PERBANDINGAN SUPERVISED LEARNING NEURAL NETWORK DAN UNSUPERVISED LEARNING AUTOENCODER UNTUK MENDETEKSI CLICK FRAUD PADA MOBILE ADVERTISING

TUGAS AKHIR

JACOBUS CALVIN KRISNA
1117044

INSTITUT
TEKNOLOGI
HARAPAN
BANGSA
Veritas vos liberabit

PROGRAM STUDI INFORMATIKA
INSTITUT TEKNOLOGI HARAPAN BANGSA
BANDUNG
2021

PERBANDINGAN SUPERVISED LEARNING NEURAL NETWORK DAN UNSUPERVISED LEARNING AUTOENCODER UNTUK MENDETEKSI CLICK FRAUD PADA MOBILE ADVERTISING

TUGAS AKHIR

Diajukan sebagai salah satu syarat untuk memperoleh
gelar sarjana dalam bidang Informatika

JACOBUS CALVIN KRISNA
1117044

INSTITUT
TEKNOLOGI
HARAPAN
BANGSA
Veritas vos liberabit

PROGRAM STUDI INFORMATIKA
INSTITUT TEKNOLOGI HARAPAN BANGSA
BANDUNG
2021

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Penggunaan *smartphone* dalam jumlah besar saat ini membuat industri *digital advertising* berkembang pesat, salah satunya adalah *mobile advertising* atau disebut juga *in-app advertisement*. Terdapat empat komponen utama dalam *mobile advertising*, yaitu: *User, Advertiser, Publisher,* dan *Ad Network*. *Advertiser* adalah pihak yang ingin mempromosikan produk bisnisnya kepada *user* dengan cara membayar jasa dari *ad network* untuk mempromosikan produk tersebut melalui tayangan iklan dalam *website* atau aplikasi. *Publisher* adalah pembuat/penyedia aplikasi *mobile* yang setuju menampilkan iklan dalam aplikasinya untuk mendapatkan sejumlah keuntungan. Sedangkan *ad network* bekerja untuk menyediakan platform yang menghubungkan *advertiser* dan *publisher* [3].

Sistem *mobile advertising* sering dihadapkan dengan masalah *click fraud* atau penipuan klik. Kurangnya atensi terhadap *fraud* khususnya pada *mobile advertising* menyebabkan kerugian sebesar 30% dari total *share digital market* [1]. Pada tahun 2017, menurut *CNBC report*, bisnis mendapat kerugian sebesar 16,4 miliar dolar akibat *fraud* tersebut [12]. Hingga tahun 2020, kerugian akibat *fraud* terus meningkat hingga mencapai 35 miliar dolar [13]. *Click fraud* merupakan tindakan melakukan klik pada sebuah tayangan iklan, namun bukan karena tertarik pada iklan tersebut, melainkan sebagai salah satu cara untuk menghasilkan keuntungan bagi *publisher*. Model biaya yang bersifat *cost-per-click* atau CPC dapat menyebabkan kerugian bagi *advertiser* apabila *publisher* yang tidak bertanggung jawab melakukan *click fraud* untuk kepentingannya sendiri. Di lain sisi, *advertiser* tetap harus membayar biaya iklan sesuai jumlah klik yang tercatat dalam platform kepada *ad network* tanpa mendapatkan atensi ataupun keuntungan dari *user* [2]. Oleh karena itu, diperlukan sistem yang dapat mendeteksi *click fraud* untuk dapat menjamin keamanan bagi *advertiser* saat mempromosikan iklan pada sebuah *ad network*.

Penelitian oleh Rakin Haider [1] bertujuan untuk klasifikasi *fraud* berdasarkan impresi sebuah iklan. Impresi adalah peristiwa saat iklan dari *advertiser* ditampilkan pada perangkat *user*. Dataset pada penelitian Haider diambil dari *European Mobile Ad Server Company* yang berjumlah 260.678 data. Metode yang digunakan adalah *Ensemble Learning* dengan tiga *classifier*, yaitu: *J48, REPTree,* dan *Random Forest*. Serta dilakukan juga teknik *Bagging* dan *Boosting* untuk

mengurangi *error rate* pada masing-masing *classifier*. Hasil yang diperoleh paling tinggi menggunakan *Boosting J48* dengan akurasi 99,32%, *precision* 96,29%, dan *recall* 84,75%. Lalu ada penelitian oleh Thejas G. S. [7] yang menggunakan metode *hybrid* antara *Cascade Forest* dan *XGBoost* (CFXGB) untuk mendeteksi *click fraud*. Dataset yang digunakan adalah TalkingData (200 juta data), Avazu (40 juta data), dan Kad (1.000 data). *Cascade Forest* merupakan model *ensemble-based* yang menggunakan *Random Forest*, *Extremely Randomized Trees*, dan *XGBoost*. *Cascade Forest* digunakan untuk *feature transformer* pada model dan *XGBoost* digunakan untuk mengklasifikasikan *click fraud*. Hasil tertinggi yang diperoleh menggunakan metode CFXGB terhadap tiga dataset tersebut adalah AUC 94,53%, *precision* 94%, *recall* 94%, dan *F1-score* 94%.

Penelitian lain oleh Marcin Gabryel [4] bertujuan membuat algoritme untuk menganalisis data klik individual dan mengklasifikasikannya menjadi *OK* atau *suspect*. Data dikumpulkan secara manual menggunakan *Javascript* yang ditanamkan pada *website advertiser* untuk mendapatkan informasi terkait *user* saat mengakses *website* tersebut. Metode yang digunakan adalah kombinasi antara algoritme *Term Frequency - Inverse Document Frequency* (TF-IDF) yang dioptimisasi oleh *Differential Evolution* (DE) dan algoritme *K-Nearest Neighbor* (KNN). Hasil dari kombinasi metode mendapatkan akurasi paling tinggi sebesar 97,71%. Lalu penelitian oleh Thejas G. S. [3] bertujuan untuk mengatasi serangan *fraud* berupa *active learning system*. Serangan tersebut bertujuan untuk mengotori data belajar dan menyebabkan sistem deteksi *click fraud* menjadi rusak/tidak efisien. Dataset terdiri dari 184 juta *real-time ad click* pada platform *mobile*. Metode yang digunakan berbasis *Deep-Learning* dengan menggabungkan *Artificial Neural Network* (ANN), *Autoencoder*, dan *Semi-supervised Generative Adversarial Network* (GAN). Hasil yang diperoleh menggunakan metode berbasis *Deep-Learning* tersebut mendapatkan akurasi paling tinggi sebesar 95%.

Namun, masalah *click fraud* lebih cenderung kepada kasus anomali daripada kasus klasifikasi. Hal tersebut menyebabkan *supervised learning* dapat menjadi kurang efektif untuk mendeteksi *click fraud*. Alasan tersebut didukung dengan penelitian Thejas G. S. [3] yang pendekatannya tidak murni *supervised learning*, namun menggunakan kombinasi antara *semi-supervised learning* dan *unsupervised learning*. Oleh karena itu, penelitian ini akan membandingkan pendekatan *supervised learning* dengan *unsupervised learning* untuk mendeteksi *click fraud*. Pendekatan *supervised learing* akan menggunakan Neural Network yang dilatih menggunakan dataset berlabel, sedangkan pendekatan *unsupervised learning* akan

menggunakan Autoencoder yang dilatih menggunakan dataset yang tidak berlabel. Di akhir penelitian akan dibandingkan *recall* dan *f1 measure* antara kedua pendekatan yang dilakukan.

1.2 Rumusan Masalah

Rumusan masalah yang terdapat pada penelitian ini, yaitu:

1. Berapakah perbandingan nilai *recall* dan *f1 measure* antara pendekatan *supervised learning* menggunakan Neural Network dan *unsupervised learning* menggunakan Autoencoder untuk mendeteksi *click fraud*?

1.3 Tujuan Penelitian

Tujuan yang ingin dicapai dalam penelitian ini adalah membandingkan pendekatan *supervised learning* yang menggunakan Neural Network dan *unsupervised learning* yang menggunakan Autoencoder dalam mendeteksi *click fraud*.

1. Melakukan *pre-processing* terhadap dataset yang *imbalanced* menggunakan SMOTE (Synthetic Minority Oversampling Technique).
2. Melakukan pendeteksian *click fraud* dengan menerapkan Neural Network yang menggunakan data berlabel dan Autoencoder yang menggunakan data tidak berlabel.
3. Membandingkan *recall* dan *f1 measure* dari penerapan Neural Network dan Autoencoder untuk mendeteksi *click fraud*.

1.4 Batasan Masalah

Batasan masalah yang diberikan pada penelitian ini, yaitu:

1. Penerapan metode untuk mendeteksi *click fraud* hanya mencakup *mobile advertising* saja dan tidak berhubungan dengan *digital advertising* lainnya. Deteksi tersebut menggunakan data historikal berisi informasi klik pada *mobile advertising* yang berasal dari perusahaan *TalkingData* melalui *Kaggle*.
2. Sistem deteksi *click fraud* bekerja secara *offline* sehingga tidak dapat digunakan pada kasus *real-time*.

1.5 Konstribusi Penelitian

Dengan dilakukannya penelitian ini, diharapkan dapat mengetahui cara dan hasil dari penerapan *supervised learning* menggunakan Neural Network dan *unsupervised learning* menggunakan Autoencoder dalam mendeteksi *click fraud*. Serta dapat memberikan perbandingan *recall* dan *f1 measure* antara Neural Network dan Autoencoder.

1.6 Metodologi Penelitian

Tahapan-tahapan yang dilakukan dalam penelitian ini, yaitu:

1. **Studi Literatur**
   Tahap ini penulis memulai penelitian dengan mempelajari literatur melalui *online* baik *paper*, jurnal, artikel, dan buku yang berkaitan dengan pendeteksian *click fraud*.

2. **Analisis Masalah**
   Tahap ini penulis melakukan analisis terhadap permasalahan yang ada, serta menentukan tujuan dan batasan yang ada pada penelitian ini.

3. **Pencarian Data**
   Tahap ini merupakan proses pencarian dataset yang akan digunakan dalam penelitian. Dataset yang digunakan merupakan data *click* yang dilakukan *user* pada *mobile advertising* pada tahun 2017. Data tersebut disediakan oleh perusahaan *Big Data* yang berasal dari China bernama TalkingData.

4. **Pemodelan Data**
   Tahap ini merupakan tahap untuk memproses data yang jumlahnya tidak seimbang menjadi seimbang. Setelah itu dilakukan juga pembagian data latih untuk *supervised learning* yaitu data berlabel dan *unsupervised learning* yaitu data tidak berlabel.

5. ***Training* Data**
   Tahap ini merupakan proses pelatihan kedua model untuk mendeteksi *click fraud* menggunakan data latih yang telah dimodelkan sebelumnya.

6. ***Testing* Data**
   Tahap ini merupakan tahap pengujian kedua model yang telah dilatih sebelumnya menggunakan data latih untuk mendeteksi *click fraud*.

7. **Evaluasi** Tahap ini merupakan proses untuk mengevaluasi hasil deteksi *click fraud* kedua model menggunakan parameter *recall* dan *f1 measure* sehingga dapat memberikan perbandingan dari pendekatan *supervised learning* dan *unsupervised learning*.

1.7 Sistematika Pembahasan

Pada penelitian ini, penulis menyusun berdasarkan sistematikan pembahasan sebagai berikut:

BAB I PENDAHULUAN

Berisi pendahuluan yang terdiri dari latar belakang, rumusan masalah, tujuan penelitian, batasan masalah, kontribusi penelitian, dan metodologi penelitian.

BAB II LANDASAN TEORI

Berisi penjelasan mengenai dasar teori yang dibutuhkan untuk mendukung penelitian ini.

BAB III ANALISIS DAN PERANCANGAN

Berisi penjelasan metode beserta algoritme yang digunakan dalam penelitian ini.

BAB IV IMPLEMENTASI DAN PENGUJIAN

Berisi implementasi dan pengujian yang dilakukan menggunakan metode terkait beserta data testing dan hasilnya.

BAB V KESIMPULAN DAN SARAN

Berisi kesimpulan dari penelitian yang dilakukan dan saran untuk penelitian lebih lanjut di masa mendatang.

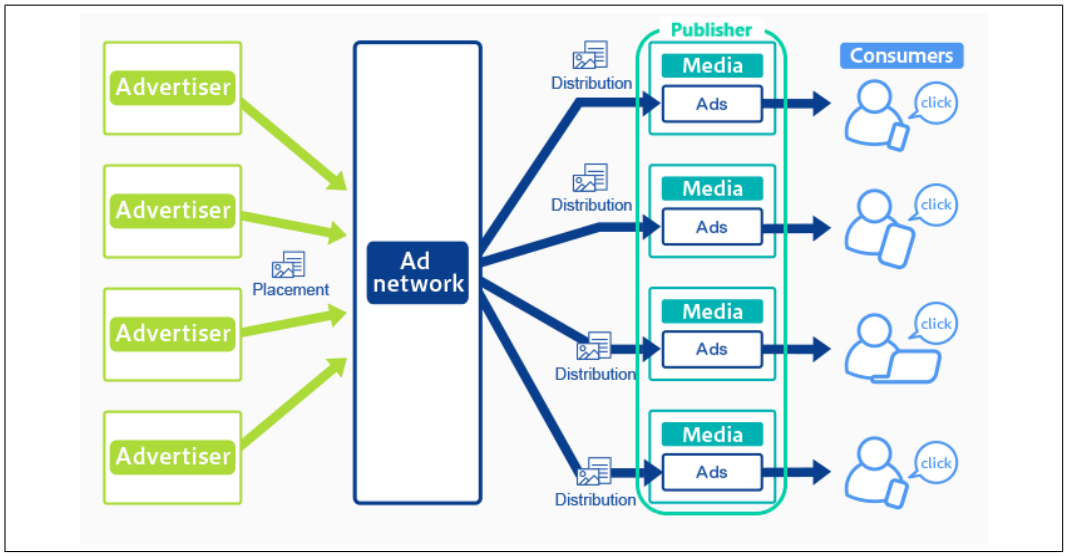
BAB 2 LANDASAN TEORI

2.1 Tinjauan Pustaka

Penelitian ini menggunakan beberapa teori terkait yang diperlukan dalam pengerjaan yang dilakukan. Penjelasan mengenai teori-teori tersebut akan dijelaskan sebagai berikut.

2.1.1 *Mobile Advertising*

Mobile Advertising atau iklan seluler merupakan bisnis yang sangat populer beberapa tahun terakhir, hal tersebut didukung dengan banyaknya masyarakat yang mulai kecanduan untuk menggunakan *smartphone* setiap harinya [3]. *Mobile advertising* adalah tayangan iklan yang ditampilkan pada perangkat *smartphone* dengan tujuan untuk mendapatkan atensi dan minat dari pengguna terhadap produk yang diiklankan. Terdapat empat komponen utama dalam *mobile advertising*, yaitu: 1) *User* yang melihat iklan, 2) *Advertiser* yang membayar agar iklannya ditampilkan pada sebuah aplikasi tertentu, 3) *Publisher* atau pembuat aplikasi yang bersedia agar iklan ditampilkan pada aplikasinya dengan sejumlah bayaran tertentu, dan 4) *Ad network* sebagai pihak yang menyediakan *platform* untuk menghubungkan *advertiser* dengan *publisher* [2]. Ekosistem *mobile advertising* dapat dilihat pada gambar 2.1.



Gambar 2.1 Ekosistem *Mobile Advertising* [18]

Industri *mobile advertising* memiliki beberapa model biaya periklanan yang

dibebankan kepada *advertiser* oleh *ad network*, di antaranya: *cost-per-click*, *cost-per-thousand-impression* dan *cost-per-action*. Model biaya *cost-per-click* atau CPC menjadi perhatian khusus saat ini karena sering disalahgunakan oleh *publisher* yang tidak bertanggung jawab untuk mendapatkan keuntungan yang besar. Setiap *click* yang dilakukan oleh *user* akan dicatat oleh *ad network*. Lalu *advertiser* berkewajiban untuk membayar biaya iklan sesuai jumlah klik yang tercatat baik itu klik *legitimate* atau *fraud* [1].

2.1.1.1 *Click Fraud*

Dengan adanya model biaya *cost-per-click* atau CPC menyebabkan terjadinya *click fraud*. *Click Fraud* atau penipuan klik merupakan tindakan *user* melakukan klik pada sebuah iklan, namun bukan karena tertarik dengan iklan tersebut, melainkan sebagai salah satu cara untuk menghasilkan keuntungan bagi *publisher* [2]. *Click fraud* dilakukan oleh *publisher* dengan cara melakukan klik dalam jumlah besar terhadap iklan pada aplikasinya seolah-olah *user* legit yang melakukannya. Hal tersebut sangat merugikan *advertiser* karena harus membayar biaya iklan sesuai jumlah klik tanpa mendapatkan keuntungan dari *user*.

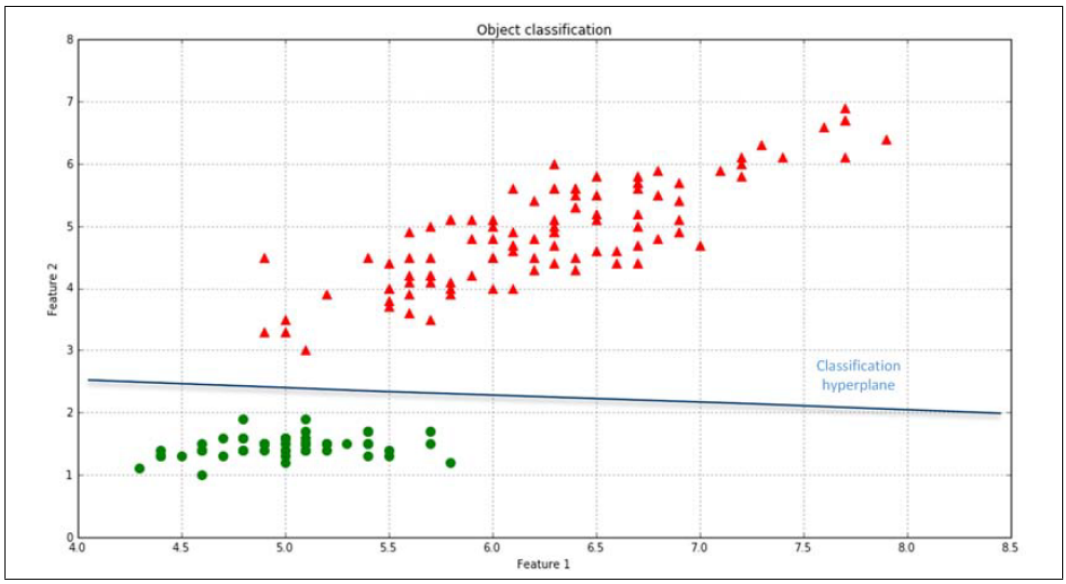
Terdapat enam jenis serangan *click fraud* yang sangat populer, di antaranya: 1) *Brute Force Attack* yaitu menggunakan mesin komputasi untuk melakukan klik secara terus menerus, 2) *Attack Through Crowdsourcing* yaitu cara memohon pertolongan *user* untuk melakukan klik terhadap iklan atas dasar belas kasihan, 3) *Attack Through Reward Traffic* yaitu memberi hadiah kepada *user* berupa kode kupon diskon atau poin bonus *game*, 4) *Click Farm Attack* yaitu cara untuk memengaruhi sekumpulan orang atau grup untuk melakukan klik terhadap iklan dengan imbalan sejumlah uang, 5) *Hit Inflation Attack* yaitu melakukan *redirect* (mengalihkan) *user* legit yang melakukan klik terhadap sebuah iklan ke halaman lain yang tidak diketahui *(unknown)*, dan 6) *Botnet Attack* yaitu penyerang profesional yang menggunakan *malware* untuk membuat perangkat *user* meng-*install* aplikasi pada *background* sehingga tidak diketahui oleh *user*.

Kerugian yang disebabkan oleh *click fraud* pada tahun 2020 secara global, menurut CHEQ, mencapai 35 miliar dolar US [13]. Kerugian tersebut diproyeksikan akan terus meningkat apabila *click fraud* tidak ditangani dengan baik. Di Indonesia, kerugian yang disebabkan oleh penipuan iklan pada tahun 2019 mencapai 120 juta dolar US [14]. Bahkan pada tahun 2017, perusahaan besar *e-commerce* seperti Tokopedia juga mengalami kerugian akibat penipuan iklan, hal tersebut diketahui

dengan adanya biaya iklan yang melonjak tinggi dan angka instalasi aplikasi tinggi namun retensi rendah (tingkat kembalinya pengguna menggunakan aplikasi) [15].

2.1.2 *Supervised Learning*

Supervised learning atau pembelajaran yang diawasi merupakan konsep guru atau pengawas yang tujuan utamanya adalah menghasilkan agen dengan pengukuran *error* yang tepat (secara langsung dibandingkan dengan nilai keluaran). Fungsi dari algoritme *supervised* adalah menggunakan data belajar yang terdiri dari masukan dan keluaran yang diharapkan. Dari informasi data belajar tersebut, agen mencoba membenarkan parameter untuk mengurangi besarnya *global loss function*. Pada setiap iterasi, jika algoritme cukup fleksibel dan elemen data koheren, maka akurasi akan meningkat dan perbedaan antara nilai prediksi dan nilai yang diharapkan akan mendekati nol (0) [8]. Contoh kasus penerapan *supervised learning* pada saat ini diantaranya: prediksi berdasarkan klasifikasi regresi atau kategorikal, deteksi spam, deteksi pola, *Natural Language Processing*, analisis sentimen, klasifikasi gambar, dan lain-lain. Pada gambar 2.2 merupakan contoh klasifikasi dua fitur menggunakan pembelajaran *supervised*.



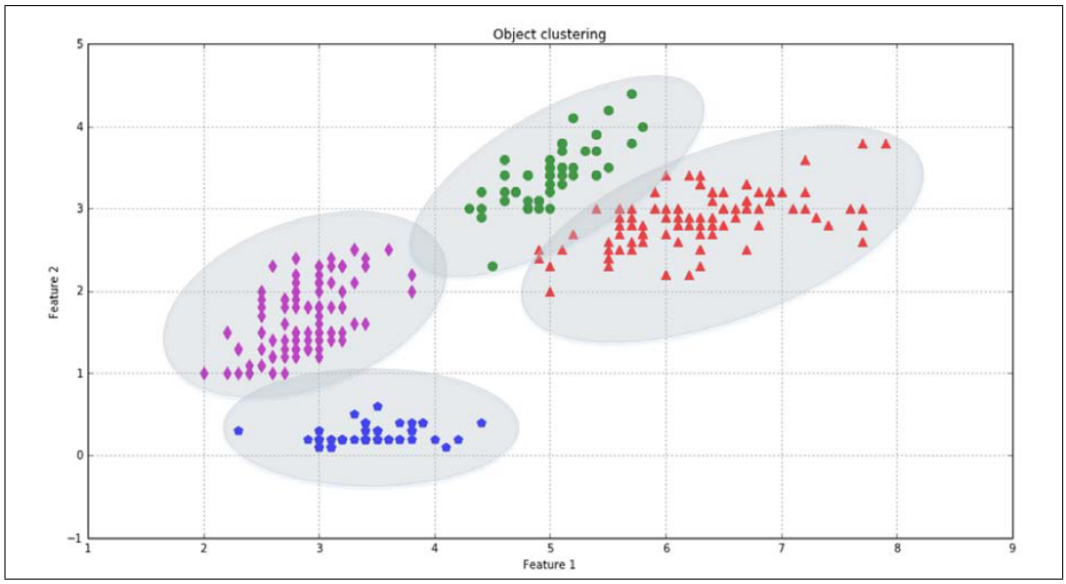
Gambar 2.2 Contoh Klasifikasi Menggunakan Pembelajaran *Supervised* [8]

Pada gambar diatas dapat dilihat algoritme klasifikasi mencoba untuk menemukan *hyperplane* (garis pembagi) yang paling baik untuk memisahkan kedua fitur. Tujuan dari pembelajaran adalah mengurangi jumlah kesalahan klasifikasi dan meningkatkan ketahanan terhadap data *noise*. Sebagai contoh, data poin warna merah yang dekat dengan garis *hyperplane* (koordinat sekitar [5.1 - 3.0]), jika fitur

2 terpengaruh dengan data *noise* dan nilainya menjadi 3.0 maka *hyperplane* dapat salah mengklasifikasikan menjadi data poin warna hijau.

2.1.3 *Unsupervised Learning*

Unsupervised learning atau pembelajaran yang tidak diawasi merupakan pendekatan yang didasarkan pada tidak adanya pengawas. Pembelajaran *unsupervised* berguna ketika perlu mempelajari bagaimana serangkaian elemen dapat dikelompokkan *(clustering)* berdasarkan kesamaannya (atau ukuran jarak) [8]. Pada gambar 2.3 merupakan contoh pembelajaran *unsupervised* yang digunakan untuk mengelompokkan data.



Gambar 2.3 Contoh Pengelompokkan Data Menggunakan Pembelajaran *Unsupervised*[8]

Pada gambar diatas dapat dilihat lingkaran warna abu-abu merupakan klaster dan setiap data poin (warna biru, ungu, hijau, dan merah) di dalamnya dapat dilabelkan serupa. Terdapat juga titik batas (seperti data poin warna merah yang tumpang tindih dengan area data poin warna hijau) yang memerlukan kriteria khusus (biasanya menggunakan *trade-off* ukuran jarak) untuk menentukan klaster yang sesuai. Dalam kasus nyata, sering kali terdapat area batas yang tumpang tindih sebagian, yang berarti bahwa beberapa data poin memiliki tingkat ketidakpastian yang tinggi karena nilai fiturnya.

2.1.4 *Artificial Neural Network*

Artificial Neural Network atau disebut *neural network* merupakan metode yang terinspirasi dari sistem saraf dan otak manusia lalu diimplementasikan pada komputer digital. Otak bersifat sangat kompleks, non-linear, dan dapat menghitung secara paralel, sehingga dapat mengatur cara kerja dari struktur sistem saraf seperti neuron yang dapat melakukan perhitungan tertentu dengan waktu yang jauh lebih cepat daripada waktu perhitungan komputer digital [11]. *Neural network* dapat disebut sebagai mesin yang dibentuk untuk membuat model yang sama dengan otak manusia dalam melakukan suatu aktivitas. Dalam *neural network* terdapat proses pembelajaran yang disebut sebagai algoritme pembelajaran dan berfungsi untuk memodifikasi bobot sinapsis *(weights)* dari *network* secara teratur hingga mencapai tujuan desain yang diinginkan. Metode pembelajaran pada *neural network* umumnya tergolong atas *supervised* dan *unsupervised*. Bila nilai keluaran telah diketahui maka termasuk metode *supervised* karena *neural network* akan berusaha untuk mendapatkan nilai tersebut, sering digunakan untuk kasus klasifikasi dan regresi. Sementara metode *unsupervised* tidak menggunakan nilai keluaran sebagai tujuan, sering digunakan untuk kasus pengelompokan dan asosiasi.

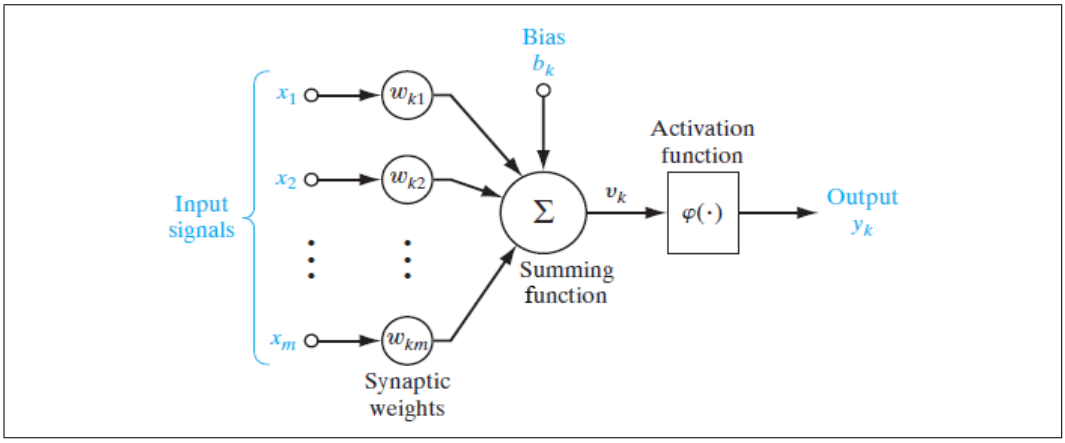
Pada penerapannya, *neural network* dapat digunakan pada sistem *offline* atau *online*/*real-time*. Contoh penerapan pada sistem *offline* seperti pada penelitian oleh Sevdalina Georgieva [17] yang menerapkan *neural network* untuk klasifikasi transaksi kartu kredit berdasarkan data historikal. Sedangkan penerapan pada sistem *online*/*real-time* seperti pada penelitian oleh Thejas G.S. [3] yang menerapkan *neural network* untuk memprediksi *user* yang akan men-*download* aplikasi berdasarkan data informasi klik *user*, kemudian sistem tersebut digunakan untuk menyeleksi *user* untuk diizinkan masuk ke *website* iklan atau tidak secara *real-time*. Lalu, dalam *neural network* terdapat neuron yang merupakan unit untuk memproses informasi. Terdapat tiga elemen dasar dalam neuron, yaitu:

1. Kumpulan sinapsis yang masing-masing memiliki bobot atau *weight*. Sinyal masukan x_j pada sinapsis j yang terhubung dengan neuron k akan dikalikan dengan bobot sinapsis w_{kj}.

2. Proses penjumlahan untuk menjumlahkan sinyal masukan yang telah diproses dengan bobotnya pada masing-masing sinapsis neuron, operasi ini disebut *linear combiner*.

3. Fungsi aktivasi untuk membatasi rentang dari keluaran neuron. Biasanya,

jarak output dari neuron dibuat antara 0 hingga 1 atau -1 hingga 1.



Gambar 2.4 Neuron non-linear [11]

Pada gambar 2.4 merupakan model neuron non-linear yang terdiri dari sinyal masukan, bias, dan keluaran. Bias memiliki fungsi untuk dapat menaikan dan menurunkan input bersih dari fungsi aktivasi yang bergantung pada positif atau negatif nilainya. Persamaan dasar *neural network* untuk proses maju atau *feed-forward* dapat dilihat dibawah ini:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.1}$$

$$v_k = u_k + b_k \tag{2.2}$$

$$y_k = \varphi(v_k) \tag{2.3}$$

Keterangan:

u_k	: keluaran *linear combiner* dari sinyal masukan.
x	: sinyal masukan.
w_k	: bobot sinapsis pada neuron k.
m	: jumlah masukan.
b_k	: nilai bias.
v_k	: keluaran fungsi *transfer*.
y_k	: sinyal keluaran dari neuron.
φ	: fungsi aktivasi.

Fungsi aktivasi yang dilambangkan dengan $\varphi(v)$ mendefinisikan keluaran dari

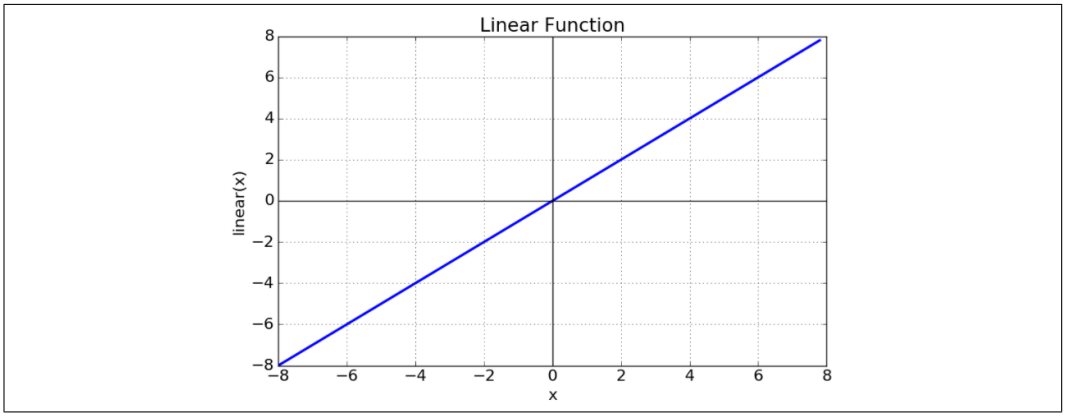
neuron dalam nilai lokal *v*. Fungsi aktivasi dibagi menjadi dua yaitu linear dan non-linear. Fungsi aktivasi linear memiliki fungsi garis atau linear dan keluaran dari fungsi tidak dibatasi di antara rentang nilai apapun. Sedangkan fungsi aktivasi yang seringkali digunakan adalah non-linear karena memudahkan model untuk melakukan generalisasi atau adaptasi dengan berbagai data [19]. Fungsi aktivasi non-linear dibagi berdasarkan jarak atau kurva. Pada kasus klasifikasi biasanya menggunakan fungsi *sigmoid* pada *output layer* hal tersebut dikarenakan menghasilkan probabilitas antara 0 sampai 1. Pada penelitian Zhaomin Chen [5] menggunakan fungsi aktivasi *sigmoid* untuk mendeteksi anomali pada jaringan yang memiliki dua probabilitas keluaran yaitu anomali dan bukan anomali. Sedangkan pada *hidden layer* secara umum saat ini menggunakan *Rectified Linear Unit* (ReLU) sebagai patokan awal namun jika hasil dari model belum baik maka dapat menggunakan fungsi aktivasi lainnya untuk proses percobaan [20]. Pada penelitian Thejas G.S. [3] menggunakan fungsi aktivasi ReLU pada *hidden layer* untuk mendeteksi *click fraud*. Berikut ini merupakan penjelasan mengenai fungsi aktivasi linear, *sigmoid*, dan ReLU:

1. Fungsi linear memiliki jarak aktivasi antara negatif tak hingga sampai positif tak hingga. Rumus linear dapat dilihat pada persamaan 2.4 dan kurva linear pada gambar 2.5.

$$f(x) = x \tag{2.4}$$

Keterangan:

f	: lambang fungsi.
x	: sinyal masukan.



Gambar 2.5 Kurva Linear [19]

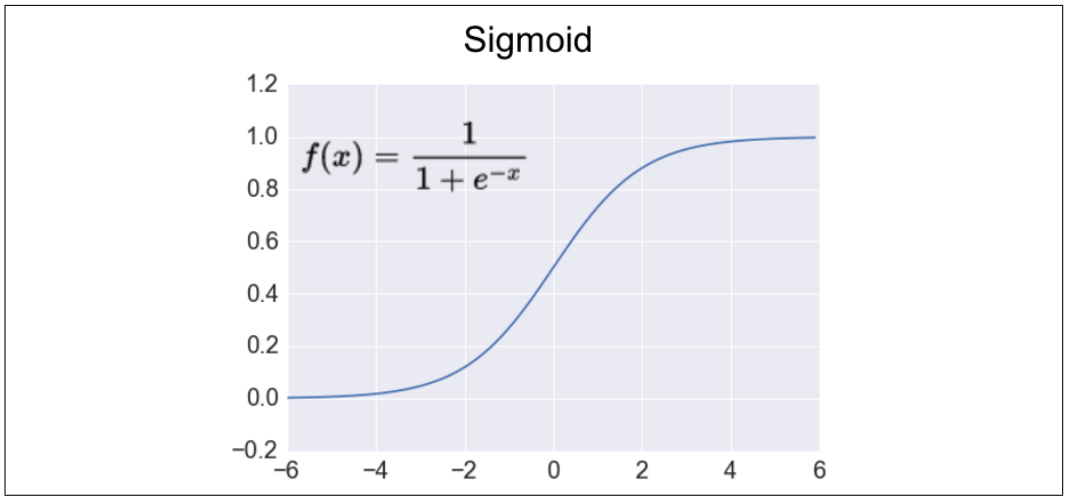
2. Fungsi *sigmoid* atau *logistic* memiliki jarak aktivasi antara 0 sampai 1 seperti pada persamaan 2.5. Rumus *sigmoid* dapat dilihat pada persamaan 2.6 dan kurva *sigmoid* pada gambar 2.6.

$$\varphi(v) = \begin{cases} 1 \text{ ; jika } v > 0,5 \\ 0 \text{ ; jika } v \leq 0,5 \end{cases} \tag{2.5}$$

$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}} \tag{2.6}$$

Keterangan:

φ	: fungsi aktivasi.
v	: keluaran fungsi *transfer*.
f	: lambang fungsi.
σ	: fungsi sigmoid.
e	: fungsi eksponensial.
x	: sinyal masukan.



Gambar 2.6 Kurva *Sigmoid* [19]

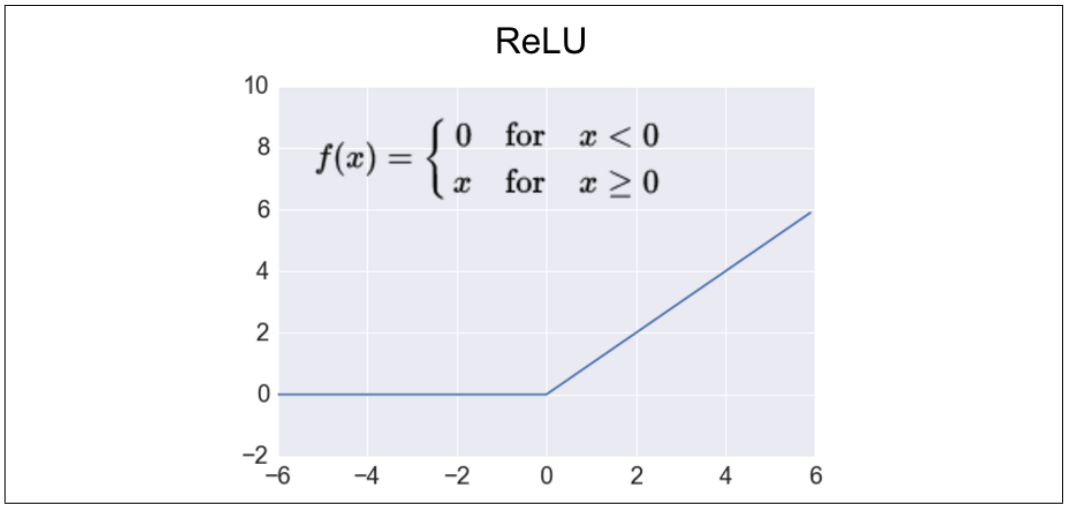
3. Fungsi ReLU *(Rectified Linear Unit)* memiliki jarak aktivasi 0 sampai tak hingga seperti pada persamaan 2.7. Rumus ReLU dapat dilihat pada persamaan 2.8 dan kurva ReLU pada gambar 2.7.

$$\varphi(v) = \begin{cases} \text{v ; jika } v > 0 \\ 0 \text{ ; jika } v \leq 0 \end{cases} \tag{2.7}$$

$$f(x) = max(0,x) \tag{2.8}$$

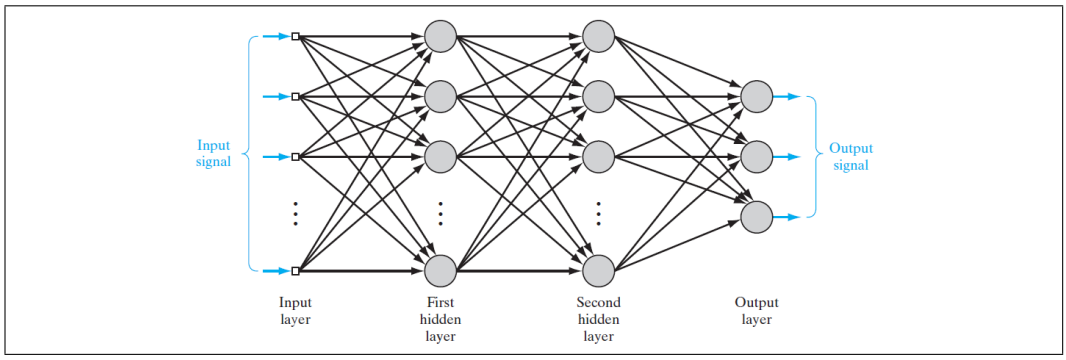
Keterangan:

φ	: fungsi aktivasi.
v	: keluaran fungsi *transfer*.
f	: lambang fungsi.
max	: nilai maksimum.
x	: sinyal masukan.



Gambar 2.7 Kurva ReLU [19]

Dalam penerapannya, *neural network* dapat memiliki lebih dari satu *layer* atau disebut juga *multilayer perceptron* seperti pada gambar 2.8. Pada gambar tersebut *neural network* terbentuk dengan dua layer tersembunyi.



Gambar 2.8 *Multilayer Perceptron* [11]

Pada gambar 2.8 di atas merupakan *multilayer perceptron* yang terdapat sejumlah *n* sinyal masukan, dua *hidden layer*, dan tiga sinyal keluaran. Gambar di atas dapat diartikan sebagai klasifikasi multi kelas (tiga kelas). Dalam *multilayer perceptron* terdapat dua jenis sinyal, yaitu:

1. *Function signal* yang berasal dari sinyal masukan (stimulus) dan disebarkan secara maju (*neuron* per *neuron*) hingga mencapai akhir dari jaringan.
2. *Error signal* yang berasal dari keluaran *neuron* dan disebarkan terbalik (*layer* per *layer*).

Dalam *neural network* diterapkan *backpropagation* yang merupakan inti dari pembelajaran. *Backpropagation* digunakan untuk memperhitungkan nilai *error*

dalam proses mundur. Nilai *error* tersebut didapat dari nilai target sebenarnya dikurangi dengan nilai *output* yang dihasilkan, sehingga menghasilkan nilai deviasi atau disebut juga dengan nilai *error*. Nilai *error* dapat dihitung menggunakan *loss function* seperti *crossentropy* untuk klasifikasi multi kelas atau *binary crossentropy* pada klasifikasi biner.

2.1.4.1 *Loss Function*

Loss function merupakan fungsi yang memperhitungkan nilai *cost* atau biaya yang dihasilkan oleh suatu fungsi komputasi. *Loss function* digunakan untuk membuat estimasi *loss* yang dihasilkan sebuah model dan dapat digunakan untuk memperbaharui nilai bobot *(weight)* sehingga memperkecil *loss* pada evaluasi selanjutnya. Selain itu, *loss function* dapat digunakan sebagai evaluasi sebuah pembelajaran model apakah baik atau buruk hasilnya. Jika grafik *loss function* menurun pada setiap iterasi, maka pembelajaran tersebut dianggap ada perkembangan yang baik. Pada penelitian ini digunakan *binary cross entropy* mengingat prediksi hanya dua nilai, yaitu: *fraud* dan non-*fraud*. Persamaan 2.9 merupakan rumus dari *binary cross entropy*:

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i)) \tag{2.9}$$

Keterangan:

N	: jumlah neuron keluaran.
i	: jumlah iterasi.
y_i	: target *output* pada neuron keluaran ke i .
$p(y_i)$	: probabilitas target output y_i.
log	: fungsi logaritma.

2.1.4.2 *Backpropagation*

Dalam *neural network* terdapat dua tahap, yaitu maju dan mundur. Tahap maju atau disebut tahap *feed-forward* yang dimulai dari *input layer* sampai pada *output layer*. Setelah itu tahap mundur atau disebut tahap *backpropagation* dilakukan secara terbalik dari *output layer* sampai pada *input layer*. *Backpropagation*

merupakan proses menghitung gradien atau turunan secara efisien. Tujuan utama dari *backpropagation* adalah untuk mengoptimalkan *neural network* dengan cara memperbarui nilai bobot dan bias pada setiap *layer* [27]. Dalam melakukan *backpropagation* terdapat *learning rate* yang sangat berpengaruh dalam memperbarui nilai bobot. *Learning rate* merupakan persentase besarnya perubahan nilai bobot yang akan diterapkan pada *neural network* terhadap nilai *error*. Pembaruan nilai bobot dapat menggunakan rumus pada persamaan 2.10.

$$w_i^{'} = w_i - (\alpha \times \delta w_i) \tag{2.10}$$

Keterangan:

$w_i^{'}$	: nilai bobot baru pada *layer* ke *i*.
w_i	: nilai bobot lama pada *layer* ke *i*.
α	: *learning rate*.
δw_i	: delta atau turunan parsial nilai bobot pada *layer* ke *i*.

Jika dijabarkan, δw_i merupakan *chain rule* dari rumus yang digunakan dalam menghitung nilai *error*. *Chain rule* merupakan rumus yang digunakan untuk menghitung turunan beberapa fungsi. Rumus turunan terhadap δw_i dapat dilihat pada persamaan 2.11.

$$\delta w_i = \frac{\delta c}{\delta w_i} = \frac{\delta c}{\delta a_i} \times \frac{\delta a_i}{\delta z_i} \times \frac{\delta z_i}{\delta w_i} \tag{2.11}$$

Keterangan:

δw_i	: delta atau turunan parsial nilai bobot pada *layer* ke *i*.
δc	: delta atau turunan parsial *cost function*.
δa_i	: delta atau turunan parsial *fungsi aktivasi* pada *layer* ke *i*.
δz_i	: delta atau turunan parsial perkalian nilai bobot dan nilai *neuron* pada *layer* ke *i*.

Perhitungan untuk pembaruan nilai bias sama dengan nilai bobot. Perbedaan perhitungan nilai bias hanya pada turunan parsial terhadap bias atau δb seperti

pada persamaan 2.12.

$$\delta b_i = \frac{\delta c}{\delta b_i} = \frac{\delta c}{\delta a_i} \times \frac{\delta a_i}{\delta z_i} \times \frac{\delta z_i}{\delta b_i} \tag{2.12}$$

Keterangan:

δb_i	: delta atau turunan parsial nilai bias pada *layer* ke *i*.
δc	: delta atau turunan parsial *cost function*.
δa_i	: delta atau turunan parsial *fungsi aktivasi* pada *layer* ke *i*.
δz_i	: delta atau turunan parsial perkalian nilai bobot dan nilai *neuron* pada *layer* ke *i*.

Algoritme pada *neural network* dibagi menjadi dua, yaitu *feedforward* dan *backpropagation*. Pada gambar 2.9 merupakan tahap awal inisialisasi *weight* dan algoritme proses *feedforward*:

Step 0. Initialize weights (set to small random values).

Step 1. While stopping condition is false, do Steps 2-9.

Step 2. For each training pair, do Steps 3-8.

Feedforward:

Step 3. Each input unit ($X_i, i = 1, \dots, n$) receives input signal x_i and broadcasts this signal to all units in the layer above (the hidden units).

Step 4. Each hidden unit ($Z_j, j = 1, \dots, p$) sums its weighted input signals,

$$z_{in_j} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$$

applies its activation function to compute its output signal,

$$z_j = f(z_{in_j})$$

and sends this signal to all units in the layer above (output units).

Step 5. Each output unit ($Y_k, k = 1, \dots, m$) sums its weighted input signals,

$$y_{in_k} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$$

applies its activation function to compute its output signal,

$$y_k = f(y_{in_k})$$

Gambar 2.9 Algoritme *feedforward* pada *neural network* [9]

Selanjutnya pada gambar 2.10 adalah algoritme proses *backpropagation* serta

pembaruan nilai *weight* dan *bias* yang merupakan lanjutan dari proses *feedforward* diatas:

Backpropagation of error:

Step 6. Each output unit ($Y_k, k = 1, \dots, m$) receives a target pattern corresponding to the Sinput training pattern, computes its error information term,

$$\delta_k = (t_k - y_k) \cdot f'(y_{in_k})$$

calculates its weight correction term (used to update w_{jk} later),

$$\Delta_{w_{jk}} = \alpha \delta_k z_j$$

calculates its bias correction term (used to update w_{0k} later),

$$\Delta_{w_{0k}} = \alpha \delta_k$$

and sends δ_k to units in the layer below.

Step 7. Each hidden unit ($Z_j, j = 1, \dots, p$) sums its delta inputs (from units in the layer above),

$$\delta_{in_j} = \sum_{k=1}^{m} \delta_k w_{jk}$$

multiplies by the derivative of its activation function to calculate its error information term,

$$\delta_j = \delta_{in_j} \cdot f'(z_{in_j})$$

and calculates its bias correction term (used to update v_{0j} later),

$$\Delta_{v_{0j}} = \alpha \delta_j$$

Updates weights and biases:

Step 8. Each output unit ($Y_k, k = 1, \dots, m$) updates its bias and weights ($j = 0, \dots, p$):

$$w_{jk}(new) = w_{jk}(old) + \Delta_{w_{jk}}$$

Each hidden unit ($Z_j, j = 1, \dots, p$) updates its bias and weights ($i = 0, \dots, n$):

$$v_{ij}(new) = v_{ij}(old) + \Delta_{v_{ij}}$$

Step 9. Test stopping condition.

Gambar 2.10 Algoritme *backpropagation* pada *neural network* [9]

2.1.5 Optimisasi Adam

Algoritme pengoptimalan untuk pembelajaran model dapat memberikan hasil yang bagus dalam waktu komputasi. Salah satu algoritme pengoptimalan yang saat ini banyak digunakan pada industri adalah Adam. Adam merupakan singkatan dari *adaptive moment estimation*, yang merupakan sebuah metode untuk pengoptimalan stokastik yang efisien dan hanya membutuhkan gradien pertama dengan kebutuhan memori yang sedikit [33]. Metode Adam menghitung *learning rate* adaptif untuk parameter yang berbeda berdasarkan estimasi momentum gradien pertama dan gradien kedua. Adam merupakan gabungan dari dua metode

optimisasi, yaitu AdaGrad *(adaptive gradient)* dan RMSProp *(root mean square propagation)*. Secara umum, AdaGrad mempertahankan *learning rate* setiap parameter yang meningkatkan performa pada masalah gradien renggang. Lalu RMSProp yang juga mempertahankan *learning rate* setiap parameter, diadaptasi berdasarkan rata-rata besaran gradien terbaru untuk nilai bobot, hal tersebut memberikan keuntungan RMSProp karena dapat bekerja dengan baik pada data *noisy*.

Adam mengadopsi keuntungan yang ada pada AdaGrad dan RMSProp. Dengan menggabungkan properti pada AdaGrad dan RMSProp, algoritme Adam menyediakan optimisasi yang dapat menangani masalah gradien renggang dan data *noisy*. Pertama, Adam mengadaptasi parameter *learning rate* berdasarkan rata-rata momentum pertama (seperti RMSProp), kedua digunakan juga rata-rata momentum kedua dari gradien. Algoritme bekerja dengan menghitung rata-rata pergerakan eksponensial dari gradien dan gradien kuadrat. Algoritme optimisasi Adam untuk memperbarui *bias* dan bobot dapat dilihat pada gambar 2.11.

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
  $v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Gambar 2.11 Algoritme optimisasi Adam [33]

2.1.6 *Autoencoder*

Autoencoder merupakan arsitektur *neural network* yang *unsupervised* dan digunakan untuk menemukan representasi data masukan yang telah dikompresi. Data masukan dapat berupa teks, gambar, suara, dan video. *Autoencoder* tidak

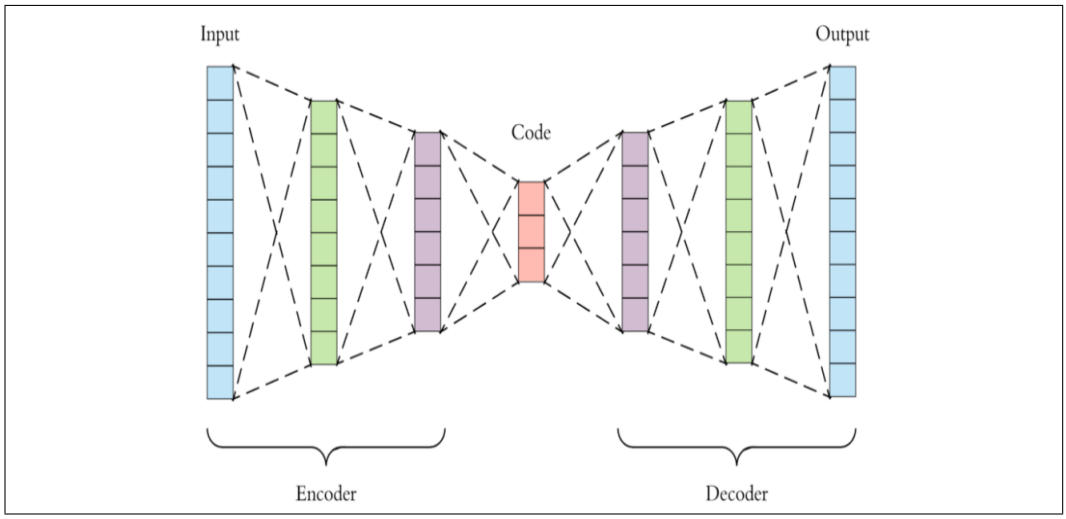
memerlukan data berlabel sebagai data belajar, namun dapat dengan sendirinya menemukan pola atau representasi baru dari sebuah dataset [16]. Sebagai contoh, *denoising autoencoder* merupakan *neural network* yang berusaha menemukan kode dan digunakan untuk mengubah data yang *noisy* menjadi data yang bersih. Dalam bentuk yang sederhana, *autoencoder* akan mempelajari representasi atau kode dengan cara menyalin masukan ke keluaran. Pertama, *autoencoder* melakukan tahap *encode* terhadap data masukan ke dalam bentuk dimensi yang lebih rendah yang biasanya berbentuk vektor. Proses *encode* tersebut akan mendekati struktur tersembunyi pada data yang diberikan, biasanya disebut sebagai representasi, kode, atau vektor. Setelah itu, proses berlanjut ke tahap *decode* yaitu mengembalikan data yang telah dikompresi menjadi data masukan asli. Hasil dari proses *decode* tidak dapat mewakili data masukan asli secara utuh, maka dari itu perbedaan antara masukan dan keluaran dapat diukur menggunakan *loss function*.

Secara umum *autoencoder* merupakan *dimensionality reduction* atau algoritme kompresi yang memiliki beberapa properti penting, yaitu:

1. Data-specific, artinya *autoencoder* hanya dapat mengompresi data secara berarti berdasarkan data latih yang diberikan.
2. *Lossy*, artinya keluaran dari *autoencoder* tidak akan sama persis dengan masukan, akan ada informasi yang hilang akibat proses kompresi.
3. *Unsupervised*, artinya untuk melatih *autoencoder* tidak diperlukan data latih yang berlabel. Proses *autoencoder* akan menghasilkan labelnya sendiri dari data pelatihan.

Autoencoder memiliki tiga *layer* bagian, seperti pada gambar 2.12, yaitu:

1. *Encoder*: berfungsi untuk melakukan kompresi terhadap data masukan dan menghasilkan *layer code*.
2. *Code*: merupakan representasi data masukan dengan dimensi yang lebih kecil. Jumlah *neuron* dalam *layer* kode dapat berbeda sesuai dengan *hyperparameter* yang diatur sebelum proses pelatihan.
3. *Decoder*: berfungsi untuk membangun kembali data masukan menggunakan *layer code*.



Gambar 2.12 Arsitektur *Autoencoder* [21]

Pada gambar 2.12 merupakan contoh arsitektur *autoencoder* yang terdiri dari *encoder*, *code*, dan *decoder*. Pada bagian *encoder*, terdapat *layer* masukan dan dua buah *hidden layer*. Di awal terdapat sepuluh sinyal masukan yang dikompresi menjadi delapan *neuron* pada *hidden layer* pertama, kemudian dikompresi kembali menjadi enam *neuron* pada *hidden layer* kedua. Pada *code layer* merupakan representasi data masukan yang paling padat dalam tiga *neuron*. Kemudian pada bagian *decoder*, data paling padat dari *layer code* direkonstruksi kembali menjadi data masukan. Jumlah *hidden layer* dan *neuron* pada *decoder* sama dengan *encoder*, namun dengan proses yang berlawanan. Dalam *autoencoder* terdapat empat *hyperparameter* yang diatur sebelum dilakukan proses latihan [21], yaitu:

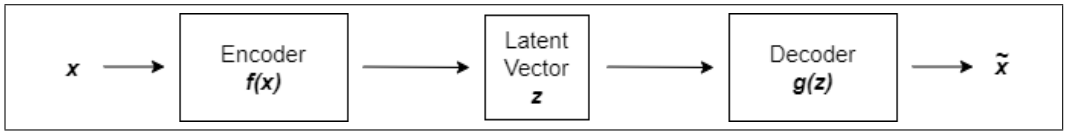
1. *Code size*: jumlah *neuron* pada *middle layer* atau disebut juga dengan *bottleneck layer*. Ukuran yang lebih kecil menghasilkan lebih banyak kompresi, sehingga lebih banyak informasi yang hilang.

2. Jumlah *hidden layer*: *autoencoder* dapat memiliki kedalaman *hidden layer* berapapun. Jumlah *hidden layer* tidak memiliki rumus atau ketentuan khusus, namun biasanya menggunakan 1 atau 2 *hidden layer* sudah memberikan hasil yang sangat baik [17]. Arsitektur *autoencoder* yang memiliki banyak *hidden layer* dapat disebut dengan *stacked autoencoder*. Jumlah *hidden layer encoder* dan *decoder* dapat diatur dengan *hyperparameter* ini.

3. Jumlah *neuron* per *hidden layer*: dalam menentukan jumlah *neuron* pada setiap *hidden layer* tidak memiliki rumus atau ketentuan khusus, untuk mendapatkan jumlah *neuron* yang optimal diperlukan percobaan berulang

[17]. Jumlah *neuron* per *hidden layer* akan berkurang pada setiap *layer encoder*, dan akan bertambah pada setiap *layer decoder*. Aturan jumlah struktur *hidden layer encoder* dan *decoder* harus simetris.

4. *Loss function*: dapat dihitung menggunakan *mean squared error* (MSE) atau *binary crossentropy*. Jika keluaran dalam jarak 0 hingga 1 maka dapat menggunakan *binary crossentropy*, selain itu dapat menggunakan MSE.



Gambar 2.13 Operator pada *Autoencoder* [16]

Gambar 2.13 di atas merupakan *Autoencoder* yang terdiri dari dua operator, yaitu:

1. *Encoder*, berfungsi untuk mengubah masukan, x, menjadi vektor laten yang berdimensi lebih rendah seperti pada persamaan 2.13. Vektor laten yang berdimensi rendah membuat *encoder* dapat mempelajari hanya fitur-fitur terpenting dari data masukan.

$$z = f(x) \tag{2.13}$$

Keterangan:

z	: vektor laten.
f	: fungsi encoder.
x	: sinyal masukan.

2. *Decoder*, berfungsi untuk mengembalikan masukan dari vektor laten yang berdimensi rendah menjadi data masukan awal, x.

$$g(z) = \tilde{x} \tag{2.14}$$

Keterangan:

g	: fungsi decoder.
z	: vektor laten.
$\tilde{x}$	: keluaran decoder (masukan yang direkonstruksi).

Encoder dan *decoder* merupakan fungsi non-linear. Dimensi z adalah ukuran jumlah fitur penting yang dapat diwakili. Dimensi tersebut biasanya lebih kecil dari dimensi masukan untuk efisiensi dan membatasi proses untuk mempelajari hanya properti yang paling menonjol. *Loss function*, $\mathcal{L}(x,\tilde{x})$, mengukur perbedaan masukan, x, dengan keluaran, $\tilde{x}$, atau disebut juga sebagai *reconstruction error*. *Mean Squared Error* merupakan contoh dari *loss function*, rumus dari MSE dapat dilihat pada persamaan 2.15.

$$\mathcal{L}(x,\tilde{x}) = MSE = \frac{1}{m}\sum_{i=1}^{i=m}(x_i - \tilde{x}_i)^2 \tag{2.15}$$

Keterangan:

$\mathcal{L}$	: *loss function*.
i	: jumlah iterasi.
x_i	: data masukan ke i.
$\tilde{x}_i$	: data keluaran ke i.
m	: dimensi keluaran.

Secara garis besar penerapan *autoencoder* digunakan untuk menghitung nilai *reconstruction* dari hasil *encoding* dan *decoding*. Gambar 2.14 merupakan algoritme *autoencoder*:

Steps	Processes
Step 1: Prepare the input date	Input Matrix X // input dataset Parameter of the matrix // parameter (w, b_x, b_h) where: w : Weight between layers, bx Encoder's parameters , b_h Decoder's Parameters
Step 2: initial Variables	h ← null // vector for hidden layer $\hat{X}$ ← null // Reconstructed x L ← null // vector for Loss Function l ← batch number i ← 0

Step 3: loop statement	While i < l do
	// Encoder function maps an input X to hidden representation h:
	$h = f(p[i].w + p[i].b_x)$
	/* Decoder function maps hidden representation h back to a Reconstruction $\hat{X}$:*/
	$\hat{X} = g(p[i].w^T + p[i].b_x)$
	/*For nonlinear reconstruction, the reconstruction loss is generally from cross-entropy :*/
	$L = -sum(x * \log(\hat{X}) + (1-x) * \log(1-\hat{X}))$
	/* For linear reconstruction, the reconstruction loss is generally from the squared error:*/
	$L = sum(X-\hat{X})^2$
	$\theta[i] = \overset{\min}{p} L(X-\hat{X})$
	End while
	Return θ

Step 4: output	θ ← <null matrix>//objective function /*Training an autoencoder involves finding parameters = (W,b_x , b_h) that minimize the reconstruction loss on the given dataset X and the objective function*/

Gambar 2.14 Algoritme *autoencoder* [6]

2.1.6.1 *Autoencoder* Untuk Klasifikasi

Dalam penerapannya *autoencoder* dapat digunakan untuk klasifikasi menggunakan data berlabel yang tidak seimbang. Penerapan *autoencoder* untuk klasifikasi serupa dengan deteksi anomali [26]. Dalam deteksi anomali, model hanya mempelajari pola dari data yang normal (selabel). Data yang tidak mengikuti pola yang telah dipelajari model maka diklasifikasikan sebagai anomali. Untuk klasifikasi biner dan menggunakan data yang tidak seimbang, dapat diterapkan deteksi anomali menggunakan *autoencoder*.

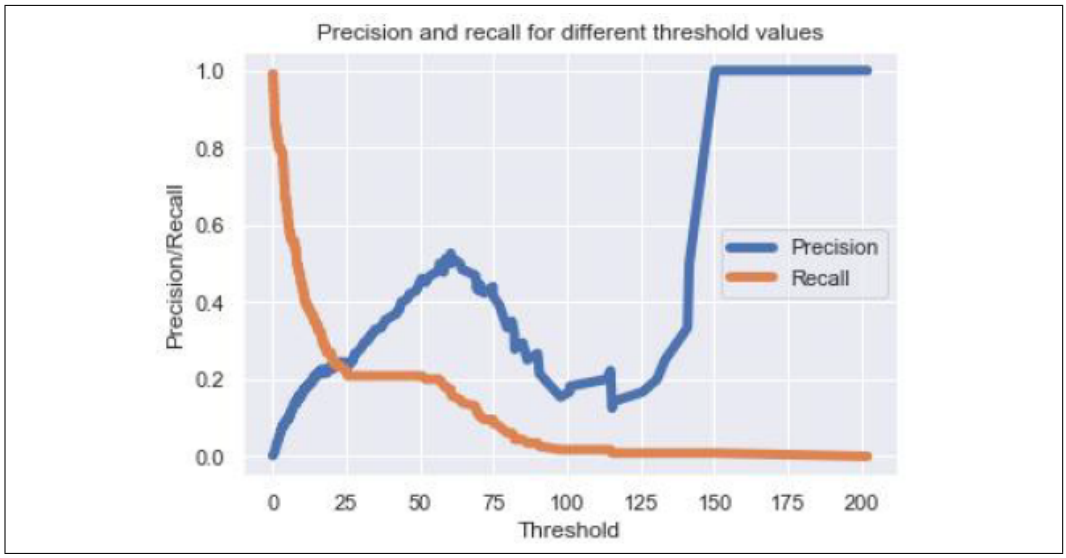
Algoritme *autoencoder* untuk mendeteksi anomali

1. Membagi data menjadi dua bagian, yaitu: label positif dan label negatif. (contoh: label *fraud* (positif) dan label non-*fraud* (negatif))
2. Menentukan kelas mayoritas antara label positif atau label negatif (contoh: label negatif sebagai kelas mayoritas).
3. Label negatif diasumsikan sebagai data normal, sementara label positif

diasumsikan sebagai anomali.

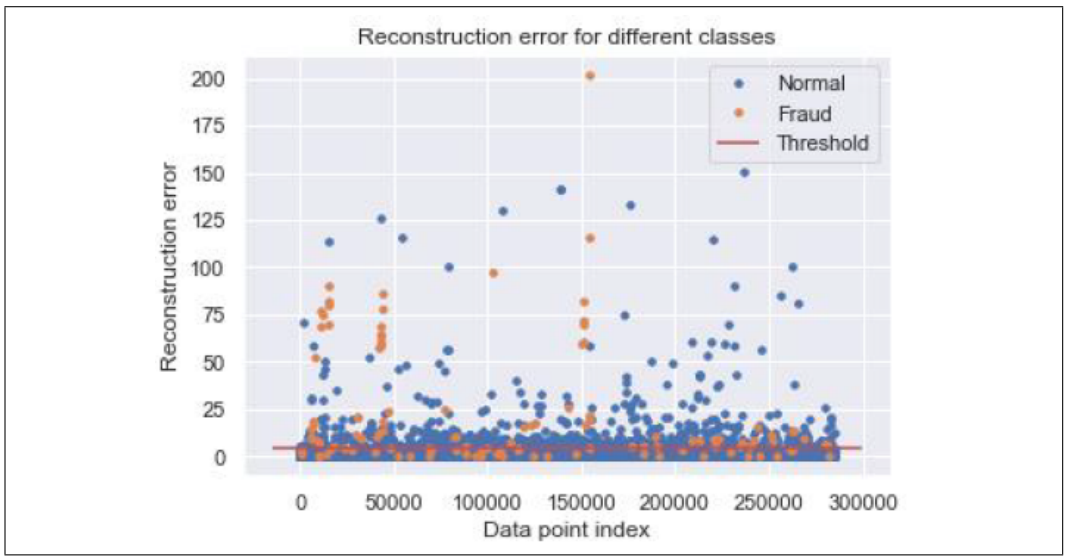
4. Label positif diabaikan selama proses pembelajaran, sementara *autoencoder* melakukan proses belajar hanya menggunakan data dengan label negatif.
5. *Autoencoder* belajar berdasarkan fitur dari data normal.
6. Dalam proses belajar, *autoencoder* yang bagus akan mempelajari data baru dengan hasil distribusi atau pola yang sama.
7. Hasil dari proses belajar adalah *reconstruction error* yang kecil.
8. Setelah itu, proses pengujian dilakukan menggunakan data dengan label positif (anomali).
9. Hasil pengujian dengan label positif akan memunculkan *reconstruction error* yang besar pada setiap data poin.
10. Data dengan *reconstruction error* yang besar akan dideteksi sebagai anomali.

Pada tahap klasifikasi, *reconstruction error* yang didapatkan dari proses belajar *autoencoder* dapat digunakan untuk mendeteksi *fraud*. *Reconstruction error* dapat dihitung menggunakan rumus *Mean Squared Error* (MSE) pada persamaan 2.15. Seperti yang telah dijelaskan sebelumnya, jika *reconstruction error* besar, maka data poin tersebut dapat diklasifikasikan sebagai anomali, maka dari itu sebuah nilai ambang atau *threshold* harus ditentukan. Nilai *threshold* digunakan untuk membatasi *reconstruction error* antara data normal dan data anomali. Dalam menentukan nilai *threshold*, dapat digunakan grafik *tradeoff* antara *precision* dan *recall* seperti pada gambar 2.15. Dan garis *threshold* yang telah ditentukan dapat dilihat pada pemetaannya terhadap *reconstruction error* pada setiap data poin seperti pada gambar 2.16.



Gambar 2.15 Grafik *Tradeoff* antara *Precision* dan *Recall* [6]

Pada gambar grafik diatas dapat dilihat bahwa dalam menentukan nilai *threshold* terdapat *tradeoff* antara *precision* dan *recall*. Apabila menginginkan *recall* yang tinggi namun *precision* rendah, maka dapat menggunakan nilai *threshold* yang mendekati nilai 0. Sebaliknya, apabila menginginkan *precision* yang tinggi namun *recall* rendah, maka dapat menggunakan nilai *threshold* diatas 150. Oleh karena itu, penentuan nilai *threshold* bergantung pada kasus yang sedang diuji untuk memilih nilai yang lebih tinggi antara *precision* atau *recall*.



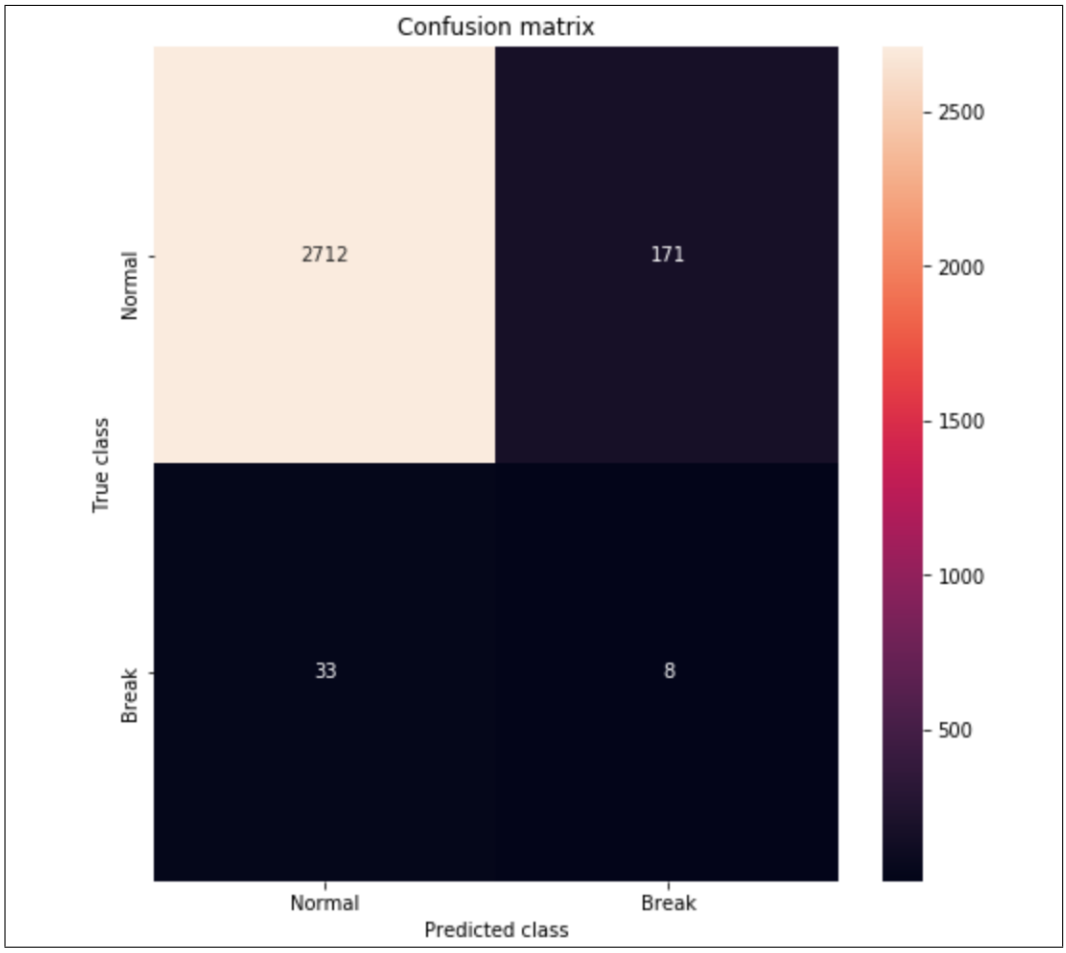
Gambar 2.16 Garis *Threshold* terhadap *Reconstruction Error* untuk Setiap Data Poin [6]

Pada gambar di atas garis *threshold* digunakan untuk menentukan batas anomali. Setiap data poin yang berada dibawah garis *threshold* akan diklasifikasikan sebagai

data normal dan data poin diatas garis *threshold* akan diklasifikasikan sebagai data anomali. Setelah nilai *threshold* ditentukan maka dapat dilihat *confusion matrix* untuk mengevaluasi model. Jumlah TP, TN, FP, dan FN dapat ditentukan dengan melihat gambar di atas, dengan ditentukan nilai *threshold* data poin yang berada dibawah garis menunjukkan TN dan FN, sedangkan data poin yang berada di atas garis menunjukkan TP dan FP.

2.1.6.2 Evaluasi *Autoencoder*

Pada penelitian M. A. Al-Shabi [6] digunakan *confusion matrix* untuk mengevaluasi model untuk mendeteksi penipuan kartu kredit. *Confusion matrix* tersebut didapatkan dari model *autoencoder* yang menerapkan deteksi anomali. Seperti yang telah dijelaskan diatas, deteksi anomali menggunakan nilai *reconstruction error* untuk menghitung perbedaan hasil *encoding* dan *decoding* antara data masukkan dan keluaran. Setelah itu digunakan nilai *threshold* untuk membatasi data normal dan data anomali lalu diterapkan *confusion matrix*. Dengan menggunakan *confusion matrix* maka dapat diukur *precision* dan *recall* untuk mengetahui kelas mana yang sering diklasifikasikan salah [22]. Contoh hasil evaluasi yang menggunakan *confusion matrix* dengan model yang dilatih menggunakan *autoencoder* dapat dilihat pada gambar 2.17. Setelah mendapat *confusion matrix* maka dapat dilakukan perhitungan akurasi, *precision*, *recall*, dan *f1 measure*.

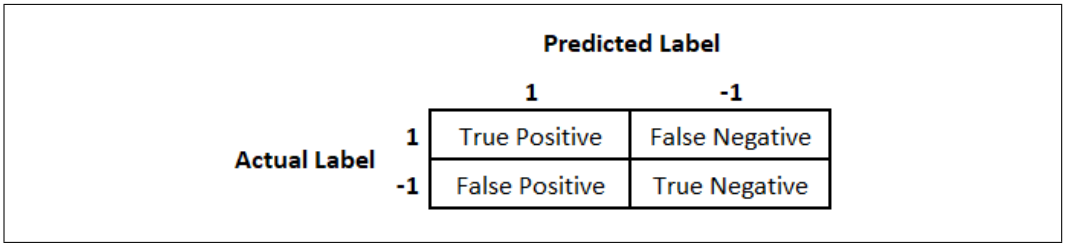


Gambar 2.17 Contoh Evaluasi *Autoencoder* [26]

Gambar di atas menunjukan *confusion matrix* sebagai evaluasi hasil klasifikasi menggunakan *autoencoder*. Dari *confusion matrix* tersebut dapat disimpulkan jumlah *true positive* (TP) sebanyak 8 data, *true negative* (TN) sebanyak 2.712 data, *false positive* (FP) sebanyak 171 data, dan *false negative* (FN) sebanyak 33 data.

2.1.7 *Confusion Matrix*

Confusion Matrix merupakan metode pengukuran untuk mengevaluasi hasil klasifikasi. Dengan melakukan klasifikasi sebanyak C kelas, dihasilkan *confusion matrix* M berukuran $C \times C$, di mana elemen M_{ij} dalam matriks menunjukkan jumlah sampel yang salah diklasifikasikan, sementara M_{ii} adalah jumlah sampel yang hasil klasifikasinya adalah benar. *Confusion matrix* pada gambar 2.18 digunakan pada kasus klasifikasi biner sehingga membentuk matriks berukuran 2×2 [10].



Gambar 2.18 *Confusion Matrix* untuk Klasifikasi Biner [10]

Kolom M_{11} pada matriks menunjukkan jumlah data yang pada kenyataannya adalah positif dan diklasifikasikan sebagai positif, sehingga disebut data *true-positive* (TP). Kolom M_{12} menunjukkan jumlah data yang pada kenyataannya adalah negatif tetapi diklasifikasikan sebagai positif, sehingga disebut data *false-positive* (FP). Kolom M_{21} menunjukkan jumlah data yang pada kenyataannya adalah positif tetapi diklasifikasikan sebagai negatif, sehingga disebut data *false-negative* (FN). Dan kolom M_{22} menunjukkan jumlah data yang kenyataannya adalah negatif dan diklasifikasikan sebagai negatif, sehingga disebut *true-negative* (TN). Untuk menghitung akurasi dapat digunakan persamaan 2.16. Hasil akurasi yang mendekati nilai 1 akan semakin baik sebaliknya jika mendekati nilai 0 maka hasilnya buruk.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.16}$$

Lalu perhitungan *precision* yang merupakan perbandingan dari hasil positif dapat dihitung dengan persamaan 2.17.

$$Precision = \frac{TP}{TP+FP} \tag{2.17}$$

Dan perhitungan *recall* atau disebut juga sebagai sensitivitas dapat dihitung dengan persamaan 2.18.

$$Recall = \frac{TP}{TP+FN} \tag{2.18}$$

Dan perhitungan *f1 measure* atau nilai rata-rata dari *precision* dan *recall* dapat dihitung dengan persamaan 2.19.

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{2.19}$$

2.1.8 *Imbalanced Class*

Imbalanced class atau kelas tidak seimbang merupakan masalah ketika jumlah observasi dari sebuah kelas lebih tinggi dibandingkan kelas lainnya [25]. Dalam pembelajaran mesin, kelas tidak seimbang sering ditemukan pada kasus klasifikasi, seperti: *fraud detection*, *spam filtering*, *disease screening*, dan lain-lain. Dalam klasifikasi dua kelas, data dikatakan seimbang apabila memiliki rasio kurang lebih 50% pada masing-masing kelas. Kelas tidak seimbang menyebabkan model pembelajaran mesin yang dibangun menjadi tidak akurat karena hanya dapat memprediksi kelas mayoritas dan gagal untuk memprediksi kelas minoritas. Sebagai contoh pada kasus *fraud detection*, kelas minoritas yaitu transaksi *fraud* hanya terdapat 400 dibandingkan dengan kelas mayoritas yaitu transaksi non-*fraud* sebanyak 90.000. Hal tersebut menggambarkan kelas tidak seimbang karena rasio persentase transaksi *fraud* terhadap transaksi non-*fraud* hanya sebesar 0,0044%.

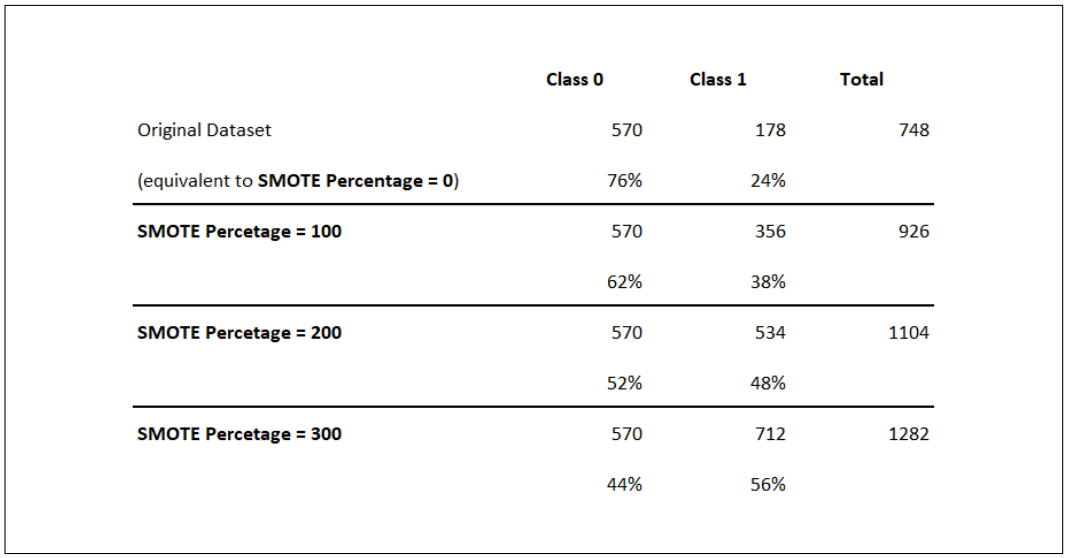
Gambar 2.19 Contoh Grafik Kelas Tidak Seimbang [25]

Pada gambar 2.19 dapat dilihat grafik jumlah kelas *no-fraud* yang sangat tinggi dibandingkan kelas *fraud*. Untuk mengatasi kelas tidak seimbang maka dapat diterapkan teknik *resampling*. Teknik *resampling* dibagi menjadi dua, yaitu: *oversampling* dan *undersampling*. Teknik *oversampling* merupakan cara untuk menduplikasi kelas minoritas agar mendekati jumlah kelas mayoritas. *Oversampling* merupakan teknik yang tepat jika diterapkan pada dataset dengan jumlah yang tidak terlalu besar. Sedangkan teknik *undersampling* merupakan cara untuk membuang sejumlah observasi dari kelas mayoritas hingga jumlah kelas mayoritas dan kelas minoritas seimbang. *Undersampling* tepat jika diterapkan pada dataset dalam jumlah yang besar, namun dengan catatan melakukan *undersampling* dapat membuang informasi yang berharga untuk pembelajaran model [25].

2.1.8.1 *Synthetic Minority Oversampling Technique* (SMOTE)

Synthetic Minority Oversampling Technique atau disebut juga SMOTE merupakan algoritme *oversampling* yang bertujuan untuk membuat sampel berlebih untuk kelas minoritas. SMOTE merupakan teknik statistik untuk meningkatkan jumlah kelas pada dataset secara seimbang. SMOTE merupakan cara yang lebih baik untuk meningkatkan jumlah kelas yang sedikit/jarang daripada sekedar menduplikasi kasus yang ada [23]. SMOTE digunakan untuk menghindari dataset yang tidak seimbang *(imbalaced)*, yaitu kategori pada kelas target yang jumlahnya sangat sedikit sehingga kelas tersebut tidak terwakili. Cara kerja SMOTE bukan hanya sekedar menduplikasi kelas minoritas yang ada, namun mengambil sampel dari ruang fitur untuk setiap kelas target dan tetangga terdekatnya. Hal tersebut akan menghasilkan sampel baru yang menggabungkan fitur kelas target dengan fitur tetangganya. Pendekatan ini mengingkatkan fitur untuk setiap kelas dan membuat sampel menjadi lebih umum *(general)*.

Masukan dari SMOTE dataset secara keseluruhan, namun hanya meningkatkan persentase kelas minoritas, dan tidak mengubah jumlah dari kelas mayoritas. Sebagai contoh jika dataset tidak seimbang memiliki target kelas A sebesar 1% dan kelas B sebesar 99%, maka untuk meningkatkan persentase kelas minoritas sebanyak dua kali dari persentase sebelumnya digunakan persentase SMOTE = 200. Penerapan SMOTE dapat dilihat pada gambar 2.20, dataset asli memiliki persentase data yang tidak seimbang antara kelas 0 (76%) sedangkan kelas 1 (24%).



	Class 0	Class 1	Total
Original Dataset	570	178	748
(equivalent to **SMOTE Percentage = 0**)	76%	24%	
SMOTE Percetage = 100	570	356	926
	62%	38%	
SMOTE Percetage = 200	570	534	1104
	52%	48%	
SMOTE Percetage = 300	570	712	1282
	44%	56%	

Gambar 2.20 Penerapan SMOTE [23]

Penggunaan persentase pada SMOTE bergantung pada setiap dataset. Meningkatkan jumlah kelas menggunakan SMOTE tidak menjamin akan menghasilkan akurasi model yang bagus. Oleh karena itu, diperlukan percobaan untuk menguji persentase, fitur yang berbeda, dan jumlah tetangga terdekat dan bagaimana pengaruhnya terhadap model yang dibangun [23]. Tabel dibawah ini merupakan daftar masukan yang diperlukan SMOTE (tabel 2.1), parameter SMOTE (tabel 2.2), dan keluaran yang dihasilkan oleh SMOTE (tabel 2.3).

Tabel 2.1 Daftar Masukan SMOTE [23]

Nama	**Tipe**	**Deskripsi**
Sample	Data Table	Sampel dataset yang digunakan

Tabel 2.2 Daftar Parameter SMOTE [23]

Nama	***Range***	**Tipe**	**Default**	**Deskripsi**
Persentase SMOTE	≥ 0	Integer	100	Jumlah *oversampling* dalam kelipatan 100

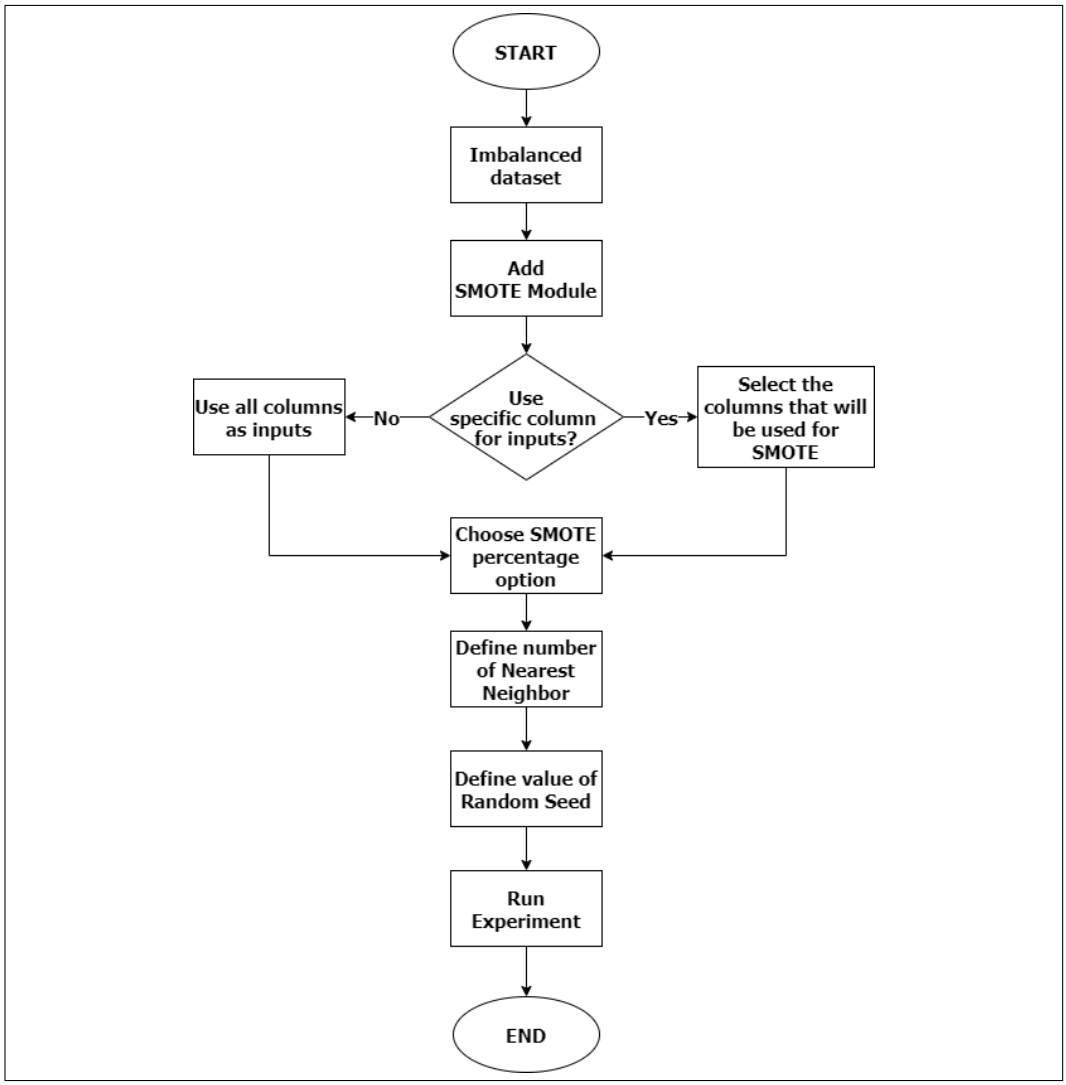
Tabel 2.2 Daftar Parameter SMOTE (Lanjutan)

Nama	***Range***	**Tipe**	**Default**	**Deskripsi**
Jumlah tetangga terdekat	≥ 1	Integer	1	Jumlah tetangga terdekat untuk menentukan fitur dari sampel yang baru
Random seed	*Any*	Integer	0	*Seed* untuk generator nomor acak

Tabel 2.3 Daftar Keluaran SMOTE [23]

Nama	**Tipe**	**Deskripsi**
Tabel	Data Table	Tabel data yang berisi sampel asli dan ditambahkan jumlah sampel dari hasil SMOTE. Jumlah dari sampel baru dapat dihitung dengan rumus $(\textit{Persentase SMOTE} \div 100) \times T$, dimana T adalah jumlah sampel kelas minoritas.

Pada gambar 2.21 merupakan *flowchart* konfigurasi SMOTE saat digunakan pada dataset tidak seimbang.



Gambar 2.21 Flowchart Konfigurasi SMOTE [23]

2.1.9 Pustaka Python

Pada bagian ini akan dijelaskan mengenai pustaka atau *library* yang digunakan dalam penelitian.

2.1.9.1 Pandas

Dalam pemrogramman komputer, Pandas merupakan pustaka perangkat lunak yang digunakan dalam pemrogramman Python untuk manipulasi dan analisis data.

Tabel 2.4 Daftar Metode yang Digunakan

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
1	read_csv	file path, file name	string, string	data frame	Membaca data dalam bentuk file .csv dan merepresentasikan dalam bentuk tabel data frame.
2	DataFrame	subset	array	data frame	Membuat representasi data masukan menjadi array 2 dimensi atau tabel dengan baris dan kolom.
3	shape	subset	data frame	data frame size (row, col)	Memberikan informasi dimensi dari sebuah DataFrame.
4	info	subset	data frame	string	Menampilkan rangkuman dari isi tabel yang terdiri dari jumlah baris, nama kolom, jumlah data null, dan tipe data.
5	count	subset, axis	data frame, integer	integer	Menampilkan jumlah data yang tidak null berdasarkan kolom pada DataFrame.
6	drop	label, axis	string, integer	data frame	Melakukan drop atau menghapus label spesifik dari sebuah baris atau kolom.
7	duplicated	subset	data frame	boolean	Memberikan nilai boolean (true atau false) berdasarkan ada atau tidaknya baris yang duplikat.

Tabel 2.4 Daftar Metode yang Digunakan (Lanjutan)

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
8	drop_duplicates	subset	data frame	data frame	Melakukan drop atau menghapus baris yang duplikat pada DataFrame.
9	apply	function, axis	string, integer	data frame	Menerapkan sebuah fungsi pada baris atau kolom pada DataFrame.
10	loc	subset, label	string, integer	data frame	Menampilkan DataFrame dengan parameter label atau array boolean yang spesifik.
11	to_datetime	array	integer, string	array	Melakukan konversi input menjadi datetime.
12	corr	method	string	data frame	Menghitung korelasi berpasangan antar kolom.

2.1.9.2 NumPy

Dalam pemrogramman komputer, NumPy merupakan pustaka untuk pemrogramman Python yang mendukung array dan matriks multi-dimensi dalam ukuran besar. Pustaka NumPy menawarkan fungsi matematika tingkat tinggi untuk beroperasi pada array.

Tabel 2.5 Daftar Metode yang Digunakan

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		

Tabel 2.5 Daftar Metode yang Digunakan (Lanjutan)

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
1	mean	array, axis	string, integer	float	Menghitung nilai rata-rata aritmatik dari sumbu input yang diberikan.
2	std	array, axis	string, integer	float	Menghitung nilai standar deviasi (ukuran penyebaran distribusi) dari input yang diberikan.
3	median	array, axis	string, integer	float	Menghitung nilai median (nilai tengah) dari input yang diberikan.
4	abs	value	integer, float	integer, float	Menghitung nilai absolut dari input x yang diberikan.
5	power	value 1, value 2	integer, float	integer, float	Menghitung elemen x1 yang dipangkatkan dengan elemen x2.
6	sum	array, axis	string, integer	integer, float	Menghitung jumlah dari input yang diberikan.
7	size	array	string	integer	Menghitung ukuran array input yang diberikan.
8	random.rand	dimension (d1, d2,..., dn)	integer	integer, float	Mengeluarkan nilai random dengan input dimensi yang diberikan.

Tabel 2.5 Daftar Metode yang Digunakan (Lanjutan)

No	Metode	Masukan		Luaran	Keterangan
		Parameter	**Variabel**		
9	random.uniform	low, high, size	integer, integer, integer	integer, float	Mengeluarkan nilai random dari distribusi uniform (seragam) dengan nilai low dan high serta ukuran array.

2.1.9.3 Matplotlib

Matplotlib merupakan pustaka Python yang digunakan untuk visualisasi data seperti pemetaan grafik. Visualisasi dari Matplotlib adalah sebuah gambar grafik yang terdapat satu sumbu atau lebih. Setiap sumbu memiliki sumbu horizontal (x) dan sumbu vertikal (y) dan data direpresentasikan menjadi warna dan *glyphs* seperti: *marker* (berbentuk lingkaran), garis, atau poligon.

Tabel 2.6 Daftar Metode yang Digunakan

No	Metode	Masukan		Luaran	Keterangan
		Parameter	**Variabel**		
1	figure	num, figsize	integer, integer	figure	Membuat figur baru atau mengaktifkan figur yang sudah ada sebelumnya.
2	plot	data, kind, figsize, labels, colors	array, string, integer, string, string	graph image	Membuat plot grafik dari input data.
3	legend	labels	string	graph legend	Membuat legenda pada grafik sesuai keterangan garis yang diberikan.

Tabel 2.6 Daftar Metode yang Digunakan (Lanjutan)

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
4	xlabel	xlabel, location	string, string	label x	Memberikan label keterangan pada sumbu x.
5	ylabel	ylabel, location	string, string	label y	Memberikan label keterangan pada sumbu y.
6	show	-	-	figure	Menampilkan plot grafik yang telah diatur sebelumnya.

2.1.9.4 Seaborn

Seaborn adalah pustaka yang merupakan pengembangan dari pustaka Matplotlib. Fungsi dari Seaborn yaitu untuk visualisasi data yang fleksibel. Seaborn diklaim lebih bagus dalam melakukan visualisasi data dan rangkaian kode yang lebih mudah dibandingkan Matplotlib.

Tabel 2.7 Daftar Metode yang Digunakan

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
1	distplot	data, x, y, hue, kind	array, vector, vector, vector, string	graph image	Membuat plot distribusi data dari input yang diberikan.
2	heatmap	data, annot, cmap	array, boolean, string	matrix image	Membuat plot data dalam bentuk persegi sebagai matriks berkode warna.

2.1.9.5 Imbalanced-learn

Imbalanced-learn atau imblearn merupakan pustaka yang secara khusus dirancang untuk menangani dataset yang tidak seimbang. Beberapa metode yang disediakan,

seperti: *undersampling*, *oversampling*, dan SMOTE untuk menangani dan menghapus ketidakseimbangan dari kumpulan data.

Tabel 2.8 Daftar Metode yang Digunakan

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
1	SMOTE	ratio, random state, k-neighbors	string, integer, integer	data frame	Melakukan teknik oversampling menggunakan SMOTE.

2.1.9.6 Scikit-learn

Scikit-learn merupakan pustaka pembelajaran mesin yang digunakan dalam bahasa pemrogramman Python. Pustaka ini mendukung algoritme, seperti: klasifikasi, regresi, dan pengelompokan.

Tabel 2.9 Daftar Metode yang Digunakan

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
1	train_test_split	data, test size, train size, random state, shuffle	array, float, float, integer, boolean	array	Melakukan pembagian dataset dalam bentuk array atau matriks secara acak menjadi train dan test subsets.
2	confusion_ matrix	y_true, y_pred	array, array	confusion matrix	Menghitung confusion matrix untuk mengevaluasi akurasi dari sebuah klasifikasi. Pada klasifikasi biner maka akan mengeluarkan perhitungan dari TN, FN, TP, dan FP.

2.2 Tinjauan Studi

Pada bagian ini akan dijelaskan mengenai perbandingan dari beberapa penelitian terkait deteksi *click fraud* pada *mobile advertising*.

2.2.1 *State Of The Art*

Tabel 2.10 *State of the Art*

No	Penelitian	Metode	Hasil Penelitian
1	Thejas G.S, Kianoosh G. Boroojeni, Kshitij Chandna, Isha Bhatia, S.S Iyengar, and N.R Sunitha, "Deep Learning-based Model to Fight Against Ad Click Fraud", Proceedings of ACM Southeast Conference, USA, 2019.	1. *Deep Learning* dengan menggabungkan *Artificial Neural Network* (ANN), *Autoencoder*, dan *Semi-supervised Generative Adversarial Network* (GAN).	Mendeteksi *click fraud* menggunakan metode Deep Learning dengan menggabungkan ANN, *Auto-encoder*, dan GAN. *Auto-encoder* digunakan ditahap awal untuk mempelajari distribusi *human attribute* sehingga dapat mendeteksi manusia atau robot. Setelah itu ANN digunakan untuk mengklasifikasikan klik sebagai *fraud* atau bukan dan dengan bantuan dari GAN yang membuat sampel serangan *click fraud* agar meningkatkan akurasi dari ANN. Hasil yang diperoleh sistem mendapatkan akurasi 95%.

Tabel 2.10 *State Of The Art* (Lanjutan)

No	Penelitian	Metode	Hasil Penelitian
2	Zhaomin Chen, Chai Kiat Yeo, Bu Sung Lee, and Chiew Tong Lauw, "Autoencoder-based Network Anomaly Detection", Wireless Telecommunications Symposium (WTS), US, 2018.	1. *Autoencoder* konvensional 2. *Convolution Autoencoder*	Mendeteksi anomali pada jaringan menggunakan *Autoencoder* konvensional (AE) dan *Convolution Autoencoder* (CAE). AE dan CAE merupakan metode reduksi dimensi yang berusaha untuk menemukan *subspace* yang dapat mendeskripsikan seluruh data dengan kategori normal. Kemudian ketika sebuah data dengan kategori tidak normal (anomali) dimasukkan ke dalam model akan menghasilkan *reconstruction error* yang besar, maka data tersebut akan dibuang sebagai anomali. Kedua model tersebut memberikan hasil yang cukup baik karena dapat menemukan korelasi non-linear antara fitur-fitur yang ada. Hasil akurasi masing-masing model, yaitu: *Autoencoder* konvensional (95.85%) dan *Convolution Autoencoder* (96.87%).

Tabel 2.10 *State Of The Art* (Lanjutan)

No	Penelitian	Metode	Hasil Penelitian
3	M. A. Al-Shabi, "Credit Card Fraud Detection Using Autoencoder Model in Unbalanced Datasets", Journal of Advances in Mathematics and Computer Science, India, 2019.	1. *Autoencoder*	Mendeteksi penipuan kartu kredit yang datanya tidak seimbang (imbalanced) menggunakan *Autoencoder*. Model *Autoencoder* yang dibangun memiliki keuntungan karena dapat bekerja pada dataset yang tidak seimbang sehingga tidak membutuhkan proses *oversampling* seperti pada model klasifikasi lainnya. Karena *Autoencoder* mendeteksi anomali pada sebuah set data, maka nilai *threshold* pada model sangatlah berpengaruh untuk pendeteksian. Pada penelitian ini berfokus untuk menekan angka *false negative* (FN) yang berarti persentase *recall* harus tinggi. Hasil paling bagus adalah menggunakan *threshold* = 0.7 yang mendapat akurasi 80%, *recall* 91%, *precision* 9%, dan *f1 score* 4%.

2.3 Tinjauan Objek

Pada bagian ini akan dipaparkan objek yang digunakan dalam deteksi *click fraud* pada *mobile advertising*.

2.3.1 *Advertising*

Advertising atau periklanan merupakan komunikasi pemasaran yang menggunakan pesan non-pribadi atau bersponsor secara terbuka untuk mempromosikan atau menjual produk, layanan, atau ide. Sponsor dari periklanan biasanya adalah bisnis yang ingin mempromosikan produk atau layanan mereka. Periklanan dikomunikasikan melalui berbagai media, seperti: koran, majalah, televisi, radio, situs web, media sosial, dan lain-lain. Tayangan iklan berupaya untuk meningkatkan konsumsi produk atau layanan melalui *branding*, yang mengaitkan nama atau gambar produk dengan kualitas tertentu di benak konsumen.

2.3.2 Aktivitas Klik *User*

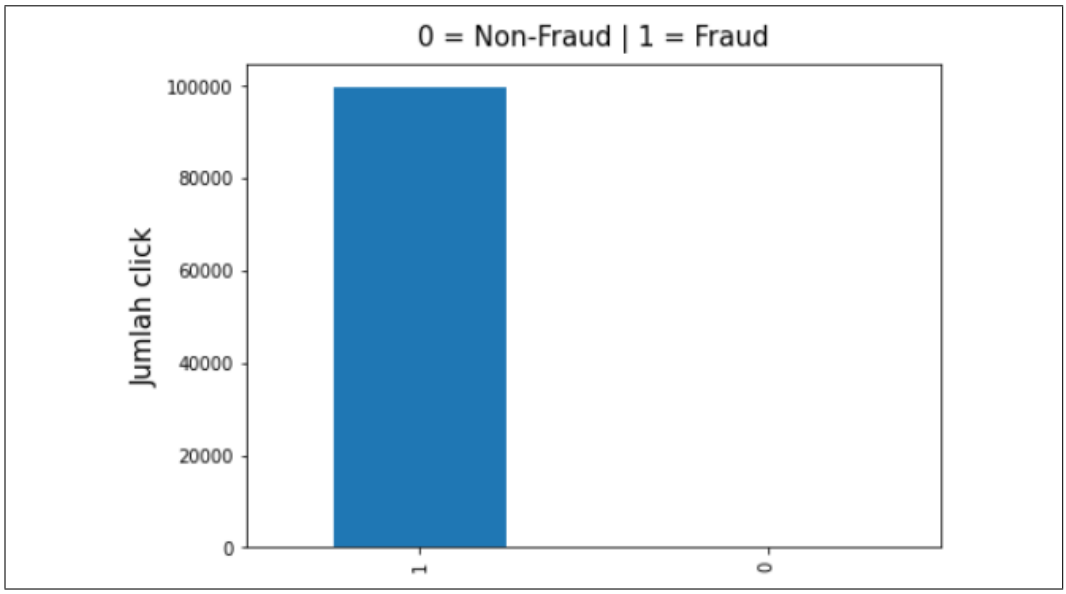
User adalah pengguna layanan teknologi informasi yang menggunakan perangkat komputasi. Seorang *user* dapat berinteraksi dengan layanan teknologi informasi, seperti: *click*, *swipe*, *pinch*, dan lain-lain. Interaksi *user* dengan layanan teknologi informasi dapat disebut sebagai aktivitas *user*. Layanan teknologi informasi tersebut salah satunya adalah tayangan iklan yang mengajak *user* untuk mengunduh sebuah aplikasi. Secara umum, interaksi yang paling sering dilakukan oleh *user* adalah klik. Data klik tersebut biasanya digunakan oleh perusahaan teknologi informasi untuk mengevaluasi perilaku *user* kemudian menentukan *user* mana yang melakukan penipuan klik. *User* yang disebut melakukan penipuan klik apabila melakukan klik terhadap sebuah iklan dan tidak mengunduh aplikasi dari iklan tersebut. Hal tersebut dikarenakan *user* tidak tertarik dengan apa yang ditawarkan iklan namun tetap melakukan klik untuk membuka iklan.

2.3.3 Dataset Penipuan Klik

Dataset yang digunakan pada penelitian ini diambil dari Kaggle. Dataset tersebut berisi data klik pada *mobile advertising* yang dikumpulkan oleh perusahaan *big data* asal China bernama TalkingData. Dataset berjumlah 100.000 data dan terdiri dari 8 fitur, yaitu: *ip*, *app*, *device*, *os*, *channel*, *click_time*, *attributed_time*, dan *is_attributed*. Label *is_attributed* merupakan kelas target yang mengindikasikan apakah *user* melakukan *download* aplikasi atau tidak, hal tersebut merupakan perilaku *user* yang tertarik atau tidak men-*download* aplikasi. Pada penelitian ini, penulis membuat label baru, yaitu *is_fraud* yang mengindikasikan secara langsung apakah *user* merupakan *fraud* atau non-*fraud*. Label *is_fraud* dibuat berdasarkan label sebelumnya yaitu *is_attributed*, dengan ketentuan sebagai berikut: jika *user*

men-*download* aplikasi (*is_attributed* = 1) maka *user* adalah non-*fraud* (*is_fraud* = 0), sedangkan jika *user* tidak men-*download* aplikasi (*is_attributed* = 0) maka *user* adalah *fraud* (*is_fraud* = 1).

Dataset memiliki kelas yang tidak seimbang antara *fraud* dan non-*fraud*. Hal tersebut dapat menyebabkan model menjadi tidak akurat dalam mendeteksi *click fraud*. Jumlah kelas tidak seimbang pada dataset dapat dilihat pada gambar 2.22.



Gambar 2.22 Kelas Tidak Seimbang Pada Dataset yang Digunakan

Pada gambar di atas dapat dilihat bahwa kelas *fraud* memiliki jumlah data yang lebih banyak dibandingkan dengan kelas non-*fraud*. Rasio jumlah kelas *fraud* adalah 99.772, sedangkan kelas non-*fraud* hanya 227. Oleh karena itu, dataset perlu dimodifikasi dengan menggunakan teknik *oversampling* untuk membuat kedua kelas menjadi seimbang. Teknik *oversampling* yang digunakan pada penelitian ini adalah SMOTE.

BAB 3 ANALISIS DAN PERANCANGAN SISTEM

Bab ini memaparkan analisis masalah yang diatasi berserta pendekatan dan alur kerja dari perangkat lunak yang dikembangkan, mengimplementasikan metode yang digunakan dan hasil yang akan ditampilkan.

3.1 Analisis Masalah

Pada bab 1 telah dijelaskan mengenai masalah yang dihadapi terkait *click fraud* pada *mobile advertising*. Pada penelitian ini, penulis akan menggunakan pendekatan *supervised* dan *unsupervised* untuk mendeteksi *click fraud*. Pendekatan *supervised* akan menggunakan Neural Network dan pendekatan *unsupervised* akan menggunakan Autoencoder. Di akhir penelitian akan dilakukan perbandingan hasil deteksi antara Neural Network dan Autoencoder.

Penelitian yang dilakukan oleh Thejas G. S. [3] adalah melakukan pendeteksian penipuan klik secara *real-time* menggunakan Deep Learning yang merupakan sistem gabungan antara Autoencoder, Neural Network, dan Generative Adversarial Network (GAN). Penggunaan Autoencoder dalam penelitian Thejas adalah untuk mendeteksi *user* sebagai *real user* atau *bot* berdasarkan informasi klik yang didapatkan ketika *user* melakukan klik pada sebuah iklan. Selanjutnya informasi klik *user* yang dideteksi sebagai *bot* akan dibuang, sedangkan yang dideteksi sebagai *real user* akan diteruskan ke Neural Network untuk diprediksi apakah *user* tersebut akan melakukan download aplikasi atau tidak. Di akhir, sistem Deep Learning tersebut digunakan untuk menyeleksi *user* yang akan diizinkan masuk ke dalam *website* iklan atau ditolak karena dideteksi melakukan penipuan klik.

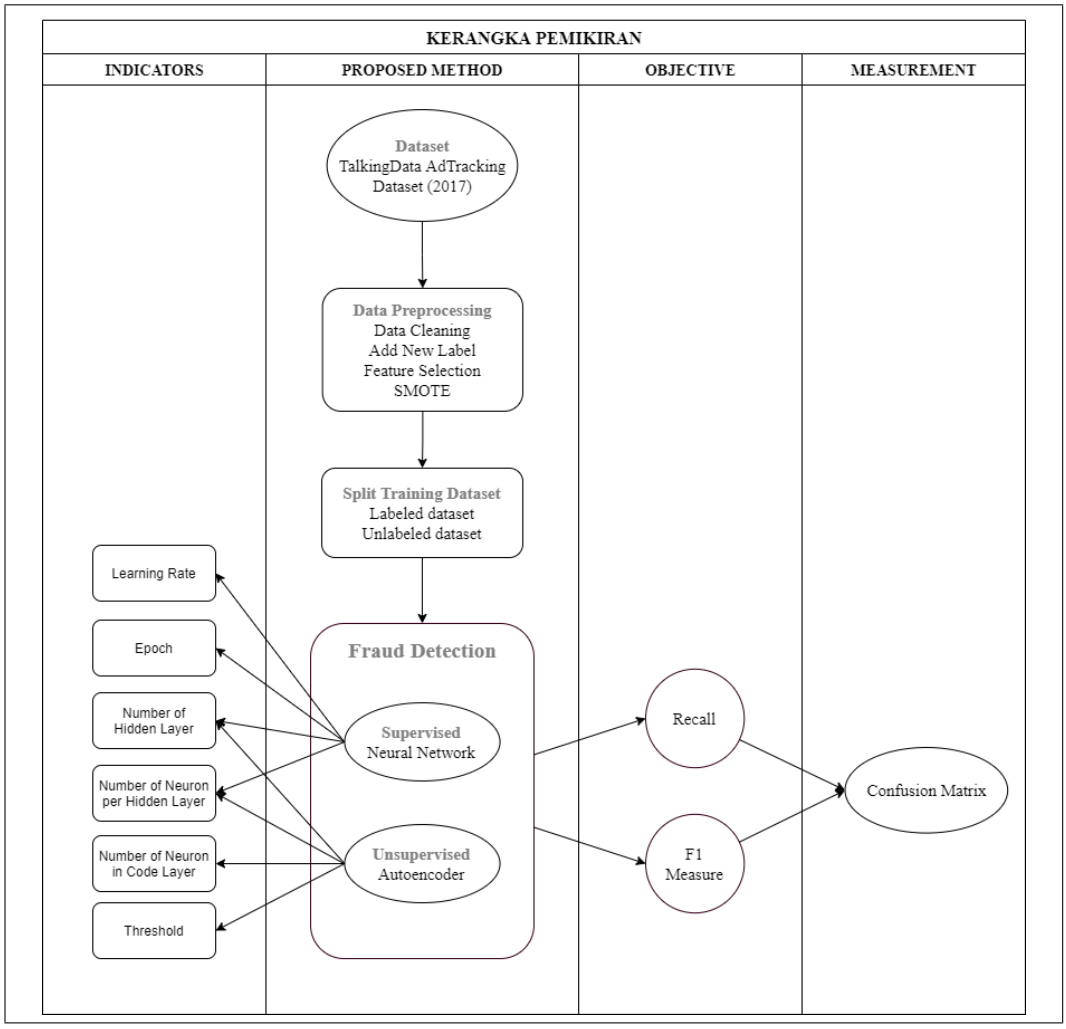
Pada penelitian yang dilakukan oleh penulis bertujuan untuk menerapkan Autoencoder untuk mendeteksi *click fraud* menggunakan informasi klik *user* dengan menerapkan teknik deteksi anomali dan bukan untuk mendeteksi *real user* atau *bot*. Model Neural Network dan Autoencoder dibangun secara terpisah untuk tujuan yang sama yaitu mendeteksi *click fraud*. Kemudian di akhir akan dibandingkan hasil deteksi antara kedua model tersebut. Sistem deteksi *click fraud* yang dibangun berjalan secara *offline* karena dalam kasus deteksi *click fraud* tidak murni *online*/*real-time*. Hal tersebut didukung dengan kasus *dispute* pada Google Adsense yang sewaktu-waktu dapat menarik komisi dari tayangan iklan jika terdeteksi adanya *click fraud*. Sistem pada Google Adsense membutuhkan waktu

untuk mendeteksi *click fraud* sehingga tidak berjalan secara *real-time*.

Dataset masukan untuk penelitian ini dibagi menjadi dua, yaitu dataset berlabel dan tidak berlabel. Pembagian tersebut dilakukan karena kedua pendekatan memiliki masukan yang berbeda. Pada pendekatan *supervised* yang menerapkan Neural Network akan menggunakan dataset berlabel, sedangkan pendekatan *unsupervised* yang menerapkan Autoencoder akan menggunakan dataset tidak berlabel. Penelitian ini akan membandingkan kedua pendekatan tersebut untuk mendeteksi *click fraud* dan dievaluasi menggunakan *recall* dan *f1 measure*.

3.2 Kerangka Pemikiran

Berikut ini adalah kerangka pemikiran dari metode yang diusulkan untuk melakukan deteksi *click fraud*.



Gambar 3.1 Kerangka Pemikiran

Pada gambar 3.1 adalah kerangka pemikiran yang telah disusun dalam bentuk

diagram:

1. Data *Preprocessing*: melakukan proses data *cleaning* untuk menghilangkan *missing values* dan *duplicate values*. Pada penelitian ini dilakukan penambahan label baru *(is_fraud)* berdasarkan label sebelumnya *(is_attributed)*. Label baru *(is_fraud)* mengindikasikan apakah *user* merupakan *fraud* atau bukan. Setelah itu dilakukan *feature selection* untuk memilih fitur paling signifikan yang akan digunakan pada proses belajar. Dan dilakukan juga teknik *oversampling* menggunakan *Synthetic Minority Oversampling Technique* (SMOTE) untuk mengatasi data yang tidak seimbang *(imbalanced)* antara label *fraud* dan non-*fraud*.

2. *Split Dataset*: melakukan proses pembagian terhadap dataset sebagai data belajar untuk pendekatan *supervised* dan *unsupervised*. Pendekatan *supervised* akan menggunakan dataset yang berlabel, sedangkan pendekatan *unsupervised* akan menggunakan dataset yang tidak berlabel.

3. *Fraud Detection*: melakukan deteksi *click fraud* menggunakan pendekatan *supervised* dengan Neural Network dan pendekatan *unsupervised* dengan Autoencoder.

4. *Indicators*: merupakan *hyperparameter* yang digunakan pada saat pembuatan model. Pada Neural Network akan menggunakan *learning rate*, *epoch*, *number of hidden layer*, dan *number of neuron per hidden layer*. Sedangkan Autoencoder akan menggunakan jumlah *number of hidden layer*, *number of neuron per hidden layer*, *number of neuron in code layer* dan *threshold*.

5. *Objective*: tujuan penelitian ini adalah membandingkan *recall* dan *f1 measure* dari kedua pendekatan *supervised* dan *unsupervised*.

6. *Measurement*: pengukuran akurasi menggunakan *confusion matrix*.

3.2.1 Penjelasan Indikator

Pada bagian ini akan dijelaskan mengenai indikator *hyperparameter* yang digunakan pada Neural Network dan Autoencoder. Berikut ini merupakan penjelasan indikator yang akan digunakan pada Neural Network:

1. *Learning rate*: digunakan untuk mengatur seberapa besar model melakukan pembaharuan nilai bobot. *Learning rate* merupakan *hyperparameter* yang

penting untuk menemukan model yang konvergen, jika nilainya terlalu kecil model dapat konvergen namun diperlukan *epoch* yang besar sehingga waktu proses belajar akan lama. Sementara *learning rate* yang terlalu besar menyebabkan model tidak dapat konvergen. Oleh karena itu, diperlukan observasi untuk menemukan jumlah *learning rate* yang tepat.

2. *Epoch*: merupakan jumlah dilakukannya algortime *feed-forward* dan *backpropagation* secara berulang. Semakin besar nilai *epoch* maka diharapkan model akan cukup belajar dari data yang diberikan sehingga dapat meningkatkan akurasi. Namun, jika nilai *epoch* terlalu tinggi, model akan terlalu banyak belajar yang menyebabkan *overfitting*. Oleh karena itu, diperlukan observasi untuk menemukan jumlah *epoch* yang tepat.

3. *Number of hidden layer*: merupakan jumlah *layer* tersembunyi yang berada diantara *input layer* dan *output layer*. *Hidden layer* merepresentasikan faktor atau variabel yang tersembunyi (tidak ada dalam dataset). Tujuan dari menambah jumlah *hidden layer* adalah membuat model dapat mempelajari fitur-fitur lain yang signifikan saat proses belajar, sehingga hasil atau *output*-nya mendekati atau sesuai dengan tujuan yang diinginkan. Jumlah *hidden layer* tidak memiliki ketentuan khusus dalam penggunaannya. Jumlah *hidden layer* yang banyak dapat menyelesaikan masalah yang kompleks karena dari segi pengenalan fitur baru akan lebih banyak, namun waktu komputasi akan lebih lama. Oleh karena itu, diperlukan observasi untuk menemukan jumlah *hidden layer* yang tepat.

4. *Number of neuron per hidden layer*: merupakan jumlah *neuron* yang terdapat pada setiap *hidden layer*. *Neuron* berperan untuk menaikkan atau menurunkan nilai bobot *(weight)* yang mempengaruhi *output* dari model. Menambah jumlah *neuron* memungkinkan model untuk mengurangi nilai *error* pada saat proses belajar. Semakin banyak *neuron* maka perhitungan model akan semakin kompleks, namun model dapat konvergen. Sementara jika *neuron* terlalu sedikit menyebabkan jaringan gagal untuk menyampaikan informasi yang benar dari *input layer* ke setiap *layer* berikutnya. Oleh karena itu, diperlukan observasi untuk menemukan jumlah *neuron* pada *hidden layer* yang tepat.

Selanjutnya, dibawah ini merupakan penjelasan indikator yang akan digunakan pada Autoencoder:

1. *Number of hidden layer*: merupakan jumlah *layer* tersembunyi yang berada

diantara *input layer* dan *output layer*. Jumlah *hidden layer* pada Autoencoder harus simetris atau sama jumlahnya antara *encoder* dan *decoder*. Hal tersebut dikarenakan pada *encoder* data masukkan akan dikompresi ke dimensi yang lebih kecil dan pada *decoder* data akan direkonstruksi kembali untuk menjadi data masukkan. Jika jumlah *hidden layer* semakin banyak, maka semakin banyak pula kompresi datanya dan menyebabkan banyak informasi yang hilang. Selain itu jumlah *hidden layer* yang terlalu banyak dapat menyebabkan komputasi lebih lama dengan hasil yang sama apabila menggunakan *hidden layer* yang sedikit. Oleh karena itu, diperlukan observasi untuk menemukan jumlah *hidden layer* yang tepat.

2. *Number of neuron per hidden layer*: merupakan jumlah *neuron* pada setiap *hidden layer*. Jumlah *neuron* harus lebih kecil dari *input neuron* serta simetris secara terbalik antara *encoder* dan *decoder*. *Neuron* pada *encoder* jumlahnya akan semakin kecil, sedangkan pada *decoder* jumlahnya akan semakin besar. *Neuron* pada *hidden layer* berfungsi sebagai representasi fitur data masukkan yang dikompresi semakin kecil pada setiap *hidden layer*. Semakin kecil jumlah *neuron* maka data akan direpresentasikan dalam dimensi yang lebih kecil. Jumlah *neuron* yang kecil dapat membuat banyak informasi yang hilang pada saat kompresi, sehingga menghasilkan *loss* yang besar. Oleh karena itu, diperlukan observasi untuk menemukan jumlah *neuron* pada *hidden layer* yang tepat.

3. *Number of neuron in code layer*: merupakan jumlah *neuron* yang merepresentasikan data masukkan dalam dimensi yang paling kecil/padat. Dalam *code layer* jumlah *neuron* harus lebih kecil daripada *input layer* dan *hidden layer*. Semakin kecil jumlah *neuron* pada *code layer* maka representasinya akan semakin padat, namun pada saat dilakukan proses *decoding* akan menghasilkan *reconstruction error* yang besar dikarenakan informasi yang dimiliki *code layer* sangat sedikit. Hal tersebut disebabkan oleh terlalu kecilnya jumlah *neuron* pada *code layer* yang harus merepresentasikan data masukkan. Oleh karena itu, diperlukan observasi untuk menemukan jumlah *neuron* pada *code layer* yang tepat.

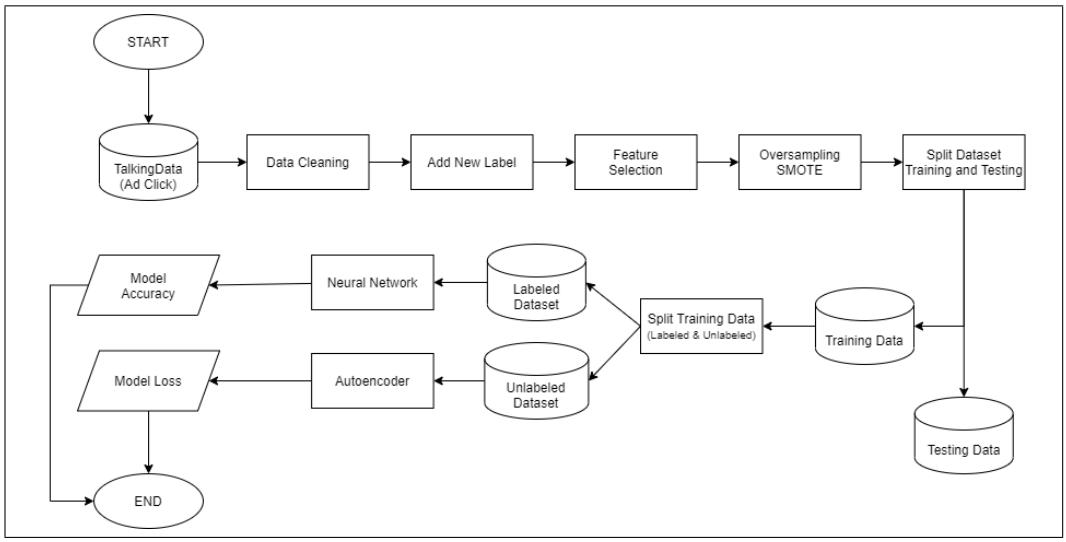
4. *Threshold*: digunakan dalam mendeteksi data *fraud* pada Autoencoder yang tujuannya untuk membuat garis batas *reconstruction error* antara data *fraud* dan data non-*fraud*. Penggunaan nilai *threshold* yang terlalu besar dapat menyebabkan model mengalami misklasifikasi data *fraud* sehingga *recall* akan rendah. Sebaliknya, jika nilai *threshold* terlalu kecil maka model akan

banyak misklasifikasi data non-*fraud* yang menyebabkan *precision* rendah. Oleh karena itu, diperlukan observasi untuk menemukan nilai *threshold* yang tepat.

3.3 Analisis Urutan Proses Global

Dalam pendeteksian *click fraud* ini terbagi menjadi dua proses yaitu proses *training* dan proses *testing*. Proses *training* dilakukan untuk mendapatkan model Neural Network dan Autoencoder yang optimal. Proses *testing* dilakukan untuk menguji kedua model untuk mendeteksi *click fraud* menggunakan dataset *testing*.

3.3.1 Proses *Training*



Gambar 3.2 *Flowchart Training*

Berikut ini adalah uraian dari *flowchart training* pada gambar 3.2 yang dilakukan pada penelitian ini:

1. Dataset yang digunakan terdiri dari 100.000 data dan memiliki delapan fitur, yaitu *ip*, *app*, *device*, *os*, *channel*, *click_time*, *attributed_time*, dan *is_attributed*.

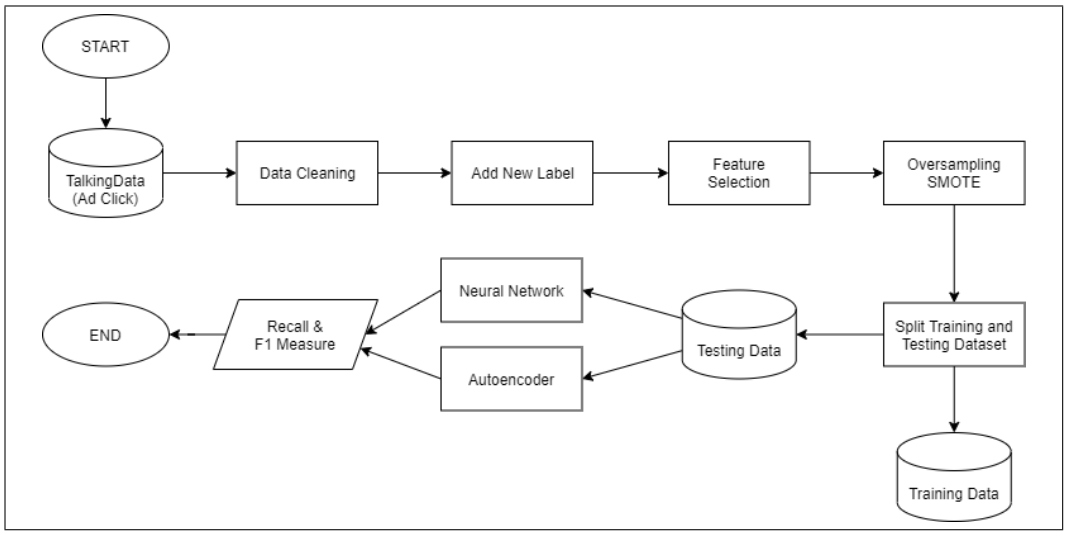
2. Data *cleaning* untuk menghilangkan *missing values* dan *duplicate values*.

3. Penambahan label baru yaitu *is_fraud* berdasarkan label sebelumnya *is_attributed*. Label baru ditambahkan dengan ketentuan sebagai berikut: jika *is_attributed* = 1 maka *is_fraud* = 0, sedangkan jika *is_attributed* = 0 maka *is_fraud* = 1. Setelah itu label *is_attributed* dibuang agar dataset tidak

memiliki dua kelas target.

4. *Feature selection* untuk memilih fitur-fitur yang paling signifikan untuk membangun model.
5. *Oversampling* SMOTE untuk mengatasi jumlah kelas target yang tidak seimbang.
6. Membagi dataset untuk proses belajar Neural Network dan Autoencoder sebesar 80% dari total dataset.
7. Dataset untuk proses belajar dibagi kembali menjadi berlabel dan tidak berlabel. Dataset berlabel digunakan sebagai data belajar untuk Neural Network yang terdiri dari kelas *fraud* dan non-*fraud*, sementara dataset tidak berlabel digunakan sebagai data belajar untuk Autoencoder yang terdiri dari kelas *fraud* saja.
8. Akurasi model untuk melihat performa hasil belajar Neural Network, sementara *loss* model digunakan untuk melihat performa hasil belajar Autoencoder.

3.3.2 Proses *Testing*



Gambar 3.3 *Flowchart Testing*

Berikut ini adalah uraian dari *flowchart testing* pada gambar 3.3 yang dilakukan pada penelitian ini:

1. Dataset yang digunakan terdiri dari 100.000 data dan memiliki delapan fitur, yaitu *ip*, *app*, *device*, *os*, *channel*, *click_time*, *attributed_time*, dan

is_attributed.

2. Data *cleaning* untuk menghilangkan *missing values* dan *duplicate values*.
3. Penambahan label baru yaitu *is_fraud* berdasarkan label sebelumnya *is_attributed*. Label baru ditambahkan dengan ketentuan sebagai berikut: jika *is_attributed* = 1 maka *is_fraud* = 0, sedangkan jika *is_attributed* = 0 maka *is_fraud* = 1. Setelah itu label *is_attributed* dibuang agar dataset tidak memiliki dua kelas target.
4. *Feature selection* untuk memilih fitur-fitur yang paling signifikan untuk membangun model.
5. *Oversampling* SMOTE untuk mengatasi jumlah kelas target yang tidak seimbang.
6. Membagi dataset untuk proses pengujian Neural Network dan Autoencoder sebesar 20% dari total dataset.
7. Neural Network dan Autoencoder diuji menggunakan data uji berlabel *fraud* dan non-*fraud*.
8. Keluaran model dalam bentuk *recall* dan *f1 measure* untuk menilai performa Neural Network dan Autoencoder dalam mendeteksi *click fraud*.

3.4 Analisis Kasus

Pada bagian ini dilakukan analisis tahapan proses dengan melakukan perhitungan manual.

3.4.1 Dataset

Dataset yang digunakan pada penelitian ini diambil dari *Kaggle* yang berjudul ”TalkingData AdTracking Fraud Detection Challenge” [28]. TalkingData adalah perusahaan layanan platform *big data* terbesar yang mencangkup lebih dari 70% perangkat seluler aktif diseluruh negara China. Jumlah data yang digunakan dalam penelitian ini adalah 100.000 data. Data tersebut merupakan data historikal aktivitas klik *user* pada *mobile advertising*. Dalam dataset terdapat 7 fitur yang telah dilakukan enkripsi terlebih dahulu untuk faktor keamanan dan 1 label, yaitu:

1. ***ip***: *ip address user* yang melakukan klik

2. ***app***: id aplikasi yang digunakan untuk *marketing*
3. ***device***: id jenis perangkat yang digunakan oleh *user* (iphone 6 plus, iphone 7, huawei, dll)
4. ***os***: id sistem operasi perangkat yang digunakan oleh *user*
5. ***channel***: id *channel* dari *publisher* iklan seluler
6. ***click_time***: waktu ketika *user* melakukan klik
7. ***attributed_time***: waktu ketika *user* melakukan *download* aplikasi setelah melakukan klik pada sebuah iklan
8. ***is_attributed***: target prediksi, mengindikasikan apakah aplikasi di-*download* oleh *user* atau tidak

Label *is_attributed* yang terdapat pada *dataset* mengindikasikan *user* yang tergolong *fraud* dan non-*fraud* berdasarkan apakah sebuah aplikasi di-*download* oleh *user* setelah melakukan klik pada sebuah iklan. Jika *user* melakukan *download* aplikasi dari iklan maka diberikan nilai 1 *(download)* yang artinya *non-fraud*, sedangkan jika *user* tidak melakukan *download* aplikasi dari iklan maka diberikan nilai 0 *(not download)* dan disebut sebagai *fraud*. Namun, label tersebut mengindikasikan perilaku *user* yang akan men-*download* aplikasi atau tidak dari sebuah iklan. Pada penelitian ini, berfokus untuk deteksi *click fraud* dari seorang *user*. Oleh karena itu, dilakukan penambahan label baru, yaitu *is_fraud* yang langsung mengindikasikan apakah *user* merupakan *fraud* atau non-*fraud* berdasarkan label sebelumnya *is_attributed*. Pada gambar 3.4 merupakan contoh dataset yang digunakan pada penelitian ini.

ip	app	device	os	channel	click_time	attributed_time	is_attributed
116067	18	1	19	121	11/9/2017 0:36		0
115581	2	1	19	205	11/7/2017 16:42		0
5314	18	1	19	107	11/9/2017 9:35		0
81009	13	1	12	477	11/8/2017 2:58		0
105170	9	1	13	232	11/7/2017 2:08		0
224120	19	0	29	213	11/8/2017 2:22	11/8/2017 2:22	1
7815	12	1	17	265	11/8/2017 14:38		0
30724	14	1	25	463	11/7/2017 0:54		0
290743	12	1	18	265	11/9/2017 14:20		0
22088	12	1	13	409	11/8/2017 3:41		0

Gambar 3.4 Contoh Dataset

3.4.2 *Preprocessing*

Pada tahap ini dilakukan *preprocessing* terhadap dataset sebelum digunakan untuk proses belajar pada Neural Network dan Autoencoder.

3.4.2.1 *Data Cleaning*

Tahap ini dilakukan untuk membersihkan data dari *missing values* dan *duplicate values* yang dapat merusak model yang akan dibangun. Dataset yang digunakan memiliki banyak *missing values* pada fitur *attributed_time*. Oleh karena itu, kolom *attributed_time* dibuang agar mendapatkan *clean* dataset. Setelah itu dilakukan konversi fitur *click_time* dengan tipe data *datetime* menjadi *integer* dan didapatkan fitur *click_day*, *click_month*, *click_year*, *click_hour*, *click_minute*, dan *click_second*. Setelah dilakukan perhitungan variansi terhadap setiap fitur yang ada, ternyata fitur *click_month* dan *click_year* memiliki nol (0) variansi maka fitur tersebut dibuang agar model dapat belajar dengan baik. Gambar 3.5 merupakan contoh dataset setelah dilakukan *data cleaning*.

ip	app	device	os	channel	click_day	click_hour	click_minute	click_second	is_attributed
87540	12	1	13	497	7	9	30	38	0
105560	25	1	17	259	7	13	40	27	0
101424	12	1	19	212	7	18	5	24	0
94584	13	1	13	477	7	4	58	8	0
68413	12	1	1	178	9	9	0	9	0

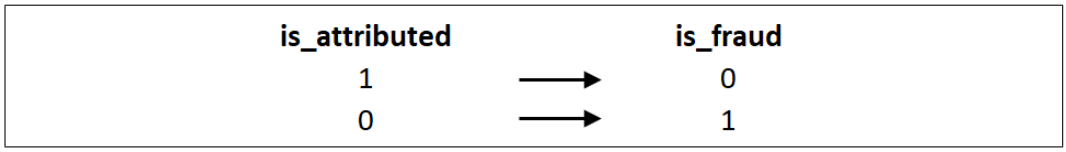
Gambar 3.5 Contoh Dataset Setelah Dilakukan *Data Cleaning*

3.4.2.2 Penambahan Label Baru

Pada dataset yang digunakan memiliki label *is_attributed* yang merupakan indikasi apakah sebuah aplikasi di-*download* oleh *user* setelah meng-klik sebuah iklan. Label tersebut merupakan perilaku *user* yang tertarik atau tidak men-*download* aplikasi. Pada penelitian Thejas G.S. [3] label *is_attributed* digunakan untuk menentukan apakah *user* melakukan *fake click* atau *valid click* berdasarkan perilaku *user* yang men-*download* atau tidak men-*download* aplikasi. Jika *user* men-*download* aplikasi dari iklan maka *user* dianggap melakukan *valid click*, sedangkan jika *user* tidak men-*download* aplikasi dari iklan maka dianggap melakukan *fake click*.

Pada penelitian ini bertujuan untuk mendeteksi *user* yang merupakan *fraud* atau non-*fraud*. *User fraud* menandakan bahwa *publisher* melakukan klik pada sebuah

iklan secara ilegal (dalam frekuensi yang besar) pada aplikasinya dan bertujuan untuk menghasilkan keuntungan berlebih. Oleh karena itu, dilakukan penambahan label baru yaitu *is_fraud* yang mengindikasikan secara langsung apakah *user* merupakan *fraud* atau non-*fraud*. Label *is_fraud* dibuat berdasarkan label sebelumnya yaitu *is_attributed*, jika *user* men-*download* aplikasi disebut non-*fraud* (*is_fraud* = 0), sedangkan jika *user* tidak men-*download* aplikasi disebut *fraud* (*is_fraud* = 1). Gambar 3.6 merupakan ketentuan pembuatan label baru. Setelah dilakukan penambahan label baru *(is_fraud)*, maka label sebelumnya *(is_attributed)* dibuang. Gambar 3.7 merupakan contoh dataset setelah dilakukan penambahan label baru.



Gambar 3.6 Ketentuan Pembuatan Label Baru

ip	app	device	os	channel	click_day	click_hour	click_minute	click_second	is_fraud
87540	12	1	13	497	7	9	30	38	1
105560	25	1	17	259	7	13	40	27	1
101424	12	1	19	212	7	18	5	24	1
94584	13	1	13	477	7	4	58	8	1
68413	12	1	1	178	9	9	0	9	1

Gambar 3.7 Penambahan Label Baru *is_fraud*

3.4.2.3 *Feature Selection*

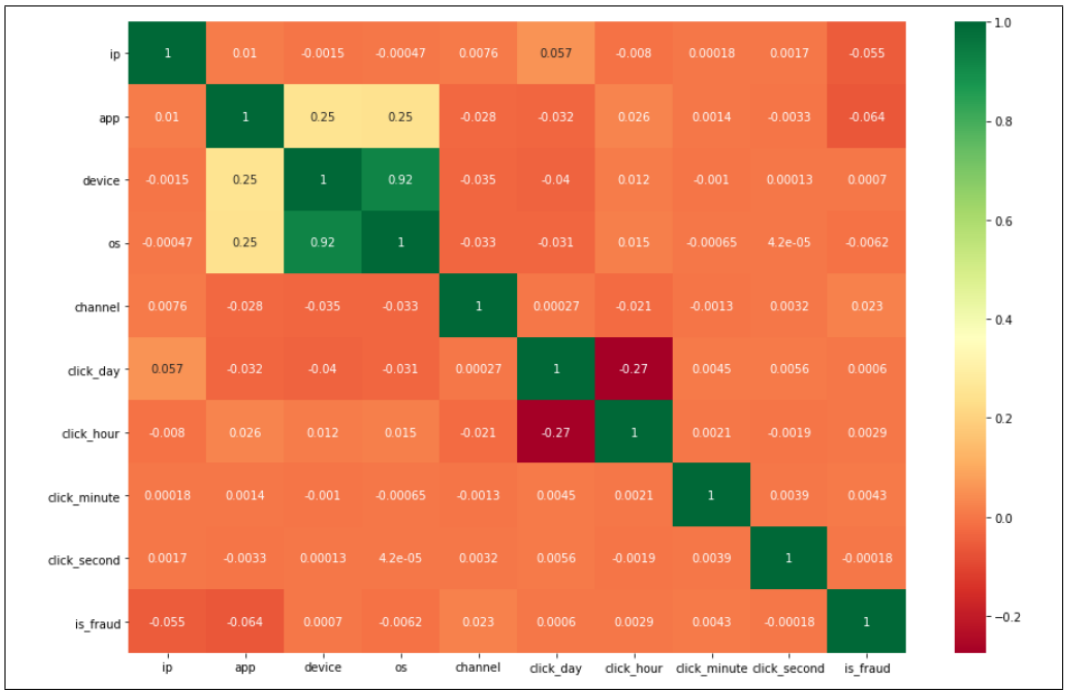
Pada tahap ini dilakukan seleksi fitur untuk memilih hanya fitur-fitur paling signifikan saja yang akan digunakan pada proses pembelajaran model. Proses seleksi fitur menggunakan *Heatmap Correlation Matrix* untuk mengetahui fitur apa saja yang paling mempengaruhi keluaran model. Untuk memunculkan matriks korelasi menggunakan *heatmap* digunakan metode *pearson* untuk mendapatkan korelasi antar fitur serta pustaka *pandas*, *seaborn*, dan *matplotlib*. Kode *python* untuk menampilkan *heatmap* dapat dilihat pada gambar 3.8.

```
corrmat = df_clean3.corr(method='pearson')
top_corr_features = corrmat.index
plt.figure(figsize=(15,10))

# plot heatmap correlation matrix
g=sns.heatmap(df_clean3[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

Gambar 3.8 Kode *python* untuk *Heatmap Correlation Matrix*

Lalu, hasil matriks korelasi antar fitur pada dataset yang digunakan dapat dilihat pada gambar 3.9.



Gambar 3.9 *Heatmap Correlation Matrix*

Dapat dilihat pada gambar di atas, bahwa label *is_fraud* memiliki korelasi dengan lima fitur, yang tertinggi adalah *channel* diikuti dengan *click_minute*, *click_hour*, *device*, dan *click_day*. Oleh karena itu, pada penelitian ini akan digunakan lima fitur tersebut. Gambar 3.10 merupakan dataset yang akan digunakan setelah proses seleksi fitur.

device	channel	click_day	click_hour	click_minute	is_fraud
1	497	7	9	30	1
1	259	7	13	40	1
1	212	7	18	5	1
1	477	7	4	58	1
1	178	9	9	0	1

Gambar 3.10 Dataset Setelah Proses Seleksi Fitur

3.4.2.4 Penerapan SMOTE

Dataset yang digunakan memiliki kelas target yang tidak seimbang antara *fraud* dan non-*fraud*. Kelas minoritas pada dataset adalah *is_fraud* = 0 atau non-*fraud* dengan persentase hanya 0.22%. Oleh karena itu, diterapkan SMOTE untuk melakukan *oversampling* terhadap kelas minoritas tersebut agar mendapatkan dataset yang seimbang. Parameter yang digunakan pada SMOTE pada penelitian ini, yaitu: *sampling_strategy* = *minority*, jumlah tetangga terdekat = 5 *(default)*, dan *random seed* = 42. Setelah dilakukan SMOTE, dataset menjadi berjumlah 199.544 data yang terdiri dari kelas *fraud* sebanyak 99.772 data dan non-*fraud* sebanyak 99.772 data. Dalam melakukan *oversampling* SMOTE digunakan pustaka *imblearn* dengan kode *python* pada gambar 3.11.

```python
# Melakukan oversampling SMOTE

sm = SMOTE(sampling_strategy="minority", k_neighbors=5, random_state=42)
X_res, y_res = sm.fit_resample(X,y)
```

Gambar 3.11 Kode *python* untuk SMOTE

Gambar 3.12 merupakan dataset setelah dilakukan proses *oversampling* dengan SMOTE.

device	channel	click_day	click_hour	click_minute	is_fraud
1	497	7	9	30	1
1	259	7	13	40	1
1	212	7	18	5	1
1	477	7	4	58	1
1	178	9	9	0	1
...	...	...	...	...	...
0	213	8	13	27	0
0	341	7	22	15	0
1	470	7	11	18	0
0	213	8	14	4	0
1	489	9	3	51	0

Gambar 3.12 Dataset Setelah Proses SMOTE

3.4.3 *Split Dataset* untuk *Training* dan *Testing*

Pada tahap ini dilakukan pembagian dataset untuk proses *training* dan *testing* yang akan digunakan pada Neural Network dan Autoencoder. Pembagian dataset yang dilakukan adalah 80% untuk proses *training* dan 20% untuk proses *testing* dan akan mempertahankan komposisi kelas *fraud* dan non-*fraud* secara seimbang pada masing-masing dataset baik *training* dan *testing*.

3.4.4 *Split Training* Data (*Labeled* dan *Unlabeled*)

Pada tahap ini dilakukan pembagian dataset yang akan digunakan pada model sebagai data belajar. Pembagian dilakukan pada data belajar yang dibagi menjadi dua kriteria, yaitu data berlabel dan tidak berlabel. Dataset berlabel digunakan untuk Neural Network yang *supervised*, sedangkan dataset tidak berlabel digunakan untuk Autoencoder yang *unsupervised*. Neural Network memerlukan label untuk mempelajari pola yang ada dalam data belajar, namun Autoencoder dapat mempelajari pola dalam dataset secara mandiri tanpa memerlukan label.

3.4.4.1 Dataset Neural Network

Neural Network menggunakan dataset yang terdiri dari lima fitur, yaitu: *device*, *channel*, *click_day*, *click_hour*, dan *click_minute*. Selain itu terdapat label *is_fraud*

yang mengindikasikan apakah sebuah *user* merupakan *fraud* atau *non-fraud*. Gambar 3.13 merupakan contoh data belajar Neural Network yang berlabel.

device	channel	click_day	click_hour	click_minute	is_fraud
1	497	7	9	30	1
1	259	7	13	40	1
1	212	7	18	5	1
1	477	7	4	58	1
1	178	9	9	0	1
...	...	...	...	...	...
0	213	8	13	27	0
0	341	7	22	15	0
1	470	7	11	18	0
0	213	8	14	4	0
1	489	9	3	51	0

Gambar 3.13 Contoh Data Belajar Neural Network

3.4.4.2 Dataset Autoencoder

Autoencoder merupakan pembelajaran *unsupervised*, sehingga tidak memerlukan label dalam data belajarnya. Namun, data yang dipelajari harus terdiri dari label yang serupa. Dalam penelitian ini, data belajar menggunakan data dengan label *is_fraud* = 0 (non-*fraud*), hal tersebut dilakukan agar Autoencoder dapat mempelajari pola dari fitur data yang serupa. Oleh karena itu, sebelum label dihilangkan untuk proses belajar, maka dilakukan dahulu proses pengelompokan data yang berlabel *is_fraud* = 0, seperti contoh pada gambar 3.14.

device	channel	click_day	click_hour	click_minute	is_fraud
0	213	8	2	22	0
1	113	8	6	10	0
0	213	7	9	54	0
0	343	9	10	58	0
0	213	7	22	19	0

Gambar 3.14 Contoh Kelompok Data Berlabel *is_fraud* = 0

Setelah itu, label *is_fraud* dihapus untuk proses belajar Autoencoder seperti contoh

pada gambar 3.15.

	device	channel	click_day	click_hour	click_minute
	0	213	8	2	22
	1	113	8	6	10
	0	213	7	9	54
	0	343	9	10	58
	0	213	7	22	19

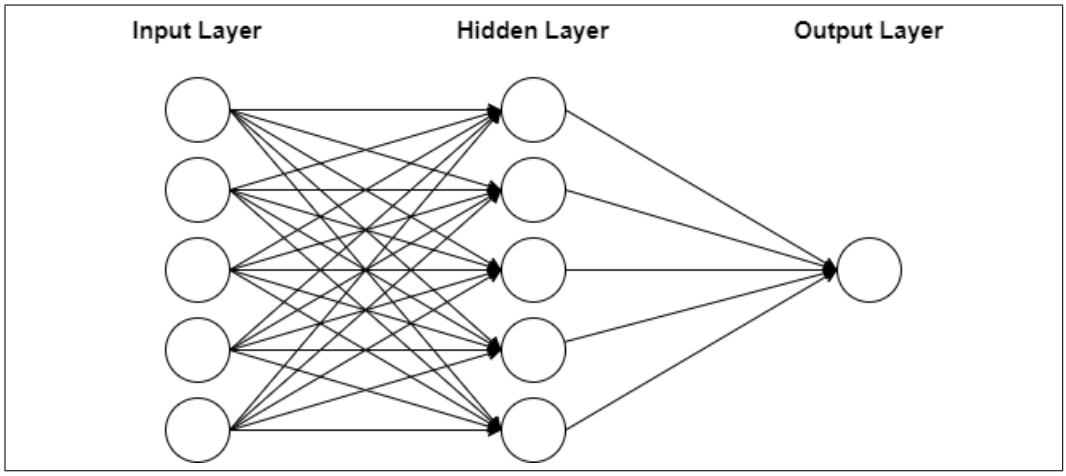
Gambar 3.15 Contoh Data Belajar Autoencoder

3.4.5 Perhitungan Neural Network

Arsitektur yang akan digunakan dalam tahapan Neural Network dibagi menjadi 3 *layer* yang terdiri dari 1 *input layer*, 1 *hidden layer*, dan 1 *output layer*. Pada masing-masing *layer* memiliki jumlah *neuron* dan fungsi aktivasi yang berbeda, yaitu:

1. *Input layer* dengan 5 *neuron* yang merepresentasikan 5 fitur dan menggunakan fungsi aktivasi ReLU.
2. *Hidden layer* dengan 5 *neuron* dan menggunakan fungsi aktivasi ReLU.
3. *Output layer* dengan 1 *neuron* dan menggunakan fungsi aktivasi *sigmoid*.

Setelah mendapatkan nilai keluaran maka dapat dihitung *loss function* menggunakan *binary crossentropy* yang digunakan untuk melakukan pembaharuan nilai bobot pada tahap *backpropagation*. Gambar arsitektur Neural Network dapat dilihat pada gambar 3.16.



Gambar 3.16 Arsitektur Neural Network

Dataset yang telah dilakukan proses *preprocessing* akan digunakan sebagai masukan dari tahap Neural Network. Dataset yang digunakan adalah dataset berlabel. Perhitungan akan menggunakan lima fitur seperti pada gambar 3.13. Pertama, dilakukan dahulu standarisasi nilai untuk masing-masing fitur yang ada. Standarisasi nilai menggunakan Z-Score atau nilai baku, hal tersebut dikarenakan dataset memiliki distribusi yang tidak normal. Tabel 3.1 merupakan perbandingan nilai fitur sebelum distandarisasi dan setelah distandarisasi untuk contoh data dengan label *is_fraud* = 0 atau non-*fraud*.

Tabel 3.1 Contoh Data Dengan 5 Fitur Sebelum dan Sesudah Standarisasi

No	**Fitur**	**Sebelum Standarisasi**	**Setelah Standarisasi**
1	device	0	-0,202599
2	channel	343	1,184274
3	click_day	9	2,143964
4	click_hour	10	0,324234
5	click_minute	58	1,783353

Pada tahap awal Neural Network, dilakukan inisialisasi dan dilanjutkan dengan perhitungan keluaran pada setiap *layer*:

Inisialisasi :

Input neuron $= 5$

Hidden layer $= 1$
Neuron pada hidden layer $= 5$

Output neuron $= 1$

Learning rate $= 0,001$
Bias $= 0$
Rentang bobot $= -0,37214855$ *sampai dengan* $+0,37214855$:
$w_{j_1k_1} = 0,080 \quad w_{j_2k_1} = -0,245 \quad w_{j_3k_1} = -0,323 \quad w_{j_4k_1} = 0,334 \quad w_{j_5k_1} = 0,346$
$w_{j_1k_2} = 0,229 \quad w_{j_2k_2} = -0,145 \quad w_{j_3k_2} = -0,299 \quad w_{j_4k_2} = 0,137 \quad w_{j_5k_2} = -0,044$
$w_{j_1k_3} = -0,281 \quad w_{j_2k_3} = -0,003 \quad w_{j_3k_3} = -0,346 \quad w_{j_43} = 0,304 \quad w_{j_53} = -0,179$
$w_{j_1k_4} = 0,120 \quad w_{j_2k_4} = -0,140 \quad w_{j_3k_4} = 0,014 \quad w_{j_4k_4} = 0,034 \quad w_{j_5k_4} = -0,234$
$w_{j_1k_5} = 0,349 \quad w_{j_2k_5} = 0,204 \quad w_{j_3k_5} = 0,327 \quad w_{j_4k_5} = 0,293 \quad w_{j_5k_5} = 0,072$

$w_{k_1l_1} = 0,313 \quad w_{k_2l_1} = -0,306 \quad w_{k_3l_1} = -0,226 \quad w_{k_4l_1} = -0,338 \quad w_{k_5l_1} = -0,130$

Berikut ini merupakan perhitungan terhadap *hidden layer*, seperti pada persamaan 2.1 dan 2.2:

$$h_1 = ((w_{j_1k_1} \times x_1) + (w_{j_2k_1} \times x_2) + (w_{j_3k_1} \times x_3) + (w_{j_4k_1} \times x_4) + (w_{j_5k_1} \times x_5)) + b$$
$$h_1 = ((0,080 \times -0,202599) + (-0,245 \times 1,184274) + (-0,323 \times 2,143964) + (0,334 \times 0,324234) + (0,346 \times 1,783353)) + 0$$
$$h_1 = -0,274319$$

$$h_2 = ((w_{j_1k_2} \times x_1) + (w_{j_2k_2} \times x_2) + (w_{j_3k_2} \times x_3) + (w_{j_4k_2} \times x_4) + (w_{j_5k_2} \times x_5)) + b$$
$$h_2 = ((0,229 \times -0,202599) + (-0,145 \times 1,184274) + (-0,229 \times 2,143964) + (0,137 \times 0,324234) + (-0,044 \times 1,783353)) + 0$$
$$h_2 = -0,895719$$

$$h_3 = ((w_{j_1k_3} \times x_1) + (w_{j_2k_3} \times x_2) + (w_{j_3k_3} \times x_3) + (w_{j_4k_3} \times x_4) + (w_{j_5k_3} \times x_5)) + b$$
$$h_3 = ((-0,281 \times -0,202599) + (-0,003 \times 1,184274) + (-0,346 \times 2,143964) + (0,304 \times 0,324234) + (-0,179 \times 1,783353)) + 0$$
$$h_3 = -0,911657$$

$$h_4 = ((w_{j_1k_4} \times x_1) + (w_{j_2k_4} \times x_2) + (w_{j_3k_4} \times x_3) + (w_{j_4k_4} \times x_4) + (w_{j_5k_4} \times x_5)) + b$$
$$h_4 = ((0,120 \times -0,202599) + (-0,140 \times 1,184274) + (0,014 \times 2,143964) + (0,034 \times 0,324234) + (-0,234 \times 1,783353)) + 0$$
$$h_4 = -0,565486$$

$$h_5 = ((w_{j_1k_5} \times x_1) + (w_{j_2k_5} \times x_2) + (w_{j_3k_5} \times x_3) + (w_{j_4k_5} \times x_4) + (w_{j_5k_5} \times x_5)) + b$$
$$h_5 = ((0,349 \times -0,202599) + (0,204 \times 1,184274) + (0,327 \times 2,143964) + (0,293 \times 0,324234) + (0,072 \times 1,783353)) + 0$$
$$h_5 = 1,098264$$

Setelah dilakukan perhitungan terhadap *hidden layer*, selanjutnya diterapkan fungsi aktivasi ReLU menggunakan rumus 2.8 pada *hidden layer* seperti pada persamaan 2.3:

$$
\begin{aligned}
&ReLU(h_1) = max(0, h_1)\\
&ReLU(-0,274319) = max(0,\ -0,274319)\\
&ReLU(-0,274319) = 0\\
\\
&ReLU(h_2) = max(0, h_2)\\
&ReLU(-0,895719) = max(0,\ -0,895719)\\
&ReLU(-0,895719) = 0\\
\\
&ReLU(h_3) = max(0, h_3)\\
&ReLU(-0,911657) = max(0,\ -0,911657)\\
&ReLU(-0,911657) = 0\\
\\
&ReLU(h_4) = max(0, h_4)\\
&ReLU(-0,565486) = max(0,\ -0,565486)\\
&ReLU(-0,565486) = 0\\
\\
&ReLU(h_5) = max(0, h_5)\\
&ReLU(1,098264) = max(0,\ 1,098264)\\
&ReLU(1,098264) = 1,098264
\end{aligned}
$$

Selanjutnya dilakukan perhitungan terhadap *output layer* menggunakan persamaan 2.1 dan 2.2:

$$
\begin{aligned}
o_1 &= ((w_{k_1 l_1} \times ReLU(h_1)) + (w_{k_2 l_1} \times ReLU(h_2)) + (w_{k_3 l_1} \times ReLU(h_3))\\
&\quad + (w_{k_4 l_1} \times ReLU(h_4)) + (w_{k_5 l_1} \times ReLU(h_5)) + b\\
o_1 &= ((0,313 \times 0) + (-0,306 \times 0) + (-0,226 \times 0)\\
&\quad + (-0,338 \times 0) + (-0,130 \times 1,098264)) + 0\\
o_1 &= -0,142781
\end{aligned}
$$

Pada *output layer* digunakan fungsi aktivasi *sigmoid function* menggunakan rumus 2.6 untuk mendapatkan hasil prediksi antara 1 dan 0 atau *fraud* dan *non-fraud*:

$$sigmoid(o_1) = \frac{1}{1+e^{-o_1}}$$
$$sigmoid(-0,142781) = \frac{1}{1+e^{-0,142781}}$$
$$sigmoid(-0,142781) = \frac{1}{1+0,866944}$$
$$sigmoid(-0,142781) = \frac{1}{1,866944}$$
$$sigmoid(-0,142781) = 0,464365$$

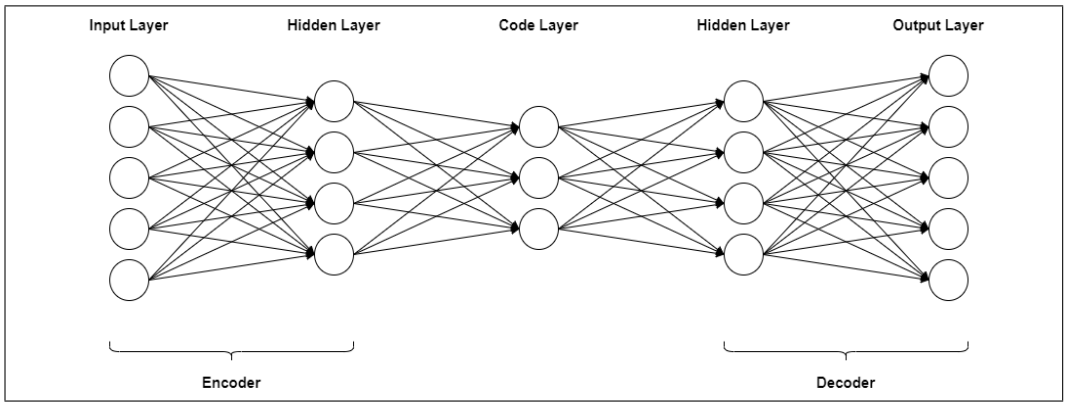
Pada contoh kasus di atas, hasil *neuron* keluaran dari Neural Network adalah 0,464365 yang berarti diklasifikasikan sebagai label 0 atau non-*fraud* seperti pada persamaan 2.5. Klasifikasi pada contoh data tersebut sesuai dengan label yang sebenarnya yaitu non-*fraud*. Namun, nilai *neuron* keluaran tersebut belum baik, maka dalam tahap belajar perlu dilakukan proses *backpropagation* untuk memperbaiki nilai bobot dan bias agar hasil probabilitas dari fungsi aktivasi dapat sesuai dengan hasil yang diinginkan, yaitu jika *fraud* maka hasil mendekati nilai 1 dan jika non-*fraud* maka hasil mendekati nilai 0.

3.4.6 Perhitungan Autoencoder

Arsitektur yang akan digunakan dalam tahapan Autoencoder dibagi menjadi 5 *layer*, yang terdiri dari 2 *layer* pada *encoder*, 1 *code layer*, dan 2 *layer* pada *decoder*. Pada masing-masing *layer* memiliki jumlah *neuron* dan fungsi aktivasi yang berbeda, yaitu:

1. *Input layer* dengan 5 *neuron* yang merepresentasikan 5 fitur dan menggunakan fungsi aktivasi ReLU.
2. *Hidden layer encoder* dengan 4 *neuron* dan menggunakan fungsi aktivasi ReLU.
3. *Code layer* dengan 3 *neuron* dan menggunakan fungsi aktivasi ReLU.
4. *Hidden layer decoder* dengan 4 *neuron* dan menggunakan fungsi aktivasi linear.
5. *Output layer* dengan 5 *neuron* dan menggunakan fungsi aktivasi linear.

Setelah mendapatkan nilai keluaran maka dapat dihitung *loss function* menggunakan *mean squared error* (MSE) yang digunakan untuk melakukan pembaharuan nilai bobot pada tahap *backpropagation*. Gambar arsitektur Autoencoder dapat dilihat pada gambar 3.17.



Gambar 3.17 Arsitektur Autoencoder

Dataset yang telah dilakukan proses *preprocessing* akan digunakan sebagai masukan dari tahap Autoencoder. Dataset dengan label *is_fraud* = 0 yang telah dihilangkan labelnya digunakan sebagai data belajar untuk Autoencoder agar dapat memahami pola secara mandiri. Contoh dataset tidak berlabel dapat dilihat pada gambar 3.15. Seperti pada Neural Network, langkah pertama yang dilakukan adalah melakukan standarisasi nilai terhadap setiap fitur yang ada. Standarisasi menggunakan Z-Score karena distribusi data tidak normal. Pada tabel 3.2 merupakan standarisasi nilai fitur menggunakan data dengan label *non-fraud*.

Tabel 3.2 Contoh Standarisasi Data Dengan Label *is_fraud* = 0

No	**Fitur**	**Sebelum Standarisasi**	**Setelah Standarisasi**
1	Device	0	-0,208672
2	Channel	213	0,064641
3	Click_day	8	0,577796
4	Click_hour	2	-1,156681
5	Click_minute	22	-0,280065

Pada tahap awal Autoencoder, dilakukan inisialisasi dan dilanjutkan dengan perhitungan keluaran pada setiap *layer*:

Inisialisasi :

Input neuron $= 5$
Encoder hidden layer $= 1$
Neuron pada encoder hidden layer $= 4$

Code layer $= 1$
Neuron pada code layer $= 3$

Decoder hidden layer $= 1$
Neuron pada decoder hidden layer $= 4$
Output neuron $= 5$

Learning rate $= 0,001$
Bias $= 0$
Rentang bobot $= -0,37214855$ *sampai dengan* $+0,37214855$:
$w_{j_1k_1} = -0,082 \quad w_{j_2k_1} = -0,170 \quad w_{j_3k_1} = 0,244 \quad w_{j_4k_1} = -0,106 \quad w_{j_5k_1} = -0,163$
$w_{j_1k_2} = 0,031 \quad w_{j_2k_2} = -0,267 \quad w_{j_3k_2} = 0,224 \quad w_{j_4k_2} = -0,316 \quad w_{j_5k_2} = 0,362$
$w_{j_1k_3} = 0,202 \quad w_{j_2k_3} = -0,224 \quad w_{j_3k_3} = -0,368 \quad w_{j_43} = 0,234 \quad w_{j_53} = 0,153$
$w_{j_1k_4} = 0,170 \quad w_{j_2k_4} = 0,201 \quad w_{j_3k_4} = -0,317 \quad w_{j_4k_4} = -0,105 \quad w_{j_5k_4} = -0,285$

$w_{k_1l_1} = 0,270 \quad w_{k_2l_1} = 0,091 \quad w_{k_3l_1} = -0,125 \quad w_{k_4l_1} = -0,324$
$w_{k_1l_2} = -0,140 \quad w_{k_2l_2} = -0,130 \quad w_{k_3l_1} = 0,170 \quad w_{k_4l_1} = 0,102$
$w_{k_1l_3} = 0,288 \quad w_{k_2l_3} = -0,020 \quad w_{k_3l_3} = -0,283 \quad w_{k_4l_3} = 0,158$

$w_{l_1m_1} = 0,194 \quad w_{l_2m_1} = 0,045 \quad w_{l_3m_1} = 0,201$
$w_{l_1m_2} = -0,004 \quad w_{l_2m_2} = 0,016 \quad w_{l_3m_2} = -0,053$
$w_{l_1m_3} = -0,353 \quad w_{l_2m_3} = -0,291 \quad w_{l_3m_3} = -0,348$
$w_{l_1m_4} = 0,101 \quad w_{l_2m_4} = -0,138 \quad w_{l_3m_4} = 0,006$

$w_{m_1n_1} = 0,303 \quad w_{m_2n_1} = -0,186 \quad w_{m_3n_1} = -0,066 \quad w_{m_4n_1} = 0,190$
$w_{m_1n_2} = -0,201 \quad w_{m_2n_2} = -0,314 \quad w_{m_3n_2} = -0,156 \quad w_{m_4n_2} = -0,252$
$w_{m_1n_3} = 0,319 \quad w_{m_2n_3} = 0,229 \quad w_{m_3n_3} = 0,099 \quad w_{m_4n_3} = 0,276$
$w_{m_1n_4} = 0,226 \quad w_{m_2n_4} = -0,233 \quad w_{m_3n_4} = 0,292 \quad w_{m_4n_4} = 0,029$
$w_{m_1n_5} = 0,228 \quad w_{m_2n_5} = 0,294 \quad w_{m_3n_5} = -0,135 \quad w_{m_4n_5} = -0,290$

Berikut ini merupakan perhitungan terhadap *hidden layer* pada bagian *encoder* menggunakan persamaan 2.1 dan 2.2:

$$hE_1 = ((w_{j_1k_1} \times x_1) + (w_{j_2k_1} \times x_2) + (w_{j_3k_1} \times x_3) + (w_{j_4k_1} \times x_4) + (w_{j_5k_1} \times x_5)) + b$$
$$hE_1 = ((-0,082 \times -0,208672) + (-0,170 \times 0,064641) + (0,244 \times -0,577796) + (-0,106 \times -1,156671) + (-0,163 \times -0,280065)) + 0$$
$$hE_1 = 0,316651$$

$$hE_2 = ((w_{j_1k_2} \times x_1) + (w_{j_2k_2} \times x_2) + (w_{j_3k_2} \times x_3) + (w_{j_4k_2} \times x_4) + (w_{j_5k_2} \times x_5)) + b$$
$$hE_2 = ((0,031 \times -0,208672) + (-0,267 \times 0,064641) + (0,224 \times -0,577796) + (-0,316 \times -1,156671) + (0,362 \times -0,280065)) + 0$$
$$hE_2 = 0,370836$$

$$hE_3 = ((w_{j_1k_3} \times x_1) + (w_{j_2k_3} \times x_2) + (w_{j_3k_3} \times x_3) + (w_{j_4k_3} \times x_4) + (w_{j_5k_3} \times x_5)) + b$$
$$hE_3 = ((0,202 \times -0,208672) + (-0,224 \times 0,064641) + (-0,368 \times -0,577796) + (0,234 \times -1,156671) + (0,153 \times -0,280065)) + 0$$
$$hE_3 = -0,584135$$

$$hE_4 = ((w_{j_1k_4} \times x_1) + (w_{j_2k_4} \times x_2) + (w_{j_3k_4} \times x_3) + (w_{j_4k_4} \times x_4) + (w_{j_5k_4} \times x_5)) + b$$
$$hE_4 = ((0,170 \times -0,208672) + (0,201 \times 0,064641) + (-0,317 \times -0,577796) + (-0,105 \times -1,156671) + (-0,285 \times -0,280065)) + 0$$
$$hE_4 = -0,003778$$

Setelah dilakukan perhitungan terhadap *hidden layer*, selanjutnya diterapkan *Rectified Linear Unit function* (ReLU) menggunakan rumus 2.8 pada *hidden layer*:

$$ReLU(hE_1) = max(0, hE_1)$$
$$ReLU(0,316651) = max(0,\ 0,316651)$$
$$ReLU(0,316651) = 0,316651$$

$$ReLU(hE_2) = max(0, hE_2)$$
$$ReLU(0,370836) = max(0,\ 0,370836)$$
$$ReLU(0,370836) = 0,370836$$

$$ReLU(hE_3) = max(0, hE_3)$$
$$ReLU(-0,584135) = max(0,\ -0,584135)$$
$$ReLU(-0,584135) = 0$$

$$ReLU(hE_4) = max(0, hE_4)$$
$$ReLU(-0,003778) = max(0,\ -0,003778)$$
$$ReLU(-0,003778) = 0$$

Selanjutnya dilakukan perhitungan terhadap *layer code* menggunakan persamaan 2.1 dan 2.2:

$$c_1 = ((w_{k_1 l_1} \times ReLU(hE_1)) + (w_{k_2 l_1} \times ReLU(hE_2)) + (w_{k_3 l_1} \times ReLU(hE_3)) + (w_{k_4 l_1} \times ReLU(hE_4))) + b$$
$$c_1 = ((0,270 \times 0,316651) + (0,091 \times 0,370836) + (-0,125 \times 0) + (-0,324 \times 0)) + 0$$
$$c_1 = 0,119609$$

$$c_2 = ((w_{k_1 l_2} \times ReLU(hE_1)) + (w_{k_2 l_2} \times ReLU(hE_2)) + (w_{k_3 l_2} \times ReLU(hE_3)) + (w_{k_4 l_2} \times ReLU(hE_4))) + b$$
$$c_2 = ((-0,140 \times 0,316651) + (-0,130 \times 0,370836) + (0,170 \times 0) + (0,102 \times 0)) + 0$$
$$c_2 = -0,092800$$

$$c_3 = ((w_{k_1 l_3} \times ReLU(hE_1)) + (w_{k_2 l_3} \times ReLU(hE_2)) + (w_{k_3 l_3} \times ReLU(hE_3)) + (w_{k_4 l_3} \times ReLU(hE_4))) + b$$
$$c_3 = ((0,288 \times 0,316651) + (-0,020 \times 0,370836) + (-0,283 \times 0) + (0,158 \times 0)) + 0$$
$$c_3 = 0,083590$$

Setelah dilakukan perhitungan terhadap *layer code*, diterapkan kembali *Rectified Linear Unit function* (ReLU) menggunakan rumus 2.8 pada *layer code*:

$$ReLU(c_1) = max(0, c_1)$$
$$ReLU(0,119609) = max(0,\ 0,119609)$$
$$ReLU(0,119609) = 0,119609$$

$$ReLU(c_2) = max(0, c_2)$$
$$ReLU(-0,092800) = max(0,\ -0,092800)$$
$$ReLU(-0,092800) = 0$$

$$ReLU(c_3) = max(0, c_3)$$
$$ReLU(0,083590) = max(0,\ 0,083590)$$
$$ReLU(0,083590) = 0,083590$$

Kemudian dilakukan perhitungan kembali terhadap *hidden layer* pada bagian *decoder* menggunakan persamaan 2.1 dan 2.2:

$$hD_1 = ((w_{l_1m_1} \times ReLU(c_1)) + (w_{l_2m_1} \times ReLU(c_2)) + (w_{l_3m_1} \times ReLU(c_3))) + b$$
$$hD_1 = ((0,194 \times 0.119609) + (0,045 \times 0) + (0,201 \times 0,083590)) + 0$$
$$hD_1 = 0,040075$$

$$hD_2 = ((w_{l_1m_2} \times ReLU(c_1)) + (w_{l_2m_2} \times ReLU(c_2)) + (w_{l_3m_2} \times ReLU(c_3))) + b$$
$$hD_2 = ((-0,004 \times 0.119609) + (0,016 \times 0) + (-0,053 \times 0,083590)) + 0$$
$$hD_2 = -0,005060$$

$$hD_3 = ((w_{l_1m_3} \times ReLU(c_1)) + (w_{l_2m_3} \times ReLU(c_2)) + (w_{l_3m_3} \times ReLU(c_3))) + b$$
$$hD_3 = ((-0,353 \times 0.119609) + (-0,291 \times 0) + (-0,348 \times 0,083590)) + 0$$
$$hD_3 = -0,071402$$

$$hD_4 = ((w_{l_1m_4} \times ReLU(c_1)) + (w_{l_2m_4} \times ReLU(c_2)) + (w_{l_3m_4} \times ReLU(c_3))) + b$$
$$hD_4 = ((0,101 \times 0.119609) + (-0,138 \times 0) + (0,006 \times 0,083590)) + 0$$
$$hD_4 = 0,012677$$

Setelah itu diterapkan fungsi aktivasi linear menggunakan rumus 2.4 terhadap *hidden layer* pada *decoder*:

$$Linear(hD_1) = hD_1$$
$$Linear(hD_1) = 0,040075$$

$$Linear(hD_2) = hD_2$$
$$Linear(hD_2) = -0,005060$$

$$Linear(hD_3) = hD_3$$
$$Linear(hD_3) = -0,071402$$

$$Linear(hD_4) = hD_4$$
$$Linear(hD_4) = 0,012677$$

Pada *layer output* dilakukan perhitungan kembali pada setiap *neuron* menggunakan persamaan 2.1 dan 2.2:

$$o_1 = ((w_{m_1n_1} \times Linear(hD_1)) + (w_{m_2n_1} \times Linear(hD_2)) + (w_{m_3n_1} \times Linear(hD_3)) + (w_{m_4n_1} \times Linear(hD_4))) + b$$
$$o_1 = ((0,303 \times 0,040075) + (-0,186 \times -0,005060) + (-0,066 \times -0,071402) + (0,190 \times 0,012677)) + 0$$
$$o_1 = 0,020275$$

$$o_2 = ((w_{m_1n_2} \times Linear(hD_1)) + (w_{m_2n_2} \times Linear(hD_2)) + (w_{m_3n_2} \times Linear(hD_3)) + (w_{m_4n_2} \times Linear(hD_4))) + b$$
$$o_2 = ((-0,201 \times 0,040075) + (-0,314 \times -0,005060) + (-0,156 \times -0,071402) + (-0,252 \times 0,012677)) + 0$$
$$o_2 = 0,001481$$

$$o_3 = ((w_{m_1n_3} \times Linear(hD_1)) + (w_{m_2n_3} \times Linear(hD_2)) + (w_{m_3n_3} \times Linear(hD_3)) + (w_{m_4n_3} \times Linear(hD_4))) + b$$
$$o_3 = ((0,319 \times 0,040075) + (0,229 \times -0,005060) + (0,099 \times -0,071402) + (0,276 \times 0,012677)) + 0$$
$$o_3 = 0,008072$$

$$o_4 = ((w_{m_1n_4} \times Linear(hD_1)) + (w_{m_2n_4} \times Linear(hD_2)) + (w_{m_3n_4} \times Linear(hD_3)) + (w_{m_4n_4} \times Linear(hD_4))) + b$$
$$o_4 = ((0,226 \times 0,040075) + (-0,233 \times -0,005060) + (0,292 \times -0,071402) + (0,029 \times 0,012677)) + 0$$
$$o_4 = -0,010253$$

$$o_5 = ((w_{m_1n_5} \times Linear(hD_1)) + (w_{m_2n_5} \times Linear(hD_2)) + (w_{m_3n_5} \times Linear(hD_3)) + (w_{m_4n_5} \times Linear(hD_4))) + b$$
$$o_5 = ((0,228 \times 0,040075) + (0,294 \times -0,005060) + (-0,135 \times -0,071402) + (-0,290 \times 0,012677)) + 0$$
$$o_5 = 0,013671$$

Lalu, diterapkan kembali fungsi aktivasi linear menggunakan rumus 2.4 terhadap *output layer*:

$$Linear(o_1) = o_1$$
$$Linear(0,020275) = 0,020275$$

$$Linear(o_2) = o_2$$
$$Linear(0,001481) = 0,001481$$

$$Linear(o_3) = o_3$$
$$Linear(0,008072) = 0,008072$$

$$Linear(o_4) = o_4$$
$$Linear(-0,010253) = -0,010253$$

$$Linear(o_5) = o_5$$
$$Linear(0,013671) = 0,013671$$

Setelah mendapatkan hasil *neuron* keluaran dari Autoencoder yang merupakan *reconstruction* dari *neuron* masukan, dapat dihitung *reconstruction error* menggunakan rumus *mean squared error* (MSE) seperti pada persamaan 2.15. Perhitungan *reconstruction error* yang dilakukan adalah sebagai berikut:

Neuron masukan :
$device = x_1 = -0,208672$
$channel = x_2 = 0,064641$
$click\_day = x_3 = 0,577796$
$click\_hour = x_4 = -1,156681$
$click\_minute = x_5 = -0,280065$

Neuron keluaran :
$device = \tilde{x}_1 = 0,020275$
$channel = \tilde{x}_2 = 0,001481$
$click\_day = \tilde{x}_3 = 0,008072$
$click\_hour = \tilde{x}_4 = -0,010253$
$click\_minute = \tilde{x}_5 = 0,013671$

$$MSE = \frac{1}{m}\sum_{i=1}^{i=m}(x_i - \tilde{x}_i)^2$$
$$MSE = \frac{1}{5} \times ((x_1 - \tilde{x}_1)^2 + (x_2 - \tilde{x}_2)^2 + (x_3 - \tilde{x}_3)^2 + (x_4 - \tilde{x}_4)^2 + (x_5 - \tilde{x}_5)^2)$$
$$\begin{aligned} MSE = \frac{1}{5} \times ((&-0,208672 - 0,020275)^2 + (0,064641 - 0,001481)^2 \\ &+ (0,577796 - 0,008072)^2 + (-1,156681 - -0,010253)^2 \\ &+ (-0,280065 - 0,013671)^2) \end{aligned}$$
$$\begin{aligned} MSE = \frac{1}{5} \times ((&-0,228947)^2 + (0,063160)^2 + (0,569724)^2 + (-1,146428)^2 \\ &+ (-0,293736)^2) \end{aligned}$$
$$MSE = \frac{1}{5} \times (0,052417 + 0,003989 + 0,324586 + 1,314298 + 0,086281)$$
$$MSE = \frac{1}{5} \times 1,781570$$
$$MSE = 0,356314$$

Setelah dilakukan perhitungan *reconstruction error* menggunakan MSE, maka dapat disimpulkan bahwa contoh data dengan label non-*fraud* di atas memiliki nilai *reconstruction error* 0,356314. Hal tersebut dapat dijadikan pedoman untuk menentukan nilai *threshold* yang membatasi *reconstruction error* antara data non-*fraud* dan data *fraud*. Sebagai contoh, apabila nilai *threshold* yang ditentukan adalah 0,4 maka data yang memiliki *reconstruction error* lebih dari atau sama

dengan *threshold* diklasifikasikan sebagai *fraud* sedangkan data yang memiliki *reconstruction error* kurang dari *threshold* diklasifikasikan sebagai non-*fraud*. Untuk meminimalkan nilai *reconstruction error* maka dapat dilakukan *backpropagation* pada saat proses belajar Autoencoder.

Selanjutnya Autoencoder yang digunakan untuk mendeteksi anomali atau data dengan label *fraud*, akan menghasilkan nilai *reconstruction error* lebih besar dari nilai *threshold* yang telah ditentukan yaitu 0,4. Sebagai contoh, digunakan data dengan label *is_fraud* = 1 untuk menguji Autoencoder dalam mendeteksi anomali. Tabel 3.3 merupakan fitur dari data dengan label *fraud* yang nilainya distandarisasi.

Tabel 3.3 Contoh Standarisasi Data Dengan Label *is_fraud* = 1

No	**Fitur**	**Sebelum Standarisasi**	**Setelah Standarisasi**
1	Device	1	-0,079945
2	Channel	178	-0,701366
3	Click_day	7	-0,968657
4	Click_hour	13	0,593897
5	Click_minute	54	1,430300

Setelah mendapatkan nilai fitur yang telah distandarisasi, dilakukan perhitungan yang sama seperti proses di atas hingga menghasilkan nilai *neuron* keluaran. Hasil keluaran tersebut digunakan kembali untuk menghitung *reconstruction error*. Berikut ini merupakan contoh perhitungan MSE seperti pada persamaan 2.15 untuk data dengan label *fraud*.

Neuron masukan :
$device = x_1 = -0,079945$
$channel = x_2 = -0,701366$
$click\_day = x_3 = -0,968657$
$click\_hour = x_4 = 0,593897$
$click\_minute = x_5 = 1,430300$

Neuron keluaran :
$device = \tilde{x}_1 = 0,000417$
$channel = \tilde{x}_2 = 0,007113$
$click\_day = \tilde{x}_3 = -0,005252$
$click\_hour = \tilde{x}_4 = -0,008944$
$click\_minute = \tilde{x}_5 = 0,010249$

$$MSE = \frac{1}{m}\sum_{i=1}^{i=m}(x_i - \tilde{x}_i)^2$$
$$MSE = \frac{1}{5} \times ((x_1 - \tilde{x}_1)^2 + (x_2 - \tilde{x}_2)^2 + (x_3 - \tilde{x}_3)^2 + (x_4 - \tilde{x}_4)^2 + (x_5 - \tilde{x}_5)^2)$$
$$\begin{aligned} MSE = \frac{1}{5} \times ((&-0,079945 - 0,000417)^2 + (-0,701366 - 0,007113)^2 \\ &+ (-0,968657 - (-0,005252))^2 + (0,593897 - (-0,008944))^2 \\ &+ (1,430300 - 0,010249)^2) \end{aligned}$$
$$\begin{aligned} MSE = \frac{1}{5} \times ((&-0,080362)^2 + (-0,708480)^2 + (-0,963405)^2 + (0,602842)^2 \\ &+ (1,420050)^2) \end{aligned}$$
$$MSE = \frac{1}{5} \times (0,006458 + 0,501944 + 0,928149 + 0,363418 + 2,016543)$$
$$MSE = \frac{1}{5} \times 3,816512$$
$$MSE = 0,763302$$

Setelah dilakukan perhitungan MSE terhadap data dengan label *fraud* didapatkan *reconstruction error* 0,763302. Nilai tersebut lebih besar daripada nilai *threshold* yang ditentukan sebelumnya yaitu 0,4. Maka dapat disimpulkan bahwa contoh data dengan label *fraud* tersebut berhasil diklasifikasikan sebagai *fraud*.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Pada bab ini akan menjelaskan mengenai proses implementasi dan pengujian terhadap sistem yang telah dibangun berdasarkan penjelasan pada bab sebelumnya.

4.1 Lingkungan Implementasi

Pada lingkungan implementasi, akan dijelaskan mengenai perangkat - perangkat yang digunakan selama proses pembangunan sistem baik dari perangkat keras maupun perangkat lunak yang digunakan.

4.1.1 Spesifikasi Perangkat Keras

Spesifikasi dari perangkat keras yang digunakan dalam pembangunan sistem deteksi *click fraud* adalah:

1. *Laptop* ASUS G732LXS.
2. *Processor* Intel Core i9-10980HK CPU @2.4GHz.
3. *Solid State Drive* kapasitas 2TB.
4. RAM dengan kapasitas 32GB.

4.1.2 Lingkungan Perangkat Lunak

Spesifikasi dari perangkat lunak yang digunakan dalam pembangunan sistem deteksi *click fraud* adalah:

1. Sistem Operasi Windows 10 Home 64-bit.
2. Jupyter Notebook 6.3.0.
3. Python 3.8.9 64-bit.

4.2 Daftar *Class* dan *Method*

Pada bab ini akan dijelaskan mengenai *class* dan *method* yang digunakan pada sistem deteksi *click fraud*.

4.2.1 *Class Preprocessing*

Class preprocessing berisikan metode yang digunakan pada tahap *data preprocessing* agar data dapat digunakan pada tahap selanjutnya. Metode yang terdapat pada *class preprocessing* dapat dilihat pada tabel 4.1.

Tabel 4.1 Daftar Metode pada *Class Preprocessing*

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
1	dropFeature	subset, feature	data frame, vector	data frame	Menghapus fitur pada array dataset yang tidak digunakan.
2	dropDuplicate	subset	data frame	data frame	Menghapus baris data yang nilainya duplikat.
3	convertObject ToInteger	subset	data frame	data frame	Mengonversi fitur click_time pada array dataset menjadi integer dengan kolom hari, bulan, tahun, jam, menit, dan detik.
4	checkIs Attributed	row data, axis	vector, integer	integer	Mengecek label is_attributed dan diubah menjadi is_fraud.
5	addNewLabel	subset	data frame	data frame	Menambahkan label baru is_fraud dan menghapus label is_attributed.
6	applySMOTE	subset	data frame	data frame	Mengimplementasikan pustaka SMOTE untuk *oversampling* data yang tidak seimbang.

4.2.2 *Class* Neural Network

Class Neural Network berisikan metode yang digunakan pada proses pembelajaran hingga pengujian. Metode yang terdapat pada *class* Neural Network dapat dilihat pada tabel 4.2.

Tabel 4.2 Daftar Metode pada *Class* Neural Network

No	Metode	Masukan		Luaran	Keterangan
		Parameter	**Variabel**		
1	init	X_train, y_train, X_valid, y_valid, X_test, y_test	array, array, array, array, array, array	-	Melakukan inisialisasi awal data belajar, data validasi, dan data uji.
2	train_NN	hn_1, hn_2, hn_3, epoch, lr, model_ number	integer, integer, integer, integer, float, integer	model, history	Melakukan proses belajar Neural Network menggunakan library sesuai parameter yang ditetapkan serta data belajar dan data validasi yang telah diinisialisasikan.
3	save_model	model, model_ number	model, integer	file path	Melakukan penyimpanan model yang telah melalui proses belajar ke dalam file dalam format .h5.
4	plot_acc_ loss_NN	history	array	graph image	Membuat plot grafik akurasi dan loss setiap epoch dari hasil belajar dan validasi.
5	predict_NN	model	model	confusion matrix, precision, recall, f1 measure	Melakukan prediksi terhadap data uji menggunakan model yang telah melalui proses belajar.

4.2.3 *Class* Autoencoder

Class Autoencoder berisikan metode yang digunakan pada proses pembelajaran hingga pengujian. Metode yang terdapat pada *class* Autoencoder dapat dilihat

pada tabel 4.3.

Tabel 4.3 Daftar Metode pada *Class* Autoencoder

No	Metode	Masukan		Luaran	Keterangan
		Parameter	**Variabel**		
1	init	X_train_0, X_valid_0, X_test, y_test	array, array, array, array	-	Melakukan inisialisasi awal data belajar dengan label non-*fraud*, data validasi dengan label non-*fraud*, dan data uji dengan label *fraud* dan non-*fraud*.
2	train_ Autoencoder	hn_1, hn_2, cn, epoch, lr, model_ number	integer, integer, integer, integer, float, integer	model	Melakukan proses belajar Autoencoder menggunakan model scratch sesuai parameter yang ditetapkan serta data belajar non-fraud dan data validasi yang telah diinisialisasikan.
3	save_model	model, model_ number	model, integer	file path	Melakukan penyimpanan model yang telah melalui proses belajar ke dalam file dalam format .parms.
4	predict_ Autoencoder	model	model	graph image	Membuat plot grafik precision recall trade off berdasarkan prediksi model menggunakan data uji.

Tabel 4.3 Daftar Metode pada *Class* Autoencoder (Lanjutan)

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
5	assign_ threshold	error_df, threshold	data frame, float	figure, confusion matrix, accuracy, precision, recall, f1 measure	Menetapkan nilai threshold untuk prediksi Autoencoder dan mengeluarkan hasil prediksi berdasarkan data uji.

4.2.4 *Class* Layer_Dense

Class Layer_Dense berisikan metode yang digunakan untuk membuat dan menginisialisasikan parameter sebuah *object* berupa *dense layer*. Metode yang terdapat pada *class* Layer_Dense dapat dilihat pada tabel 4.4.

Tabel 4.4 Daftar Metode pada *Class* Layer_Dense

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
1	init	n_inputs, n_neurons	integer, integer	-	Melakukan inisialisasi awal parameter yang ada pada dense layer.
2	forward	inputs, training	array, boolean	-	Melakukan proses perhitungan feed-forward pada dense layer.
3	backward	dvalues	array	-	Melakukan proses perhitungan backpropagation untuk memperbaharui nilai bobot dan bias.
4	get_ parameters	-	-	-	Mengambil parameter bobot dan bias pada kelas Layer_Dense.

Tabel 4.4 Daftar Metode pada *Class* Layer_Dense (Lanjutan)

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
5	set_ parameters	weights, biases	array, array	-	Set nilai bobot dan bias pada kelas Layer_Dense.

4.2.5 *Class* Layer_Input

Class Layer_Input berisikan metode yang digunakan untuk melakukan proses *feed-forward*. Metode yang terdapat pada *class* Layer_Input dapat dilihat pada tabel 4.5.

Tabel 4.5 Daftar Metode pada *Class* Layer_Input

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
1	forward	inputs, training	array, boolean	-	Set nilai input untuk melakukan proses feed-forward.

4.2.6 *Class* Activation_ReLU

Class Activation_ReLU berisikan metode yang digunakan untuk menerapkan fungsi aktivasi ReLU pada proses *feed-forward*, *backpropagation*, dan prediksi. Metode yang terdapat pada *class* Activation_ReLU dapat dilihat pada tabel 4.6.

Tabel 4.6 Daftar Metode pada *Class* Activation_ReLU

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
1	forward	inputs, training	array, boolean	-	Melakukan perhitungan fungsi aktivasi ReLU untuk proses *feed-forward*.
2	backward	dvalues	array	-	Melakukan perhitungan turunan fungsi aktivasi ReLU untuk proses *backpropagation*.

Tabel 4.6 Daftar Metode pada *Class* Activation_ReLU (Lanjutan)

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
3	predictions	outputs	float	float	Mengeluarkan nilai prediksi jika fungsi aktivasi ReLU digunakan pada output layer.

4.2.7 *Class* Activation_Linear

Class Activation_Linear berisikan metode yang digunakan untuk menerapkan fungsi aktivasi Linear pada proses *feed-forward*, *backpropagation*, dan prediksi. Metode yang terdapat pada *class* Activation_Linear dapat dilihat pada tabel 4.7.

Tabel 4.7 Daftar Metode pada *Class* Activation_Linear

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
1	forward	inputs, training	array, boolean	-	Set nilai input dan output sesuai fungsi aktivasi Linear untuk proses *feed-forward*.
2	backward	dvalues	array	-	Melakukan perhitungan turunan fungsi aktivasi Linear untuk proses *backpropagation*.
3	predictions	outputs	float	float	Mengeluarkan nilai prediksi jika fungsi aktivasi Linear digunakan pada output layer.

4.2.8 *Class* Optimizer_Adam

Class Optimizer_Adam berisikan metode yang digunakan untuk menerapkan fungsi optimisasi Adam *(adaptive moment estimation)* yang dapat menangani gradien renggang pada data *noisy*. Metode yang terdapat pada *class*

Optimizer_Adam dapat dilihat pada tabel 4.8.

Tabel 4.8 Daftar Metode pada *Class* Optimizer_Adam

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
1	init	learning_rate, decay, epsilon, beta_1, beta_2	float, float, float, float, float	-	Melakukan inisialisasi awal parameter default optimizer Adam.
2	pre_update_params	-	-	-	Cek nilai decay untuk mengupdate nilai learning_rate yang digunakan pada proses belajar.
3	update_params	layer	list	-	Melakukan update nilai parameter setiap layer menggunakan optimisasi Adam.
4	post_update_params	-	-	-	Menambah iterasi untuk menghitung momentum bobot dan bias terbaru.

4.2.9 *Class* Loss

Class Loss berisikan metode untuk menghitung nilai *loss* pada layer output. Metode yang terdapat pada *class* Loss dapat dilihat pada tabel 4.9.

Tabel 4.9 Daftar Metode pada *Class* Loss

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
1	remember_trainable_layers	trainable_layers	list	-	Menyimpan informasi trainable layer.

Tabel 4.9 Daftar Metode pada *Class* Loss (Lanjutan)

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
2	calculate	output, y	array, array	float	Menghitung data dan regularization loss pada output layer.
3	calculate_ accumulated	-	-	float	Menghitung nilai loss rata-rata untuk data loss dan regularization loss.
4	new_pass	-	-	-	Reset nilai variable menjadi 0 untuk menghitung akumulasi loss baru.

4.2.10 *Class* Loss_Mean_Squared_Error

Class Loss_Mean_Squared_Error berisikan metode untuk menghitung nilai *loss* menggunakan *Mean Squared Error* (MSE) pada layer output. Metode yang terdapat pada *class* Loss_Mean_Squared_Error dapat dilihat pada tabel 4.10.

Tabel 4.10 Daftar Metode pada *Class* Loss_Mean_Squared_Error

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
1	forward	y_pred, y_true	array, array	float	Menghitung nilai loss menggunakan MSE pada layer output berdasarkan nilai prediksi dan nilai sebenarnya.
2	backward	dvalues, y_true	array, array	-	Menghitung nilai turunan input untuk proses backpropagation.

4.2.11 *Class* Model

Class Model berisikan metode untuk membuat model yang digunakan pada proses belajar, evaluasi, prediksi, dan menyimpan model. Metode yang terdapat pada *class* Model dapat dilihat pada tabel 4.11.

Tabel 4.11 Daftar Metode pada *Class* Model

No	Metode	Masukan		Luaran	Keterangan
		Parameter	Variabel		
1	init	-	-	-	Membuat list object untuk setiap layer pada model.
2	add	layer	list	-	Menambahkan object layer ke dalam model.
3	set	*, loss, optimizer	class, class	-	Set kelas loss dan kelas optimizer yang akan digunakan pada model.
4	finalize	-	-	-	Menetapkan model dengan menyambungkan layer-layer dalam bentuk list yang telah dibentuk sebelumnya.
5	train	X, y, *, epochs, batch_size, print_every, validation_ data	array, array, integer, integer, integer, array	-	Melakukan proses belajar menggunakan model yang telah di-finalize sebelumnya.
6	evaluate	X_val, y_val, *, batch_size	array, array, integer	-	Mengevaluasi hasil belajar model menggunakan data validasi apabila diinisialisasikan.

Tabel 4.11 Daftar Metode pada *Class* Model (Lanjutan)

No	Metode	Masukan		Luaran	Keterangan
		Parameter	**Variabel**		
7	predict	X, *, batch_size	array, integer	array	Melakukan prediksi menggunakan model yang telah melalui proses belajar dan mengeluarkan hasilnya berupa hasil prediksi.
8	forward	X, training	array, boolean	list	Melakukan proses feed-forward dari input layer sampai pada output layer.
9	backward	output, y	array, array	-	Melakukan proses backpropagation dari output layer sampai pada input layer.
10	save_ parameters	path	string	-	Menyimpan parameter model yang telah melalui proses belajar.
11	load_ parameters	path	string	-	Mengambil parameter model yang telah disimpan sebelumnya.
12	get_parameters	-	-	list	Mengambil parameter trainable layer dari model yang digunakan pada proses belajar.
13	set_parameters	parameters	list	-	Mengupdate parameter setiap layer dengan parameter yang terbaru.

4.3 Implementasi Perangkat Lunak

Pada bab ini akan dijelaskan mengenai cara implementasi sistem deteksi *click fraud* menggunakan Neural Network dan Autoencoder:

4.3.1 Implementasi Penggunaan Dataset

Seperti yang telah dijelaskan pada bagian 3.4.1, dataset untuk penelitian ini berasal dari perusahaan TalkingData [28] yang diambil melalui Kaggle. Dataset tersebut berisikan data historikal informasi klik *user* pada *mobile advertising*. Dataset yang digunakan berjumlah 100.000 data yang terdiri dari 7 fitur, yaitu: *ip, app, device, os, channel, click_time,* dan *attributed_time* serta 1 label *is_attributed*. Pada gambar 4.1 merupakan contoh dataset murni sebelum dilakukan tahap *preprocessing*.

ip	app	device	os	channel	click_time	attributed_time	is_attributed
116067	18	1	19	121	11/9/2017 0:36		0
115581	2	1	19	205	11/7/2017 16:42		0
5314	18	1	19	107	11/9/2017 9:35		0
81009	13	1	12	477	11/8/2017 2:58		0
105170	9	1	13	232	11/7/2017 2:08		0
224120	19	0	29	213	11/8/2017 2:22	11/8/2017 2:22	1
7815	12	1	17	265	11/8/2017 14:38		0
30724	14	1	25	463	11/7/2017 0:54		0
290743	12	1	18	265	11/9/2017 14:20		0
22088	12	1	13	409	11/8/2017 3:41		0

Gambar 4.1 Contoh Dataset

4.3.2 Implementasi *Preprocessing*

Sebelum dilakukan proses pembelajaran baik Neural Network dan Autoencoder, dilakukan terlebih dahulu tahap *preprocessing* terhadap dataset. *Preprocessing* terdiri dari *data cleaning*, penambahan label baru, *feature selection*, dan penerapan SMOTE. Selain itu dilakukan juga standarisasi nilai fitur agar dapat digunakan pada proses pembelajaran selanjutnya.

4.3.3 Implementasi Pembelajaran Neural Network

Pada bagian ini akan dilakukan pembelajaran Neural Network menggunakan dataset yang telah dilakukan *preprocessing* sebelumnya. Pembelajaran Neural Network akan menggunakan 80% data belajar yang komposisi kelasnya sama dengan data uji. Dari 80% data belajar tersebut dibagi kembali menjadi 60% untuk data belajar dan 20% untuk data validasi dan tetap mempertahankan komposisi yang sama antar kelas. Jumlah data belajar yaitu 119.726 data yang terdiri dari kelas *fraud* berjumlah 59.752 data dan kelas non-*fraud* berjumlah 59.974 data. Lalu jumlah data validasi yaitu 39.909 data yang terdiri dari kelas *fraud* berjumlah 19.962 data dan kelas non-*fraud* berjumlah 19.947 data. Berikut ini merupakan proses yang dilakukan dalam pembelajaran Neural Network:

Proses pembelajaran Neural Network

1. Menetapkan jumlah *hidden layer*, jumlah *neuron* per *hidden layer*, *epoch*, *learning rate* yang akan digunakan untuk pembelajaran.
2. Melakukan perhitungan menggunakan masukan dari data hasil *preprocessing* menggunakan algoritme *feed-forward* serta perhitungan *error* selama tahap pembelajaran.
3. Melakukan proses pembaruan bobot dengan menggunakan algoritme *backpropagation*.
4. Proses pembelajaran akan menggunakan kombinasi dari:

   Jumlah *hidden layer* = 1 dan 2.

   Jumlah *neuron* per *hidden layer* = 8, 16, 32, 64, 128, 256, dan 512.

   Epoch = 50 dan 100.

   Learning rate = 0.001, 0.005, dan 0.01.
5. Menyimpan hasil pembelajaran model Neural Network sesuai kombinasi pada poin diatas.

4.3.3.1 Implementasi Pengujian Neural Network

Pada bagian ini akan dilakukan pengujian Neural Network dengan menggunakan 20% data uji yang komposisi kelasnya sama dengan data belajar. Jumlah data uji yaitu 39.909 data yang terdiri dari kelas *fraud* berjumlah 20.058 data dan kelas non-*fraud* berjumlah 19.851 data. Berikut ini merupakan proses yang dilakukan dalam pengujian Neural Network:

Proses pengujian Neural Network

1. Pengujian menggunakan 20% data uji.
2. Model hasil pembelajaran sebelumnya digunakan untuk proses pengujian.
3. Melakukan proses pengujian dengan menggunakan algoritme *feed-forward* dan memetakan label hasil prediksi terhadap label sebenarnya.
4. Menghitung nilai *recall* dan *f1 measure* berdasarkan jumlah *true positive*, *true negative*, *false positive*, dan *false negative* yang didapatkan dari hasil pengujian.

4.3.3.2 Deskripsi *Hyperparameter* Neural Network

Pada bagian ini akan dijelaskan pemilihan nilai *hyperparameter* yang digunakan pada pembelajaran Neural Network. Pemilihan nilai *hyperparameter* yang dilakukan berdasarkan referensi buku atau *paper*, diantaranya:

1. Jumlah *hidden layer*: 1 dan 2.
   Menurut penelitian Sevdalina [17], penggunaan 1 atau 2 *hidden layer* digunakan secara luas saat ini dan bekerja sangat baik karena model dapat mencapai tujuan yang diinginkan. Penggunaan *hidden layer* yang besar dapat membuat model menemukan *hidden* faktor atau variabel yang berpengaruh pada hasil atau *output*. Namun, penggunaan jumlah *hidden layer* yang terlalu besar dapat menyebabkan *overfitting* dikarenakan model terlalu banyak mempelajari fitur yang serupa pada data belajar sehingga berdampak buruk saat melakukan pengujian dengan data baru [36]. Lalu pada buku Wang [29], telah dibuktikan bahwa Neural Network dengan 1 *hidden layer* dapat melakukan sebagian besar pekerjaan dengan baik. Untuk pengujian disarankan untuk menggunakan 1 *hidden layer* diawal, jika dinilai performanya belum baik dapat digunakan 2 *hidden layer*.

2. Jumlah *neuron* per *hidden layer*: 8, 16, 32, 64, 128, 256, dan 512.
   Merujuk kembali pada penelitian Sevdalina [17], tidak ada rumus untuk menentukan jumlah *neuron* pada *hidden layer*. Pernyataan tersebut didukung dengan buku Wang [29] yang mengharuskan dilakukannya beberapa percobaan dari peneliti untuk menemukan jumlah *neuron* pada *hidden layer* yang tepat. Berdasarkan percobaan yang dilakukan peneliti, penggunaan jumlah *neuron* yang kecil pada *hidden layer* (contoh: 1, 2, 3, 4, 5) mendapatkan hasil yang buruk. Seperti pada penelitian Imran [37], ketika menggunakan *neuron* dalam jumlah kecil (2, 3, 4, 5, dst), maka model tidak dapat mencapai konvergensi dan baru mendapat model yang konvergen dengan menggunakan 35 *neuron*. Oleh karena itu, peneliti menggunakan beberapa jumlah *neuron* yang besar pada *hidden layer* untuk melihat perbedaan hasilnya.

3. *Epoch*: 50 dan 100.
   Pada buku Rowel [16], beberapa model Neural Network dengan menggunakan *epoch* kecil dapat memperoleh hasil yang bagus (contoh: 10,

20, dst). Namun, hal tersebut kembali pada pengujian yang dilakukan apakah model sudah mencapai konvergensi atau belum. Penggunaan *learning rate* yang kecil membutuhkan *epoch* yang besar agar model dapat konvergen, begitu juga sebaliknya. Oleh karena itu, peneliti menggunakan *epoch* 50 dan 100 untuk melihat perbedaan hasilnya.

4. *Learning rate*: 0.001, 0.005, dan 0.01.
   Pada buku Rowel [16] direkomendasikan untuk melakukan percobaan terhadap *learning rate* dari besar ke kecil (contoh: 0.1 ke 0.001). Sementara pada penelitian Marco [30], menggunakan *learning rate* 0.001. Berdasarkan percobaan yang dilakukan peneliti, menggunakan *learning rate* 0.1 dan 0.05 menyebabkan model tidak dapat konvergen dan *loss*-nya besar. Oleh karena itu, peneliti menggunakan *learning rate* yang lebih kecil diantaranya 0.01, 0.005, dan 0.001.

4.3.4 Implementasi Pembelajaran Autoencoder

Pada bagian ini akan dilakukan pembelajaran Autoencoder menggunakan dataset yang telah dilakukan *preprocessing* sebelumnya. Pembelajaran Autoencoder akan menggunakan 80% data belajar yang komposisi kelasnya sama dengan data uji. Dari 80% data belajar tersebut juga dibagi kembali menjadi 60% untuk data belajar dan 20% untuk data validasi serta tetap mempertahankan komposisi yang sama antar kelas. Jumlah data belajar yaitu 119.726 data yang terdiri dari kelas *fraud* berjumlah 60.121 data dan kelas non-*fraud* berjumlah 59.605 data. Lalu jumlah data validasi yaitu 39.909 data yang terdiri dari kelas *fraud* berjumlah 19.907 data dan kelas non-*fraud* berjumlah 20.002 data. Setelah itu data belajar dan data validasi diambil hanya data berlabel non-*fraud* untuk proses belajar Autoencoder. Berikut ini merupakan proses yang dilakukan dalam pembelajaran Autoencoder:

Proses pembelajaran Autoencoder

1. Menetapkan jumlah *hidden layer encoder* dan *decoder*, jumlah *neuron* per *hidden layer encoder* dan *decoder*, jumlah *neuron* pada *code layer*, *epoch*, *learning rate* yang akan digunakan untuk pembelajaran.
2. Melakukan perhitungan menggunakan masukan dari data hasil *preprocessing* menggunakan algoritme *feed-forward* serta perhitungan *error* selama tahap pembelajaran.
3. Melakukan proses pembaruan bobot dengan menggunakan algoritme *backpropagation*.

4. Proses pembelajaran akan menggunakan kombinasi dari:

   Jumlah *hidden layer encoder* = 1 dan 2.

   Jumlah *hidden layer decoder* = 1 dan 2.

   Jumlah *neuron hidden layer encoder* = 2, 3 dan 4.

   Jumlah *neuron hidden layer decoder* = 2, 3 dan 4.

   Jumlah *neuron code layer* = 1, 2 dan 3.

5. Menyimpan hasil pembelajaran model Autoencoder sesuai kombinasi pada poin diatas.

4.3.4.1 Implementasi Pengujian Autoencoder

Pada bagian ini akan dilakukan pengujian Autoencoder dengan menggunakan 20% data uji yang komposisi kelasnya sama dengan data belajar. Jumlah data uji yaitu 39.909 data yang terdiri dari kelas *fraud* berjumlah 19.744 data dan kelas non-*fraud* berjumlah 20.165 data. Berikut ini merupakan proses yang dilakukan dalam pengujian Autoencoder:

Proses pengujian Autoencoder

1. Pengujian menggunakan 20% data uji.
2. Model hasil pembelajaran sebelumnya digunakan untuk proses pengujian.
3. Melakukan proses pengujian dengan menggunakan algoritme *feed-forward*.
4. Menghitung nilai *reconstruction error* menggunakan persamaan MSE 2.15 berdasarkan data masukan dan data keluaran.
5. Melakukan plot grafik *tradeoff* antara *precision* dan *recall* untuk menganalisa nilai *threshold*. Terdapat 5 nilai *threshold* yang ditetapkan oleh penulis, yaitu: 0.001, 0.01, 0.1, 0.5, dan 1.
6. Memetakan garis *threshold* terhadap *reconstruction error* untuk setiap data poin.
7. Menggunakan nilai *threshold* yang telah ditentukan untuk menghitung nilai *recall* dan *f1 measure* berdasarkan jumlah *true positive*, *true negative*, *false positive*, dan *false negative*.

4.3.4.2 Deskripsi *Hyperparameter* Autoencoder

Pada bagian ini akan dijelaskan pemilihan nilai *hyperparameter* yang digunakan pada pembelajaran Autoencoder. Pemilihan nilai *hyperparameter* yang dilakukan berdasarkan referensi buku atau *paper*, diantaranya:

1. Jumlah *hidden layer encoder* dan *decoder*: 1 dan 2.
   Autoencoder merupakan Neural Network sehingga dalam menentukan jumlah *hidden layer*-nya pun sama. Jumlah *hidden layer* diuji untuk melihat pengaruhnya terhadap hasil kompresi berupa *reconstruction error* yang dilakukan oleh Autoencoder. Pada penelitian M. A. Al-Shabi [6], menggunakan 1 *hidden layer* pada *encoder/decoder* dan sudah mencapai tujuan/hasil penelitian.

2. Jumlah *neuron hidden layer encoder* dan *decoder*: 2, 3, dan 4.
   Pada penelitian Marco [30], struktur *neuron* yang digunakan pada *encoder* jumlahnya mengecil sampai pada *code layer* dan membesar jumlahnya pada *decoder*. Lalu jumlah *neuron* pada *encoder* dan *decoder* pun simetris atau sama secara terbalik. Hal tersebut sesuai dengan ketentuan jumlah *neuron* pada *encoder* dan *decoder* pada Autoencoder [21]. Peneliti menguji semua kemungkinan kombinasi jumlah *neuron* pada *encoder* dan *decoder* sesuai jumlah *hidden layer* pada poin 1 diatas untuk melihat perbedaan hasil yang didapatkan antara kombinasi yang diuji.

3. Jumlah *neuron code layer*: 1, 2, dan 3.
   Berdasarkan kombinasi pada poin 1 dan 2 diatas yang disusun oleh penguji, dengan menggunakan *neuron hidden layer encoder* dan *decoder* yaitu 2, 3, dan 4. Serta ketentuan jumlah *neuron encoder* yang mengecil sampai *code layer* [21]. Maka jumlah *neuron* pada *code layer* yang dapat diuji yaitu 1, 2, dan 3.

4. *Threshold*: 0.001, 0.01, 0.1, 0.5, dan 1.
   Pada buku M. A. Al-Shabi [6], dilakukan pengujian dengan beberapa nilai *threshold* untuk mendapatkan hasil klasifikasi yang bagus. Salah satu cara untuk menentukan nilai *threshold* adalah dengan melihat grafik *tradeoff* antara *precision* dan *recall* seperti gambar 2.15 pada Bab 2. Berdasarkan percobaan yang dilakukan oleh peneliti sesuai dengan grafik *tradeoff precision* dan *recall*, diuji 5 nilai *threshold*, yaitu 0.001, 0.01, 0.1, 0.5, dan 1 untuk melihat perbedaan hasil klasifikasi terhadap setiap arsitektur Autoencoder.

4.4 Pengujian

Pada bab ini, pengujian dilakukan untuk mendapatkan hasil *recall* dan *f1 measure* yang terbaik dengan menentukan *hyperparameter* dan juga nilai-nilai yang dapat berpengaruh terhadap metode Neural Network dan Autoencoder. Pada tahap ini juga akan dibuktikan pengaruh kombinasi *hidden layer*, *neuron* per *hidden layer*, *learning rate*, dan *epoch* untuk mendapatkan hasil Neural Network yang optimal. Lalu kombinasi *hidden layer*, *neuron* per *hidden layer*, *learning rate*, dan *threshold* untuk Autoencoder.

Hasil pengujian *recall* dan *f1 measure* digunakan untuk menilai performa baik Neural Network dan Autoencoder dalam mendeteksi *click fraud*. *Recall* berperan untuk mengukur tingkat kesalahan dalam mengklasifikasikan *fraud* sebagai non-*fraud*. Semakin tinggi persentase nilai *recall* maka kesalahan mengklasifikasikan *fraud* sebagai non-*fraud* akan lebih sedikit jumlahnya, begitu pula sebaliknya. Selain itu, terdapat hasil *f1 measure* yang merupakan nilai rata-rata antara *precision* dan *recall*. *Precision* digunakan untuk mengukur tingkat kesalahan dalam mengklasifikasikan non-*fraud* sebagai *fraud*. Hasil pengujian dengan *hyperparameter* yang dinilai terbaik yaitu memiliki *recall* yang tinggi karena pada kasus *click fraud* diperlukan jumlah *false negative* yang sangat rendah.

4.4.1 Skenario Pengujian

Pada bagian ini, akan dijelaskan skenario pengujian yang dilakukan untuk membandingkan hasil deteksi antara Neural Network dengan Autoencoder.

4.4.1.1 Skenario Pengujian Neural Network

Pengujian terhadap Neural Network akan dibagi menjadi 1 *hidden layer* dan 2 *hidden layer*. Pada gambar 4.2 merupakan skenario pengujian 1 *hidden layer* yang berjumlah 42. Lalu pada gambar 4.3 merupakan skenario pengujian 2 *hidden layer* yang penerapan jumlah *neuron* pada *hidden layer* ke-2 lebih kecil atau sama terhadap *hidden layer* ke 1 sehingga skenarionya berjumlah 168. Maka, total skenario untuk pengujian Neural Network dengan 1 *hidden layer* dan 2 *hidden layer* adalah 210.

Kombinasi	*Neuron Hidden Layer*	*Learning Rate*	*Epoch*
1	8	0.001	50
2	16	0.005	100
3	32	0.01	
4	64		
5	128		
6	256		
7	512		
Jumlah Skenario (7 Kombinasi × 3 *Learning Rate* × 2 *Epoch*)			42

Gambar 4.2 Skenario Pengujian Neural Network 1 *Hidden Layer*

Kombinasi	*Neuron*		*Learning Rate*	*Epoch*
	Hidden Layer 1	*Hidden Layer 2*		
1	8	8	0.001	50
2	16	8	0.005	100
3	16	16	0.01	
4	32	8		
5	32	16		
6	32	32		
7	64	8		
8	64	16		
9	64	32		
10	64	64		
11	128	8		
12	128	16		
13	128	32		
14	128	64		
15	128	128		
16	256	8		
17	256	16		
18	256	32		
19	256	64		
20	256	128		
21	256	256		
22	512	8		
23	512	16		
24	512	32		
25	512	64		
26	512	128		
27	512	256		
28	512	512		
Jumlah Skenario (28 Kombinasi × 3 *Learning Rate* × 2 *Epoch*)				168

Gambar 4.3 Skenario Pengujian Neural Network 2 *Hidden Layer*

4.4.1.2 Skenario Pengujian Autoencoder

Pengujian terhadap Autoencoder akan dibagi menjadi 1 *hidden layer* dan 2 *hidden layer* dengan jumlah *neuron* yang sama sesuai aturannya baik pada *encoder* dan *decoder*. Pada gambar 4.4 merupakan skenario pengujian 1 *hidden layer* dengan kombinasi *neuron* pada *hidden layer encoder* dan *hidden layer decoder* yang sama beserta *threshold* yang digunakan, maka jumlah skenarionya adalah 30. Sementara pada gambar 4.5 merupakan skenario pengujian 2 *hidden layer* dan dengan tetap mempertahankan jumlah *neuron* yang sama pada *hidden layer encoder* dan *hidden layer decoder* serta penerapan *threshold*, maka jumlah skenarionya adalah 20. Total skenario pengujian Autoencoder adalah 50.

Kombinasi	Neuron			Threshold
	Hidden Layer Encoder	Code Layer	Hidden Layer Decoder	
1	4	3	4	0.001
2	4	2	4	0.01
3	4	1	4	0.1
4	3	2	3	0.5
5	3	1	3	1
6	2	1	2	
Jumlah Skenario (6 Kombinasi × 5 *Threshold*)				30

Gambar 4.4 Skenario Pengujian Autoencoder 1 *Hidden Layer*

Kombinasi	Neuron					Threshold
	Hidden Layer 1 Encoder	Hidden Layer 2 Encoder	Code Layer	Hidden Layer 1 Decoder	Hidden Layer 2 Decoder	
1	4	3	2	3	4	0.001
2	4	3	1	3	4	0.01
3	4	2	1	2	4	0.1
4	3	2	1	2	3	0.5
						1
Jumlah Skenario (4 Kombinasi × 5 *Threshold*)						20

Gambar 4.5 Skenario Pengujian Autoencoder 2 *Hidden Layer*

4.4.2 Pengujian Neural Network

Pada bagian ini, akan dijelaskan mengenai hasil pengujian pada Neural Network yang menggunakan kombinasi jumlah *hidden layer*, jumlah *neuron* per *hidden layer*, *learning rate*, dan *epoch* sesuai yang telah dijelaskan sebelumnya.

4.4.2.1 Pengujian 1 *Hidden Layer*

Pengujian pertama, pada tabel 4.12 adalah pengujian Neural Network menggunakan 1 *hidden layer* yang dikombinasikan dengan *neuron* berjumlah 8, 16, 32, 64, 128, 256, dan 512, lalu *learning rate* 0.001, 0.005, dan 0.01, serta *epoch* 50 dan 100. Hasil pengujian yaitu *precision*, *recall*, dan *f1 measure* yang dihitung berdasarkan jumlah *true positive*, *true negative*, *false positive*, dan *false negative* hasil dari *confusion matrix*.

Tabel 4.12 Hasil Pengujian Neural Network Dengan 1 Hidden Layer

Neuron Hidden Layer	***Learning Rate***	***Epoch***	**Hasil**		
			***Precision* (%)**	***Recall* (%)**	***F1 Measure* (%)**
8	**0.001**	**50**	71.50	74.63	73.03
		100	78.08	73.43	75.68
	0.005	**50**	74.27	70.74	72.46
		100	73.85	65.05	69.17
	0.01	**50**	79.14	90.40	84.40
		100	75.41	75.41	65.07
16	**0.001**	**50**	70.55	67.52	69.01
		100	70.67	69.09	69.87
	0.005	**50**	83.93	68.67	75.53
		100	81.67	77.60	79.58
	0.01	**50**	82.15	88.18	85.06
		100	84.74	63.76	72.77
32	**0.001**	**50**	82.69	72.16	77.06
		100	82.85	77.16	79.90
	0.005	**50**	79.25	84.27	81.68
		100	85.19	78.97	81.96
	0.01	**50**	85.42	72.57	78.47
		100	83.16	77.42	80.19
	0.001	**50**	86.24	77.73	81.76
		100	88.56	83.49	85.95

Tabel 4.12 Hasil Pengujian Neural Network Dengan 1 Hidden Layer (Lanjutan)

Neuron Hidden Layer	***Learning Rate***	***Epoch***	**Hasil**		
			***Precision* (%)**	***Recall* (%)**	***F1 Measure* (%)**
	0.005	**50**	84.37	84.08	84.23
		100	88.16	80.84	84.34
	0.01	**50**	87.69	84.11	85.86
		100	88.43	86.38	87.39
128	**0.001**	**50**	89.57	78.33	83.57
		100	89.13	84.82	86.92
	0.005	**50**	87.93	84.90	86.39
		100	92.30	89.39	90.82
	0.01	**50**	87.39	76.78	81.74
		100	83.63	**92.84**	87.99
256	**0.001**	**50**	93.79	79.31	85.95
		100	90.10	87.26	88.66
	0.005	**50**	88.88	84.44	86.61
		100	91.74	89.76	90.74
	0.01	**50**	88.51	86.85	87.67
		100	86.10	90.94	88.46
512	**0.001**	**50**	88.63	87.86	88.24
		100	92.54	90.07	91.29
	0.005	**50**	88.52	87.21	87.86
		100	93.93	87.15	90.41
	0.01	**50**	84.02	83.85	83.94
		100	91.84	83.64	87.55

Pada tabel 4.12 diatas, dapat dilihat bahwa Neural Network dengan menggunakan 1 *hidden layer* menunjukkan hasil yang cukup baik. Nilai *learning rate* sangat berpengaruh pada performa Neural Network dalam memperbaharui nilai bobot pada setiap *epoch*. Hasil *recall* tertinggi didapatkan dengan kombinasi *neuron* 128, *learning rate* 0.01, dan *epoch* 100 yang mendapatkan *recall* 92.84% dan *f1 measure* 87.99%.

4.4.2.2 Pengujian 2 *Hidden Layer*

Selanjutnya pada pengujian kedua yaitu pengujian Neural Network menggunakan 2 *hidden layer* seperti pada tabel 4.13. Pada pengujian tersebut digunakan *neuron* berjumlah 8, 16, 32, 64, 128, 256, dan 512, lalu *learning rate* 0.001, 0.005, dan 0.01, serta *epoch* 50 dan 100. Hasil pengujian sama seperti 1 *hidden layer*, yaitu *precision*, *recall*, dan *f1 measure* yang dihitung berdasarkan jumlah *true positive*, *true negative*, *false positive*, dan *false negative* hasil dari *confusion matrix*.

Tabel 4.13 Hasil Pengujian Neural Network Dengan 2 Hidden Layer

Neuron Hidden Layer 1	***Neuron Hidden Layer 2***	***Learning Rate***	***Epoch***	**Hasil**		
				***Precision* (%)**	***Recall* (%)**	***F1* Measure (%)**
8	**8**	**0.001**	**50**	77.84	69.88	73.64
			100	76.90	72.09	74.42
		0.005	**50**	78.20	89.96	83.67
			100	82.17	80.48	81.32
		0.01	**50**	83.42	70.38	76.34
			100	80.19	92.84	86.05
16	**8**	**0.001**	**50**	82.88	82.54	82.71
			100	84.77	82.23	83.48
		0.005	**50**	87.15	83.76	85.42
			100	86.31	89.35	87.80
		0.01	**50**	86.85	79.03	82.76
			100	85.14	84.80	84.97
16	**16**	**0.001**	**50**	86.27	83.54	84.88
			100	90.60	78.44	84.08
		0.005	**50**	88.16	84.17	86.12
			100	92.34	78.28	84.73
		0.01	**50**	83.45	84.47	83.95
			100	83.66	88.53	86.03
		0.001	**50**	89.72	79.76	84.45
			100	84.18	83.34	86.68
32	**8**					

Tabel 4.13 Hasil Pengujian Neural Network Dengan 2 Hidden Layer (Lanjutan)

Neuron Hidden Layer 1	***Neuron Hidden Layer 2***	***Learning Rate***	***Epoch***	**Hasil**		
				***Precision* (%)**	***Recall* (%)**	***F1* Measure (%)**
		0.005	**50**	90.39	85.59	87.92
			100	88.77	88.10	88.43
		0.01	**50**	91.58	81.79	86.41
			100	92.17	80.98	86.21
32	**16**	**0.001**	**50**	87.05	87.68	87.36
			100	89.86	85.87	87.82
		0.005	**50**	89.87	87.67	88.75
			100	89.42	88.17	88.79
		0.01	**50**	89.72	84.59	87.08
			100	91.58	92.71	92.14
32	**32**	**0.001**	**50**	90.06	89.29	89.67
			100	91.22	91.18	91.20
		0.005	**50**	93.51	87.21	90.25
			100	94.39	93.29	93.83
		0.01	**50**	95.15	81.60	87.85
			100	92.16	87.53	89.78
64	**8**	**0.001**	**50**	92.39	83.04	84.47
			100	92.02	92.42	92.22
		0.005	**50**	90.89	87.74	89.29
			100	94.91	89.09	91.91
		0.01	**50**	95.08	81.01	87.48
			100	89.99	89.89	89.94
64	**16**	**0.001**	**50**	93.23	85.02	88.94
			100	92.52	89.92	90.68
		0.005	**50**	95.04	87.58	91.16
			100	94.01	91.39	92.68
		0.01	**50**	93.81	86.22	89.86

Tabel 4.13 Hasil Pengujian Neural Network Dengan 2 Hidden Layer (Lanjutan)

Neuron Hidden Layer 1	***Neuron Hidden Layer 2***	***Learning Rate***	***Epoch***	**Hasil**		
				***Precision* (%)**	***Recall* (%)**	***F1* Measure (%)**
			100	96.58	83.38	89.50
64	**32**	**0.001**	**50**	92.86	86.57	89.60
			100	91.21	93.71	92.44
		0.005	**50**	94.04	91.93	92.97
			100	96.04	92.46	94.22
		0.01	**50**	94.43	89.22	91.75
			100	90.53	89.93	90.23
64	**64**	**0.001**	**50**	91.40	91.92	91.66
			100	97.14	89.83	93.34
		0.005	**50**	95.73	90.83	93.22
			100	97.99	92.94	95.40
		0.01	**50**	93.31	90.59	91.93
			100	96.40	88.84	92.47
128	**8**	**0.001**	**50**	93.96	85.27	89.40
			100	93.10	90.69	91.88
		0.005	**50**	94.20	88.16	91.08
			100	93.75	90.95	92.33
		0.01	**50**	95.06	87.65	91.21
			100	92.17	88.24	90.16
128	**16**	**0.001**	**50**	94.62	90.39	92.46
			100	95.66	92.47	94.04
		0.005	**50**	91.28	91.23	91.25
			100	96.17	90.85	93.43
		0.01	**50**	93.86	87.89	90.77
			100	95.02	90.04	92.46
128	**32**	**0.001**	**50**	96.42	88.03	92.04
			100	97.22	93.03	95.08

Tabel 4.13 Hasil Pengujian Neural Network Dengan 2 Hidden Layer (Lanjutan)

Neuron Hidden Layer 1	***Neuron Hidden Layer 2***	***Learning Rate***	***Epoch***	**Hasil**		
				***Precision* (%)**	***Recall* (%)**	***F1* Measure (%)**
		0.005	**50**	94.53	90.43	92.44
			100	96.18	90.19	93.09
		0.01	**50**	95.17	87.91	91.39
			100	95.37	90.34	92.79
128	**64**	**0.001**	**50**	96.91	90.44	93.56
			100	97.13	92.67	94.85
		0.005	**50**	95.69	91.34	93.46
			100	94.47	94.60	94.53
		0.01	**50**	91.41	83.53	87.29
			100	97.39	91.11	94.14
128	**128**	**0.001**	**50**	94.64	92.93	93.77
			100	93.69	96.11	94.88
		0.005	**50**	95.27	90.64	92.90
			100	96.99	93.77	95.35
		0.01	**50**	97.04	85.68	91.01
			100	95.07	92.20	93.61
256	**8**	**0.001**	**50**	90.67	93.13	91.89
			100	94.30	94.83	94.57
		0.005	**50**	93.43	87.83	90.55
			100	94.28	92.00	93.13
		0.01	**50**	92.79	87.10	89.85
			100	93.70	90.47	92.05
256	**16**	**0.001**	**50**	94.67	92.53	93.59
			100	97.31	93.49	95.36
		0.005	**50**	93.39	92.24	92.81
			100	97.31	93.34	95.28
		0.01	**50**	92.23	86.95	89.98

Tabel 4.13 Hasil Pengujian Neural Network Dengan 2 Hidden Layer (Lanjutan)

Neuron Hidden Layer 1	***Neuron Hidden Layer 2***	***Learning Rate***	***Epoch***	**Hasil**		
				***Precision* (%)**	***Recall* (%)**	***F1* Measure (%)**
			100	95.08	90.04	92.49
256	**32**	**0.001**	**50**	95.01	93.43	94.22
			100	94.99	95.08	95.03
		0.005	**50**	96.71	91.60	94.09
			100	97.58	91.79	94.60
		0.01	**50**	93.14	90.81	91.96
			100	94.71	91.69	93.18
256	**64**	**0.001**	**50**	96.25	93.59	94.90
			100	97.55	96.04	96.79
		0.005	**50**	95.42	90.35	92.81
			100	97.37	94.43	95.87
		0.01	**50**	95.34	89.11	92.12
			100	96.02	88.65	92.18
256	**128**	**0.001**	**50**	96.50	95.22	95.86
			100	97.66	95.80	96.72
		0.005	**50**	95.51	92.88	94.18
			100	98.86	91.59	95.09
		0.01	**50**	94.44	88.96	91.62
			100	94.19	89.21	91.63
256	**256**	**0.001**	**50**	96.26	94.40	95.32
			100	97.57	96.14	96.85
		0.005	**50**	97.70	92.22	94.88
			100	97.89	94.55	96.19
		0.01	**50**	94.65	91.58	93.09
			100	96.67	91.12	93.81
512	**8**	**0.001**	**50**	94.82	91.69	93.23
			100	97.32	92.40	94.80

Tabel 4.13 Hasil Pengujian Neural Network Dengan 2 Hidden Layer (Lanjutan)

Neuron Hidden Layer 1	***Neuron Hidden Layer 2***	***Learning Rate***	***Epoch***	**Hasil**		
				***Precision* (%)**	***Recall* (%)**	***F1* Measure (%)**
		0.005	**50**	97.33	84.95	90.72
			100	95.66	90.86	93.20
		0.01	**50**	92.26	86.56	89.32
			100	92.87	93.87	93.37
512	**16**	**0.001**	**50**	94.55	91.92	93.22
			100	97.16	94.93	96.03
		0.005	**50**	94.22	91.65	92.92
			100	98.05	91.59	94.71
		0.01	**50**	95.29	86.79	90.84
			100	97.48	89.17	93.14
512	**32**	**0.001**	**50**	97.96	91.14	94.43
			100	98.13	93.65	95.84
		0.005	**50**	97.09	91.73	94.33
			100	95.66	94.22	94.93
		0.01	**50**	90.85	91.05	90.95
			100	96.05	91.42	93.68
512	**64**	**0.001**	**50**	97.62	93.34	95.44
			100	97.29	96.15	96.72
		0.005	**50**	97.02	89.24	92.96
			100	95.90	93.71	94.79
		0.01	**50**	92.98	91.30	92.13
			100	97.67	89.64	93.45
512	**128**	**0.001**	**50**	96.45	95.59	96.02
			100	97.08	96.20	96.64
		0.005	**50**	96.92	92.95	94.89
			100	98.02	93.46	95.69
		0.01	**50**	96.70	88.94	92.66

Tabel 4.13 Hasil Pengujian Neural Network Dengan 2 Hidden Layer (Lanjutan)

Neuron Hidden Layer 1	***Neuron Hidden Layer 2***	***Learning Rate***	***Epoch***	**Hasil**		
				Precision **(%)**	***Recall*** **(%)**	***F1*** **Measure (%)**
			100	95.11	89.12	92.02
512	**256**	**0.001**	**50**	98.31	94.59	96.41
			100	98.04	96.12	97.07
		0.005	**50**	98.03	91.30	94.55
			100	97.62	94.10	95.83
		0.01	**50**	90.50	92.52	91.50
			100	90.98	93.29	92.12
512	**512**	**0.001**	**50**	97.65	94.52	96.06
			100	97.41	**96.35**	96.88
		0.005	**50**	97.90	91.76	94.73
			100	98.40	92.80	95.52
		0.01	**50**	93.61	89.69	91.61
			100	97.33	89.92	93.48

Pada pengujian menggunakan 2 *hidden layer*, hasil yang didapatkan akan bagus jika menggunakan jumlah *neuron* yang besar. Dengan catatan, penggunaan jumlah *neuron* yang besar dapat memperlambat waktu komputasi. Penggunaan *learning rate* dan *epoch* kembali menjadi faktor yang sangat berpengaruh terhadap hasil yang didapatkan. Dengan *learning rate* rendah (0.001) dan *epoch* yang besar (100), Neural Network dapat menemukan nilai bobot yang optimal sehingga model dapat memprediksi dengan baik. Hasil *recall* tertinggi pada pengujian 2 *hidden layer* yaitu dengan kombinasi *hidden layer* 1 menggunakan 512 *neuron*, *hidden layer* 2 menggunakan 512 *neuron*, *learning rate* 0.001, dan *epoch* 100 yang mendapat *recall* 96.35% dan *f1 measure* 96.88%.

4.4.3 Pengujian Autoencoder

Pada bagian ini, akan dijelaskan mengenai hasil pengujian Autoencoder yang menggunakan kombinasi jumlah *hidden layer*, jumlah *neuron* per *hidden layer*, jumlah *neuron* pada *code layer*, dan *threshold* untuk membatasi data *fraud* dan non-*fraud*.

4.4.3.1 Pengujian 1 *Hidden Layer*

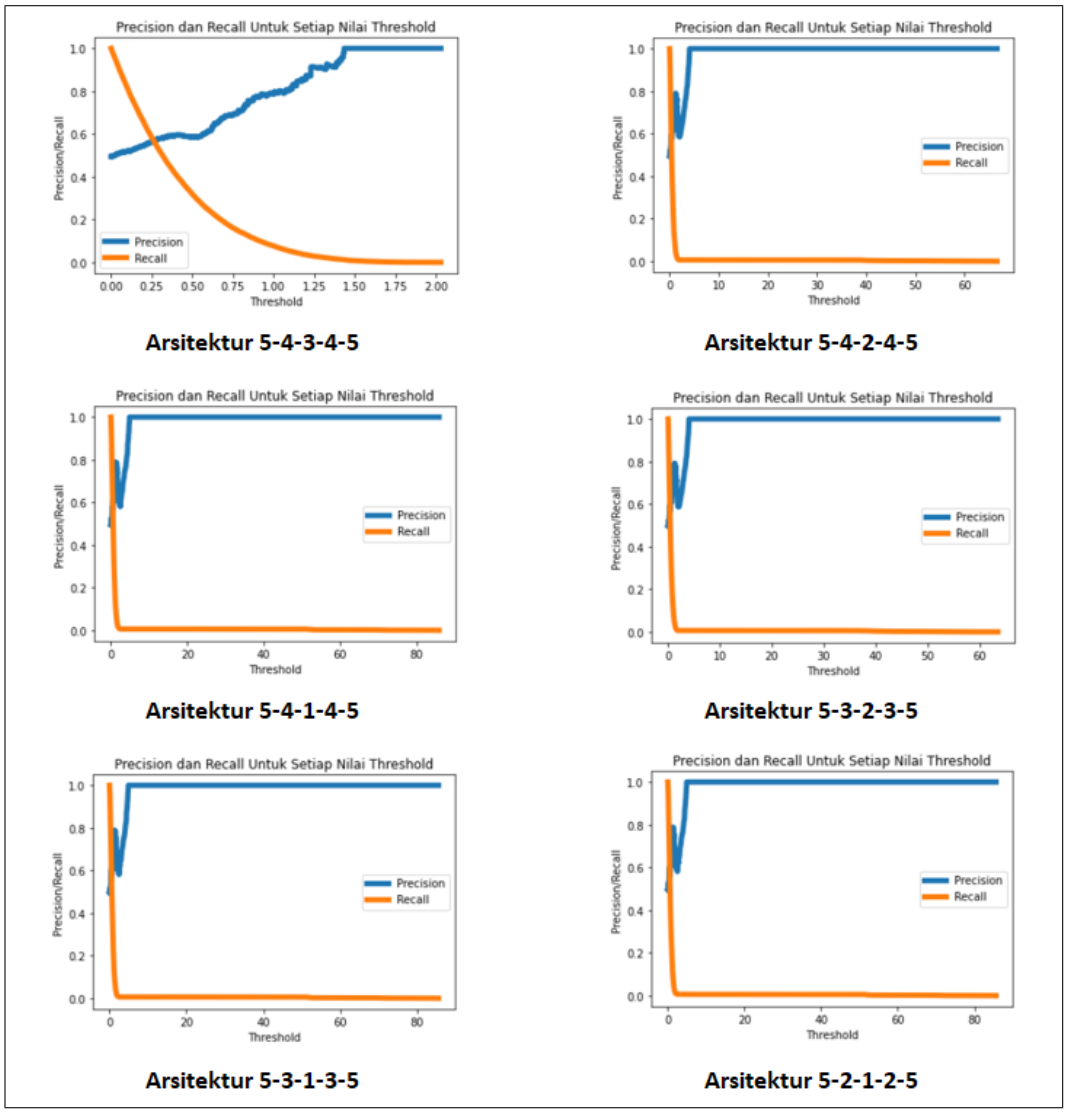
Seperti pada pengujian Neural Network, pengujian Autoencoder dibagi menjadi 1 *hidden layer* dan 2 *hidden layer*. Pada penerapannya, jumlah *neuron* pada *hidden layer encoder* dan *hidden layer decoder* jumlahnya harus simetris. Oleh karena itu, berdasarkan 5 fitur yang ada, maka dapat dilakukan pengujian terhadap 6 kombinasi arsitektur untuk 1 *hidden layer* seperti pada gambar 4.4. Arsitektur yang diuji dengan urutan *neuron* (*input layer - hidden layer encoder - code layer - hidden layer decoder - output layer*) diantaranya: 5-4-3-4-5, 5-4-2-4-5, 5-4-1-4-5, 5-3-2-3-5, 5-3-1-3-5, dan 5-2-1-2-5. Sementara jumlah *neuron input layer* dan *neuron output layer* sama dengan jumlah fitur yang digunakan yaitu 5. Hasil pertama yang didapatkan dari proses pengujian adalah nilai *reconstruction error* untuk masing-masing arsitektur, seperti pada tabel 4.14.

Tabel 4.14 *Reconstruction error* Pengujian Autoencoder 1 *Hidden Layer*

Jumlah *Neuron*					***Reconstruction Error***
Input Layer	***Hidden Layer Encoder***	***Code Layer***	***Hidden Layer Decoder***	***Output Layer***	
5	**4**	**3**	**4**	**5**	0.325
5	**4**	**2**	**4**	**5**	0.521
5	**4**	**1**	**4**	**5**	0.729
5	**3**	**2**	**3**	**5**	0.521
5	**3**	**1**	**3**	**5**	0.729
5	**2**	**1**	**2**	**5**	0.729

Pada tabel *reconstruction error* diatas, arsitektur yang memiliki kompresi minimal (3 *neuron* di *code layer*) yaitu arsitektur 5-4-3-4-5 mendapatkan *reconstruction error* sebesar 0.325. Hal tersebut dikarenakan tidak banyak informasi yang hilang dari hasil *encoding* dan *decoding*. Kompresi tersebut mencoba merepresentasikan masukkan 5 *neuron* ke dalam 3 *neuron*, sehingga nilai *reconstruction error* yang didapatkan paling rendah. Berbeda jika dibandingkan dengan arsitektur yang memiliki kompresi maksimal seperti arsitektur 5-2-1-2-5. Kompresi tersebut mencoba merepresentasikan masukkan 5 *neuron* ke dalam 1 *neuron* sehingga banyak informasi yang hilang saat proses *encoding* dan *reconstruction error* yang dihasilkan pun tinggi yaitu 0.729.

Neuron pada *code layer* layer memiliki korelasi dengan *reconstruction error* yang dihasilkan model. Jika digunakan 1 *neuron* pada *code layer* maka *reconstruction error* yang dihasilkan paling tinggi yaitu 0.729. Lalu, jika digunakan 2 *neuron* pada *code layer* maka *reconstruction error* yang dihasilkan menurun menjadi 0.521. Sedangkan, jika digunakan 3 *neuron* pada *code layer* maka *reconstruction error* menjadi paling rendah yaitu 0.325. Selanjutnya, pada gambar 4.6 dilakukan plot grafik *tradeoff* antara *precision* dan *recall* terhadap masing-masing arsitektur untuk menentukan nilai *threshold*. Nilai *threshold* digunakan untuk membatasi antara data *fraud* dan non-*fraud*.



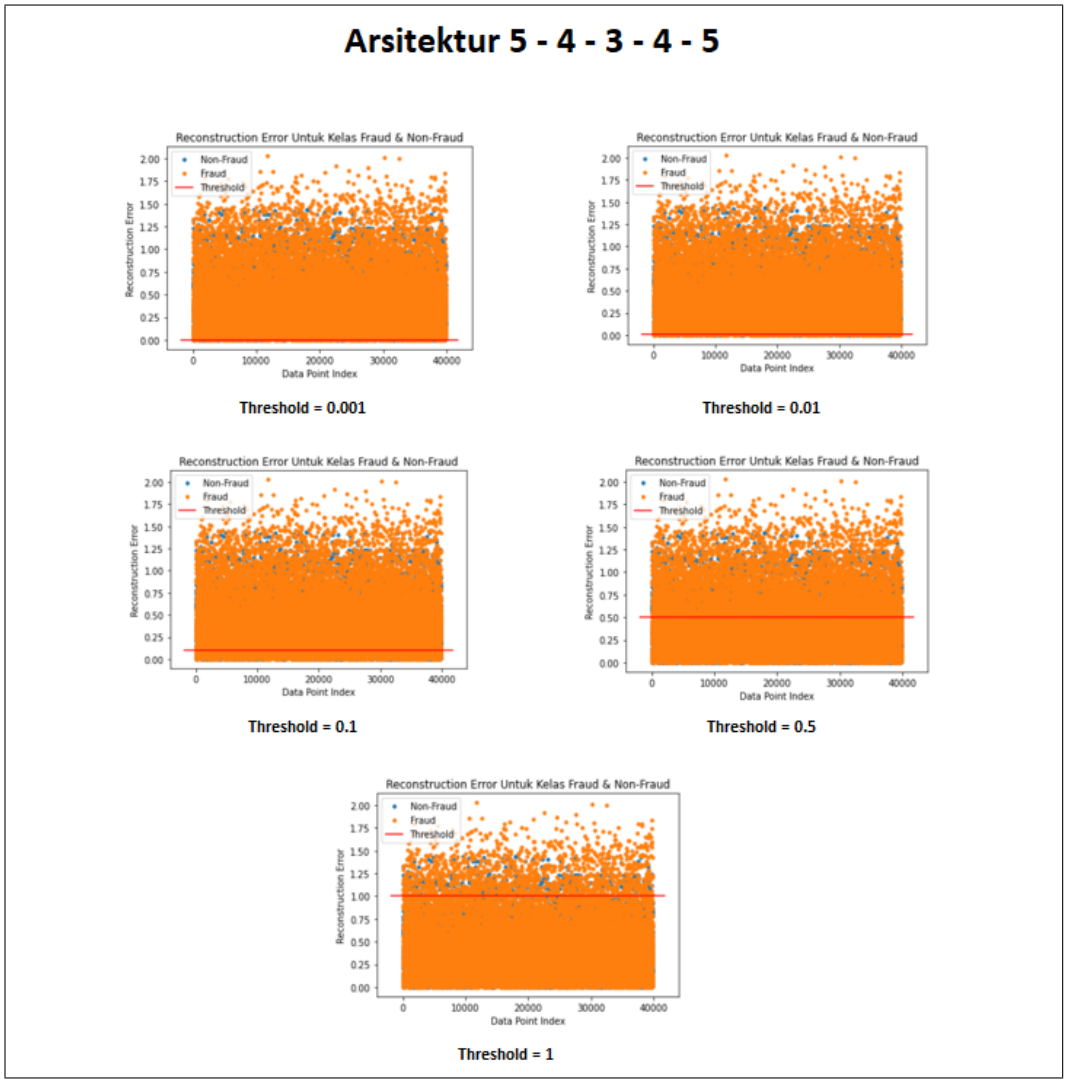
Gambar 4.6 Grafik *Tradeoff* Antara *Precision* Dan *Recall*

Pada gambar grafik *tradeoff* antara *precision* dan *recall* diatas, dapat disimpulkan bahwa dalam menentukan nilai *threshold* harus memilih antara *precision* yang tinggi namun *recall* rendah, atau *recall* tinggi namun *precision* rendah. Semakin

kecil nilai *threshold* atau mendekati 0, maka *recall* akan semakin tinggi dan *precision* rendah. Sebaliknya, semakin besar nilai *threshold* maka *precision* akan semakin tinggi dan *recall* rendah.

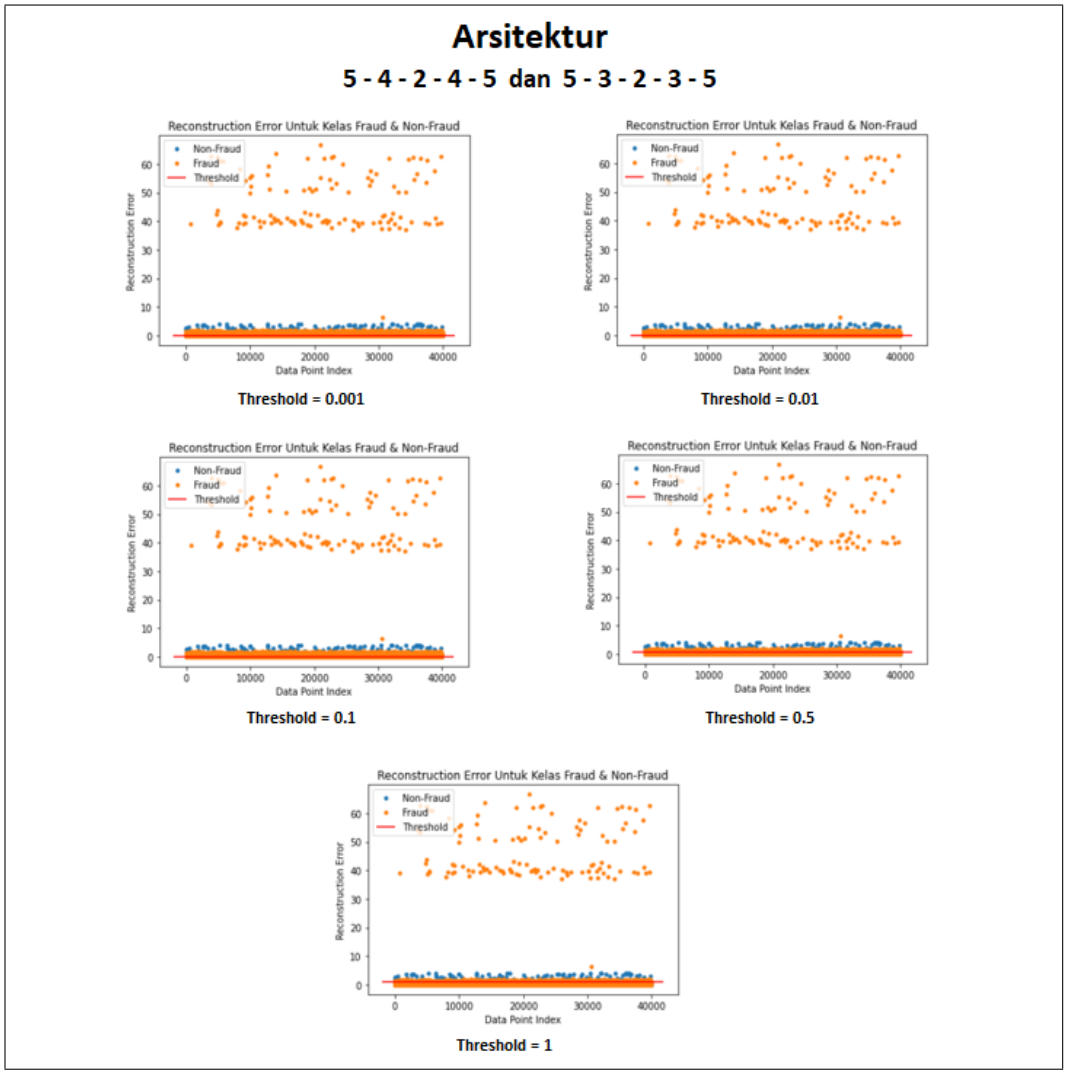
Kesimpulan lain yang dapat diambil dari gambar grafik diatas yaitu hubungannya dengan *reconstruction error* pada setiap arsitektur. Pada arsitektur yang menggunakan 3 *neuron code layer* (5-4-3-4-5) dengan *reconstruction error* yang paling rendah yaitu 0.325, nilai *threshold* paling tinggi berada dikisaran 2. Hal tersebut mengindikasikan bahwa nilai *reconstruction error* untuk data *fraud* paling tinggi berada dikisaran 2. Selanjutnya, pada arsitektur dengan 2 *neuron code layer* (5-4-2-4-5 dan 5-3-2-3-5) dengan *reconstruction error* sebesar 0.521, nilai *threshold* tertinggi berada dikisaran 60. Hal tersebut berarti *reconstruction error* untuk data *fraud* tertinggi berada dikisaran 60. Lalu sama halnya pada arsitektur dengan 1 *neuron code layer* (5-4-1-4-5, 5-3-1-3-5, dan 5-2-1-2-5) yang memiliki *reconstruction error* paling tinggi yaitu 0.729, nilai *threshold* tertinggi pada kisaran 80 yang berarti *reconstruction error* data *fraud* tertinggi yaitu dikisaran 80.

Merujuk pada kasus *click fraud* yang memerlukan nilai *recall* tinggi, maka digunakan nilai *threshold* kecil atau mendekati 0. Nilai *threshold* yang diuji yaitu: 0.001, 0.01, 0.1, 0.5, dan 1. Kelima nilai *threshold* tersebut diuji agar dapat diketahui pengaruhnya terhadap hasil deteksi pada setiap arsitektur. Dengan menggunakan nilai *threshold* kecil (mendekati 0) maka *recall* akan semakin tinggi. Berikutnya dilakukan pemetaan garis *threshold* setiap arsitektur terhadap *reconstruction error* untuk masing-masing data poin baik *fraud* maupun non-*fraud*. Pertama dimulai dengan arsitektur 5-4-3-4-5 seperti pada gambar 4.7.



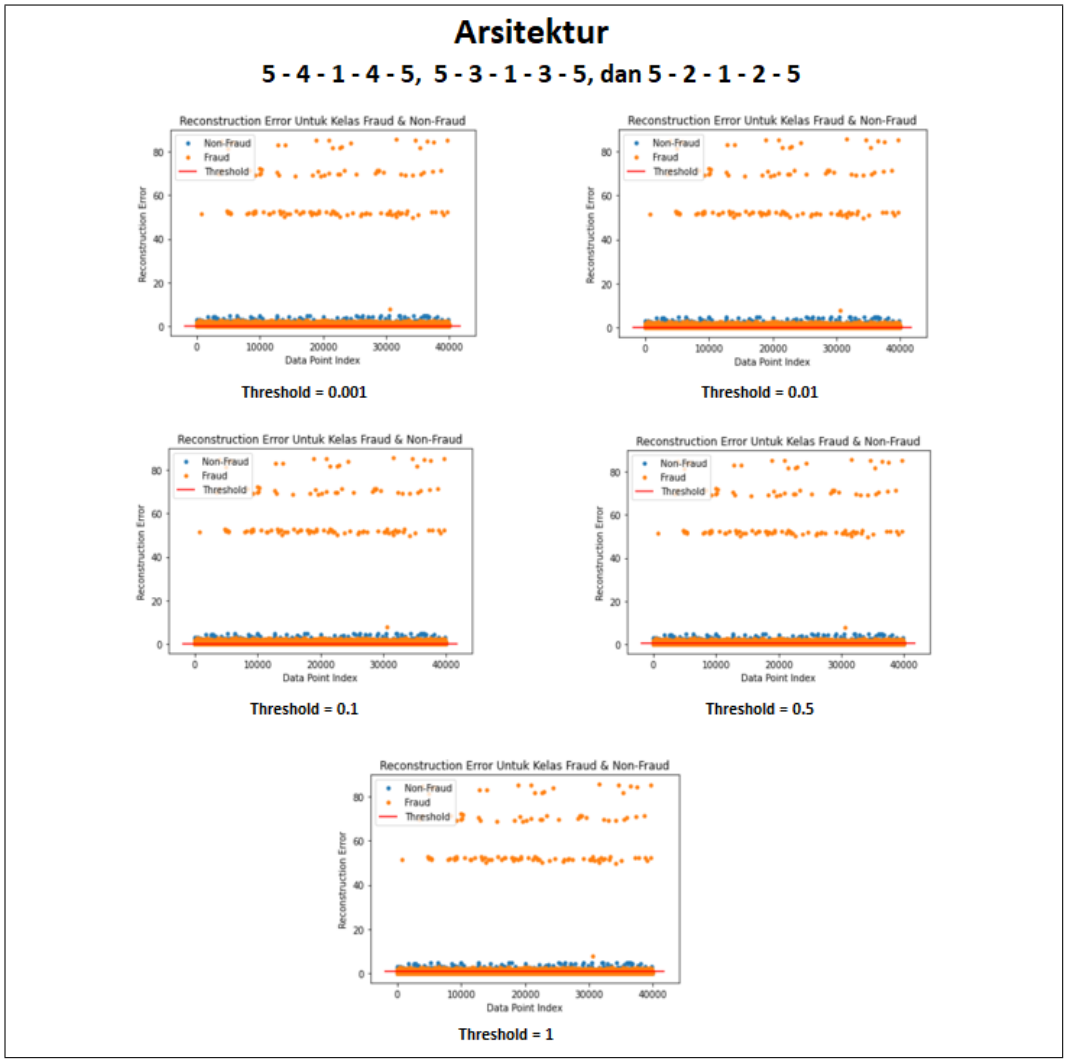
Gambar 4.7 Pemetaan Garis *Threshold* Pada Arsitektur 5-4-3-4-5

Pada gambar 4.7 diatas, dapat dilihat penggunaan nilai *threshold* yang besar yaitu 1 membuat model deteksi salah mengklasifikasikan banyak data *fraud* sehingga menyebabkan nilai *recall* akan rendah. Kemudian, pada arsitektur 5-4-2-4-5 dan 5-3-2-3-5 yang memiliki jumlah *neuron code layer* sama yaitu 2, maka nilai *reconstruction error* tertingginya pun sama. Sehingga pemetaan garis *threshold* untuk kedua arsitektur tersebut sama seperti pada gambar 4.8.



Gambar 4.8 Pemetaan Garis *Threshold* Pada Arsitektur 5-4-2-4-5 dan 5-3-2-3-5

Lalu, pada gambar 4.8 diatas, penggunaan nilai *threshold* besar yaitu 1 juga dapat membuat model deteksi salah mengklasifikasikan banyak data *fraud*. Selanjutnya, sama halnya pada arsitektur 5-4-1-4-5, 5-3-1-3-5 dan 5-2-1-2-5 yang memiliki jumlah *neuron code layer* sama yaitu 1, maka nilai *reconstruction error*-nya pun sama. Sehingga pemetaan garis *threshold* untuk ketiga arsitektur tersebut sama seperti pada gambar 4.9.



Gambar 4.9 Pemetaan Garis *Threshold* Pada Arsitektur 5-4-1-4-5, 5-3-1-3-5 dan 5-2-1-2-5

Dapat dilihat pada ketiga gambar pemetaan garis *threshold* terhadap *reconstruction error* setiap data poin 4.7, 4.8, dan 4.9 diatas, distribusi *reconstruction error* data *fraud* dan *non-fraud* kebanyakan saling tumpang tindih dibawah atau mendekati nilai 0. Sementara hanya sedikit distribusi *reconstruction error* data *fraud* yang nilainya besar dan tidak menimpa data non-*fraud*.

Diakhir, setelah dilakukan pemetaan garis *threshold* terhadap *reconstruction error*, nilai *threshold* digunakan untuk menghitung nilai *precision*, *recall*, dan *f1 measure* berdasarkan jumlah *true positive*, *true negative*, *false positive*, dan *false negative* hasil dari *confusion matrix*. Hasil akhir pengujian Autoencoder dengan 1 *hidden layer* dapat dilihat pada tabel 4.15.

Tabel 4.15 Hasil Pengujian Autoencoder Dengan 1 *Hidden Layer*

Jumlah *Neuron*					***Thres-hold***	**Hasil**		
Input Layer	***Hidden Layer Encoder***	***Code Layer***	***Hidden Layer Encoder***	***Output Layer***		***Precision* (%)**	***Recall* (%)**	***F1 Measure* (%)**
5	**4**	**3**	**4**	**5**	**0.001**	49.49	**99.79**	**66.16**
					0.01	49.50	97.97	65.77
					0.1	51.98	81.59	63.51
					0.5	58.78	31.87	41.33
					1	78.96	7.73	14.09
5	**4**	**2**	**4**	**5**	**0.001**	49.47	99.98	66.19
					0.01	49.57	**99.77**	**66.23**
					0.1	51.16	91.87	65.72
					0.5	60.37	40.90	48.76
					1	75.10	9.73	17.23
5	**4**	**1**	**4**	**5**	**0.001**	49.47	100.00	66.20
					0.01	49.54	**99.84**	**66.22**
					0.1	49.95	95.84	65.67
					0.5	57.96	58.79	58.38
					1	74.82	22.08	34.09
5	**3**	**2**	**3**	**5**	**0.001**	49.47	99.99	66.20
					0.01	49.57	**99.72**	**66.22**
					0.1	51.15	91.94	65.73
					0.5	60.62	41.23	49.08
					1	75.35	9.88	17.46
5	**3**	**1**	**3**	**5**	**0.001**	49.47	100.00	66.20
					0.01	49.54	**99.84**	**66.22**
					0.1	49.95	95.84	65.67
					0.5	57.97	58.82	58.39
					1	74.79	22.07	34.08
5	**2**	**1**	**2**	**5**	**0.001**	49.47	100.00	66.20
					0.01	49.54	**99.83**	**66.22**

Tabel 4.15 Hasil Pengujian Autoencoder Dengan 1 Hidden Layer (Lanjutan)

Jumlah *Neuron*						**Hasil**		
Input Layer	***Hidden Layer Encoder***	***Code Layer***	***Hidden Layer Encoder***	***Output Layer***	***Thres -hold***	***Precision* (%)**	***Recall* (%)**	***F1 Measure* (%)**
					0.1	49.97	95.83	65.69
					0.5	58.02	58.80	58.41
					1	74.71	22.04	34.04

Berdasarkan hasil pengujian Autoencoder dengan 1 *hidden layer* diatas dapat disimpulkan bahwa penggunaan kelima nilai *threshold* pada setiap arsitektur dengan *reconstruction error* 0.325, 0.521, dan 0.729 akan berbeda hasilnya. Pada arsitektur 5-4-3-4-5 dengan *reconstruction error* 0.325 hasilnya paling bagus jika menggunakan *threshold* 0.001 yang hasilnya mendapatkan *recall* 99.79% dan *f1 measure* 66.16%. Lalu pada arsitektur 5-4-2-4-5 dan 5-3-2-3-5 dengan *reconstruction error* 0.521, jika menggunakan *threshold* 0.001 akan terlalu rendah, memang hasilnya mencapai *recall* 99.99% namun hasil tersebut terlalu *overfit*, alhasil hampir seluruh data non-*fraud* salah diklasifikasikan menjadi *fraud*. Oleh karena itu, dapat digunakan *threshold* 0.01 yang mendapatkan hasil *recall* 99.77% dan *f1 measure* 66.23%. Lalu, pada arsitektur 5-4-1-4-5, 5-3-1-3-5, dan 5-2-1-2-5 dengan *reconstruction error* 0.729 jika menggunakan *threshold* 0.001 juga terlalu rendah dan lebih buruk karena seluruh data non-*fraud* salah diklasifikasikan menjadi *fraud*, sehingga dapat digunakan *threshold* 0.01 dengan hasil *recall* 99.84% dan *f1 measure* 66.22%.

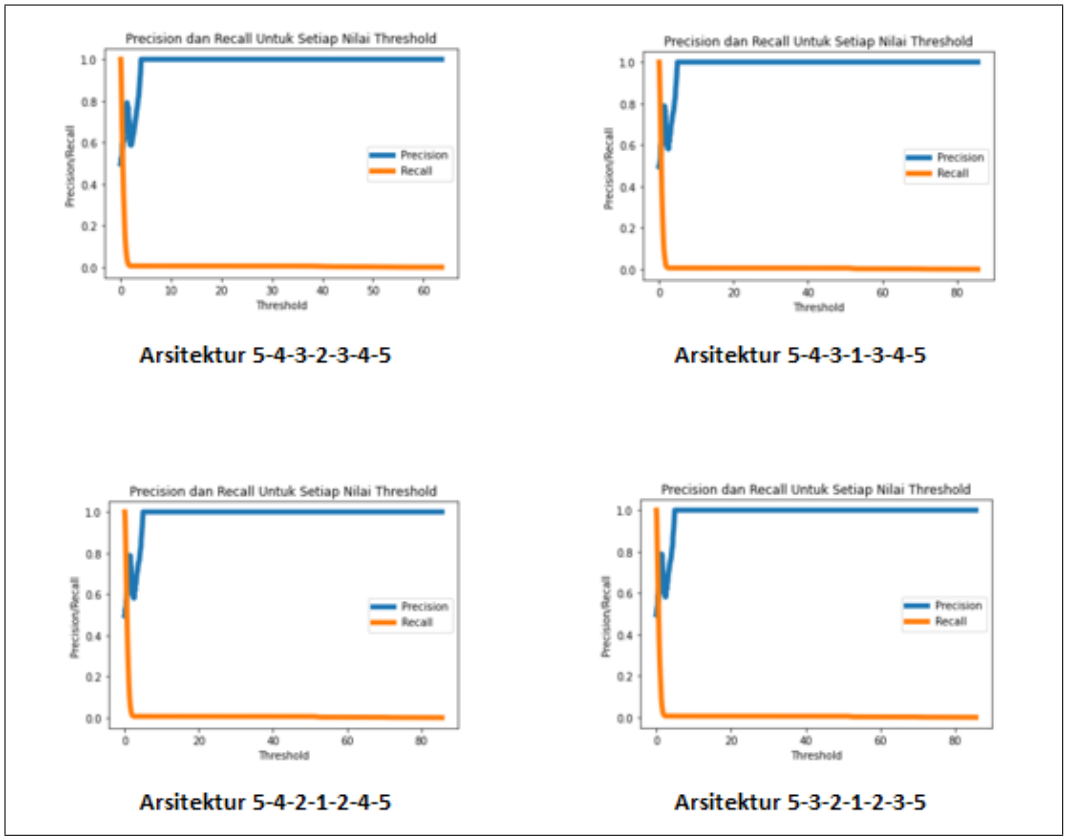
4.4.3.2 Pengujian 2 *Hidden Layer*

Selanjutnya, akan diuji Autoencoder menggunakan 2 *hidden layer* dengan arsitektur seperti pada gambar 4.5. Terdapat 4 kombinasi arsitektur untuk pengujian 2 *hidden layer*. Arsitektur yang diuji dengan urutan *neuron* (*input layer - hidden layer 1 encoder - hidden layer 2 encoder - code layer - hidden layer 1 decoder - hidden layer 2 decoder - output layer*) diantaranya: 5-4-3-2-3-4-5, 5-4-3-1-3-4-5, 5-4-2-1-2-4-5, dan 5-3-2-1-2-3-5. Hasil *reconstruction error* setiap arsitektur 2 *hidden layer* dapat dilihat pada tabel 4.16.

Tabel 4.16 *Reconstruction error* Pengujian Autoencoder 2 *Hidden Layer*

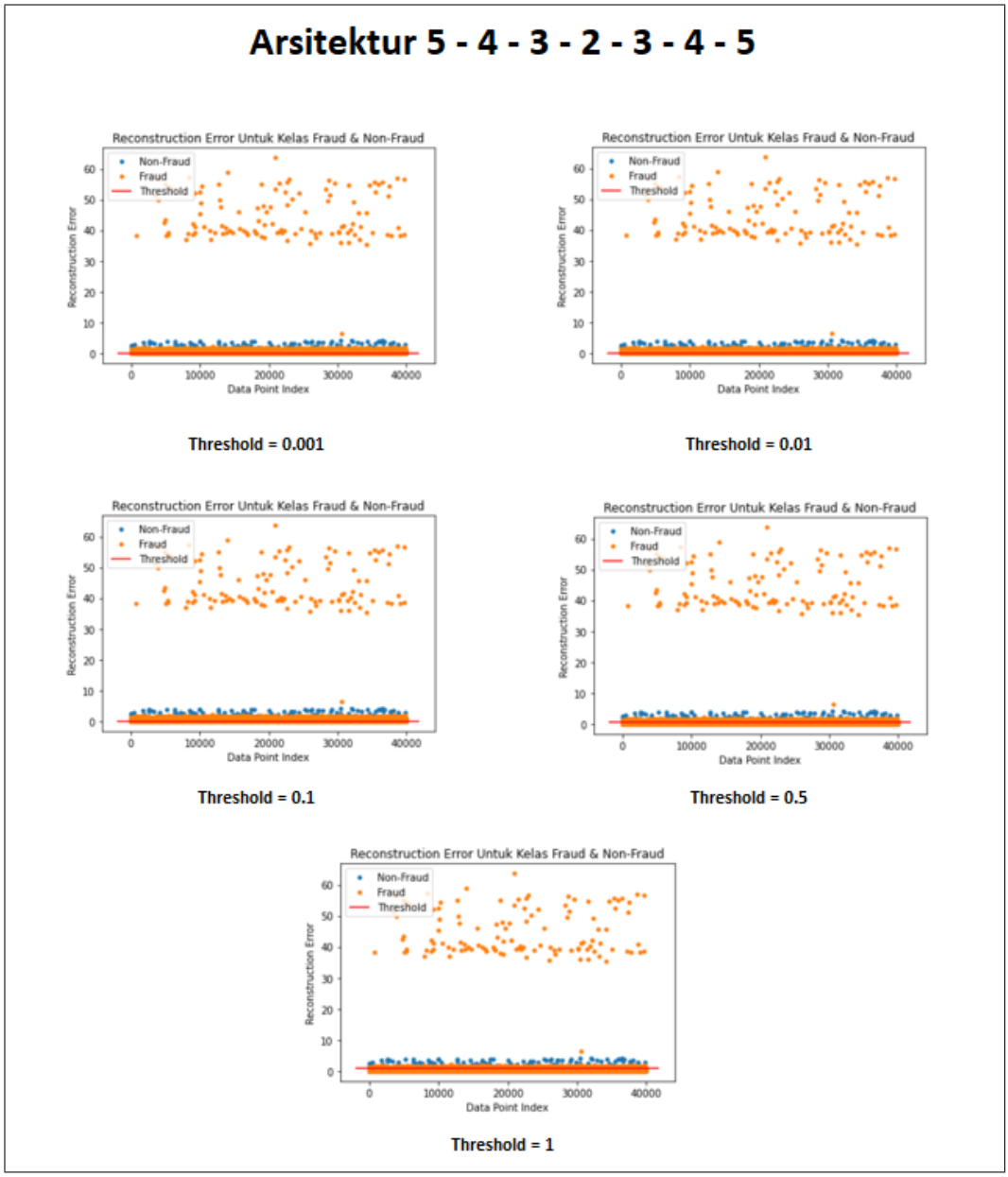
Jumlah *Neuron*							***Reconstruction Error***
Input Layer	***Hidden Layer 1 Encoder***	***Hidden Layer 2 Encoder***	***Code Layer***	***Hidden Layer 1 Decoder***	***Hidden Layer 2 Decoder***	***Output Layer***	
5	**4**	**3**	**2**	**3**	**4**	**5**	0.521
5	**4**	**3**	**1**	**3**	**4**	**5**	0.729
5	**4**	**2**	**1**	**2**	**4**	**5**	0.729
5	**3**	**2**	**1**	**2**	**3**	**5**	0.729

Berdasarkan tabel 4.16 dapat dilihat bahwa *reconstruction error* untuk arsitektur 2 *hidden layer* sama dengan 1 *hidden layer*. Kembali peran dari *neuron* pada *code layer* menjadi penentu besarnya *reconstruction error* yang dihasilkan pada setiap arsitektur. Pada arsitektur dengan 2 *neuron* pada *code layer* yaitu 5-4-3-2-3-4-5 menghasilkan *reconstruction error* sebesar 0.521. Sementara arsitektur dengan 1 *neuron* pada *code layer* seperti 5-4-3-1-3-4-5, 5-4-2-1-2-4-5, dan 5-3-2-1-2-3-5 menghasilkan *reconstruction error* 0.729. Kemudian dilakukan plot grafik *tradeoff* antara *precision* dan *recall* terhadap masing-masing arsitektur dengan 2 *hidden layer* seperti pada gambar 4.10.



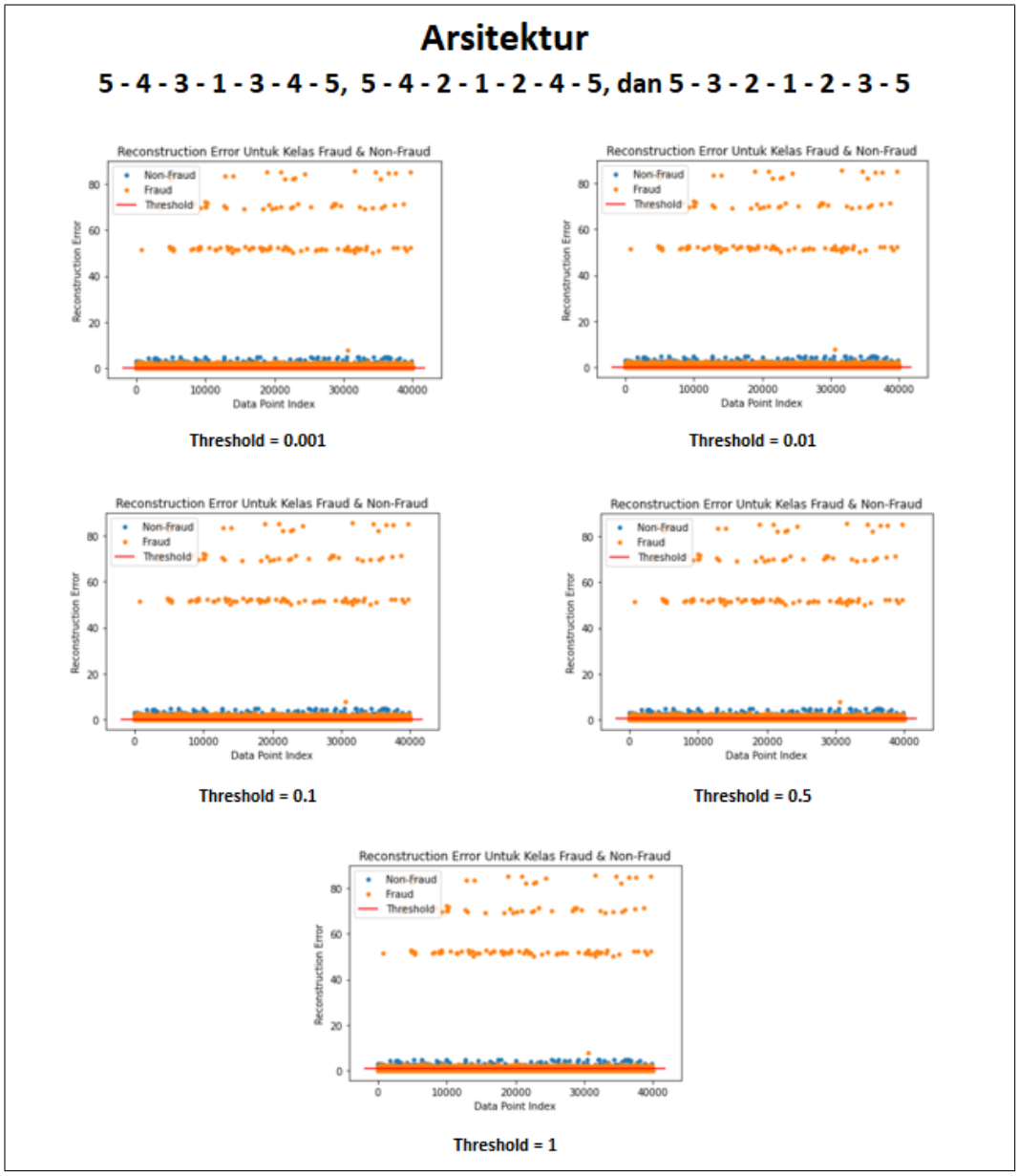
Gambar 4.10 Grafik *Tradeoff* Antara *Precision* Dan *Recall*

Pada grafik diatas pada arsitektur dengan 2 *hidden layer* ternyata *tradeoff* antara *precision* dan *recall* tetap sama. Arsitektur dengan *reconstruction error* 0.521, memiliki nilai *threshold* tertinggi pada kisaran 60. Sementara arsitektur dengan *reconstruction error* 0.729 memiliki nilai *threshold* tertinggi pada kisaran 80. Kemudian, dilakukan pemetaan yang sama untuk garis *threshold* terhadap *reconstruction error* masing-masing data poin untuk setiap arsitektur. Gambar 4.11 merupakan pemetaan garis *threshold* untuk arsitektur 5-4-3-2-3-4-5.



Gambar 4.11 Pemetaan Garis *Threshold* Pada Arsitektur 5-4-3-2-3-4-5

Pada gambar 4.11, penggunaan nilai *threshold* besar yaitu 1 tetap membuat hasil deteksi salah mengklasifikasikan banyak data *fraud* yang membuat *recall* rendah sehingga harus menggunakan nilai *threshold* kecil. Kemudian pada arsitektur 5-4-3-1-3-4-5, 5-4-2-1-2-4-5, dan 5-3-2-1-2-3-5 dengan *reconstruction error* yang sama, pemetaan garis *threshold*-nya pun sama seperti pada gambar 4.12.



Gambar 4.12 Pemetaan Garis *Threshold* Pada Arsitektur 5-4-3-1-3-4-5, 5-4-2-1-2-4-5, dan 5-3-2-1-2-3-5

Berdasarkan kedua gambar pemetaan garis *threshold* 4.11 dan 4.12 diatas, distribusi *reconstruction error* data *fraud* dan non-*fraud* saling menumpuk dibawah mendekati nilai 0 dan sedikit yang berada diatas nilai 40. Selanjutnya, dilakukan perhitungan nilai *precision*, *recall*, dan *f1 measure* berdasarkan *confusion matrix*. Hasil akhir pengujian Autoencoder dengan 2 *hidden layer* dapat dilihat pada tabel 4.17.

Tabel 4.17 Hasil Pengujian Autoencoder Dengan 2 *Hidden Layer*

Jumlah *Neuron*							***Thres -hold***	**Hasil**		
Input Layer	***Hidden Layer 1 Encoder***	***Hidden Layer 2 Encoder***	***Code Layer***	***Hidden Layer 1 Decoder***	***Hidden Layer 2 Decoder***	***Output Layer***		***Precision (%)***	***Recall (%)***	***F1 Measure (%)***
5	**4**	**3**	**2**	**3**	**4**	**5**	**0.001**	49.47	99.98	66.19
							0.01	49.57	**99.78**	**66.23**
							0.1	51.12	91.84	65.68
							0.5	60.41	40.91	48.78
							1	75.06	9.74	17.24
5	**4**	**3**	**1**	**3**	**4**	**5**	**0.001**	49.47	100.00	66.20
							0.01	49.50	**99.85**	**66.19**
							0.1	49.93	95.83	65.65
							0.5	57.94	58.68	58.31
							1	74.70	21.97	33.96
5	**4**	**2**	**1**	**2**	**4**	**5**	**0.001**	49.47	100.00	66.20
							0.01	49.52	**99.85**	**66.21**
							0.1	49.93	95.83	65.65
							0.5	57.95	58.69	58.31
							1	74.70	21.97	33.96
5	**3**	**2**	**1**	**2**	**3**	**5**	**0.001**	49.47	100.00	66.20
							0.01	49.53	**99.84**	**66.21**
							0.1	49.94	95.83	65.66
							0.5	57.95	58.74	58.34
							1	74.73	22.01	34.00

Berdasarkan hasil pengujian Autoencoder 2 *hidden layer* diatas, ternyata dengan menggunakan arsitektur 2 *hidden layer* didapatkan hasil *reconstruction error* yang sama dengan 1 *hidden layer*. Hal tersebut dipengaruhi oleh jumlah *neuron* pada *code layer*. Dengan menggunakan 2 *neuron* pada *code layer* seperti arsitektur 5-4-3-2-3-4-5 digunakan *threshold* 0.01 untuk mendapatkan hasil klasifikasi yang bagus yaitu *recall* 99.78% dan *f1 measure* 66.23%. Sedangkan arsitektur 5-4-3-1-3-4-5, 5-4-2-1-2-4-5, dan 5-3-2-1-2-3-5 yang menggunakan 1 *neuron* pada *code layer*

mendapatkan hasil yang bagus dengan *threshold* 0.01, hasil tersebut yaitu *recall* 99.85% dan *f1 measuure* 66.21%.

4.4.4 Pembahasan Pengujian

Pada tabel 4.18 merupakan rangkuman hasil pengujian yang memiliki nilai *recall* dan *f1 measure* terbaik dengan menggunakan Neural Network 1 *hidden layer* dan 2 *hidden layer* serta Autoencoder 1 *hidden layer* dan 2 *hidden layer*. Hasil yang dapat disimpulkan untuk Neural Network bahwa semakin besar jumlah *neuron* pada *hidden layer* maka hasilnya dapat semakin bagus, namun membutuhkan waktu belajar yang cukup lama karena banyak perhitungan untuk memperbarui nilai bobot. Sementara kesimpulan yang didapatkan dari hasil Autoencoder yaitu jumlah *neuron* pada *code layer* sangat mempengaruhi *reconstruction error* yang dihasilkan oleh model. Semakin kecil jumlah *neuron* pada *code layer* maka *reconstruction error*-nya akan semakin besar, demikian sebaliknya. Dan pemilihan nilai *threshold* pada Autoencoder sangat mempengaruhi hasil klasifikasi model, oleh karena itu harus dilakukan beberapa percobaan untuk menemukan nilai *threshold* yang paling sesuai.

Tabel 4.18 Rangkuman Hasil Pengujian Terbaik Neural Network Dan Autoencoder

Metode	**Jumlah *Hidden Layer***	**Arsitektur *Network***	***Learning Rate***	***Epoch***	**Hasil**		
					***Precision* (%)**	***Recall* (%)**	***F1 Measure* (%)**
Neural Network	**1**	**5 - 128 - 5**	**0.01**	**100**	83.63	92.84	87.99
Neural Network	**2**	**5 - 512 - 512 - 5**	**0.001**	**100**	97.41	96.35	**96.88**
Autoencoder (*Th* = 0.01)	**1**	**5 - 4 - 1 - 4 - 5** **5 - 3 - 1 - 3 - 5**	**0.001**	**50**	49.54	99.84	66.22
Autoencoder (*Th* = 0.01)	**2**	**5 - 4 - 2 - 1 - 2 - 4 - 5**	**0.001**	**50**	49.52	**99.85**	66.21

Pada tabel rangkuman 4.18 diatas, dapat disimpulkan bahwa pendeteksian dengan menggunakan Autoencoder 2 *hidden layer* dengan arsitektur 5-4-2-1-2-4-5 dan *threshold* 0.01 mendapatkan hasil *recall* yang paling tinggi dibandingkan model lainnya, yaitu 99.85%. Sementara hasil *f1 measure* tertinggi didapatkan dengan

Neural Network 2 *hidden layer* dengan arsitektur 5-512-512-5, *learning rate* 0.001 dan *epoch* 100, yaitu 96.88%.

Pada penelitian ini, karena sistem digunakan untuk mendeteksi *click fraud* yang mementingkan nilai *recall* maka digunakan Autoencoder. Namun, jika melihat nilai *f1 measure* pada Autoencoder yaitu 66.21%, lebih cenderung *overfit* karena *recall* tinggi namun *precision* sangat rendah. Hal tersebut didukung oleh penelitian Martin [34], yang menerapkan Autoencoder untuk mendeteksi anomali pada laporan keuangan dan hasil modelnya mendapatkan nilai *recall* yang tinggi (93%) namun *precision* rendah (32%), alhasil membuat nilai *f1 measure*-nya pun rendah (48%). Hasil yang diperoleh tersebut dikarenakan Autoencoder diatur untuk lebih banyak mendeteksi pada kasus *fraud* dengan benar sehingga membuat nilai *recall* tinggi dan *precision* rendah, alhasil *f1 measure*-nya juga rendah.

Pada penerapannya, Autoencoder dapat digunakan untuk mendeteksi *click fraud* dengan menerapkan teknik deteksi anomali, karena secara umum kasus *fraud* merupakan kejadian yang dapat disebut sebagai anomali (kejadian yang diluar pola biasanya). Namun, karena mayoritas distribusi *reconstruction error* antara data *fraud* dan non-*fraud* saling menyerupai (mendekati nilai 0), menyebabkan Autoencoder hanya dapat mendeteksi sedikit sekali data non-*fraud* dengan benar. Kasus serupa ditemukan pada penggunaan Autoencoder di penelitian Thejas [3], kelemahan dari Autoencoder dalam mendeteksi anomali ketika distribusi *loss* antara kedua kelas saling menyerupai, sehingga Autoencoder tidak dapat mendeteksi sebuah kelas dengan benar.

Pengukuran lain pada penelitian ini yaitu menggunakan *f1 measure* yang bertujuan untuk mengukur sensivitas model agar tidak *overfit*. Hasil *f1 measure* yang tinggi mengindikasikan bahwa hasil *precision* dan *recall* sama-sama tinggi atau mendekati dan tidak *overfit*. Seperti hasil dengan menggunakan Neural Network yang mendapatkan *precision* 97.41% dan *recall* 96.35% maka hasil *f1 measure*-nya tinggi yaitu 96.88%. Hasil tinggi yang didapat dikarenakan pada umumnya Neural Network digunakan untuk sistem klasifikasi dan hasilnya sudah teruji bagus seperti pada penelitian Saurabh [35]. Penelitian tersebut menerapkan ANN untuk mendeteksi penipuan kartu kredit, yang hasilnya mendapatkan *precision* 99.96%, *recall* 99.96%, dan *f1 measure* 99.96%. Hasil tersebut menjelaskan bahwa ANN dapat mendeteksi hampir seluruh data baik *fraud* maupun non-*fraud* dengan benar, sehingga nilai *f1 measure* yang didapatkan tinggi. Pada penelitian tersebut, ketiga metrik pengukuran mendapatkan nilai yang sama yaitu 99.96%, hal tersebut dikarenakan jumlah *false positive* hampir sama dengan *false negative* (FP = FN).

BAB 5 KESIMPULAN DAN SARAN

Pada bab ini berisi kesimpulan berdasarkan penelitian, pelatihan, dan pengujian yang dilakukan oleh peneliti. Selain itu terdapat juga saran yang dapat digunakan atau dipertimbangkan pada saat melakukan penelitian di masa mendatang.

5.1 Kesimpulan

Kesimpulan dari pembuatan sistem deteksi *click fraud* menggunakan Neural Network dan Autoencoder melalui pelatihan dan pengujian yang telah dilakukan adalah sebagai berikut:

1. Penerapan Autoencoder dalam mendeteksi *click fraud* mendapatkan hasil *recall* tertinggi yaitu 99.85%. Namun, mendapatkan *f1 measure* yang rendah, dikarenakan Autoencoder diatur untuk mendeteksi data *fraud* dengan benar dan menyebabkan banyak data non-*fraud* salah diklasifikasikan. Data non-*fraud* memiliki distribusi *reconstruction error* yang kebanyakan serupa dengan data *fraud* (mendekati nilai 0). Sehingga pada saat menentukan nilai *threshold* terjadi *tradeoff* antara *precision* dan *recall* sehingga nilai *f1 measure* rendah.

2. Neural Network mendapatkan hasil *f1 measure* tertinggi yaitu 96.88%. Hasil *f1 measure* tersebut tinggi dikarenakan *precision* dan *recall* memiliki nilai yang seimbang yaitu 97.41% dan 96.35%. Hasil tersebut menandakan bahwa Neural Network dapat mendeteksi baik data *fraud* dan non-*fraud* secara seimbang pada kasus *click fraud*.

3. Berdasarkan hasil pengujian yang diperoleh, Autoencoder lebih bagus dalam mendeteksi *click fraud* karena dapat mengklasifikasikan hampir seluruh data *fraud* dengan benar dan menghasilkan nilai *recall* tinggi. Sedangkan Neural Network mampu mengklasifikasikan hampir seluruh data *fraud* dan non-*fraud* dengan benar sehingga menghasilkan nilai *f1 measure* yang tinggi. Jika dibandingkan *recall* antara kedua model, dalam kasus *click fraud*, maka akan lebih tepat jika menggunakan Autoencoder. Lain halnya jika menginginkan nilai *f1 measure* yang tinggi maka dapat digunakan Neural Network.

5.2 Saran

Saran dari penulis untuk pengembangan sistem deteksi *click fraud* pada penelitian di masa mendatang adalah:

1. Menggunakan dataset yang kelasnya murni antara data *fraud* dan non-*fraud*, sehingga pada penerapan Autoencoder, distribusi *reconstruction error* antara kedua kelas akan terlihat perbedaannya lebih jelas.

2. Menerapkan ekstraksi fitur pada saat *preprocessing* untuk memberikan fitur-fitur yang lebih berbobot untuk proses belajar. Dengan ekstraksi fitur, model dapat mempelajari pola baru yang lebih signifikan terhadap hasil belajar model, dibandingkan hanya menggunakan fitur murni dari dataset.

DAFTAR REFERENSI

[1] Ch. Md. Rakin Haider, Anindya Iqbal, Atif Hasan Rahman, and M. Sohel Rahman, "An Ensemble Learning Based Approach for Impression Fraud Detection in Mobile Advertising", *2018 Journal of Network and Computer Applications*, United States, 2018.

[2] Riwa Mouawi, Mariette Awad, Ali Chehab, Imad H. El Hajj, and Ayman Kayssi, "Towards a Machine Learning Approach for Detecting Click Fraud in Mobile Advertizing", *The 13th International Conference on Innovations in Information Technology (IIT)*, United States, 2018, pp. 88-92.

[3] Thejas G. S., Kianoosh G. Boroojeni, Kshitij Chandna, Isha Bhatia, S. S. Iyengar, and N. R. Sunitha, "Deep Learning-based Model to Fight Against Ad Click Fraud", *in Proceedings of the ACM Southeast Conference (ACMSE)*, Kennesaw, GA, USA, 2019, pp. 176-181.

[4] Marcin Gabryel, "Data Analysis Algorithm for Click Fraud Recognition", *in Proceedings of the 24th International Conference on Information Systems and Technologies (ICIST)*, Vilnus, Lithuania, 2018, pp. 437-446.

[5] Zhaomin Chen, Chai Kiat Yeo, Bu Sung Lee, and Chiew Tong Lauw, "Autoencoder-based Network Anomaly Detection", *Wireless Telecommunications Symposium (WTS)*, Phoenix, Arizona, US, 2018.

[6] M. A. Al-Shabi, "Credit Card Fraud Detection Using Autoencoder Model in Unbalanced Datasets", *Journal of Advances in Mathematics and Computer Science*, India, 2019.

[7] Thejas G. S., Surya Dheeshjith, S. S. Iyengar, N. R. Sunitha, and Prajwal Badrinath, "A Hybrid and Effective Learning Approach for Click Fraud Detection", *Journal of Machine Learning with Applications*, 2020.

[8] Giuseppe Bonaccorso, *Machine Learning Algorithms*, 2017.

[9] Laurene V. Fausett, *Fundamental of Neural Networks: Architectures, Algorithms, and Applications*, 1993.

[10] Hamed Habibi Aghdam and Elnaz Jahani Heravi, *Guide to Convolutional Neural Networks*, 2017.

[11] Simon Haykin, *Neural Network and Learning Machines 3rd edition*, 2009.

[12] L. Handley, ”Businesses could lose $16.4 billion to online advertising fraud in 2017: Report”, CNBC, 13 April 2017. [Online]. Available: https://www.cnbc.com/2017/03/15/businesses-could-lose-164-billion-to-online-advert-fraud-in-2017.html. [Accessed: 13 February 2021]

[13] Jonathan Marciano, ”The global growth of ad fraud in 15 countries”, CHEQ, 7 January 2021. [Online]. Available: https://www.cheq.ai/the-global-growth-of-ad-fraud-in-15-countries. [Accessed: 13 February 2021]

[14] Rahmad Fauzan, ”2019, Kerugian Akibat Penipuan Iklan di Indonesia Tembus US$120 Juta”, Teknologi Bisnis, 12 August 2019. [Online]. Available: https://teknologi.bisnis.com/read/20190812/282/1135333/2019-kerugian-akibat-penipuan-iklan-di-indonesia-tembus-us120-juta. [Accessed: 14 February 2021]

[15] Kustin Ayuwuragil, ”Ad Fraud yang Rugikan Ratusan Miliar Sempat Jerat Tokopedia”, CNN Indonesia, 4 Desember 2017. [Online]. Available: https://www.cnnindonesia.com/teknologi/20171130212125-185-259373/ad-fraud-yang-rugikan-ratusan-miliar-sempat-jerat-tokopedia. [Accessed: 14 February 2021]

[16] Rowel Atienza, *Advanced Deep Learning with Keras*, 2018.

[17] Sevdalina Georgieva, Maya Markova, and Velizar Pavlov, ”Using Neural Network For Credit Card Fraud Detection”, *Sixth International Conference on New Trends in the Applications of Differential Equations in Sciences (NTADES’19)*, Bulgaria, 2019.

[18] Ruchika Sharma, ”111 Top Ad Networks That Work With Publishers [Updated]”, iZooto. [Online]. Available: https://www.izooto.com/blog/111-top-ad-networks. [Accessed: 15 February 2021]

[19] Sagar Sharma, ”Activation Functions in Neural Networks”, Towards Data Science, 2017. [Online]. Available: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. [Accessed: 16 February 2021]

[20] Dishashree Gupta, ”Fundamentals of Deep Learning – Activation Functions and When to Use Them?”, Analytics Vidhya, 30 January 2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/01/fundamentals-

deep-learning-activation-functions-when-to-use-them/. [Accessed: 16 February 2021]

[21] Arden Dertat, "Applied Deep Learning - Part 3: Autoencoders", Towards Data Science, 2017. [Online]. Available: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798. [Accessed: 16 February 2021]

[22] Aditya Sharma, "Autoencoder as a Classifier using Fashion-MNIST Dataset", Data Camp, 2018. [Online]. Available: https://www.datacamp.com/community/tutorials/autoencoder-classifier-python. [Accessed: 18 February 2021]

[23] Microsoft's Team, "SMOTE", Microsoft, 2019. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/smote. [Accessed: 18 February 2021]

[24] Daniel Godoy, "Understanding binary cross-entropy / log loss: a visual explanation", Towards Data Science, 2018. [Online]. Available: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a. [Accessed: 10 March 2021]

[25] Benai Kumar, "10 Techniques to deal with Imbalanced Classes in Machine Learning", Analytics Vidhya, 23 July 2020. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/. [Accessed: 19 March 2021]

[26] Chitta Ranjan, "Extreme Rare Event Classification using Autoencoder in Keras", Towards Data Science, 2019. [Online]. Available: https://towardsdatascience.com/extreme-rare-event-classification-using-autoencoders-in-keras-a565b386f098. [Accessed: 22 March 2021]

[27] Casper Hansen, "Neural Networks: Feedforward and Backpropagation Explained & Optimization", ML From Scratch, 2019. [Online]. Available: https://mlfromscratch.com/neural-networks-explained/#/. [Accessed: 23 March 2021]

[28] TalkingData, "TalkingData AdTracking Fraud Detection Challenge", Kaggle, 2018. [Online]. Available: https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data. [Accessed: 1 April 2020]

[29] Sun-Chong Wang, "Interdisciplinary Computing in Java Programming", 2003.

[30] Marco Schreyer, Timur Sattarov, Damian Borth, Andreas Dengel, and Bernd Reimer, “Detection of Anomalies in Large-Scale Accounting Data using Deep Autoencoder Networks”, *German Research Center for Artificial Intelligence (DFKI)*, Germany, 2017.

[31] Ali Moradi Vartouni, Saeed Sedighian Kashi, and Mohammad Teshnehlab, “An Anomaly Detection Method to Detect Web Attacks Using Stacked Auto-Encoder”, *6th Iranian Joint Congress on Fuzzy and Intelligent System (CFIS)*, Kerman, Iran, 2018.

[32] Harrison Kinsley and Daniel Kukiela, ”Neural Networks from Scratch in Python”, 2020

[33] Diederik P. Kingma, Jimmy Lei Ba, ”Adam: A Method For Stochastic Optimization”, *3rd International Conference on Learning Representation (ICLR)*, San Diego, 2015.

[34] Martin Schultz and Marina Tropmann-Frick, ”Autoencoder Neural Networks versus External Auditors: Detecting Unusual Journal Entries in Financial Statement Audits”, *Proceedings of the 53rd Hawaii International Conference on System Sciences*, Hawaii, US, 2020.

[35] Saurabh C. Dubey, Ketan S. Mundhe, and Aditya A. Kadam, ”Credit Card Fraud Detection using Artificial Neural Network and BackPropagation”, *Proceedings of the International Conference on Intelligent Computing and Control Systems (ICICCS 2020)*, Dhulapally, India, 2020.

[36] Muhammad Uzair and Noreen Jamil, ”Effects of Hidden Layers on the Efficiency of Neural Networks”, *IEEE 23rd International Multitopic Conference (INMIC)*, Pakistan, 2020.

[37] Imran Shafi, Jamil Ahmad, MIEEE, Syed Ismail Shah, Sr. MIEEE, and Faisal M. Kashif, ”Impact of Varying Neurons and Hidden Layers in Neural Network Architecture for a Time Frequency Application”, *IEEE International Multitopic Conference (INMIC)*, Pakistan, 2006.