. Untuk penerapan teknik *resampling*, digunakan *library imbalance-learn*. *Library* tersebut dapat diinstal pada *environment* yang sudah disiapkan dengan menggunakan *pip*.

e. Pelatihan Model

Pada langkah ini, dilakukan *training* model menggunakan beberapa algoritma model *machine learning*, yaitu: *Decision Tree*, *Random Forest*, *Gradient Boosting*, dan *XGBoost*. Algoritma tersebut dipilih karena biasa digunakan oleh peneliti yang melakukan penelitian terkait *network intrusion detection* dan memiliki performa yang baik untuk deteksi serangan. *Library* yang digunakan untuk model *machine learning* adalah *scikit-learn*. Selanjutnya, dilakukan juga *training* menggunakan algoritma model *deep learning* CNN, lebih spesifik *One Dimension Convolutional Neural Network* (1D-CNN) seperti yang dilakukan oleh Azizjon dkk. (2020). Algorithm 1D-CNN dipilih karena sangat cocok untuk tugas yang melibatkan *sequential data*, seperti *time series analysis*, *signal processing*, dan *natural language processing* (Azizjon dkk., 2020). Sementara itu, data yang digunakan pada eksperimen ini berupa *network traffic data* yang mana sangat cocok dengan 1D-CNN. Arsitektur dari model 1D-CNN yang digunakan dapat dilihat pada Gambar 3.3.



Gambar 3.3 Arsitektur 1D-CNN (Azizjon dkk., 2020)

Berdasarkan Gambar 3.3, dapat diketahui arsitektur dari 1D-CNN berupa input dengan panjang 182, *layer convolutional*, *layer max pooling*, *layer flatten*, *layer batch normalization*, *layer dropout*, dua *layer fully connected*, dan terakhir *layer output* dengan 9 kelas.

f. Pengujian

Pada langkah ini dilakukan *testing* menggunakan *Testing Set* yang sudah diolah dan disiapkan pada langkah *pre-processing*. Langkah ini akan menghasilkan hasil prediksi kelas serangan.

g. Evaluasi

Langkah terakhir, dilakukan evaluasi dengan melihat skor matriks. Hasil *testing* pada langkah sebelumnya yang berupa prediksi, dicocokkan dengan nilai aslinya. Kemudian nilai pada matriks dikalkulasikan untuk mendapatkan skor akurasi, *recall*, presisi, dan F1-*score*. Pada penelitian ini, digunakan *library scikit-learn* untuk menghitung skor-skor tersebut.

3.2.4 Analisis Hasil Eksperimen

Hasil eksperimen akan dianalisis secara mendalam untuk menggali pemahaman yang lebih baik terkait performa model yang telah dikembangkan. Analisis ini mencakup evaluasi performa model dengan menggunakan matriks seperti akurasi, *recall*, presisi dan F1-*score*.

Setiap anomali atau temuan menarik yang muncul selama eksperimen akan diperinci dan penjelasan akan diberikan untuk menggambarkan faktor-faktor yang mungkin memengaruhi hasil tersebut. Seluruh analisis ini akan membentuk dasar untuk menyusun kesimpulan yang baik.

3.2.5 Dokumentasi Eksperimen dan Membuat Laporan

Setelah selesai melakukan tahapan eksperimen dan analisis, langkah selanjutnya adalah dokumentasi eksperimen. Model *machine learning* dan hasil prediksi akan disimpan menggunakan *library joblib*, sedangkan model *deep learning* akan disimpan menggunakan *library keras*. Kemudian, disajikan temuan dan hasil penelitian secara terstruktur melalui pembuatan laporan. Pada tahap ini, dirinci struktur dan format yang akan digunakan dalam penyusunan laporan, termasuk hal-hal seperti judul, pendahuluan, landasan teori, metodologi penelitian, hasil, dan kesimpulan. Proses ini penting untuk memastikan bahwa laporan penelitian mampu menyampaikan informasi secara jelas dan efektif kepada pembaca. Keseluruhan proses ini akan memastikan bahwa kontribusi penelitian ini dapat diakses, dipahami, dan diaplikasikan oleh masyarakat ilmiah dan pemangku kepentingan terkait.

BAB IV
HASIL DAN PEMBAHASAN

Bab ini berisi pembahasan dari setiap aktivitas yang dilakukan dalam pelaksanaan eksperimen. Bab ini juga berisi hasil evaluasi untuk mengetahui kelebihan dan kekurangan dari teknik *resampling* yang sudah diimplementasi. Analisis performa dari teknik *resampling* yang diterapkan berdasarkan hasil evaluasi dipaparkan di akhir bab ini.

4.1 Lingkungan Eksperimen

Pada langkah pertama, dibuat lingkungan eksperimen yang dapat mendukung kelancaran eksperimen. Spesifikasi dari *hardware* yang digunakan pada penelitian ini dapat dilihat pada Tabel 3.1. Selanjutnya, digunakan *software Miniconda3* sebagai wadah untuk *environment* pada penelitian ini. Terdapat dua *environment* yang dibuat, yaitu *myenv* yang digunakan untuk algoritma *machine learning* dan *tf_gpu* yang digunakan untuk algoritma *deep learning*. Kemudian dilakukan instalasi library yang diperlukan seperti pada Tabel 3.2 di masing-masing *environment*. Pembuatan *environment* dilakukan menggunakan terminal *Miniconda3* dengan perintah seperti pada Gambar 4.1. Selanjutnya dilakukan instalasi *library* pada *environment myenv* menggunakan perintah seperti pada Gambar 4.2.

```
conda create -n myenv python=3.10
```

Gambar 4.1 Kode untuk membuat *environment myenv*

```
pip install pandas matplotlib seaborn imbalanced-learn scikit-learn xgboost joblib
```

Gambar 4.2 Kode untuk instalasi *library* pada *environment myenv*

Adapun pada *environment tf_gpu*, agar kalkulasi bisa dijalankan menggunakan GPU pada *hardware* dengan sistem operasi *windows*, diperlukan tambahan *software Cuda Toolkit* v11.2 dan *library cuDNN* v8.1. Kedua tambahan tersebut dapat diunduh dari web *Nvidia Developer*. Untuk versi *Cuda Toolkit* dan *cuDNN* yang diperlukan dapat dilihat pada web *TensorFlow*. Kemudian dilakukan instalasi *Cuda Toolkit*, setelah itu *library cuDNN* diekstrak dan dipindahkan ke folder tempat instalasi *Cuda Toolkit*. Selanjutnya dibuka *Edit the system environment variables*, diklik pada *Environment Variables*, ditambahkan *path* baru untuk folder *bin* dan *libnvvp* yang berada di lokasi instalasi *Cuda Toolkit*, dan dilakukan *restart*

windows. Setelah itu, dilakukan pembuatan *environment tf_gpu* menggunakan perintah seperti pada Gambar 4.3. Kemudian dilakukan instalasi *library* yang diperlukan untuk *environment tf_gpu* menggunakan perintah pada Gambar 4.4.

```
conda create -n tf_gpu python=3.10
```

Gambar 4.3 Kode untuk membuat *environment tf_gpu*

```
pip install pandas matplotlib seaborn numpy imbalanced-learn scikit-learn
tensorflow==2.10.0 tensorflow_addons keras joblib
```

Gambar 4.4 Kode untuk instalasi *library* pada *environment tf_gpu*

4.2 Dataset

Langkah kedua eksperimen pada penelitian ini adalah mencari dataset untuk digunakan. Dataset pada penelitian ini didapatkan dari website resmi *University of New South Wales* (Moustafa & Slay, 2015). Subset yang digunakan adalah *Training Set* dan *Testing Set* yang berupa file berformat csv. Setelah didapatkan dataset, selanjutnya dibaca masing-masing subset dan dijalankan kode seperti yang terlihat pada Gambar 4.5.

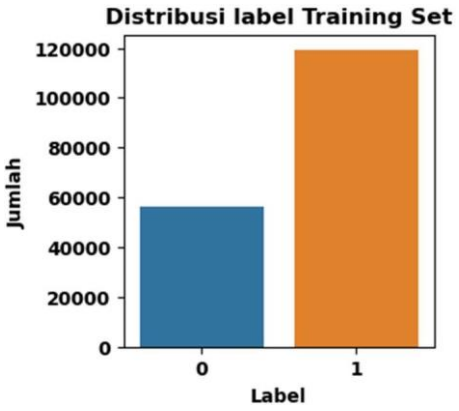
```
data_train = pd.read_csv('/Data/Visual Studio Code/Data Science/Dataset/UNSW-
NB15/UNSW_NB15_training-set.csv')
data_test = pd.read_csv('/Data/Visual Studio Code/Data Science/Dataset/UNSW-
NB15/UNSW_NB15_testing-set.csv')
```

Gambar 4.5 Kode untuk *read* file *Training Set* dan *Testing Set*

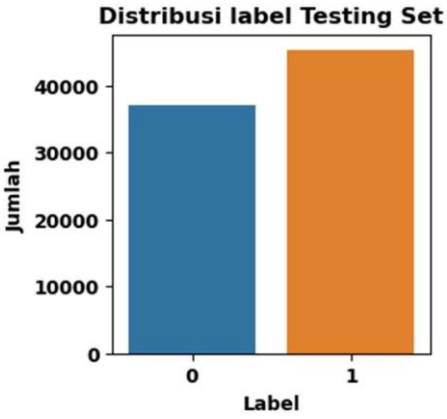
Dilakukan *read file* csv dari masing-masing subset menggunakan *library pandas* dengan *method read_csv*. Kemudian hasilnya disimpan pada variabel bernama *data_train* dan *data_test*.

4.3 Pre-Processing

Langkah ketiga eksperimen pada penelitian ini adalah menyiapkan data agar dapat diproses oleh model dengan melakukan *cleaning*, *encoding*, *normalization*, dan *feature selection*. Sebelum mulai tahapan *pre-processing*, dilihat jumlah dari label normal dan serangan dari *data_train* dan *data_test* yang dapat dilihat pada Gambar 4.6 dan Gambar 4.7.



Gambar 4.6 Distribusi label pada *Training Set*



Gambar 4.7 Distribusi label pada *Testing Set*

```
print(data_train.value_counts('label'))
print(data_test.value_counts('label'))
```

Gambar 4.8 Kode untuk melihat distribusi label

Berdasarkan kode pada Gambar 4.8 dapat diketahui jumlah pasti dari sampel normal dan serangan pada *data_train* dan *data_test*. Sampel dengan label 0 adalah sampel normal dan label 1 adalah sampel serangan. Pada *data_train* terdapat 56000 sampel normal dan 119341 sampel serangan, sedangkan pada *data_test* terdapat 37000 sampel normal dan 45332 sampel serangan. Selanjutnya tidak ditemukan *missing value*, duplikat, dan *null* setelah dilakukan pengecekan dengan menjalankan kode seperti pada Gambar 4.9.

```
print(data_train.duplicated().value_counts())
print(data_test.duplicated().value_counts())
print(data_train.isna().value_counts())
print(data_test.isna().value_counts())
print(data_train.isnull().value_counts())
```

```
print(data_test.isnull().value_counts())
```

Gambar 4.9 Kode untuk cek *missing value*, duplikat, dan *null*

4.3.1 Cleaning

Setelah mengetahui jumlah pasti dari sampel normal dan sampel serangan, selanjutnya dilakukan *cleaning* pada *data_train* dan *data_tes*t. Pada eksperimen, hanya akan digunakan sampel serangan pada variabel *data_train* dan *data_test*. Dengan demikian, sampel normal tidak akan digunakan. Maka dari itu, dijalankan kode seperti pada Gambar 4.10.

```
data_train_attack = data_train[data_train['label'] == 1]
data_test_attack = data_test[data_test['label'] == 1]
```

Gambar 4.10 Kode untuk menyimpan sampel serangan pada variabel

Kode pada Gambar 4.10 dapat diartikan bahwa cari semua sampel dengan label 1 dan simpan ke variabel yang sudah ditentukan. Pada kode, variabel tersebut adalah *data_train_attack* dan *data_test_attack*. Untuk model dengan algoritma 1D-CNN, diperlukan proses *cleaning* tambahan pada fitur *state*, hal ini dilakukan karena terdapat *unique value* yang berbeda pada *data_train_attack* dan *data_test_attack* yang nantinya akan mengakibatkan *error* pada *input shape* ketika proses *training*. Maka dari itu, dijalankan kode seperti pada Gambar 4.11 untuk mengecualikan sampel dengan *unique value* RST pada variabel *data_train_attack*, dan sampel dengan *unique value* ACC dan CLO pada variabel *data_test_attack*.

```
data_train_attack = data_train_attack[(data_train_attack['state'] != 'RST')]
data_test_attack = data_test_attack[(data_test_attack['state'] != 'ACC') &
(data_train_attack['state'] != 'CLO')]
```

Gambar 4.11 Kode untuk mengecualikan sampel dengan *unique value* RST, ACC, dan CLO pada fitur *state*

4.3.2 Encoding

Proses selanjutnya dilakukan *encoding* pada fitur *attack_cat*, *state*, *service*, dan *proto* menggunakan *LabelEncoder* dari *library Scikit-Learn* untuk model dengan algoritma *Decision Tree*, *Random Forest*, *Gradient Boosting*, dan *XGBoost*. Kode dari proses ini dapat dilihat pada Gambar 4.12. Perbandingan sebelum dan sesudah *encoding* pada fitur *attack_cat* dapat dilihat pada Tabel 4.1.

```
le = LabelEncoder()

data_train_attack['attack_cat'] = le.fit_transform(data_train_attack['attack_cat'])
data_train_attack['state'] = le.fit_transform(data_train_attack['state'])
data_train_attack['service'] = le.fit_transform(data_train_attack['service'])
data_train_attack['proto'] = le.fit_transform(data_train_attack['proto'])

data_test_attack['attack_cat'] = le.fit_transform(data_test_attack['attack_cat'])
data_test_attack['state'] = le.fit_transform(data_test_attack['state'])
data_test_attack['service'] = le.fit_transform(data_test_attack['service'])
data_test_attack['proto'] = le.fit_transform(data_test_attack['proto'])
```

Gambar 4.12 Kode untuk *encoding* fitur *attack_cat*, *state*, *service*, dan *proto*

Tabel 4.1 Sebelum dan sesudah *encoding* fitur *attack_cat*

Encoding	
Sebelum	**Sesudah**
Analysis	0
Backdoor	1
DoS	2
Exploits	3
Fuzzers	4
Generic	5
Reconnaissance	6
Shellcode	7
Worms	8

Sementara itu, untuk model dengan algoritma 1D-CNN *encoding* dilakukan pada fitur *state*, *service*, dan *proto* dengan *One-hot Encoding*. Untuk kode dari *One-hot Encoding* dapat dilihat pada Gambar 4.13.

```
cols = ['proto','state','service']

data_train_attack = pd.get_dummies(data_train_attack, columns=cols)
data_test_attack = pd.get_dummies(data_test_attack, columns=cols)
```

Gambar 4.13 Kode untuk *One-hot Encoding* fitur *state*, *service*, dan *proto*

4.3.3 Normalization

Selanjutnya pada proses ini dilakukan normalisasi pada variabel *data_train_attack* dan variabel *data_test_attack* menggunakan *MinMaxScaler* dari *library Scikit-Learn*. Kode untuk proses ini dapat dilihat pada Gambar 4.14.

```
scaler = MinMaxScaler()

cols_to_norm = ['dur', 'spkts', 'dpkts', 'sbytes', 'dbytes', 'rate', 'sttl',
'dttl', 'sload', 'dload', 'sloss', 'dloss', 'sinpkt', 'dinpkt', 'sjit', 'djit',
'swin', 'stcpb', 'dtcpb', 'dwin', 'tcprtt', 'synack', 'ackdat', 'smean', 'dmean',
'trans_depth', 'response_body_len', 'ct_srv_src', 'ct_state_ttl', 'ct_dst_ltm',
'ct_src_dport_ltm', 'ct_dst_sport_ltm', 'ct_dst_src_ltm', 'is_ftp_login',
'ct_ftp_cmd', 'ct_flw_http_mthd', 'ct_src_ltm', 'ct_srv_dst', 'is_sm_ips_ports']

data_train_attack[cols_to_norm] =
scaler.fit_transform(data_train_attack[cols_to_norm])
data_test_attack[cols_to_norm] =
scaler.fit_transform(data_test_attack[cols_to_norm])
```

Gambar 4.14 Kode untuk *Normalization*

4.3.4 Feature Selection

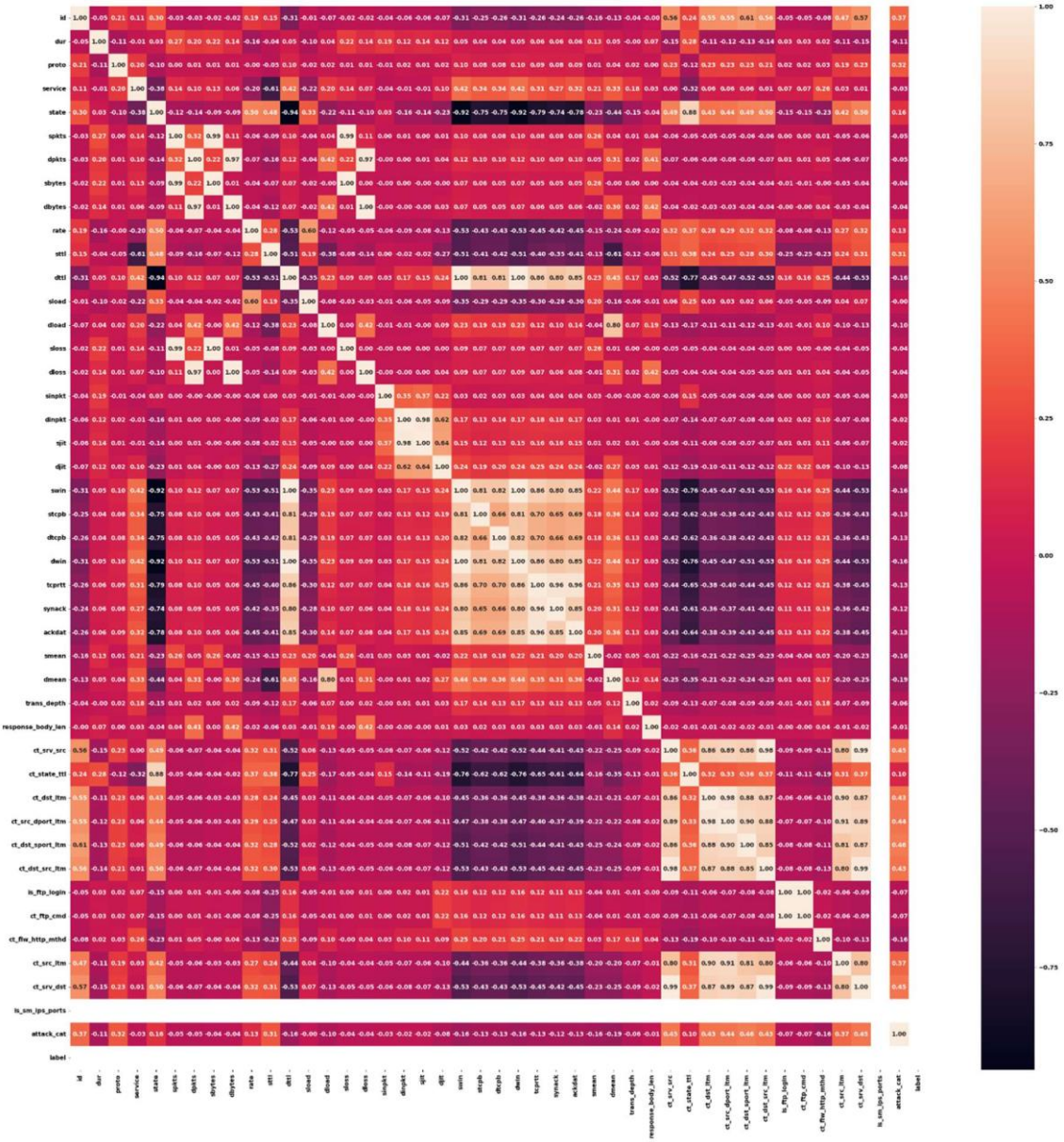
Proses terakhir dari langkah *pre-processing* adalah seleksi fitur-fitur yang akan digunakan untuk model. Dilakukan seleksi fitur dengan nilai korelasi antar fitur menggunakan *corr()* dengan *parameter method* menggunakan *pearson correlation*, serta menampilkannya ke dalam bentuk *heatmap* menggunakan *library seaborn* dan *matplotlib* yang dapat dilihat pada Gambar 4.15.

```
plt.figure(figsize=[32,32])
sns.heatmap(feature.corr(), annot=True, fmt=".2f")
plt.show()
```

Gambar 4.15 Kode untuk *pearson correlation* dalam bentuk *heatmap*

Setelah mempertimbangkan dan menganalisis, dilakukan penghapusan fitur id, label, *dinpkt*, *ct_ftp_cmd* dan *is_sm_ips_ports*. Fitur id dihapus karena hanya berisi informasi index setiap sampel data, sedangkan fitur label dan *is_sm_ips_ports* tidak menampilkan nilai korelasi apapun seperti yang terlihat di *heatmap* pada Gambar 4.16. Kemudian fitur *dinpkt* dihapus karena nilai korelasi dengan *sjit* tinggi (sedikit di bawah fitur *sjit*) sehingga kedua fitur tersebut dianggap mirip. Selanjutnya, fitur *ct_ftp_cmd*, yang memiliki nilai korelasi 1 dengan fitur *is_ftp_login* sehingga kedua fitur tersebut dianggap mirip, juga dihapus. *Heatmap* nilai korelasi

antarfitur dapat dilihat pada Gambar 4.16 dan kode untuk proses seleksi fitur dapat dilihat pada Gambar 4.17.



Gambar 4.16 *Heatmap* nilai korelasi antar fitur

```
feature = data_train_attack.drop(['id', 'dinpkt', 'ct_ftp_cmd', 'is_sm_ips_ports',
'label'], axis=1)
```

Gambar 4.17 Kode untuk proses *feature selection*

4.4 Resampling Data

Langkah keempat eksperimen pada penelitian ini adalah melakukan *resampling* pada data yang digunakan untuk *training*, dalam hal ini data yang akan digunakan adalah data yang tersimpan pada variabel *feature*. Sebelum melakukan *resampling*, dilakukan *initialize* variabel *X_train*, *y_train*, *X_test*, dan *y_test* yang nantinya akan digunakan untuk *training* dan *testing* menggunakan kode yang dapat dilihat pada Gambar 4.18. Variabel *X_train* berisi semua data yang ada di variabel *feature* kecuali kolom *attack_cat*, variabel *y_train* hanya berisi kolom *attack_cat* dari variable *feature*. Sementara itu, variabel *X_test* berisi semua data yang ada di variabel *data_test_attack* kecuali semua kolom yang dihapus pada proses *feature selection* dan kolom *attack_cat*. Variabel *y_test* hanya berisi kolom *attack_cat* dari variabel *data_test_attack.*

```
X_train = feature.drop(columns=['attack_cat'])
y_train = feature["attack_cat"]

X_test = data_test_attack.drop(columns=['attack_cat', 'id', 'dinpkt', 'ct_ftp_cmd',
'is_sm_ips_ports', 'label'])
y_test = data_test_attack["attack_cat"]
```

Gambar 4.18 Kode untuk *initialize X_train*, *y_train*, *X_test*, dan *y_test*

Pada dataset untuk pelatihan model dengan algoritma 1D-CNN, fitur *attack_cat* dipisahkan menggunakan *method pop()* dari *library pandas*. Kemudian data untuk pelatihan model disimpan pada variabel bernama *train_attack_cat* sedangkan data untuk pengujian disimpan pada variabel bernama *test_attack_cat*. Kode *initialize* untuk model dengan algoritma 1D-CNN dapat dilihat pada Gambar 4.19.

```
train_attack_cat = data_train_attack.pop('attack_cat')
test_attack_cat = data_test_attack.pop('attack_cat')

X_train = feature
y_train = train_attack_cat

X_test = data_test_attack.drop(columns=['dinpkt', 'ct_ftp_cmd', 'is_sm_ips_ports'])
y_test = test_attack_cat
```

Gambar 4.19 Kode untuk *initialize X_train*, *y_train*, *X_test*, dan *y_test* pada 1D-CNN

Setelah itu, dilakukan *resampling* data pada variabel *X_train* dan *y_train*. Hal ini dilakukan karena kedua variabel tersebut yang akan digunakan dalam langkah *training*. *Library* yang digunakan untuk *resampling* data pada eksperimen ini adalah *Imbalanced-learn*. Digunakan parameter *default* untuk setiap teknik *resampling* yang digunakan seperti yang terlihat pada

Tabel 4.2. Sementara itu, untuk kode yang digunakan pada langkah *resampling* data ini dapat dilihat pada Gambar 4.20. Serta sebagai pembanding, distribusi setiap kelas serangan sebelum dilakukan *resampling* data dapat dilihat pada Gambar 4.21 sedangkan kode untuk menampilkan distribusi tersebut dapat dilihat pada Gambar 4.22. Adapun jenis serangan dapat dilihat pada Tabel 4.1.

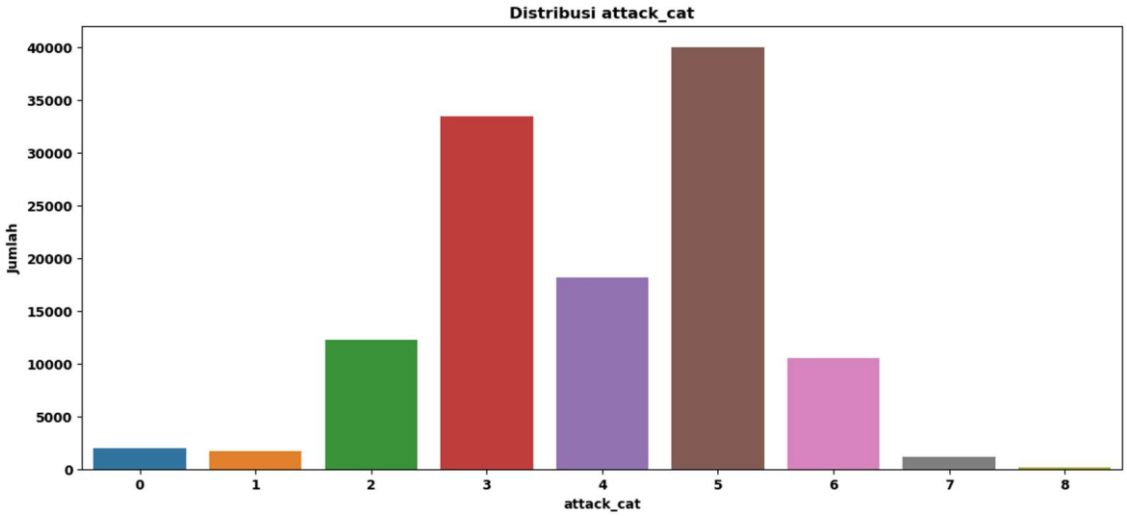
Tabel 4.2 Parameter setiap teknik *resampling* yang digunakan

Teknik Resampling	
Nama	**Parameter**
ADASYN	`ADASYN(*, sampling_strategy='auto', random_state=None, n_neighbors=5, n_jobs=None)`
Borderline SMOTE	`BorderlineSMOTE(*, sampling_strategy='auto', random_state=None, k_neighbors=5, n_jobs=None, m_neighbors=10, kind='borderline-1')`
Random Oversampling	`RandomOverSampler(*, sampling_strategy='auto', random_state=None, shrinkage=None)`
Random Undersampling	`RandomUnderSampler(*, sampling_strategy='auto', random_state=None, replacement=False)`
SMOTE	`SMOTE(*, sampling_strategy='auto', random_state=None, k_neighbors=5, n_jobs=None)`
SMOTE-Tomek	`SMOTETomek(*, sampling_strategy='auto', random_state=None, smote=None, tomek=None, n_jobs=None)`
SVM-SMOTE	`SVMSMOTE(*, sampling_strategy='auto', random_state=None, k_neighbors=5, n_jobs=None, m_neighbors=10, svm_estimator=None, out_step=0.5)`
Tomek Links	`TomekLinks(*, sampling_strategy='auto', n_jobs=None)`

```
from imblearn.over_sampling import ADASYN, BorderlineSMOTE, RandomOverSampler,
SMOTE, SVMSMOTE
from imblearn.under_sampling import RandomUnderSampler, TomekLinks
from imblearn.combine import SMOTETomek


adasyn = ADASYN()
X_adasyn, y_adasyn = adasyn.fit_resample(X_train, y_train)

borderlinesmote = BorderlineSMOTE()
X_borderlinesmote, y_borderlinesmote = borderlinesmote.fit_resample(X_train,
y_train)

ros = RandomOverSampler()
X_ros, y_ros = ros.fit_resample(X_train, y_train)

rus = RandomUnderSampler()
```

```
X_rus, y_rus = rus.fit_resample(X_train, y_train)

smote = SMOTE()
X_smote, y_smote = smote.fit_resample(X_train, y_train)

stl = SMOTETomek()
X_stl, y_stl = stl.fit_resample(X_train, y_train)

svmsmote = SVMSMOTE()
X_svmsmote, y_svmsmote = svmsmote.fit_resample(X_train, y_train)

tomek = TomekLinks()
X_tomek, y_tomek = tomek.fit_resample(X_train, y_train)
```

Gambar 4.20 Kode untuk teknik *resampling* ADASYN, BSMOTE, ROS, RUS, SMOTE, SMOTE-*Tomek*, SVM-SMOTE, dan *Tomek Links*



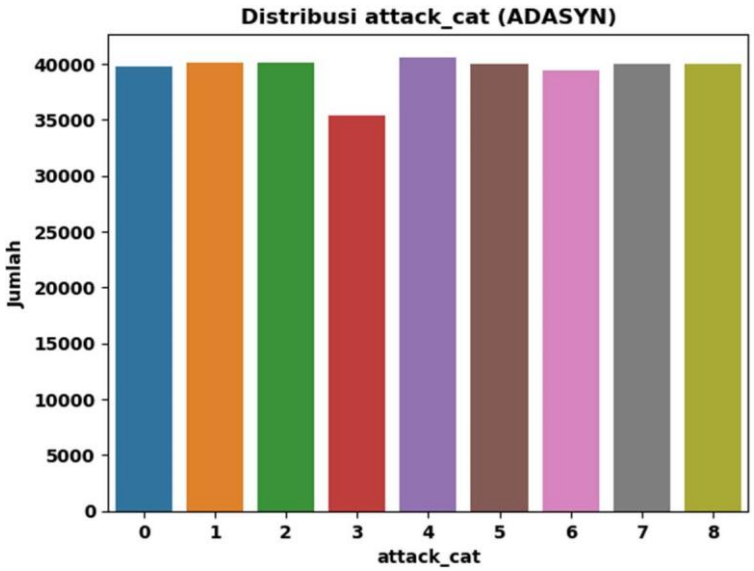
Gambar 4.21 Distribusi setiap kelas serangan sebelum *resampling* data

```
plt.figure(figsize=(14, 6))
sns.countplot(data=data_train_attack, x="attack_cat")
plt.ylabel("Jumlah")
plt.title("Distribusi attack_cat")
```

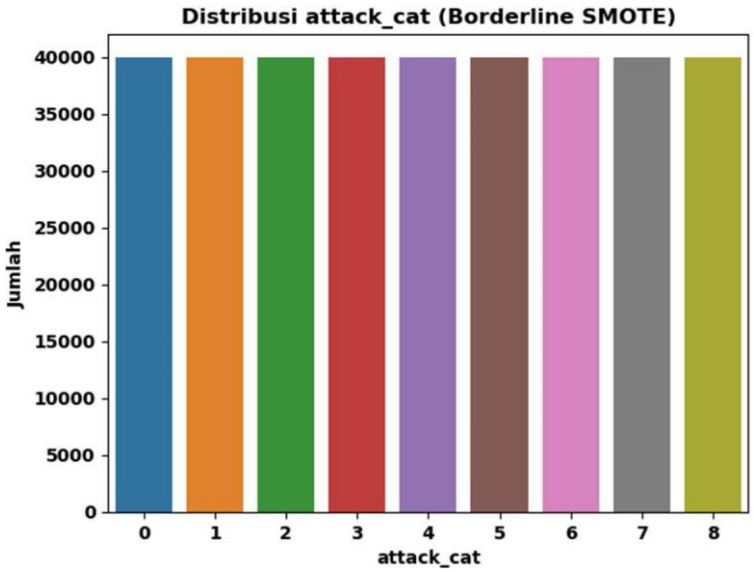
Gambar 4.22 Kode untuk menampilkan distribusi serangan

Setelah menerapkan berbagai macam teknik *resampling*, diketahui ada perbedaan yang terjadi pada distribusi data. Ketika teknik ADASYN diterapkan, distribusi setiap kelas akan terlihat seperti pada Gambar 4.23, yaitu semua kelas seimbang dengan jumlah setiap kelasnya hampir sama, yaitu sekitar 39 ribu sampai dengan 40 ribu kecuali untuk kelas *Exploits* yang berada di sekitar 35 ribu. Adapun hasil penerapan teknik *resampling Borderline* SMOTE, *Random Oversampling*, SMOTE, dan SVM-SMOTE membuat distribusi setiap kelas serangan memiliki jumlah sampel yang sama persis seperti yang terlihat pada Gambar 4.24, Gambar

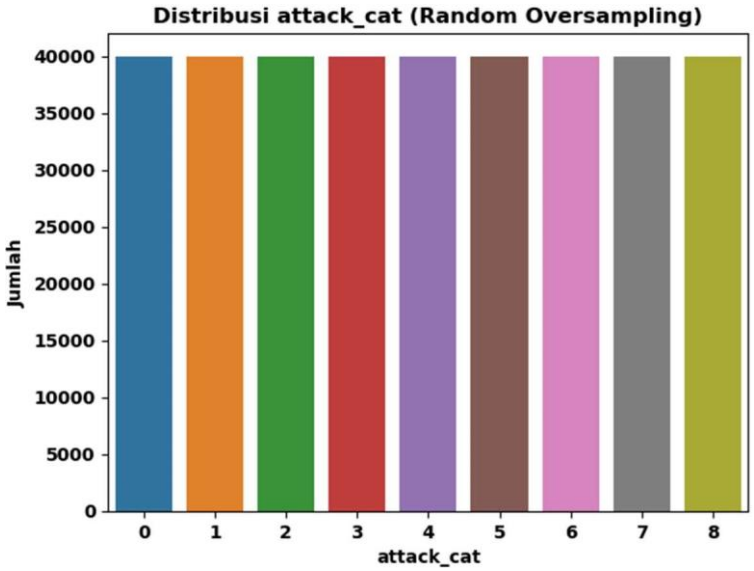
4.25, Gambar 4.26, dan Gambar 4.27, yaitu 40 ribu sampel. Kemudian, ketika diterapkan teknik *Random Undersampling*, distribusi data berkurang drastis menjadi 130 sampel di setiap kelas serangan seperti yang terlihat pada Gambar 4.28.



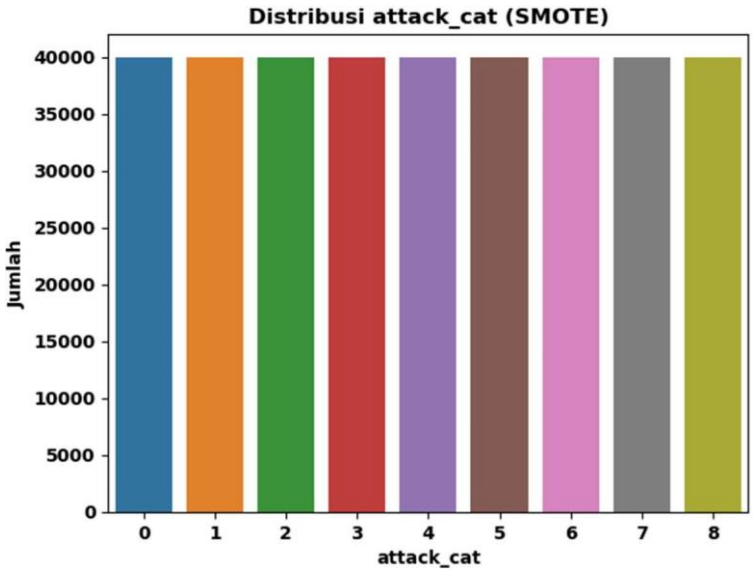
Gambar 4.23 Distribusi setelah menerapkan teknik ADASYN



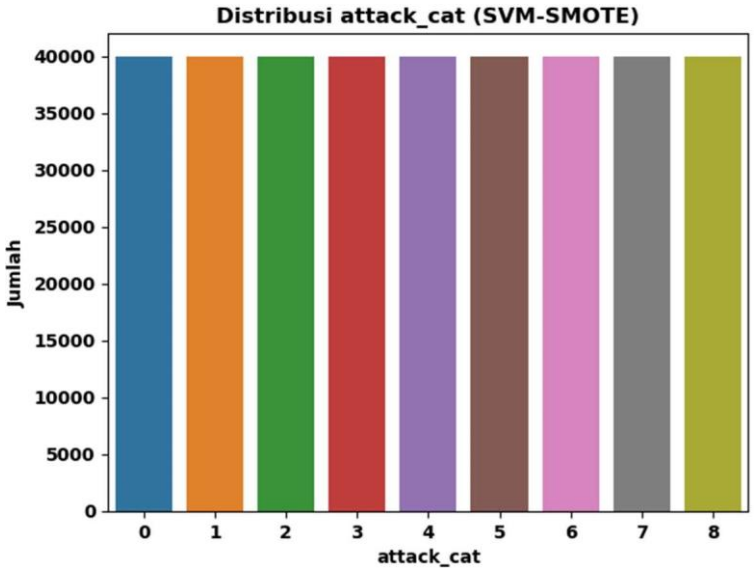
Gambar 4.24 Distribusi setelah menerapkan teknik *Borderline* SMOTE



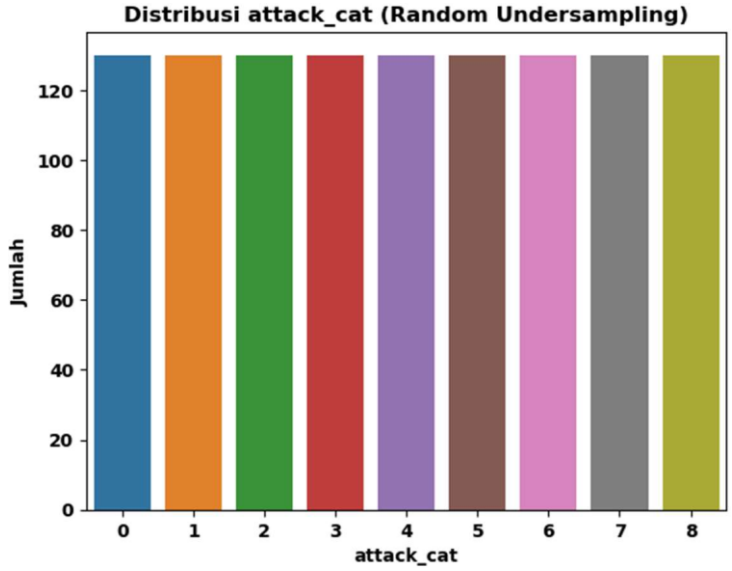
Gambar 4.25 Distribusi setelah menerapkan teknik *Random Oversampling*



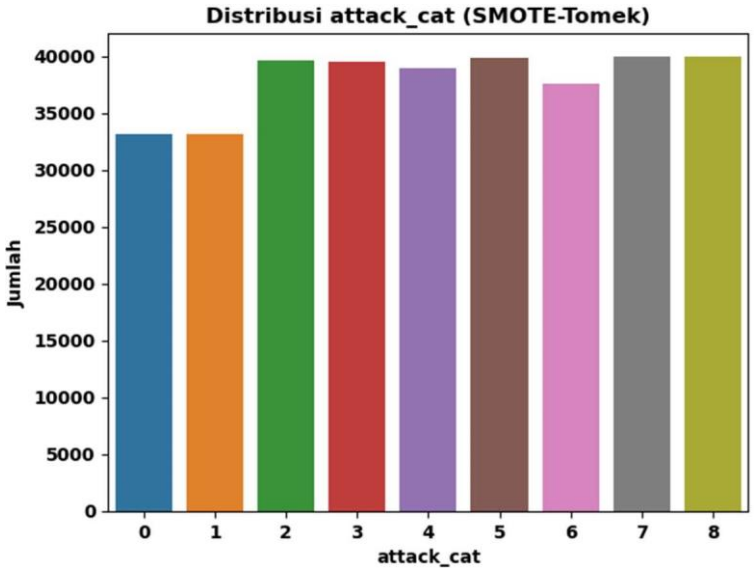
Gambar 4.26 Distribusi setelah menerapkan teknik SMOTE



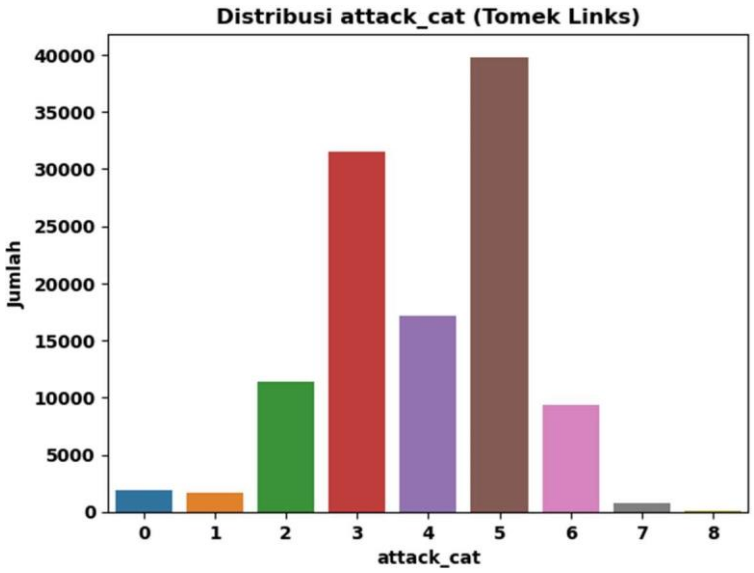
Gambar 4.27 Distribusi setelah menerapkan teknik SVM-SMOTE



Gambar 4.28 Distribusi setelah menerapkan teknik *Random Undersampling*



Gambar 4.29 Distribusi setelah menerapkan teknik SMOTE-*Tomek*



Gambar 4.30 Distribusi setelah menerapkan teknik *Tomek Links*

Selanjutnya, penerapan teknik gabungan antara *oversampling* dan *undersampling* yaitu SMOTE-Tomek menghasilkan distribusi data seperti yang terlihat pada Gambar 4.29. Dari gambar tersebut, dapat diketahui distribusi kelas 2 sampai 8 menjadi lebih seimbang dibandingkan sebelumnya dengan sampel sekitar 38 ribu sampai dengan 40 ribu sampel. Sementara itu, kelas 0 dan 1 memiliki sampel sekitar 33 ribu sampel. Penerapan teknik *resampling Tomek Links* dapat dilihat pada Gambar 4.30. Dari gambar tersebut, dapat diketahui teknik *Tomek Links* terlihat tidak terlalu memengaruhi distribusi dari data sama sekali jika

digunakan secara terpisah dari SMOTE. Namun, nyatanya sampel pada kelas 0 berkurang 75, kelas 1 berkurang 100, kelas 2 berkurang 854, kelas 3 berkurang 1863, kelas 4 berkurang 1052, kelas 5 berkurang 190, kelas 6 berkurang 1168, kelas 7 berkurang 330, dan kelas 8 tidak berkurang sama sekali.

4.5 Pelatihan Model

Langkah kelima eksperimen pada penelitian ini adalah melakukan *training* menggunakan data yang sudah diterapkan *resampling*. Sebelum memulai, dilakukan pembuatan fungsi untuk menampilkan matriks akurasi, presisi, *recall*, dan F1-*score*, serta untuk menyimpan skor-skor tersebut. Kode dari fungsi ini dapat dilihat pada Gambar 4.31.

```
from sklearn import metrics
ML_Model = []
accuracy = []
precision = []
recall = []
f1_score = []

def storeResults(model, a, b, c, d):
    ML_Model.append(model)
    accuracy.append(round(a, 5))
    precision.append(round(b, 5))
    recall.append(round(c, 5))
    f1_score.append(round(d, 5))

def model_report(modelName, y_train, y_test, p_train, p_test):
    print("Model:{}\n".format(modelName))

    acc_train = metrics.accuracy_score(y_train, p_train)
    acc_test = metrics.accuracy_score(y_test, p_test)
    print("Accuracy on training Data: {:.5f}".format(acc_train))
    print("Accuracy on test Data: {:.5f}\n".format(acc_test))

    precision_score_train = metrics.precision_score(y_train, p_train,
average='macro')
    precision_score_test = metrics.precision_score(y_test, p_test, average='macro')
    print("Precision on training Data: {:.5f}".format(precision_score_train))
    print("Precision on test Data: {:.5f}\n".format(precision_score_test))

    recall_score_train = metrics.recall_score(y_train, p_train, average='macro')
    recall_score_test = metrics.recall_score(y_test, p_test, average='macro')
    print("Recall score on training Data: {:.5f}".format(recall_score_train))
    print("Recall score on test Data: {:.5f}\n".format(recall_score_test))

    f1_score_train = 2 * (precision_score_train * recall_score_train) /
(precision_score_train + recall_score_train)
    f1_score_test = 2 * (precision_score_test * recall_score_test) /
(precision_score_test + recall_score_test)
    print("F1 score on training Data: {:.5f}".format(f1_score_train))
    print("F1 score on test Data: {:.5f}\n".format(f1_score_test))

    print("Classification Report")
    print(metrics.classification_report(y_test, p_test))
```

```
    storeResults(modelName, acc_test, precision_score_test, recall_score_test,
f1_score_test)
```

Gambar 4.31 Kode untuk menampilkan dan menyimpan skor akurasi, presisi, *recall*, dan F1-*score*

Setelah itu, dijalankan *training* menggunakan algoritma model *Decision Tree*, *Random Forest*, *Gradient Boosting*, dan *XGBoost* dengan *parameter default* seperti yang dapat dilihat pada Tabel 4.3. Namun, untuk *parameter max_depth* pada *Decision Tree* dan *n_estimators* pada *Random Forest* berbeda untuk setiap teknik *resampling* yang diterapkan. Hal ini dapat dilihat pada Tabel 4.4. Kemudian, ditentukan *max_depth* pada *Decision Tree* secara manual dengan cara membandingkan akurasi pada data *training* dan data *testing* dengan *range* 1 sampai 30 seperti pada Gambar 4.32. Sementara itu, untuk menentukan *n_estimators* secara manual cukup mengubah classifier pada Gambar 4.32 menggunakan algoritma *Random Forest* dan variabel *range* menjadi 1 sampai 20 seperti yang dapat dilihat pada Gambar 4.33. Perlu diketahui bahwa parameter *max_depth* dan *n_estimators* ini dapat berubah setiap kali kode dijalankan. Contoh hasil dari menjalankan kode pada Gambar 4.32 dan Gambar 4.33 dapat dilihat pada Gambar 4.35 dan Gambar 4.36. Parameter *max_depth* dan *n_estimators* yang digunakan pada penelitian ini dapat dilihat pada Tabel 4.4. Kode untuk *training* menggunakan algoritma *Decision Tree*, *Random Forest*, *Gradient Boosting*, dan *XGBoost* secara garis besar dapat dilihat pada Gambar 4.34.

```
training_accuracy = []
test_accuracy = []

depth = range(1, 30)
for n in depth:
    destree_test = DecisionTreeClassifier(max_depth=n)

    destree_test.fit(X_train, y_train)
    training_accuracy.append(destree_test.score(X_train, y_train))
    test_accuracy.append(destree_test.score(X_test, y_test))

plt.plot(depth, training_accuracy, label="training accuracy")
plt.plot(depth, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("max_depth")
plt.legend()
```

Gambar 4.32 Kode untuk menentukan *max_depth* pada *Decision Tree* secara manual di setiap penerapan teknik *resampling*

```
training_accuracy = []
test_accuracy = []

n_est = range(1, 20)
for n in n_est:
    ranfor_test = RandomForestClassifier(n_estimators=n)

    ranfor_test.fit(X_train, y_train)
    training_accuracy.append(ranfor_test.score(X_train, y_train))
    test_accuracy.append(ranfor_test.score(X_test, y_test))

plt.plot(n_est, training_accuracy, label="training accuracy")
plt.plot(n_est, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_estimators")
plt.legend()
```

Gambar 4.33 Kode untuk menentukan *n_estimators* pada *Random Forest* secara manual di setiap penerapan teknik *resampling*

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier

destree = DecisionTreeClassifier(max_depth=1)
destree.fit(X_train, y_train)

ranfor = RandomForestClassifier(n_estimators=18)
ranfor.fit(X_train, y_train)

gbc = GradientBoostingClassifier()
gbc.fit(X_train, y_train)

xgb = XGBClassifier()
xgb.fit(X_train, y_train)
```

Gambar 4.34 Kode untuk *training* menggunakan algoritma *Decision Tree*, *Random Forest*, *Gradient Boosting*, dan *XGBoost*

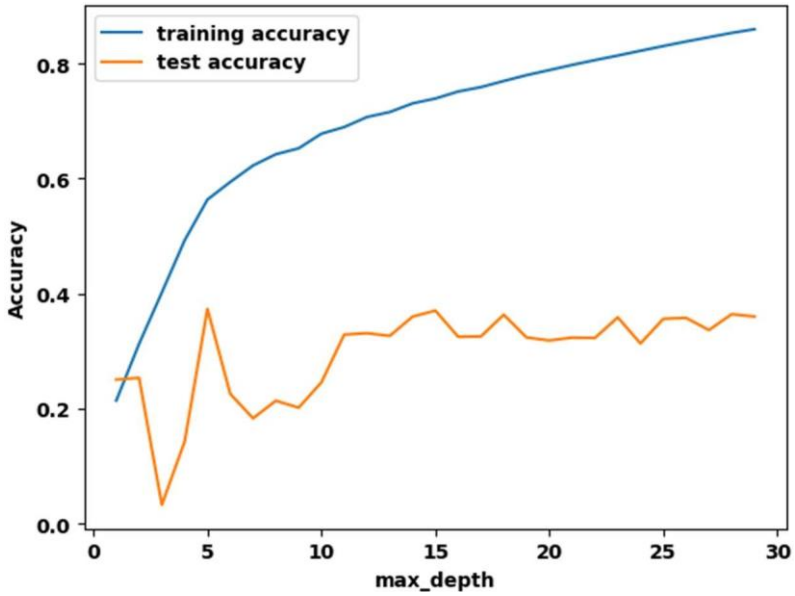
Tabel 4.3 Parameter *default* algoritma *Decision Tree*, *Random Forest*, *Gradient Boosting*, dan *XGBoost*

Algoritma	**Parameter**
Decision Tree	`DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0)`
Random Forest	`RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)`

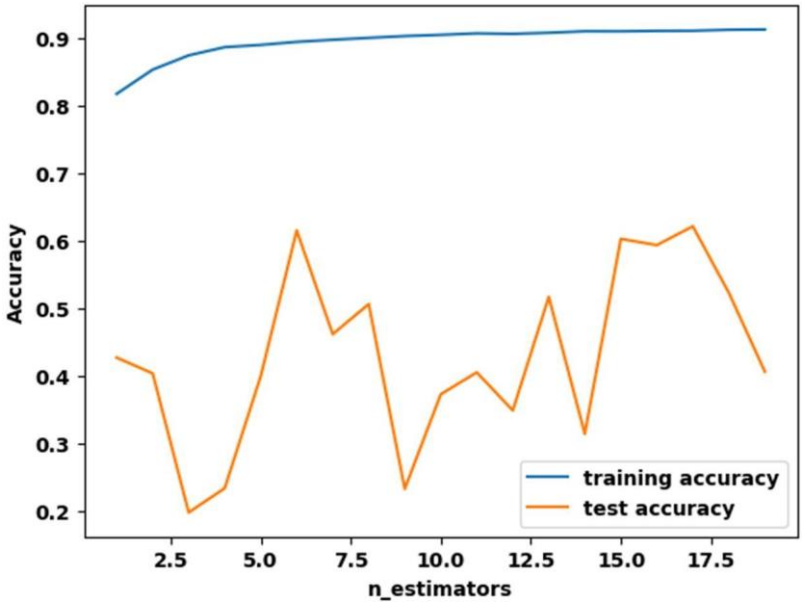
Gradient Boosting	`GradientBoostingClassifier(*, loss='log_loss', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, validation_fraction=0.1, n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0)`
XGBoost	`XGBClassifier(base_score=None, booster=None, callbacks=None, colsample_bylevel=None, colsample_bynode=None, colsample_bytree=None, device=None, early_stopping_rounds=None, enable_categorical=False, eval_metric=None, feature_types=None, gamma=None, grow_policy=None, importance_type=None, interaction_constraints=None, learning_rate=None, max_bin=None, max_cat_threshold=None, max_cat_to_onehot=None, max_delta_step=None, max_depth=None, max_leaves=None, min_child_weight=None, missing=nan, monotone_constraints=None, multi_strategy=None, n_estimators=None, n_jobs=None, num_parallel_tree=None, objective='multi:softprob', ...)`

Tabel 4.4 Parameter *max_depth* dan *n_estimators* pada setiap penerapan teknik *resampling*

Parameter Decision Tree dan Random Forest	
Teknik Resampling	**Parameter**
ADASYN	`DecisionTreeClassifier(max_depth=5)`
	`RandomForestClassifier(n_estimators=17)`
Borderline SMOTE	`DecisionTreeClassifier(max_depth=28)`
	`RandomForestClassifier(n_estimators=15)`
Random Oversampling	`DecisionTreeClassifier(max_depth=12)`
	`RandomForestClassifier(n_estimators=9)`
Random Undersampling	`DecisionTreeClassifier(max_depth=23)`
	`RandomForestClassifier(n_estimators=15)`
SMOTE	`DecisionTreeClassifier(max_depth=23)`
	`RandomForestClassifier(n_estimators=11)`
SMOTE-Tomek	`DecisionTreeClassifier(max_depth=17)`
	`RandomForestClassifier(n_estimators=8)`
SVM-SMOTE	`DecisionTreeClassifier(max_depth=20)`
	`RandomForestClassifier(n_estimators=14)`
Tomek Links	`DecisionTreeClassifier(max_depth=12)`
	`RandomForestClassifier(n_estimators=20)`



Gambar 4.35 Contoh untuk menentukan *max_depth*



Gambar 4.36 Contoh untuk menentukan *n_estimators*

Selanjutnya, untuk *training* model 1D-CNN menggunakan arsitektur yang digunakan oleh Azizjon, dkk. (2020) dengan *batch_size* 32 dan dilakukan 50 *training epoch*. Untuk kode dari arsitektur 1D-CNN yang digunakan pada proses ini dapat dilihat pada Gambar 4.37. Adapun kode untuk *training* model menggunakan algoritma 1D-CNN dapat dilihat pada Gambar 4.38.

```
# Define the input shape
input_shape = (182, 1)
```

```
# Create the sequential model
model = Sequential()

# Convolutional layers
model.add(Conv1D(filters=32, kernel_size=5, activation='sigmoid',
input_shape=input_shape))

# MaxPooling1D layer
model.add(MaxPooling1D(pool_size=2, strides=1))

# Flatten layer
model.add(Flatten())

# Batch Normalization layer
model.add(BatchNormalization(axis=-1))

# Dropout layer with dropout 0.5
model.add(Dropout(0.5))

# Two Fully-connected layers
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))

# Output
model.add(Dense(units=9, activation='softmax'))

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy', Recall(), Precision(), tfa.metrics.F1Score(num_classes=9,
average='macro')])
```

Gambar 4.37 Kode untuk arsitektur 1D-CNN yang digunakan

```
epochs = 50
batch_size = 32

history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,
validation_data=(X_test, y_test))
```

Gambar 4.38 Kode untuk *training* menggunakan 1D-CNN

4.6 Pengujian

Langkah keenam dari eksperimen yang dilakukan adalah melakukan *testing* menggunakan data yang sudah disiapkan, yaitu data yang tersimpan pada variabel *X_test* dan *y_test*. Pada proses ini, pengujian semua model yang sudah dilatih yaitu, model *Decision Tree*, *Random Forest*, *Gradient Boosting*, *XGBoost* dilakukan dengan kode seperti pada Gambar 4.39. Sementara itu, kode yang digunakan untuk *testing* pada model 1D-CNN dapat dilihat pada Gambar 4.40. Kemudian, dari proses *testing* tersebut dihasilkan nilai prediksi yang disimpan pada sebuah variabel.

```
p_test_destree = destree.predict(X_test)
p_test_ranfor = ranfor.predict(X_test)
p_test_gbc = gbc.predict(X_test)
p_test_xgb = xgb.predict(X_test)
```

Gambar 4.39 Kode untuk *testing* model *Decision Tree*, *Random Forest*, *Gradient Boosting*, dan *XGBoost*

```
predictions = model.predict(X_test)
```

Gambar 4.40 Kode untuk *testing* model 1D-CNN

4.7 Evaluasi

Langkah ketujuh adalah melakukan evaluasi dengan menganalisis skor akurasi, presisi, *recall*, dan F1-*score*. Untuk melakukan proses tersebut, dijalankan kode seperti pada Gambar 4.41 dan menghasilkan *output* berupa skor akurasi, presisi, *recall*, dan F1-*score*. Kode yang dijalankan ini merupakan fungsi yang telah dibuat seperti pada Gambar 4.31, dan hanya digunakan pada model *Decision Tree*, *Random Forest*, *Gradient Boosting*, dan *XGBoost*. Sementara itu, untuk 1D-CNN dijalankan dengan kode seperti pada Gambar 4.42.

```
model_report(str(destree), y_train, y_test, p_train_destree, p_test_destree)
model_report(str(ranfor), y_train, y_test, p_train_ranfor, p_test_ranfor)
model_report(str(gbc), y_train, y_test, p_train_gbc, p_test_gbc)
model_report(str(xgb), y_train, y_test, p_train_xgb, p_test_xgb)
```

Gambar 4.41 Kode untuk menampilkan skor akurasi, presisi, *recall*, dan F1-*score*

```
loss, accuracy, recall, precision, f1_score = model.evaluate(X_test, y_test)
f1_score = 2 * (precision * recall) / (precision + recall)

print('Test Loss:', loss)
print('Test Accuracy:', accuracy)
print('Test Recall:', recall)
print('Test Precision:', precision)
print('Test F1-Score:', f1_score)
```

Gambar 4.42 Kode untuk menampilkan *loss*, akurasi, presisi, *recall*, dan F1-*score* pada model 1D-CNN

Setelah itu, dikumpulkan semua skor dari setiap penerapan teknik *resampling* menggunakan kelima algoritma model tersebut. Agar lebih mudah, digunakanlah alat seperti *excel* atau *spreadsheet*. Selanjutnya, diekstrak hasil tersebut dan didapatkan berbagai macam informasi yang kemudian disajikan dalam bentuk tabel.

Tabel 4.5 Skor setiap teknik *resampling* menggunakan model *Decision Tree*

	Decision Tree			
	Accuracy	**Recall**	**Precision**	**F1-Score**
Tanpa Resampling	**0.70701**	0.39129	**0.46084**	**0.42323**
ADASYN	0.37336	**0.43778**	0.29441	0.35206
Borderline SMOTE	0.37320	0.27647	0.26771	0.27202
Random Oversampling	0.41578	0.34264	0.33168	0.33707
Random Undersampling	0.42101	0.40861	0.36460	0.38535
SMOTE	0.26961	0.32880	0.24638	0.28169
SMOTE-Tomek	0.36169	0.35412	0.30285	0.32648
SVM-SMOTE	0.26670	0.28512	0.26895	0.27680
Tomek Links	0.69229	0.34805	0.35253	0.35028

Tabel 4.6 Skor setiap teknik *resampling* menggunakan model *Random Forest*

	Random Forest			
	Accuracy	**Recall**	**Precision**	**F1-Score**
Tanpa Resampling	0.35981	0.27760	0.39143	0.32483
ADASYN	0.47651	0.36410	0.29812	0.32782
Borderline SMOTE	0.53929	0.36304	0.32149	0.34100
Random Oversampling	0.58184	0.34236	0.39914	0.36858
Random Undersampling	0.64142	**0.51381**	0.40568	**0.45339**
SMOTE	0.48612	0.38076	0.31905	0.34718
SMOTE-Tomek	0.52590	0.39903	0.36165	0.37942
SVM-SMOTE	0.34311	0.33447	0.33790	0.33617
Tomek Links	**0.70992**	0.39145	**0.48933**	0.43495

Tabel 4.7 Skor setiap teknik *resampling* menggunakan model *Gradient Boosting*

	Gradient Boosting			
	Accuracy	**Recall**	**Precision**	**F1-Score**
Tanpa Resampling	0.36213	0.34393	**0.38231**	0.36211
ADASYN	0.25926	0.38226	0.29857	0.33527
Borderline SMOTE	0.25121	0.39751	0.35360	0.37427
Random Oversampling	0.25280	0.40203	0.30741	0.34841
Random Undersampling	0.49508	0.45249	0.33418	0.38444
SMOTE	0.28232	0.38635	0.29651	0.33552
SMOTE-Tomek	0.49821	0.42688	0.31244	0.36080
SVM-SMOTE	**0.57705**	**0.48912**	0.36605	**0.41873**
Tomek Links	0.34997	0.33354	0.34002	0.33675

Tabel 4.8 Skor setiap teknik *resampling* menggunakan model *XGBoost*

	XGBoost			
	Accuracy	**Recall**	**Precision**	**F1-Score**
Tanpa Resampling	**0.70409**	0.40448	**0.41353**	0.40895
ADASYN	0.45893	0.40046	0.28486	0.33291
Borderline SMOTE	0.50893	0.40783	0.30056	0.34607
Random Oversampling	0.50205	0.37852	0.34648	0.36179
Random Undersampling	0.52678	**0.50907**	0.35502	**0.41831**
SMOTE	0.55343	0.43299	0.30360	0.35693
SMOTE-Tomek	0.50024	0.44732	0.35286	0.39452
SVM-SMOTE	0.61021	0.46898	0.35220	0.40229
Tomek Links	0.70254	0.41179	0.48949	0.44729

Tabel 4.9 Skor setiap teknik *resampling* menggunakan model 1D-CNN

	1D-CNN			
	Accuracy	**Recall**	**Precision**	**F1-Score**
Tanpa Resampling	0.75137	**0.67573**	0.86773	0.75979
ADASYN	0.63032	0.59664	0.76800	0.67156
Borderline SMOTE	0.65386	0.61435	0.75738	0.67841
Random Oversampling	0.64436	0.59847	0.77022	0.67357
Random Undersampling	0.61870	0.57817	0.71225	0.63825
SMOTE	0.64246	0.60383	0.75610	0.67144
SMOTE-Tomek	0.65947	0.62090	0.78771	0.69443
SVM-SMOTE	0.66509	0.62327	0.75510	0.68288
Tomek Links	**0.75270**	0.67478	**0.87584**	**0.76227**

Dari Tabel 4.5, dapat diketahui bahwa penerapan teknik *resampling* ADASYN pada dataset untuk model *Decision Tree* mendapatkan skor *recall* terbaik dengan sedikit peningkatan skor pada *recall* sekitar 4.65%. Sementara itu, skor akurasi, presisi, dan F1-*score* terbaik justru diraih oleh model *Decision Tree* dengn data latih tanpa dilakukan *resampling*. Adapun pada Tabel 4.6, dapat diketahui bahwa penerapan teknik *resampling Tomek Links* pada model *Random Forest* mendapatkan skor akurasi dan presisi terbaik dengan peningkatan skor yang signifikan pada akurasi sekitar 35.01% dan pada presisi sekitar 9.79%. Sementara itu, penerapan teknik *resampling Random Undersampling* pada model *Random Forest* mendapatkan skor *recall* dan F1-*score* terbaik dengan peningkatan, yaitu sekitar 23.62% dan 12.85% dari yang sebelumnya tanpa *resampling*. Pada Tabel 4.7, dapat diketahui bahwa penerapan teknik *resampling* SVM-SMOTE pada model *Gradient Boosting* mampu menghasilkan skor akurasi, *recall*, dan F1-*score* terbaik dengan peningkatan akurasi sekitar 21.49%, *recall* sekitar 14.52%, dan F1-*score* sekitar 5.66%. Sementara itu, skor presisi pada model *Gradient Boosting* tidak mengalami peningkatan apapun ketika diterapkan teknik *resampling*. Pada Tabel 4.8, dapat diketahui bahwa penerapan teknik *Random Undersampling* pada model *XGboost* menghasilkan skor *recall* dan F1-*score* terbaik dengan peningkatan pada *recall* sekitar 10.46% dan F1-*score* sekitar 0.94%. Sementara itu, skor akurasi dan presisi pada model *XGBoost* tidak mengalami peningkatan apapun ketika diterapkan teknik *resampling*.

Dari Tabel 4.9, dapat diketahui bahwa penerapan teknik *Tomek Links* pada model 1D-CNN mendapatkan skor akurasi, presisi, dan F1-score terbaik, yaitu dengan sedikit peningkatan pada akurasi sebesar 0.14%, presisi sebesar 0.81%, dan F1-*score* sebesar 0.25% dibanding sebelum dilakukan teknik *resampling*. Sementara itu, skor *recall* pada model 1D-CNN tidak mengalami peningkatan apapun ketika diterapkan teknik *resampling*. Secara keseluruhan dari Tabel 4.5, Tabel 4.6, Tabel 4.7, Tabel 4.8, dan Tabel 4.9, dapat diketahui bahwa teknik *Tomek Links* pada model 1D-CNN mendapatkan skor akurasi, presisi, dan F1-*score* terbaik, sedangkan skor *recall* terbaik dihasilkan oleh model 1D-CNN yang tanpa diterapkan teknik *resampling*.

4.8 Analisis Hasil Eksperimen

Hasil eksperimen penerapan teknik *resampling* yang diterapkan pada model *Decision Tree* mengakibatkan menurunnya skor akurasi, presisi, dan F1-*score*. Ketika dilakukan *oversampling* menggunakan teknik ADASYN, skor *recall* meningkat, tetapi dengan mengorbankan skor akurasi, presisi, dan F1-*score*. Sementara itu, teknik *oversampling* lainnya tidak meningkatkan skor apapun. Ketika diterapkan *undersampling* menggunakan teknik *Random Undersampling*, skor *recall* meningkat, tetapi dengan mengorbankan skor akurasi, presisi, dan F1-*score*. Sementara itu, *hybrid sampling* tidak meningkatkan skor apapun pada model *Decision Tree*.

Adapun hasil eksperimen penerapan teknik ADASYN, *Borderline* SMOTE, SMOTE, dan SVM-SMOTE pada model *Random Forest* dapat meningkatkan akurasi, *recall*, dan F1-*score*, tetapi membuat skor presisi menurun dibanding sebelum dilakukan teknik *resampling*. Teknik *Random Oversampling* mampu meningkatkan skor akurasi, *recall*, presisi, dan F1-*score*, tetapi tidak dapat mencapai skor tertinggi. Begitu juga dengan teknik *hybrid sampling* SMOTE-*Tomek* yang dapat meningkatkan skor akurasi, *recall*, presisi, dan F1-*score*, tetapi tidak dapat mencapai skor tertinggi. Skor tertinggi justru didapatkan dengan menerapkan teknik *undersampling*, yaitu *Random Undersampling* dan *Tomek Links*.

Pada hasil eksperimen penerapan teknik *resampling* pada model *Gradient Boosting*, teknik ADASYN, *Random Oversampling*, dan SMOTE mampu meningkatkan skor *recall*, tetapi dengan mengorbankan skor akurasi, presisi, dan F1-*score* dibanding sebelum dilakukan teknik *resampling*. Teknik *Borderline* SMOTE mampu meningkatkan skor *recall* dan F1-*score*, tetapi membuat skor akurasi dan presisi menurun. Teknik SVM-SMOTE mampu mencapai skor akurasi, *recall*, dan F1-*score* tertinggi, tetapi membuat skor presisi menurun. Teknik SMOTE-*Tomek*, mampu meningkatkan skor akurasi, *recall*, dan F1-*score* tetapi mengorbankan skor

presisi. Teknik *Random Undersampling* mampu meningkatkan skor akurasi, *recall*, dan F1-*score*, tetapi dengan menurunnya skor presisi. Sementara itu, teknik *Tomek Links* membuat skor akurasi, *recall*, presisi, dan F1-*score* menurun dibandingkan dengan sebelum dilakukan teknik *resampling*.

Pada hasil eksperimen penerapan teknik *resampling* yang diterapkan pada model *XGBoost*, teknik *Random Undersampling* mampu mencapai skor *recall* dan F1-*score* tertinggi dengan mengorbankan skor akurasi dan presisi. Sementara itu, teknik *Tomek Links* menurunkan skor akurasi, *recall*, presisi, dan F1-*score*. Begitu juga dengan teknik ADASYN dan *Random Oversampling* yang menurunkan skor akurasi, *recall*, presisi, dan F1-*score* dibanding sebelum dilakukan teknik *resampling*. Teknik *Borderline* SMOTE, SMOTE, dan SVM-SMOTE mampu meningkatkan skor *recall* tetapi dengan mengorbankan skor akurasi, presisi, dan F1-*score*. Teknik *hybrid sampling* SMOTE-*Tomek* mampu meningkatkan skor *recall*, tetapi membuat skor akurasi, presisi, dan F1-*score* menurun dibandingkan sebelum dilakukan *resampling*.

Pada hasil eksperimen penerapan teknik *resampling* yang diterapkan pada model 1D-CNN, teknik *Tomek Links* mencapai skor akurasi, presisi, dan F1-*score* tertinggi dengan sedikit penurunan pada skor *recall*. Sementara itu, teknik *Random Undersampling* menurunkan skor akurasi, *recall*, presisi, dan F1-*score* dibanding sebelum dilakukan teknik *resampling*. Adapun penerapan semua teknik *oversampling* dan teknik *hybrid sampling* pada model 1D-CNN tidak meningkatkan skor apapun.

Terdapat perbandingan hasil eksperimen pada penelitian ini dengan penelitian terkait yang menggunakan dataset yang sama, yaitu UNSW-NB15. Pada penelitian yang dilakukan oleh Rahma, dkk. (2023), penerapan teknik *Random Oversampling* pada model CNN-BiLSTM mendapatkan skor *recall* dan F1-*score* tertinggi, yaitu sebesar 0.677 dan 0.589. Jika dibandingkan dengan hasil eksperimen pada penelitian ini, penerapan teknik *Tomek Links* pada model 1D-CNN menghasilkan skor F1-*score* lebih tinggi, yaitu sebesar 0.762. Sementara itu, skor *recall* tertinggi pada penelitian ini, yaitu sebesar 0.675 dengan model 1D-CNN tanpa *resampling* hampir sama dengan yang dicapai oleh Rahma, dkk. (2023).

Pada penelitian yang dilakukan oleh Bagui & Li. (2021), penerapan teknik *Random Undersampling* pada model ANN mendapatkan skor presisi tertinggi sebesar 0.545. Penerapan teknik *Random Oversampling* mendapatkan skor F1-*score* tertinggi sebesar 0.453. Penerapan teknik *hybrid sampling* RU-RO (*Random Undersampling* dan *Random Oversampling*) mendapatkan skor *recall* tertinggi sebesar 0.775. Jika dibandingkan dengan hasil eksperimen

pada penelitian ini, penerapan teknik *Tomek Links* pada model 1D-CNN mendapatkan skor presisi dan F1-*score* lebih tinggi, yaitu sebesar 0.875 dan 0.762. Sementara itu, untuk skor *recall* pada penelitian ini lebih rendah dibanding dari hasil yang didapatkan Bagui & L. (2021), yaitu sebesar 0.675.

Pada penelitian yang dilakukan oleh Khan dkk. (2023), penerapan teknik Random Oversampling pada model *Extra Tree* mendapatkan skor akurasi dan F1-score tertinggi, yaitu sebesar 0.999 dan 0.999. Penerapan teknik SMOTE pada model *Random Forest* mendapatkan skor presisi tertinggi, yaitu sebesar 0.997. Penerapan *Random Oversampling*, ADASYN, SMOTE-*Tomek*, dan SMOTE pada model OE-IDS mendapatkan skor *recall* yang sama persis, yaitu 1.00. Jika dibandingkan dengan hasil eksperimen pada penelitian ini, skor akurasi, presisi, dan F1-*score* yang didapat pada penelitian ini jauh lebih rendah dibandingkan dengan skor yang didapatkan pada penelitian (Khan dkk,. 2023). Begitu juga untuk skor *recall* yang didapatkan pada penelitian ini, jauh lebih rendah dibandingkan dengan skor yang didapat pada penelitian (Khan dkk., 2023). Hasil ini bisa saja terjadi karena classifier yang digunakan pada penelitian Khan, dkk. (2023) telah dioptimisasi khusus untuk deteksi intrusi jaringan.

4.9 Dokumentasi Eksperimen

Dokumentasi eksperimen dilakukan untuk menghemat waktu jika di kemudian hari model dan hasil prediksi diperlukan. Model *Decision Tree*, *Random Forest*, *Gradient Boosting*, *XGBoost* dan hasil prediksinya disimpan menggunakan *library joblib* dengan menjalankan kode seperti pada Gambar 4.43. Sementara itu, model 1D-CNN dan hasil prediksinya disimpan menggunakan *library keras* dengan menjalankan kode seperti pada Gambar 4.44. Kemudian semua dokumentasi disimpan pada repository github yang dapat diakses dengan tautan: https://github.com/rajasa13/Data-Science

```
import joblib

joblib.dump(destree, '/Data-Science/Notebook/UNSW-
NB15/Imbalanced/Saved/destree.joblib', compress=('zlib', 3))
joblib.dump(ranfor, '/Data-Science/Notebook/UNSW-
NB15/Imbalanced/Saved/ranfor.joblib', compress=('zlib', 3))
joblib.dump(gbc, '/Data-Science/Notebook/UNSW-NB15/Imbalanced/Saved/gbc.joblib',
compress=('zlib', 3))
joblib.dump(xgb, '/Data-Science/Notebook/UNSW-NB15/Imbalanced/Saved/xgb.joblib',
compress=('zlib', 3))

joblib.dump(p_test_destree, '/Data-Science/Notebook/UNSW-
NB15/Imbalanced/Saved/p_test_destree.joblib', compress=('zlib', 3))
joblib.dump(p_test_ranfor, '/Data-Science/Notebook/UNSW-
NB15/Imbalanced/Saved/p_test_ranfor.joblib', compress=('zlib', 3))
```

```
joblib.dump(p_test_gbc, '/Data-Science/Notebook/UNSW-
NB15/Imbalanced/Saved/p_test_gbc.joblib', compress=('zlib', 3))
joblib.dump(p_test_xgb, '/Data-Science/Notebook/UNSW-
NB15/Imbalanced/Saved/p_test_xgb.joblib', compress=('zlib', 3))
```

Gambar 4.43 Kode untuk menyimpan model *Decision Tree*, *Random Forest*, *Gradient Boosting*, *XGBoost* dan hasil prediksinya

```
from keras.models import load_model

model.save('/Data-Science/Notebook/UNSW-
NB15/CNN/Imbalanced/GPU_Saved/CNN_2/cnn1d.keras')

import joblib

joblib.dump(predicted_classes, '/Data-Science/Notebook/UNSW-
NB15/CNN/Imbalanced/GPU_Saved/CNN_2/predicted_classes.joblib', compress=('zlib',
3))
```

Gambar 4.44 Kode untuk menyimpan model 1D-CNN dan hasil prediksinya

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Ketidakseimbangan dataset deteksi intrusi jaringan dapat diatasi dengan teknik *resampling*. Pada penelitian ini, teknik *resampling* yang digunakan adalah ADASYN, *Borderline* SMOTE, *Random Oversampling*, *Random Undersampling*, SMOTE, SMOTE-*Tomek*, SVM-SMOTE, dan *Tomek Links*. Setiap teknik *resampling* memiliki kelebihan dan kekurangan tersendiri. Oleh karena itu, eksperimen perlu dilakukan untuk mengetahui teknik *resampling* yang paling efektif dan mengevaluasi performa model deteksi intrusi jaringan setelah penerapan teknik *resampling*. Hal ini dilakukan untuk memastikan bahwa penerapan teknik *resampling* dapat meningkatkan kinerja model. Teknik *resampling* dipilih karena mudah diterapkan serta tidak memerlukan perubahan pada algoritma *machine learning* yang digunakan. Selain itu, teknik *resampling* memiliki berbagai pendekatan yang dapat disesuaikan dengan karaktersitik dataset. Adapun kontribusi penelitian ini, yaitu menganalisis kelebihan dan kekurangan dari masing-masing teknik *resampling* yang digunakan dan membuktikan bahwa penerapan teknik *resampling* dapat meningkatkan kinerja model deteksi intrusi pada jaringan. Berdasarkan eksperimen yang sudah dilakukan, dapat disimpulkan bahwa penerapan teknik *resampling* dapat dilakukan melalui pendekatan *oversampling*, *undersampling*, gabungan keduanya yaitu *hybrid sampling* untuk meningkatkan representasi kelas minoritas atau mengurangi jumlah sampel pada kelas mayoritas. Penerapan teknik *resampling* mampu meningkatkan keseimbangan dataset deteksi intrusi jaringan. Sehingga meminimalkan bias yang mungkin muncul akibat perbedaan distribusi kelas. Hal ini juga dapat berdampak positif pada kinerja model deteksi jenis serangan jaringan karena memungkinkan algoritma *machine learning* lebih efektif dalam memahami dan mengklasifikasikan sampel dari kelas minoritas. Pada penelitian ini, teknik *resampling* yang paling baik adalah *Tomek Links* yang diterapkan pada model 1D-CNN dengan skor akurasi sebesar 75.27%, *recall* sebesar 67.47%, presisi sebesar 87.58%, dan F1-*socre* sebesar 76.22%. Namun, pemilihan teknik *resampling* harus cermat karena setiap pendekatan memiliki kelebihan dan kekurangan tersendiri. Dengan demikian, pemahaman mendalam tentang karakteristik dataset deteksi intrusi jaringan dan evaluasi teliti terhadap performa model setelah penerapan teknik *resampling* sangat diperlukan untuk mencapai hasil yang optimal dalam mendeteksi intrusi pada jaringan.

5.2 Saran

Penelitian yang telah dilakukan masih memiliki banyak kekurangan. Dengan demikian, agar penelitian selanjutnya dapat lebih dikembangkan, berikut beberapa saran yang dapat dipertimbangkan, yaitu:

a. Melakukan parameter *tuning* pada algoritma model *machine learning* dan *deep learning* agar dapat menghasilkan performa yang lebih baik.
b. Mempertimbangkan penggunaan matriks penilaian lain seperti *Receiver Operating Characteristic* (ROC), *Curve and Area Under the Curve* (AUC), dan uji signifikansi secara statistik untuk evaluasi model.
c. Mempertimbangkan penggunaan teknik *resampling* yang lain seperti SMOTE-ENN (SMOTE *Edited Nearest Neighbors*) dan GAN (Generative Adversarial Network).
d. Mempertimbangkan untuk menggunakan dataset tidak seimbang lain seperti NSL-KDD, CIC-IDS2017, dan KDD-Cup 1999.

DAFTAR PUSTAKA

Abdelkhalek, A., & Mashaly, M. (2023). Addressing the class imbalance problem in network intrusion detection systems using data resampling and deep learning. *The Journal of Supercomputing*, 1–34.

Abdulkareem, S. A., Foh, C. H., Carrez, F., & Moessner, K. (2022). SMOTE-Stack for Network Intrusion Detection in an IoT Environment. *2022 IEEE Symposium on Computers and Communications (ISCC)*, 1–6. https://doi.org/10.1109/ISCC55528.2022.9912910

Ahmad, Z., Shahid Khan, A., Wai Shiang, C., Abdullah, J., & Ahmad, F. (2021). Network intrusion detection system: A systematic study of machine learning and deep learning approaches. *Transactions on Emerging Telecommunications Technologies*, *32*(1), e4150.

Azizjon, M., Jumabek, A., & Kim, W. (2020). 1D CNN based network intrusion detection with normalization on imbalanced data. *2020 international conference on artificial intelligence in information and communication (ICAIIC)*, 218–224. IEEE.

Bagui, S., & Li, K. (2021). Resampling imbalanced data for network intrusion detection datasets. *Journal of Big Data*, *8*(1), 1–41.

Batchu, R. K., & Seetha, H. (2021). A generalized machine learning model for DDoS attacks detection using hybrid feature selection and hyperparameter tuning. *Computer Networks*, *200*, 108498.

Bharathi. (2021). Confusion matrix for multi-class classification.

Boskamp, E. (2023, Juni 15). 30 crucial cybersecurity statistics [2023]: Data, trends and more.

Brownlee, J. (2019, November 26). How to choose a feature selection method for machine learning.

Cavin, A. (2022, Agustus 4). 6 ways to encode features for machine learning algorithms.

Chapaneri, R., & Shah, S. (2022). Enhanced detection of imbalanced malicious network traffic with regularized generative adversarial networks. *Journal of Network and Computer Applications*, *202*, 103368.

Chawla, N. V, Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, *16*, 321–357.

Chen, W., Wang, H., Fei, M., Du, D., & Rakić, A. (2022). An Intrusion Detection Method Using ADASYN and Bayesian Optimized LightGBM. *2022 34th Chinese Control and Decision Conference (CCDC)*, 4622–4627. IEEE.

Darina, L. (2023, Maret 7). How Fast Is Technology Growing Statistics [Updated 2023]. Diambil 28 Mei 2023, dari Leftronic website: https://leftronic.com/blog/how-fast-is-technology-growing-statistics/

deepak_jain. (2019, Maret 12). Data preprocessing in data mining.

Ding, H., Chen, L., Dong, L., Fu, Z., & Cui, X. (2022). Imbalanced data classification: A KNN and generative adversarial networks-based hybrid approach for intrusion detection. *Future Generation Computer Systems*, *131*, 240–254.

Evidently AI Team. (2023). Accuracy, precision, and recall in multi-class classification.

Ghorbani, R., & Ghousi, R. (2020). Comparing different resampling methods in predicting students' performance using machine learning techniques. *IEEE Access*, *8*, 67899–67911.

Han, H., Wang, W.-Y., & Mao, B.-H. (2005). Borderline-SMOTE: a new over-sampling method in imbalanced data sets learning. *International conference on intelligent computing*, 878–887. Springer.

He, H., Bai, Y., Garcia, E. A., & Li, S. (2008). ADASYN: Adaptive synthetic sampling approach for imbalanced learning. *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, 1322–1328. Ieee.

javaTpoint. (2021). Feature selection techniques in machine learning - javaTpoint.

Jiajia, F., Jiangfeng, X., & Junfeng, Z. (2021). Intrusion Detection Model Based on SAE and BALSTM. *2021 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, 1192–1197. IEEE.

Khan, M. A., Iqbal, N., Jamil, H., & Kim, D.-H. (2023). An optimized ensemble prediction model using AutoML based on soft voting classifier for network intrusion detection. *Journal of Network and Computer Applications*, *212*, 103560.

Lee, B.-S., Kim, J.-W., & Choi, M.-J. (2022). Experimental Comparison of Hybrid Sampling Methods for an Efficient NIDS. *2022 23rd Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 1–4. IEEE.

Lemaitre, G. (2016a). ADASYN — imbalanced-learn 0.3.0.dev0 documentation. Diambil dari imbalanced-learn website: https://glemaitre.github.io/imbalanced-learn/auto_examples/over-sampling/plot_adasyn.html

Lemaitre, G. (2016b). Random over-sampling — imbalanced-learn 0.3.0.dev0 documentation. Diambil dari imbalanced-learn website: https://glemaitre.github.io/imbalanced-learn/auto_examples/over-sampling/plot_random_over_sampling.html

Lemaitre, G. (2016c). Random under-sampling — imbalanced-learn 0.3.0.dev0 documentation. Diambil dari imbalanced-learn website: https://glemaitre.github.io/imbalanced-learn/auto_examples/under-sampling/plot_random_under_sampler.html

Lemaitre, G. (2016d). SMOTE + Tomek — imbalanced-learn 0.3.0.dev0 documentation. Diambil dari imbalanced-learn website: https://glemaitre.github.io/imbalanced-learn/auto_examples/combine/plot_smote_tomek.html

Lemaitre, G. (2016e). SMOTE — imbalanced-learn 0.3.0.dev0 documentation. Diambil dari imbalanced-learn website: https://glemaitre.github.io/imbalanced-learn/auto_examples/over-sampling/plot_smote.html

Lemaitre, G. (2016f). Tomek links — imbalanced-learn 0.3.0.dev0 documentation. Diambil dari imbalanced-learn website: https://glemaitre.github.io/imbalanced-learn/auto_examples/under-sampling/plot_tomek_links.html

Li, F., Ma, W., Li, H., & Li, J. (2022). Improving Intrusion Detection System Using Ensemble Methods and Over-Sampling Technique. *2022 4th International Academic Exchange Conference on Science and Technology Innovation (IAECST)*, 1200–1205. IEEE.

Loukas, S. (2020, Mei 28). Everything you need to know about Min-Max normalization: A Python tutorial.

Mahalakshmi, M., Ramkumar, M. P., & GSR, E. S. (2022). SCADA Intrusion Detection System using Cost Sensitive Machine Learning and SMOTE-SVM. *2022 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*, 332–337. IEEE.

Mbow, M., Koide, H., & Sakurai, K. (2021). An intrusion detection system for imbalanced dataset based on deep learning. *2021 Ninth International Symposium on Computing and Networking (CANDAR)*, 38–47. IEEE.

McCain, A. (2023, Januari 11). How fast is technology advancing? [2023]: Growing, evolving, and accelerating at exponential rates.

Moustafa, N., & Slay, J. (2015). UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). *2015 military communications and information systems conference (MilCIS)*, 1–6. IEEE.

Ndichu, S., Ban, T., Takahashi, T., & Inoue, D. (2021). A machine learning approach to detection of critical alerts from imbalanced multi-appliance threat alert logs. *2021 IEEE International Conference on Big Data (Big Data)*, 2119–2127. IEEE.

Ndichu, S., Ban, T., Takahashi, T., & Inoue, D. (2022). Security-Alert Screening with Oversampling Based on Conditional Generative Adversarial Networks. *2022 17th Asia Joint Conference on Information Security (AsiaJCIS)*, 1–7. IEEE.

Nettleton, D. (2014). Selection of Variables and Factor Derivation. Dalam *Commercial Data Mining* (hlm. 79–104). Elsevier. https://doi.org/10.1016/B978-0-12-416602-8.00006-6

Paluszek, M., & Thomas, S. (2020). What Is Deep Learning? Dalam *Practical MATLAB Deep Learning* (hlm. 1–24). Berkeley, CA: Apress. https://doi.org/10.1007/978-1-4842-5124-9_1

Panesar, A. (2019). What Is Machine Learning? Dalam *Machine Learning and AI for Healthcare* (hlm. 75–118). Berkeley, CA: Apress. https://doi.org/10.1007/978-1-4842-3799-1_3

Rahma, F., & Pratama, A. R. (2020). *Keamanan Siber Dan Informasi : Prinsip Dasar Dan Ancaman Terkini* (1 ed.). Yogyakarta: UII Press.

Rahma, F., Rachmadi, R. F., Pratomo, B. A., & Purnomo, M. H. (2023). Assessing the Effectiveness of Oversampling and Undersampling Techniques for Intrusion Detection on an Imbalanced Dataset. *IEEE Industrial Electronics and Applications Conference (IEACon)*. Penang, Malaysia.

Rajasa, M. C., Rahma, F., Rachmadi, R. F., Pratomo, B. A., & Purnomo, M. H. (2023). A Review of Imbalanced Datasets and Resampling Techniques in Network Intrusion Detection System. *International Conference on Information Technology and Digital Applications (ICITDA)*. Yogyakarta.

Sabari, K. K., & Shrivastava, S. (2022). Anomaly-based Intrusion Detection using GAN for Industrial Control Systems. *2022 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, 1–6. IEEE.

SagarDhandare. (2022, Maret 28). What is encoding? And its importance in data science!

Sharma, B., Sharma, L., & Lal, C. (2022). Anomaly based network intrusion detection for IoT attacks using convolution neural network. *2022 IEEE 7th International conference for Convergence in Technology (I2CT)*, 1–6. IEEE.

Suripto, Rahmanita, R. N., & Kirana, A. S. (2022, Agustus 26). Teknik pre-processing dan classification dalam data science.

Tanha, J., Abdi, Y., Samadi, N., Razzaghi, N., & Asadpour, M. (2020). Boosting methods for multi-class imbalanced data classification: an experimental review. *Journal of Big Data, 7*, 1–47.

Thiyam, B., & Dey, S. (2023). Efficient Feature Evaluation Approach for a class-imbalanced dataset using Machine learning. *Procedia Computer Science, 218*, 2520–2532.

Zuech, R., Hancock, J., & Khoshgoftaar, T. M. (2021). Detecting web attacks in severely imbalanced network traffic data. *2021 IEEE 22nd International Conference on Information Reuse and Integration for Data Science (IRI)*, 267–273. IEEE.

LAMPIRAN