PENERAPAN *DEEP NEURAL NETWORK* DENGAN *DROPOUT* DAN COST-SENSITIVE LEARNING UNTUK PREDIKSI SEORANG TERKENA PENYAKIT STROKE

TUGAS AKHIR

Cynthia Caroline
1118004

INSTITUT
TEKNOLOGI
HARAPAN
BANGSA
Veritas vos liberabit

PROGRAM STUDI INFORMATIKA
INSTITUT TEKNOLOGI HARAPAN BANGSA
BANDUNG
2022

PENERAPAN *DEEP NEURAL NETWORK* DENGAN *DROPOUT* DAN COST-SENSITIVE LEARNING UNTUK PREDIKSI SEORANG TERKENA PENYAKIT STROKE

TUGAS AKHIR

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

Cynthia Caroline
1118004

INSTITUT
TEKNOLOGI
HARAPAN
BANGSA
Veritas vos liberabit

PROGRAM STUDI INFORMATIKA
INSTITUT TEKNOLOGI HARAPAN BANGSA
BANDUNG
2022

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Stroke merupakan penyakit penyebab kematian yang menduduki peringkat kedua dan penyakit penyebab disabilitas ketiga di dunia. Selain itu, menurut data dari Riset Kesehatan Dasar (Riskesdas) Kementrian Kesehatan Republik Indonesia tahun 2018, meningkat jika dibandingkan data tahun 2013, yaitu dari 7% menjadi 10,9%. Jika dilihat dari keuangan, kasus stoke ini juga sangat berdampak. Menurut Badan Penyelenggara Jaminan Sosial (BPJS) Kesehatan, kasus stroke pada tahun 2016 sampai tahun 2018 sudah menghabiskan dana sekitar 4 triliun rupiah [1]. Oleh karena itu, diperlukan sebuah sistem yang dapat mendeteksi penyakit stroke lebih awal.

Sudah ada penelitian yang menggunakan algoritme *machine learning* dan *deep learning* untuk mendeteksi penyakit stroke. Pada penelitian sebelumnya, prediksi penyakit stroke dibuat dengan menggunakan algoritme seperti *Decision Tree*, *Gaussian Naïve Bayes*, *Random Forest*, *Expectation Maximization*, *Logistic Regression*, *K-Nearest Neighbors* (KNN), *Support Vector Machine* (SVM), dan *Deep Neural Network* (DNN) [2] [3] [4] [5] [6] [7] [8], namun model prediksi menggunakan *machine learning* belum dapat menghasilkan akurasi yang baik, sehingga diperlukan model untuk memprediksi penyakit stroke dengan menggunakan *deep learning* yang ditambahkan metode *regularization dropout* yang berfungsi untuk mencegah *overfitting*.

Pada penelitian [2], dijelaskan bahwa penulis membandingkan beberapa metode *machine learning*, seperti *Logistic Regression*, *Decision Tree*, *Random Forest Classification*, *K-Nearest Neighbor* (KNN), *Support Vector Classification* (SVM), dan *Naïve Bayes*. *Dataset* pada penelitian ini diambil dari *website* Kaggle.com dengan nama *Stroke Prediction Dataset*, yang memiliki jumlah data sebanyak 5.110, di mana 249 stroke dan 4.861 tidak stroke. Penulis juga memakai teknik *undersampling* untuk mengatasi *imbalanced class*. Hasil akurasi dan *recall* paling tinggi didapat oleh algoritme *Naïve Bayes*, yaitu dengan akurasi 82% dan *recall* 85.7

Penelitian [3] menjelaskan bahwa penulis membandingkan beberapa metode *machine learning* seperti *Decision Tree*, *Logistic Regression*, dan *Random Forest*. Data didapatkan dari *dataset* di situs Kaggle.com dengan nama *Healthcare stroke Patients in Python*, yang terdiri dari 12 kolom dan 62.001 baris. *Random*

Forest meraih akurasi dan recall tertinggi, yaitu 99.98% dan 99%. Hasil dari penelitian ini terlihat mengalami *overfitting* yang sangat tinggi.

Penelitian lain [4] membandingkan algoritme lain, yaitu *Decision Tree*, *Expectation Maximization*, *Random Forest*, *Gaussian Naive Bayes*, dan *Deep Neural Network* (DNN). Penulis juga memakai *Principal Component Analysis* (PCA) sebagai teknik *feature extraction*. Data didapat dari banyak rumah sakit di Banglore dan *medical center*, dengan total sebesar 1.500 data. Penulis menyatakan model yang dibuat dengan DNN dan *feature extraction* PCA mendapatkan hasil akurasi dan recall terbaik, yaitu 86.42% dan 74.89%, namun penulis juga menyatakan bahwa DNN memiliki kelemahan, yaitu waktu *training* yang lambat sehingga kita harus meningkatkan performa.

Pada penelitian [5], penulis mencari algoritme yang paling cocok untuk kasus dataset yang sangat besar, sekitar 800 ribu data. Penulis membandingkan DNN, *Gradient Boosting Decision Tree* (GBDT), *Logistic Regression*, dan SVM. Penulis mendapat data dari National Health Insurance Research Database (NHIRD) dan penulis memakai 2.007 fitur dari total keseluruhan 7.932 fitur. Model DNN mendapat hasil terbaik dengan akurasi 87.3%, *recall* 84.5%, dan AUC 91.5%

Penelitian [6], penulis membandingkan AUC antara ANN tanpa scaling dengan ANN menggunakan bermacam-macam scaling (normalizer, min-max, standard, robust), SVM, XGB, Binary Logistic Regression. Hasil terbaik didapat oleh model ANN tanpa *scaling*, dengan akurasi 87.8%, *recall* 96.7%, ROC 84%.

Dalam penelitian [7], penulis mencari kombinasi hyperparameter terbaik untuk mendapatkan akurasi tertinggi pada model DNN. Penulis mendapatkan data dari Imam Khomeini Hospital, Ardabil, Iran, dengan jumlah 332 pasien. Penulis melakukan 81 percobaan terkait kombinasi *activation function*, *hidden layer*, *epoch*, *momentum*, dan *learning rate*. hasil terbaik diraih dari kombinasi *activation function* tanh, *hidden layer* berjumlah 10, *epoch* berjumlah 400, *momentum* sebesar 0.5, *learning rate* 0.1, dengan akurasi sebesar 99.5%, *recall* 98%, dan ROC area 97%.

Penelitian lain [8], penulis membuat model menggunaka DNN dan PCA agar dapat mencari variabel yang berperan paling penting dalam kasus penyakit stroke. Data didapat dari *Korean National Hospital Discharge In-depth Injury Survey* (KNHDS). KNHDS mengambil data dari *Korea Centers for Disease Control and Prevention* (KCDC), yang dikumpulkan dari tahun 2013 hingga tahun

2016. Penulis memakai 15.099 data dan 11 variabel. Model DNN yang dibuat penulis mendapat akurasi 84.03%, *recall* 64.32%, dan AUC 83.48%.

Dalam penelitian [9], penulis membandingkan 10 *dataset* yang semuanya bersifat *imbalanced class*, dan hasilnya 6 data mendapatkan G-Mean yang terbaik dengan teknik *cost-sensitive learning with moving threshold* dengan metode pengukuran ROC *curve*, jika dibandingkan dengan metode *random oversampling*, *random undersampling*, SMOTE, *cost-sensitive learning with moving threshold* dengan metode pengukuran *imbalance ratio*

Pada penelitian ini, akan digunakan metode *Deep Neural Network* (DNN) dengan memperhatikan *dropout* [10] [11], *cost-sensitive learning*, dan *probability tuning*. Teknik *dropout* digunakan untuk mengatasi kelemahan DNN, yaitu mudah mengalami *overfitting* [7] dan waktu *training* yang lambat [4] [8]. *Cost-sensitive learning* dan *probability tuning* digunakan karena *dataset* yang digunakan bersifat *imbalance*.

1.2 Rumusan Masalah

Berikut adalah rumusan masalah yang akan dibahas di dalam penelitian ini.

1. Bagaimana pengaruh *dropout* dalam mengatasi *overfitting* pada metode DNN untuk memprediksi seseorang terkena stroke?
2. Bagaimana pengaruh *cost-sensitive* dan *probability tuning* dalam mengatasi *dataset* yang bersifat *imbalance* pada metode DNN untuk memprediksi seseorang terkena stroke?
3. Berapa nilai ROC terbaik pada model DNN untuk memprediksi seseorang terkena stroke?

1.3 Tujuan Penelitian

Berikut adalah tujuan penelitan dalam penelitian ini.

1. Mengetahui pengaruh *dropout* dalam mengatasi *overfitting* dengan metode DNN.
2. Mengetahui pengaruh *cost-sensitive* dan *probability tuning* dalam mengatasi *dataset* yang bersifat *imbalance* pada metode DNN.
3. Mengetahui nilai ROC terbaik pada model DNN.

1.4 Batasan Masalah

Agar penelitian ini menjadi lebih terarah, maka penulis membatasi masalah yang akan dibahas sebagai berikut.

1. Dataset yang digunakan berasal dari Kaggle, dengan judul *Cerebral Stroke Prediction-Imbalanced Dataset*.
2. *Overfitting* atau tidaknya suatu model akan dilihat dari *learning curve*.
3. Model akan dilihat performanya dari nilai *Receiver Operating Characteristic* (ROC) *Curve*.

1.5 Kontribusi Penelitian

Kontribusi yang diberikan pada penelitian ini adalah sebagai berikut.

1. Melakukan pengujian apakah metode DNN cocok untuk prediksi penyakit stroke pada seseorang.
2. Melihat seberapa berpengaruh *overfitting* pada dataset tabular dengan algoritme DNN terhadap akurasi data uji.
3. Melakukan pengujian apakah *dropout* dapat benar-benar mengatasi *overfitting*.

1.6 Metodologi Penelitian

Penelitian ini dibuat dengan metode penelitian sebagai berikut.

1. Studi Literatur

   Penulisan tugas akhir ini dimulai dengan melakukan studi kepustakaan yaitu dengan cara mengumpulkan bahan-bahan referensi seperti jurnal penelitian, *paper*, dan buku terkait dengan topik.
2. Eksplorasi Dataset

   Pada tahap ini penulis akan mempelajari isi dan karakteristik dari dataset *Cerebral Stroke Prediction-Imbalanced Dataset* yang akan digunakan untuk memprediksi kemungkinan penyakit stroke pada seseorang.
3. Analisis Masalah

   Pada tahap ini akan dilakukan analisis permasalahan yang ada berdasarkan batasan masalah yang sudah dibuat.
4. Perancangan dan Implementasi Algoritme

   Pada tahap ini akan dilakukan pembuatan model dengan algoritme DNN dan
5. Pengujian

   Pada tahap ini akan dilakukan pengujian terhadap hasil akurasi prediksi stroke dengan cara hasil akurasi dan ROC *curve* akan dibandingkan antara algoritme DNN dengan teknik *regularization dropout* dan DNN yang tidak menggunakan *dropout*.
6. Dokumentasi

   Pada tahap ini akan dilakukan dokumentasi hasil analisis dan implementasi secara tertulis dalam bentuk laporan tugas akhir.

1.7 Sistematika Pembahasan

Penelitian ini dibuat dengan sistematika sebagai berikut.

BAB 1 PENDAHULUAN: Bab ini berisi latar belakang, rumusan masalah, tujuan penelitian, batasan masalah, kontribusi penelitian, metodologi penelitian, dan sistematika pembahasan

BAB 2 LANDASAN TEORI: Bab ini berisi penjelasan dasar mengenai teori yang mendukung untuk implementasi penelitian ini.

BAB 3 METODOLOGI PENELITIAN: Bab ini berisi analisis algoritme DNN dengan *regularization dropout* dan *handling imbalance class* menggunakan *cost-sensitive learning* dan *probability tuning* untuk membangun model prediksi orang terkena penyakit stroke.

BAB 4 IMPLEMENTASI DAN PENGUJIAN: Bab ini berisi implementasi dan pengujian dari algoritme DNN, *regularization dropout*, *cost-sensitive learning*, dan *probability tuning* terhadap *dataset* stroke, melihat performa model dengan ROC *curve*, dan melihat *overfitting* atau tidaknya suatu model menggunakan *learning curve*.

BAB 5 KESIMPULAN DAN SARAN: Bab ini berisi kesimpulan dari penlitian yang dilakukan berdasarkan hasil dari pengujian dan saran untuk penelitian di waktu mendatang.

BAB 2 LANDASAN TEORI

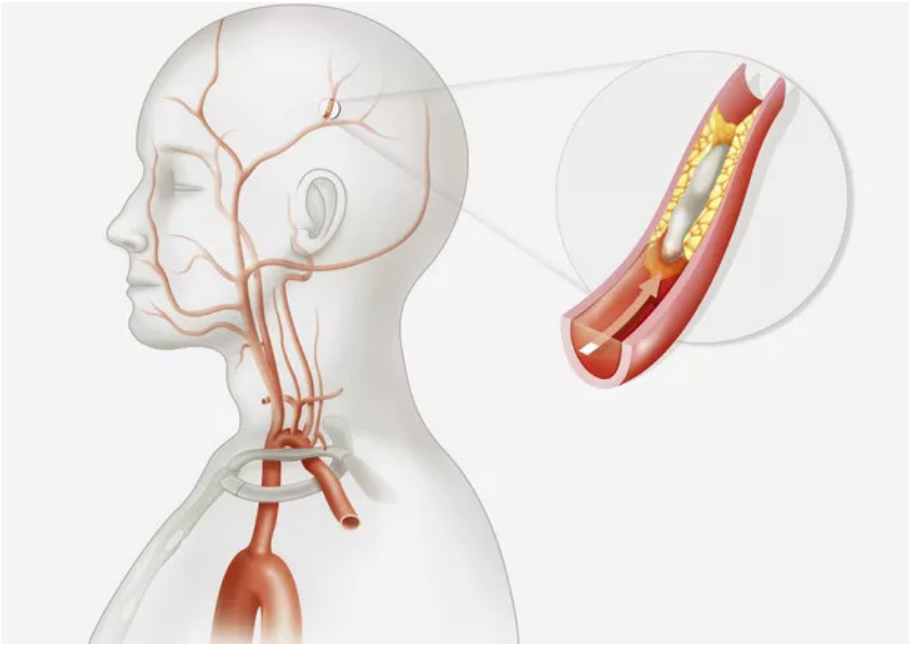
2.1 Tinjauan Pustaka

Bagian ini menjelaskan teori dasar yang digunakan dalam penelitian ini. Teori diambil dari jurnal terkait dan beberapa buku referensi.

2.1.1 Stroke

Stroke adalah kondisi yang terjadi ketika pasokan darah ke otak berkurang akibat penyumbatan (stroke iskemik) atau pecahnya pembuluh darah (stroke hemoragik) [12]. Tanpa darah, otak tidak akan mendapatkan asupan oksigen dan nutrisi, sehingga sel-sel pada area otak yang terdampak akan segera mati. Stroke dapat disebabkan oleh sejumlah faktor, seperti [13]:

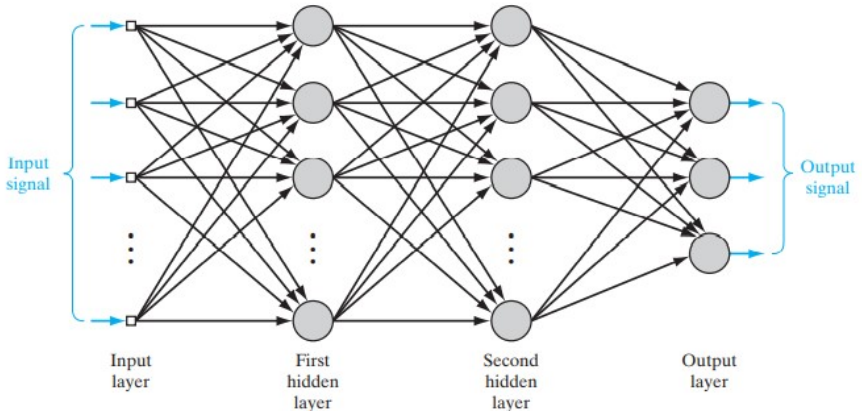
1. Faktor kesehatan, seperti tekanan hipertensi (lebih dari 140/90), mengidap penyakit jantung (gagal jantung, penyakit jantung bawaan, infeksi jantung, atau aritmia), obesitas, kolestrol tinggi, mengidap penyakit diabetes, *sleep apnea*, dan pernah mengalami TIA (stroke ringan) atau serangan jantung sebelumnya.
2. Faktor gaya hidup, seperti kebiasaan merokok, kurangnya olahraga, konsumsi obat-obatan terlarang, dan kecanduan alkohol.
3. Faktor lain, seperti faktor keturunan dan usia.



Gambar 2.1 Ilustrasi pembuluh darah yang terkena stroke

2.1.2 Deep Neural Network

Neural network adalah sebuah arsitektur yang cara kerjanya terinspirasi dari cara kerja otak. Otak terdiri dari kumpulan neuron yang saling terhubung. Setiap neuron menerima *input* dari *output* neuron lain dan kemudian melakukan perhitungan. *Neural network* terdiri dari kumpulan perceptron [15]. Perceptron adalah bagian terkecil dari arsitektur *neural network* [15]

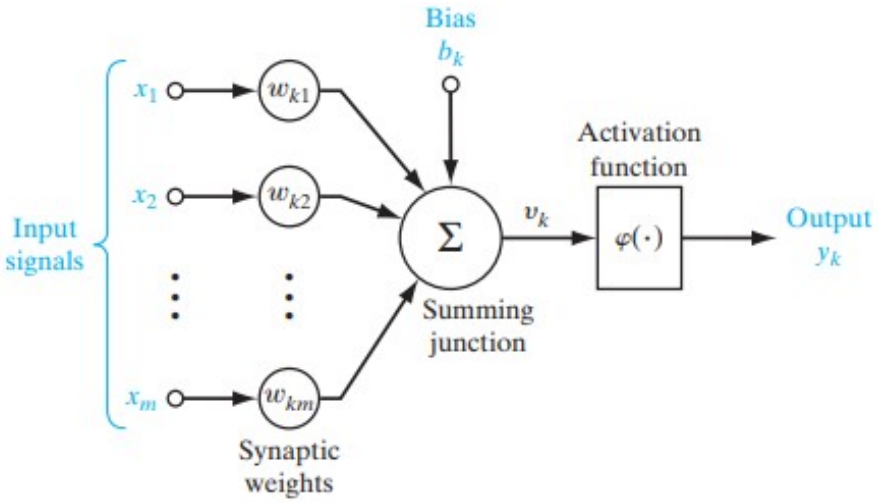


Gambar 2.2 *Neural Network* [16]

Gambar 3.14 menunjukkan arsitektur sederhana *neural network*, di mana, terdapat 3 *layer*, yaitu *input layer* (*layer* yang berfungsi menerima *input* data), *hidden layer* (*layer* yang berada diantara *input layer* dan *output layer*), *output layer* (*layer* paling akhir, yang berfungsi mengeluarkan output neuron).

Dalam *neural network*, terdapat 3 elemen dasar, yaitu [16]:

1. Kumpulan sinapsis yang masing-masing memiliki *weight*. Sinyal *input* x_j pada sinapsis j yang terhubung dengan neuron k, akan dikalikan dengan sinapsis *weight* w_{kj}.
2. Penjumlahan yang menjumlahkan sinyal *input*, dengan *weight* masing-masing sinapsis neuron, disebut *linear combiner*.
3. *Activation function*, yang berfungsi untuk membatasi rentang *output* neuron. Umumnya, *range output* neuron antara 0 sampai 1 dan -1 sampai 1.



Gambar 2.3 Neuron *nonlinear* [16]

Gambar 2.3 merupakan neuron *nonlinear* yang terdiri dari *input*, bias, dan *output*. Bias berfungsi untuk meningkatkan atau menurunkan hasil *input* dari *activation function*, tergantung hasilnya apakah positif atau negatif. Persamaan *neural network* dapat dilihat di bawah ini:

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

$$y_k = \varphi(u_k + b_k) \tag{2.2}$$

$$v_k = (u_k + b_k) \tag{2.3}$$

Keterangan :

u_k : *output linear combiner*

y_k : *output* neuron

v_k : *output* sebelum dimasukkan ke *activation function*

w_k : *weight*

x : *input*

b_k : bias

φ : *activation function*

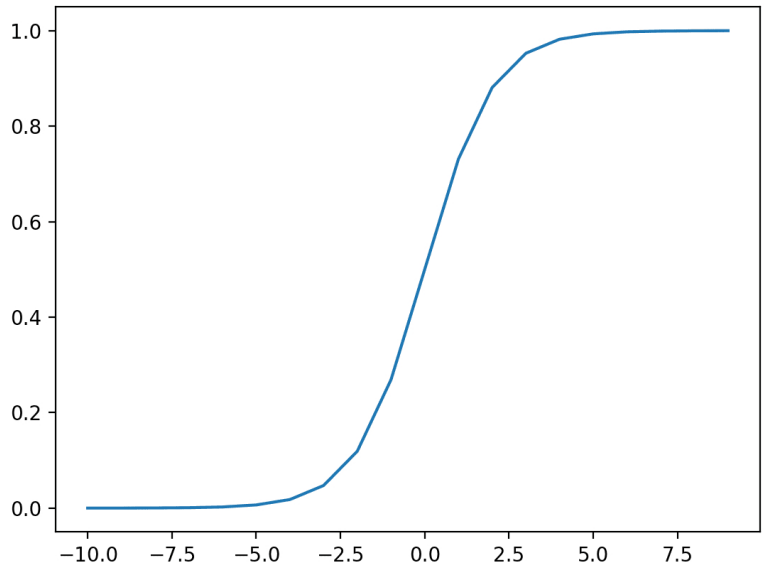
2.1.2.1 Activation Function

Beberapa *activation function* yang populer dan sering dipakai, diantaranya sebagai berikut [16]:

1. *Logistic* atau *sigmoid*, yang memiliki rentang antara 0 sampai 1. Rumus dan kurva *sigmoid* dapat dilihat pada persamaan 2.4 dan gambar 2.4.

$$\sigma(z) = 1/(1+exp(-z)) \tag{2.4}$$

Keterangan	:
z	: input
σ	: fungsi *sigmoid*

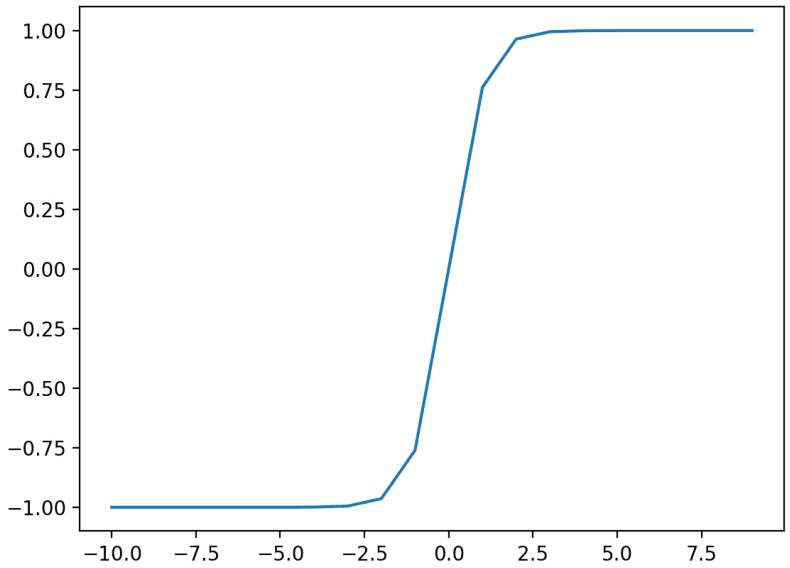


Gambar 2.4 Fungsi *sigmoid*

2. *Hyperbolic tangent* (tanh), yang memiliki rentang antara -1 sampai 1. Rumus dan kurva *tanh* dapat dilihat pada persamaan 2.5 dan gambar 2.5

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.5}$$

Keterangan :
x : input



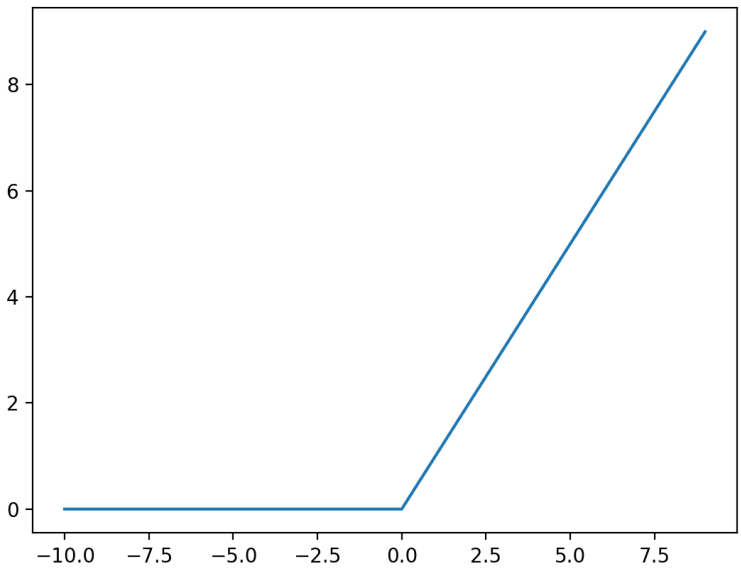
Gambar 2.5 Fungsi *tanh*

‘

3. *Rectified Linear Unit* (ReLU), yang memiliki rentang antara 0 sampai tak hingga. Rumus dan kurva ReLU dapat dilihat pada persamaan 2.6 dan gambar 2.6

$$ReLU(z) = max(0, z) \tag{2.6}$$

Keterangan :
z : input
σ : fungsi *ReLU*



Gambar 2.6 Fungsi ReLU

‘

2.1.2.2 Dropout

Dropout merupakan salah satu teknik *regularization* yang bekerja dengan cara menonaktifkan neuron dalam *neural network* secara acak, dengan tujuan mengurangi *overfitting*. Algoritme dropout adalah, disetiap fase *training*, setiap neuron (kecuali neuron *output*), mempunyai probabilitas p yang sementara diputus, dengan maksud akan diabaikan terlebih dahulu selama fase *training* kali ini, tetapi mungkin akan aktif saat fase berikutnya [14]. Rumus *dropout* dapat dilihat pada persamaan 2.9 berikut.

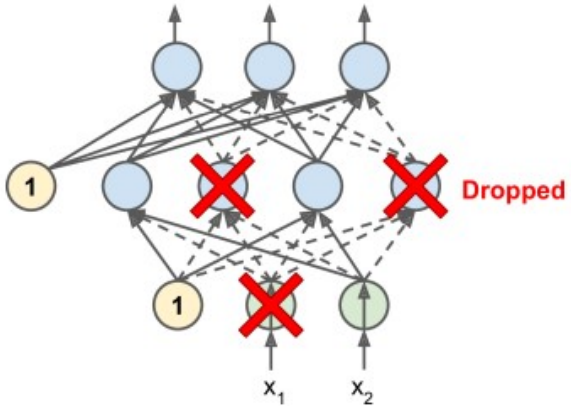
$$r_j^l \sim Bernoulli(p) \tag{2.7}$$

$$y^{\sim(l)} = r^{(l)} * y^{(l)} \tag{2.8}$$

$$z_i^{(l+1)} = w_i^{(l+1)} y^{\sim(l)} + b_i^{(l+1)} \tag{2.9}$$

$$y_i^{(l+1)} = f(z_i^{l+1}) \tag{2.10}$$

Keterangan	:
$z^{(l)}$	: *input* menuju *hidden layer*
$y^{(l)}$	: *output* dari *hidden layer*
$r^{(l)}$	: variabel Bernoulli *random* yang memiliki probabilitas $p = 1$
l	: *hidden layer*
$w^{(l)}$	: *weight* dalam *hidden layer*
$b^{(l)}$	: *bias* dalam *hidden layer*



Gambar 2.7 Dropout [14]

Dropout memiliki *hyperparameter* p yang disebut *rate*. Nilai yang direkomendasikan untuk *rate* adalah 0.1 untuk *input layer* dan 0.5 sampai 0.8 untuk *output layer* . Dalam menggunakan dropout, terdapat beberapa penyesuaian terhadap *hyperparameter*, yaitu [10]:

1. Meningkatkan ukuran *network*, karena dropout mengurangi unit selama *training*, jadi untuk menyeimbangkannya, jumlah neuron harus dinaikkan dengan *rate*, yaitu sebesar neuron dikalikan dengan 1/*rate*.
2. Meningkatkan *learning rate* dan *momentum*, karena dropout menghasilkan *noise* yang bisa menyebabkan saling membatalkan. Meningkatkan *learning rate* antara 10 sampai 100 kali dan menambahkan momentum antara 0.95 sampai 0.99 akan sangat mengimbangi.
3. Menambahkan *max-norm regularization*, karena meningkatkan *learning rate* dan *momentum* dapat membuat *weight* semakin besar pula. Hal ini dapat dicegah dengan menambahkan *max-norm regularization* yang dapat menetralkan efek itu.

2.1.3 Teknik untuk mengatasi *Imbalance Class*

Bagian ini akan menjelaskan berbagai macam teknik untuk mengatasi *imbalance class* dan alasannya baik teknik tersebut cocok digunakan, maupun tidak cocok digunakan dalam kasus prediksi penyakit stroke.

2.1.3.1 *Data Sampling Algorithms*

Data sampling algorithms merupakan algoritme yang mengganti komposisi *dataset* agar performa *machine learning* meningkat. Algoritme ini dibagi 3, yaitu [17]:

1. Data *oversampling*, bekerja dengan cara untuk duplikat kelas minoritas atau mensitesis contoh baru dari kelas minor yang bertujuan untuk membuat data baru. Metode yang paling populer adalah SMOTE. Hal yang paling penting dalam melakukan *oversampling* adalah *tuning hyperparameter*. Contoh metode *oversampling*, yaitu *random oversampling*, SMOTE, *Borderline* SMOTE, SVM SMOTE, *k-Means* SMOTE, dan ADASYN.
2. Data *undersampling*, bekerja dengan cara menghapus data, baik secara acak maupun menggunakan algoritme untuk memilih data mana yang akan dihapus dari kelas mayoritas. Metode yang paling populer adalah *edited nearest neighbors* dan *Tomek links*. Contoh metode *undersampling* adalah, *random undersampling*, *condensed nearest neighbor*, *Tomek links*, *edited nearest neighbors*, *neighborhood cleaning rule*, dan *one-sided selection*.
3. Kombinasi *oversampling* dan *undersampling*, contohnya adalah SMOTE dan *random undersampling*, SMOTE dan *Tomek links*, SMOTE dan *edited nearest neighbors*.

Dalam kasus prediksi penyakit stroke, *data sampling*, baik *oversampling* maupun *undersampling* tidak cocok digunakan. Hal ini dikarenakan jika kasus penyakit dilakukan *oversampling*, maka artinya sudah dilakukan penambahan data selain data aslinya, yang menyebabkan datanya sudah tidak akurat lagi, sedangkan jika dilakukan *undersampling*, artinya akan dilakukan penghapusan data yang menyebabkan datanya menjadi sangat sedikit mengikuti kelas minoritas.

2.1.3.2 Cost-Sensitive Learning

Cost-sensitive learning adalah sebuah metode yang memperhitungkan kesalahan prediksi saat *training* model sebagai *cost*. *Cost-sensitive learning* cocok digunakan untuk masalah yang lebih mementingkan *false negative*. *Cost-sensitive learning* juga berusaha mengurangi kesalahan saat *training* data, yang disebut *error minimization*. Tujuan *cost-sensitive learning* adalah meminimalkan *cost* saat *training dataset* [19].

Ada 3 istilah umum dalam *cost-sensitive learning*, yaitu [18]:

1. *Error minimization*, yaitu tujuan dari *training machine learning* adalah untuk mengurangi kesalahan pada model.
2. *Cost*, yaitu penalti dari prediksi yang salah. Tujuannya adalah untuk meminimalkan *cost* saat data *training* dan setiap kesalahan prediksi mempunyai *cost* yang berbeda.
3. *Cost minimization*, yaitu tujuan dari *cost-sensitive learning* untuk meminimalkan *cost* pada model saat *training dataset*.

Cost-sensitive learning untuk *imbalance class* difokuskan pertama-tama adalah menetapkan *cost* yang berbeda untuk setiap jenis kesalahan klasifikasi, kemudian menggunakan metode khusus untuk menghitung *cost* tersebut. Untuk menghitung kesalahan klasifikasi dapat menggunakan *cost matrix* yang didasari oleh *confusion matrix* [20].

Confusion matrix adalah tabel rangkuman dari model hasil prediksi. *Confusion matrix* dibagi ke dalam 4 kelas, yaitu *true negative*, *false negative*, *false positive*, dan *true positive*. Dalam *imbalanced class*, kesalahan prediksi yang paling sering terjadi adalah *false negative*. Pada *cost-sensitive learning*, *cost* dapat dipetakan dalam matriks, yang disebut *cost matrix*.

Ada 3 kelompok dalam *cost-sensitive learning*, yaitu [20]:

1. *Cost-sensitive resampling*, yaitu mengubah komposisi dari data *training* agar memenuhi ekspetasi dari *cost matrix*.
2. Algoritme *cost-sensitive*, yaitu meminimalkan kesalahan dengan cara menggunakan *weight*.
3. *Cost-sensitive ensembles*, metode ini disebut *wrapper methods* karena sifatnya yang membungkus klasifikasi *machine learning* standar. Metode ini juga disebut *meta-learners* atau *ensembles* karena mereka belajar menggabungkan atau menggunakan prediksi dari model lain. Contohnya adalah algoritme *bagging* dan *boosting* versi *cost-sensitive AdaBoost*, yaitu *AdaCost*.

Dalam penelitian [19], dijelaskan bahwa, *cost-sensitive* pada *neural network* memiliki rumus sebagai berikut:

$$O_i^*(x) = \eta \sum_{j=1}^{M} O_i(x) C(i,j) \tag{2.11}$$

$$costsensitive = argmax_i O_i^*(x) \tag{2.12}$$

Keterangan	:
O_i	: *output* dari *Neural Network*
O_i^*	: *cost* dari kesalahan klasifikasi
$C(i,j)$	: nilai dari *cost matrix*
η	: normalisasi untuk *scale output cost-sensitive* ke $\sum_{j=1}^{M} O_i = 1$ dan $0 \leq O_i^* \leq 1$
$C(i,j)$	: *imbalance ratio*, di mana i adalah kelas minoritas dan j adalah kelas mayoritas

Algoritme *cost-sensitive learning* dinilai cocok untuk digunakan, karena algoritme ini dapat mengubah-ubah *weight* agar *cost* pada kelas mayoritas lebih kecil daripada *cost* pada kelas minoritas.

2.1.3.3 *Probability Tuning Algorithms*

Probability Tuning Algorithms dibagi 2 cara, yaitu [19]:

1. *Calibrating probabilities*, yaitu kalibrasi yang dilakukan untuk klasifikasi biner yang memiliki *output* probabilitas, misalnya ROC AUC atau *precision recall AUC*
2. *Tuning the classification threshold*, yaitu mengubah *threshold* agar performa *machine learning* dapat berjalan dengan baik. Hasil probabilitas di bawah *threshold* akan masuk ke kelas 0 dan sisanya masuk ke kelas 1. *Threshold default* adalah 0.5.

Probability tuning dapat mengurangi *overfitting* karena pada saat memakai metode pengukuran ROC *curve*, kita bisa menghitung nilai G-Mean terbesar yang akan digunakan untuk menentukan *threshold* terbesar. *Probability tuning* akan digunakan bersamaan dengan ROC *curve* dengan menghitung *true positive rate*, *false positive rate*, dan G-Mean yang akan dijelaskan lebih detail di bagian ROC *curve*.

2.1.4 *Receiver Operating Characteristic* (ROC) *Curve*

Receiver Operating Characteristic (ROC) *curve* adalah metode untuk melakukan pengukuran, yang tujuannya melakukan *plotting true positive rate* (TPR) atau *recall* atau *sensitivity* dengan *false positive rate* (FPR). FPR sendiri adalah rasio negatif yang salah diklasifikasikan sebagai kelas positif. FPR

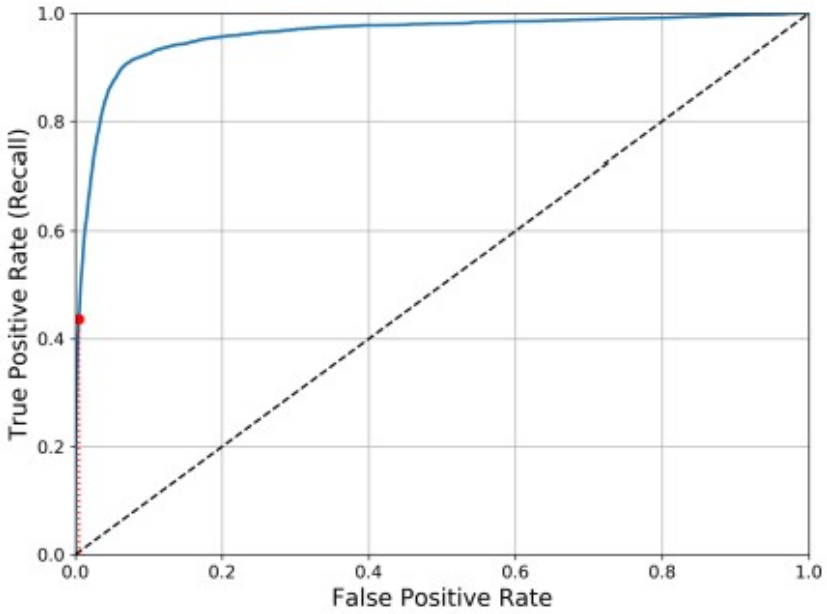
diperoleh dari perhitungan $1 - specificity$ [16]. G-Mean atau Geometric Mean adalah sebuah pengukuran yang berfungsi untuk mengukur kelas *imbalanced* yang menyeimbangkan antara *sensitivity* dan *specificity*. Agar lebih jelas, bisa dilihat pada persamaan 2.13, 2.14, dan 2.15 berikut.

$$TPR = \frac{TP}{TP+FN} \tag{2.13}$$

$$FPR = \frac{FP}{FP+TN} \tag{2.14}$$

$$G-Mean = \sqrt{TPR*(1-FPR)} \tag{2.15}$$

Keterangan :
TPR : *True Positive Rate*
FPR : *False Positive Rate*
TP : *True Positive*
TN : *True Negative*
FP : *False Positive*
FN : *False Negative*



Gambar 2.8 ROC curve [16]

Salah 1 cara untuk membandingkan hasil ROC adalah menggunakan *Area Under the Curve* (AUC). Metode ROC AUC digunakan dalam penelitian ini

dengan alasan AUC merupakan *classification-threshold-invariant*, artinya AUC mengukur kualitas prediksi model, terlepas *threshold* klasifikasi apapun yang dipilih [20]. Metode pengukuran *precision* dan *recall* kurang cocok untuk kasus *imbalanced class*, karena metode *precision* dan *recall* lebih berfokus kepada kelas minoritas saja, sedangkan ROC *curve* mencakup kedua kelas [21]. Dalam kasus ini, tidak hanya kelas positif (menderita stroke) saja yang penting, namun kedua kelas penting agar model bisa mengenali pola penderita stroke dan bukan penderita stroke dengan baik. Jika kedua kelas penting, maka pengukuran dapat dilakukan dengan ROC *curve* [22].

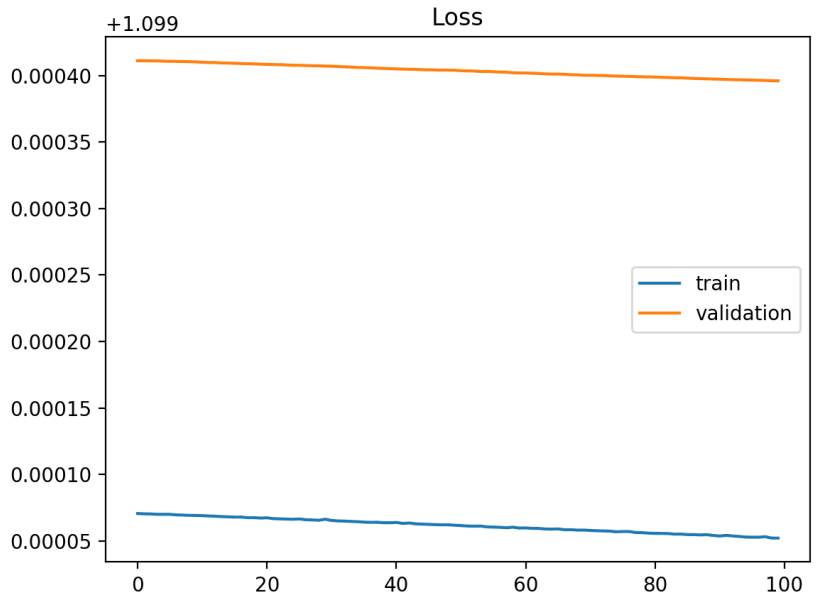
2.1.5 Learning Curves

Learning curves adalah kurva yang mengukur model berdasarkan performanya. *Learning curve* digunakan untuk mengukur hasil dari *training* model pada *machine learning* secara bertahap. Dalam *learning curves*, terdapat 2 grafik, yaitu:

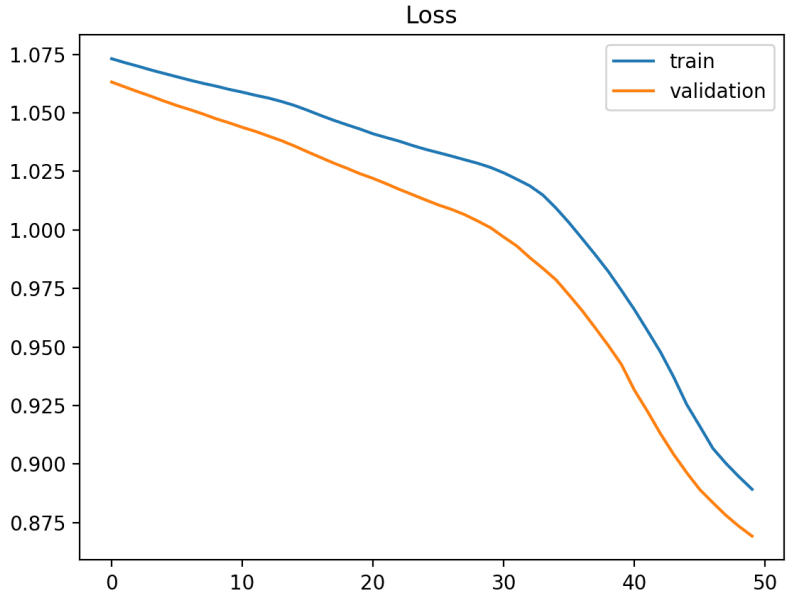
1. *Train learning curve*, yaitu *learning curves* yang dihitung dari *dataset training* yang berfungsi untuk menggambarkan seberapa baik model belajar.
2. *Validation learning curve*, yaitu *learning curves* yang dihitung dari *dataset validation* yang berfungsi utnuk menggambarkan seberapa baik model digeneralisasi.

Terdapat 3 dinamika model pada *learning curves* yang dapat diamati, yaitu:

1. *Underfit*, yaitu model tidak dapat mempelajari *dataset training*. Gambar 2.9 dan 2.10 adalah contoh model yang *underfitting*. Model yang *underfitting* dapat digambarkan dengan kurva yang hanya terdiri dari garis lurus saja. Model yang *underfitting* juga dapat digambarkan dengan *training loss* yang terus menurun.



Gambar 2.9 *Learning curves* yang mengalami *underfitting*

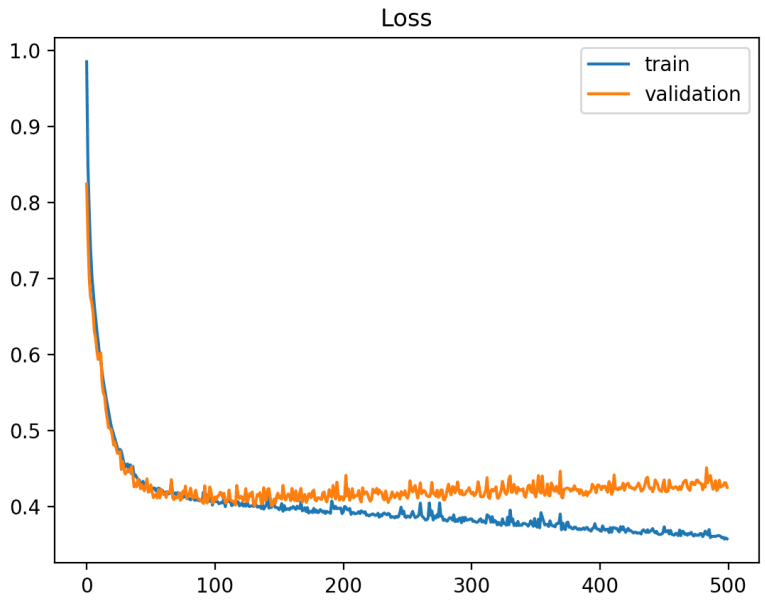


Gambar 2.10 *Learning curves* yang mengalami *underfitting*

Gambar 2.9 di atas menjelaskan bahwa model yang dibuat tidak dapat mempelajari data sama sekali,sedangkan pada gambar 2.10 menjelaskan bahwa model sebenarnya mampu belajar lebih lanjut dan hal ini dapat terjadi karena proses *training* yang sudah dihentikan sebelum waktunya.

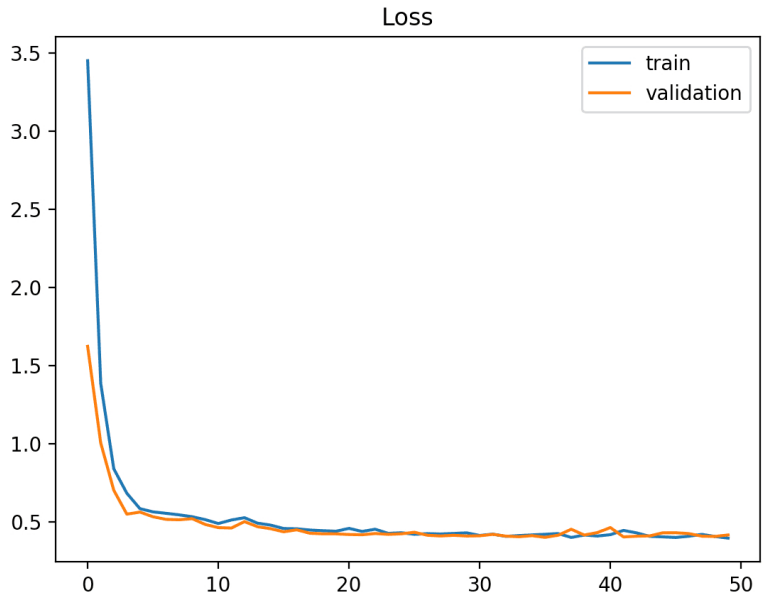
2. *Overfit*, yaitu model yang mempelajari *dataset training* dengan terlalu baik, termakusd *noise* pada *dataset training*. Masalah yang terjadi pada *overfitting*

adalah model terlalu memperhatikan data *training*, sehingga kurang mampu beradaptasi dengan data baru, yang mengakibatkan kesalahan prediksi. Hal ini dapat terjadi karena model yang dilatih terlalu lama. Gambar 2.11 adalah contoh *learning curves* yang mengalami *overfitting*.



Gambar 2.11 *Learning curves* yang mengalami *overfitting*

3. *Good fit*, yaitu model yang tidak *underfitting* dan *overfitting*. Model yang *good fit* dapat dilihat dari *training* dan *validation loss* yang stabil. Posisi kedua kurva hanya terdapat jarak yang sedikit. Gambar X adalah contoh *learning curves* yang bersifat *good fit*.



Gambar 2.12 *Learning curves good fit*

2.1.6 Pustaka Python

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

2.1.6.1 Pandas

Pandas adalah *library* Python yang digunakan untuk *preprocessing* dan analisis data.

Tabel 2.1 Daftar metode yang digunakan dari *library* Pandas

No	Metode	Masukan	Luaran	Keterangan
1	read_csv	file path: string	DataFrame	Membaca data dalam bentuk file .csv dan mengeluarkan hasil tabel data frame.
2	drop	label: string, array, axis: int, inplace: boolean	DataFrame	Menghapus baris atau kolom dari sebuah dataframe.

3	isnull	-	DataFrame	Mendeteksi apakah terdapat *missing values* atau tidak
4	sum	-	Series atau DataFrame	Menjumlahkan suatu nilai dalam DataFrame.
5	replace	to_replace: str, regex, list, dict, Series, int, float, or None, value: scalar, dict, list, str, regex, default None	DataFrame	Mengganti data dalam DataFrame
6	drop_duplicates	subset: array, inplace: boolean	DataFrame	Menghapus data atau baris yang duplikat pada DataFrame
7	describe	-	Series atau DataFrame	Mengeluarkan informasi statistik mengenai data.
8	value_counts	-	Series	Menghitung jumlah data yang unik.

2.1.6.2 Seaborn

Seaborn adalah *library* untuk membuat grafik dan statistik pada Python.

Tabel 2.2 Daftar metode yang digunakan dari *library* Seaborn

No	**Metode**	**Masukan**	**Luaran**	**Keterangan**
1	boxplot	x: array	Axes (object)	Untuk melihat distribusi suatu variabel.

2	barplot	x: int, y: int	Axes (object)	Untuk melihat kecenderungan jumlah data antara variabel berdasarkan tingginya.

2.1.6.3 Numpy

Numpy adalah *library* untuk melakukan operasi matematika.

Tabel 2.3 Daftar metode yang digunakan dari *library* Seaborn

No	Metode	Masukan	Luaran	Keterangan
1	nan	-	Axes (object)	Mengecek apakah suatu variabel bukan nomor (*not a number*).

2.1.6.4 Matplotlib

Matplotlib adalah *library* yang digunakan untuk membuat visualisasi data.

Tabel 2.4 Daftar metode yang digunakan dari *library* Matplotlib

No	Metode	Masukan	Luaran	Keterangan
1	figure	-	Figure	Membuat sebuah figur baru.
2	plot	x: array atau scalar, y: array atau scalar	list	Melakukan *plot* data dari sumbu x dan sumbu y.
3	xlabel	xlabel: str	-	Menentukan label untuk sumbu x

4	ylabel	ylabel: str	-	Menentukan label untuk sumbu y
5	title	label: str	-	Menentukan judul dari plot sebuah data.
6	legend	loc: str	-	Membuat legenda pada sebuah *plot* visualisasi data.
7	show	-	Figure	Menampilkan suatu visualisasi dari data yang telah didefinisikan sebelumnya.

2.1.6.5 Scikit-Learn

Scikit-Learn adalah *library* yang digunakan untuk *machine learning*, baik *supervised learning* maupun *unsupervised learning*.

Tabel 2.5 Daftar metode yang digunakan dari *library* Scikit-Learn

No	Metode	Masukan	Luaran	Keterangan
1	train_test_split	data: array, test size: float, random state: int	array	Membagi *dataset* menjadi data *training* dan *testing*
2	confusion_matrix	y_true: array, y_pred: array	confusion matrix	Menampilkan *confusion matrix* untuk evaluasi dari sebuah model.
3	roc_curve	y_true: array, y_score: array, pos_label: int	float	Menghitung ROC dari model hasil prediksi.

4	auc	x: array, y: array	float	Membuat point pada kurva.

2.2 Tinjauan Studi

Pada Tabel 2.6 diberikan penjelasan mengenai studi terkait dalam penelitian:

Tabel 2.6 Tinjauan Studi

No	Judul	Rumusan Masalah	Metode	Hasil
1	S. Cheon, J. Kim, and J. Lim, "The Use of Deep Learning to Predict Stroke Patient Mortality," *International Journal of Environmental Research and Public Health*, vol. 16, no. 11, 2019. [8]	Apakah dengan menggunakan model berbasis DNN dan PCA dapat mencari variabel yang paling berperan penting dalam kasus stroke?	1. *Deep Neural Network* + *Principal Component Analysis* 2. AUC	Model yang dibuat dengan DNN dan *feature extraction* PCA meraih akurasi, *recall*, dan AUC tertinggi yaitu 84.03%, 64.32%, dan 83.48%
2	A. Ashiquzzaman, et al., "Reduction of Overfitting in Diabetes Prediction Using Deep Learning Neural Network," *IT Convergence and Security*, pp. 35-43, 2017. [24]	Apakah dengan menggunakan *regularization dropout* dapat mengurangi *overfitting*?	1. *Deep Neural Network* + *dropout*	Model yang dibuat dengan DNN dan *dropout* meraih akurasi tertinggi jika dibandingkan dengan model DNN saja, yaitu dengan akurasi 88.41%

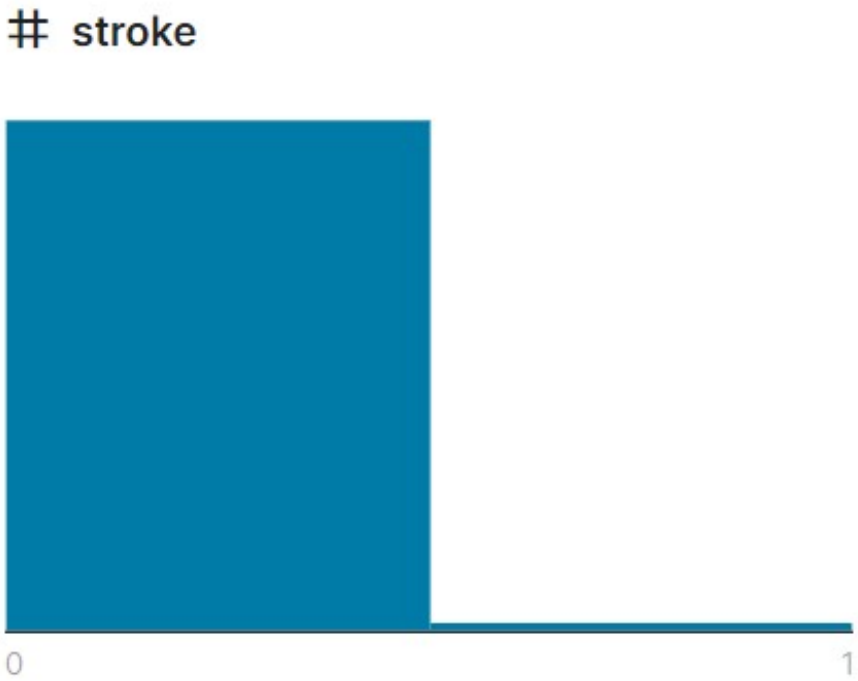
3	N. Someeh, et al., ”The outcome in patients with brain stroke: A deep learning neural network modeling,” *Journal of Research in Medical Sciences*, vol. 25, no. 1, pp. 1-7, 2020. [7]	Bagaimana kombinasi *hyperparameter* terbaik untuk mendapatkan akurasi tertinggi pada model DNN?	1. *Deep Neural Network*	Akurasi tertinggi didapat dari kombinasi *hyperparameter activation function tanh*, *hidden layer* berjumlah 10, *epoch* berjumlah 400, *momentum* sebesar 0.5, *learning rate* 0.1, dengan akurasi sebesar 99.5%.

2.3 Tinjauan Objek

Pada bagian ini akan dibahas mengenai objek terkait dengan prediksi penyakit stroke.

2.3.1 Dataset Penyakit Stroke

Data yang didapatkan merupakan *dataset* bernama *Cerebral Stroke Prediction* yang didapatkan dari di situs Kaggle [2]. Dataset berjumlah berjumlah 43.400 data dan memiliki 12 variabel yaitu *id*, *gender*, *age*, *hypertension*, *heart_disease*, *ever_married*, *work_type*, *residence_type*, *avg_glucose_level*, *bmi*, *smoking_status*, dan *stroke*. Isi data dalam dataset ini merupakan data asli yang diambil dari situs HealthData.gov. Dataset ini bersifat *imbalance*. Untuk lebih jelasnya dapat dilihat pada gambar 2.13.



Gambar 2.13 *Imbalance class* pada dataset [23]

Pada gambar 2.13, dapat dilihat bahwa dataset tersebut bersifat *imbalance* dikarenakan pada kelas stroke hanya terdapat 783 data, dan 42.617 data sisanya merupakan data orang tidak menderita stroke. Maka dari itu, perlu penanganan khusus untuk mengatasi *imbalance class*, yaitu *cost-sensitive learning* dan *probability tuning*.

BAB 3 ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Masalah

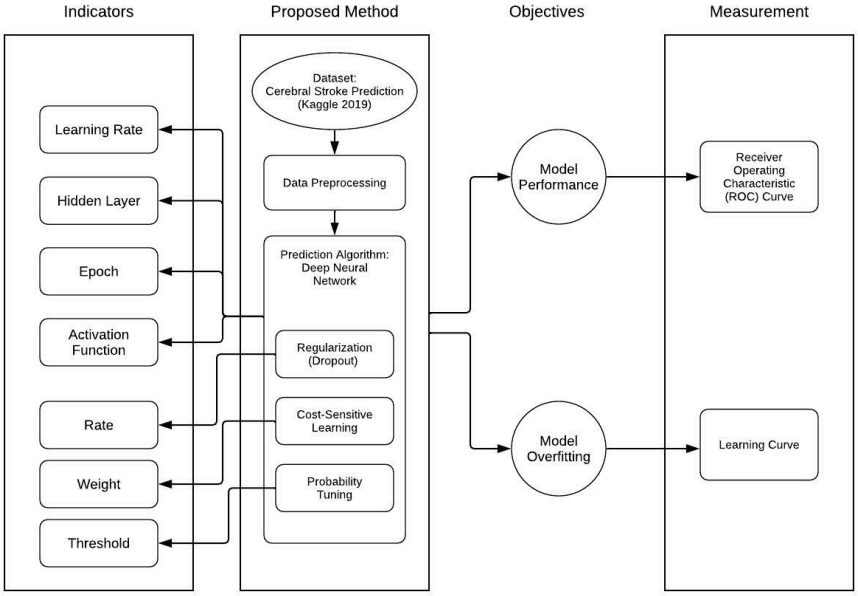
Seperti yang sudah dijelaskan pada bab 1, banyak penelitian sebelumnya yang sudah menggunakan beberapa metode *machine learning* untuk prediksi stroke, namun metode tersebut tentu saja memiliki banyak kelemahan yang dapat diatasi oleh metode *deep learning*, khususnya *Deep Neural Network* (DNN). *Dataset* dalam penelitian ini bersifat *imbalanced*, sehingga akan digunakan metode tambahan untuk *handling imbalance class*, yaitu *cost-sensitive learning* [9] dan *probability tuning*. Metode DNN dapat mendeteksi hubungan yang kompleks dan mendapatkan akurasi yang lebih tinggi, dibandingkan metode *machine learning* tradisional, namun DNN juga memiliki kelemahan, yaitu mudah mengalami *overfitting* [7] dan waktu *training* yang lambat [4] [8]. Untuk mengatasi kelemahan DNN, maka penggunaan *regularization dropout* [10] [11] dan pencarian kombinasi *hyperparameter* terbaik dinilai dapat mengatasi permasalahan tersebut.

Penelitian ini akan membangun, menguji, dan membandingkan antara model DNN biasa dan model DNN dengan menggunakan *dropout*, serta melihat seberapa besar pengaruh *overfitting* terhadap *dataset tabular* dalam kasus prediksi orang terkena penyakit stroke dengan menggunakan ROC *curve*.

Input untuk sistem ini adalah data-data penunjang yang sudah ditentukan, seperti *gender*, *age*, *hypertension*, *heart_disease*, *avg_glucose_level*, *bmi*, dan *smoking_status*. *Output* sistem ini berupa hasil prediksi bahwa seseorang menderita stroke atau tidak.

3.2 Kerangka Pemikiran

Pada gambar 3.1 diberikan gambar mengenai kerangka pemikiran dalam penelitian ini.



Gambar 3.1 Kerangka Pemikiran

Berikut akan dijelaskan setiap bagian yang ada pada gambar 3.1

1. *Indicators* adalah variabel-variabel yang digunakan dan akan memengaruhi hasil akhir. Indikator yang digunakan dalam penelitian ini adalah sebagai berikut.
   (a) *Learning rate*, berfungsi untuk mengatur seberapa besar model melakukan *update weight*. Semakin kecil model dapat konvergen, tapi diperlukan *epoch* yang besar, otomatis waktunya akan semakin lama. Jika terlalu besar, model tidak dapat konvergen.
   (b) *Hidden layer*, berfungsi untuk pola-pola yang tidak terlihat di *neural network*. Semakin banyak, semakin lambat, tetapi bisa menemukan pola yang kompleks.
   (c) *Epoch*, merupakan iterasi 1 siklus program selesai dijalankan. Semakin besar semakin bisa meningkatkan akurasi, namun akan semakin lama, dan dapat menyebabkan *overfitting*.
   (d) *Activation function*, berfungsi untuk menentukan *output* dari *neural network*, dengan *range* 0 sampai 1, -1 sampai 1, 0 sampai x. Nilai *range* tergantung dari masing-masing *activation function*.
   (e) *Rate*, merupakan probabilitas mempertahankan unit. Probabilitas = 1 artinya tidak ada *dropout*, dan semakin kecil nilai probabilitasnya, semakin banyak *neuron* yang di*dropout*. Semakin besar nilai probabilitas, maka artinya nilai *dropout* tidak cukup untuk mencegah *overfitting*, sedangkan semakin kecil nilai probabilitas, artinya semakin membutuhkan

banyak *neuron* yang otomatis akan memperlambat *training* dan hasil akan cenderung *underfitting*.

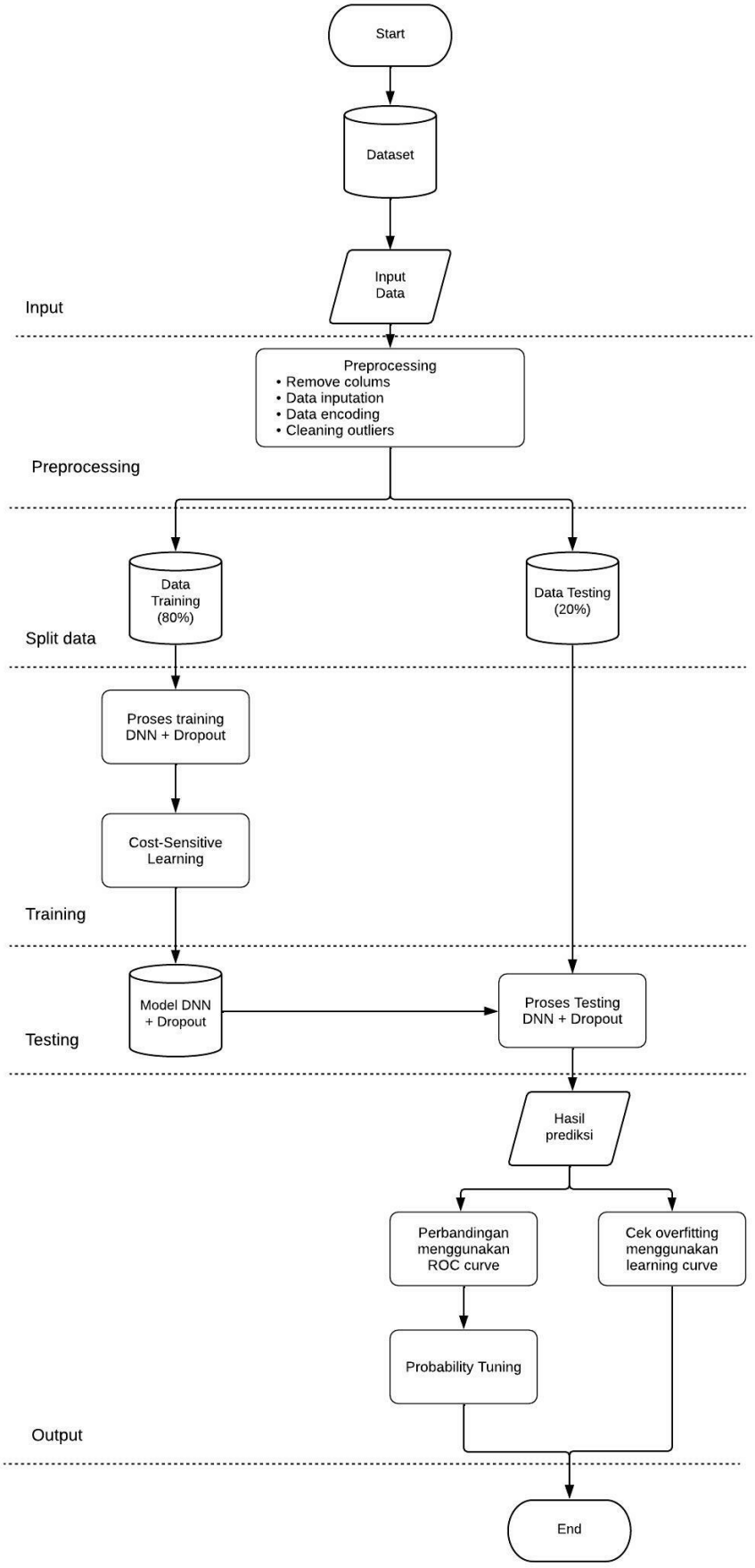
(f) *Weight*, berfungsi untuk mengatur nilai *cost* di setiap kelas agar kelas minoritas memiliki *cost* pada setiap kesalahan yang lebih kecil daripada *cost* pada kelas mayoritas.

(g) *Threshold*, berfungsi untuk menjadi pembatas antara kelas 0 dan kelas 1.

2. *Proposed Method* adalah bagian yang menjelaskan proses penelitian dari awal hingga akhir. Proses pertama kali adalah melakukan *preprocessing*, yaitu dengan menghapus *outliers*, melakukan *data inputation* dikarenakan terdapat variabel yang memiliki *missing value*, dan *handling imbalance class*. Setelah itu akan dibuat model dengan arsitektur *Deep Neural Network* yang ditambah dengan metode *regularization dropout* untuk mencegah *overfitting* pada model.
3. *Objectives* adalah bagian yang menjelaskan acuan pengukuran. Penelitian ini menggunakann acuan performa.
4. *Measurement* adalah bagian yang menjelaskan ukuran yang dipakai pada bagian *objectives*. Penelitian ini menggunakan *Receiver Operating Characteristic* (ROC) *curve* .

3.3 Urutan Proses Global

Pada gambar 3.2 diberikan *flowchart* mengenai urutan proses dalam penelitian ini.

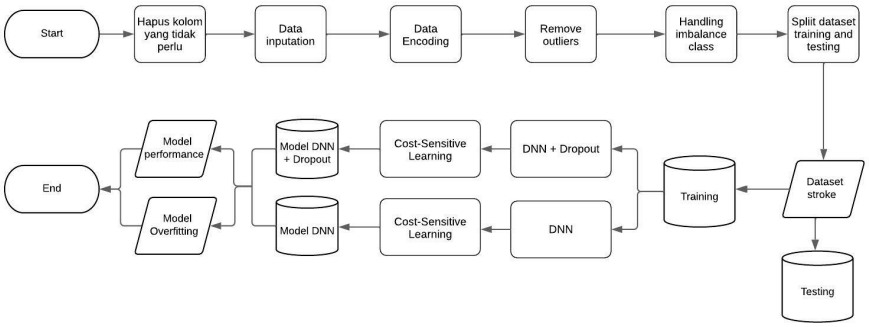

Start
Dataset
Input Data
Input
Preprocessing
• Remove colums
• Data inputation
• Data encoding
• Cleaning outliers
Preprocessing
Data Training (80%)
Data Testing (20%)
Split data
Proses training DNN + Dropout
Cost-Sensitive Learning
Training
Model DNN + Dropout
Proses Testing DNN + Dropout
Testing
Hasil prediksi
Perbandingan menggunakan ROC curve
Cek overfitting menggunakan learning curve
Probability Tuning
Output
End

Gambar 3.2 Urutan Proses Global

Model prediksi stroke ini dibangun menggunakan algoritma DNN dengan menggunakan *regularization dropout*. Setelah dilakukan *training*, model diharapkan dapat memprediksi kemungkinan orang yang terkena stroke dengan akurat dan cepat. Seperti pada gambar 3.2, dimulai dari *preprocessing dataset*, lalu melakukan *training* dan *testing* untuk membuat model. Setelah model selesai dibuat, maka akan dilakukan proses *testing* dengan data yang bersumber dari data *testing*. Jika proses *testing* selesai, maka sistem akan menghasilkan prediksi, yang akan dicek performanya menggunakan ROC *curve*.

3.3.1 Proses *Training*

Pada penelitian ini, proses *training* model digambarkan pada gambar 3.3.



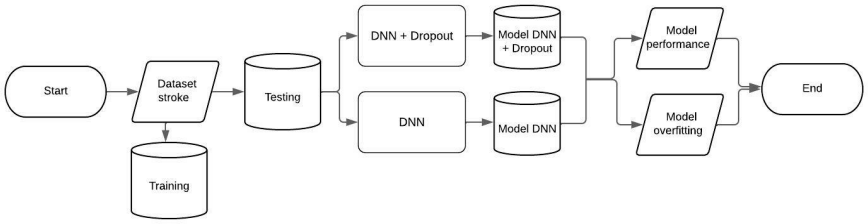
Gambar 3.3 *Flowchart* proses *training*

1. Dataset yang digunakan terdiri dari 43.400 data yang memiliki 12 variabel, yaitu *id*, *gender*, *age*, *hypertension*, *heart_disease*, *ever_married*, *work_type*, *residence_type*, *avg_glucose_level*, *bmi*, *smoking_status*, dan *stroke*.
2. Selanjutnya, akan dilakukan penghapusan terhadap variabel tertentu, yaitu *id*, *ever_married*, *work_type*, dan *residence_type*.
3. *Data inputation* dilakukan menggunakan *mean inputation* dan mengganti nilai *nan* dengan kategori *unknown*.
4. Setelah melakukan data *imputation*, maka akan dilakukan data *encoding* yaitu mengganti isi dari variabel kategorikal menjadi angka, yang berguna agar model dapat fit pada saat membuat model Deep Neural Network (DNN).
5. Setelah melakukan data *encoding*, maka akan dilakukan penghapusan terhadap *outliers*.
6. Dikarenakan kelas stroke dan tidak terkena stroke tidak seimbang, maka akan dilakukan teknik *oversampling* untuk menyeimbangkan jumlah data kedua kelas.
7. Membagi *dataset* sebesar 80% untuk proses *training Deep Neural Network* (DNN).

8. Akan menghasilkan 2 model, yaitu model DNN biasa yang akan dibandingkan dengan model DNN yang ditambahkan *dropout*.
9. Terakhir, akan diukur performa dari masing-masing model menggunakan ROC *curve* dan dibandingkan menggunakan AUC.

3.3.2 Proses *Testing*

Pada penelitian ini, proses *testing* model digambarkan pada gambar 3.4.



Gambar 3.4 *Flowchart* proses *testing*

1. *Input* yang digunakan terdiri dari 43.400 data yang memiliki 8 fitur, yaitu *gender*, *age*, *hypertension*, *heart_disease*, *avg_glucose_level*, *bmi*, *smoking_status*, dan *stroke*.
2. Membagi *dataset* sebesar 20% untuk proses *training Deep Neural Network* (DNN).
3. Akan menghasilkan 2 model, yaitu model DNN biasa yang akan dibandingkan dengan model DNN yang ditambahkan *dropout*.
4. Terakhir, akan diukur performa dari masing-masing model menggunakan ROC *curve* dan dibandingkan menggunakan AUC.

3.4 Analisis Manual

Pada bagian ini akan dijelaskan analisis tahapaan proses yang dilakukan dalam sistem.

3.4.1 Dataset

Data yang didapatkan merupakan *dataset* bernama *Cerebral Stroke Prediction* yang didapatkan dari Mendeley Data dan diterbitkan di situs Kaggle [23]. Isi data dalam dataset ini merupakan data asli yang diambil dari situs HealthData.gov. HealtData.gov adalah sebuah situs web pemerintah Amerika Serikat yang dikelola oleh *U.S. Department of Health & Human Services*. Dataset berjumlah 43.400 data dan memiliki 12 variabel, yaitu sebagai berikut.

1. *id*: id sebagai angka unik.
2. *gender*: jenis kelamin (*male*, *female*, *other*).
3. *age*: umur pasien.
4. *hypertension*: hipertensi seseorang (0 jika tidak memiliki hipertensi, 1 jika

memiliki hipertensi)

5. *heart_disease*: penyakit jantung (0 jika tidak memiliki penyakit jantung, 1 jika memiliki penyakit jantung)
6. *ever_married*: status pernikahan (*no* dan *yes*)
7. *work_type*: status pekerjaan (*children*, *govt_jov*, *never_worked*, *private* or *self-employed*)
8. *residence_type*: tempat tinggal seseorang (*rural* atau *urban*)
9. *avg_glucose_level*: tingkat glukosa rata-rata dalam darah
10. *bmi*: *body mass index*
11. *smoking_status*: status merokok pada seseorang (*formerly smoked*, *never smoked*, *smokes*)
12. *stroke*: status menderita stroke pada seseorang 0 jika tidak menderita stroke, 1 jika menderita stroke)

3.4.2 *Preprocessing*

Pada tahap ini dilakukan *preprocessing* terhadap dataset sebelum dilakukan *training* dan *testing* pada metode Deep Neural Network (DNN) dan *dropout*.

3.4.2.1 Menghapus Beberapa Kolom

Pada tahap ini dilakukan penghapusan kolom *id*, *ever_married*, *work_type*, dan *residence_type* karena kolom tersebut dinilai tidak memiliki keterkaitan dengan penyakit stroke. Pada tahap ini dilakukan juga pembersihan dari *duplicate values*. Gambar 3.5 merupakan kode penghapusan kolom pada Python dan gambar 3.6 merupakan contoh dataset setelah dilakukan penghapusan beberapa kolom.

```
df.drop(['id','ever_married','work_type','Residence_type'], axis=1, inplace=True)
df
```

Gambar 3.5 Penghapusan kolom menggunakan kode Python

	gender	age	hypertension	heart_disease	avg_glucose_level	bmi	smoking_status	stroke
0	Male	3.0	0	0	95.12	18.0	NaN	0
1	Male	58.0	1	0	87.96	39.2	never smoked	0
2	Female	8.0	0	0	110.89	17.6	NaN	0
3	Female	70.0	0	0	69.04	35.9	formerly smoked	0
4	Male	14.0	0	0	161.28	19.1	NaN	0
...	...	...	...	...	...	...	...	...
43395	Female	10.0	0	0	58.64	20.4	never smoked	0
43396	Female	56.0	0	0	213.61	55.4	formerly smoked	0
43397	Female	82.0	1	0	91.94	28.9	formerly smoked	0
43398	Male	40.0	0	0	99.16	33.2	never smoked	0
43399	Female	82.0	0	0	79.48	20.6	never smoked	0

Gambar 3.6 Penghapusan kolom *id*, *ever_married*, *work_type*, dan *residence_type*

3.4.2.2 Data *Imputation*

Pada fitur *bmi* terdapat *missing_values* sedangkan pada fitur *smoking_status* terdapat *NaN*. Oleh karena itu, untuk fitur *bmi* dilakukan data *imputation* dengan cara *mean imputation* dan untuk fitur *smoking_status* dilakukan dengan cara mengganti *NaN* dengan *unknown*. Gambar 3.7 merupakan kode data *imputation* pada Python dan gambar 3.8 merupakan contoh dataset setelah dilakukan data *imputation*.

```
1 df.isnull().sum()
```

```
gender                 0
age                    0
hypertension           0
heart_disease          0
avg_glucose_level      0
bmi                 1462
smoking_status     13292
stroke                 0
dtype: int64
```

```
1 df['bmi'].fillna(df['bmi'].mean(), inplace=True)
2 df['smoking_status'] = df['smoking_status'].replace(np.nan, 'unknown')
```

```
1 df.isnull().sum()
```

```
gender             0
age                0
hypertension       0
heart_disease      0
avg_glucose_level  0
bmi                0
smoking_status     0
stroke             0
dtype: int64
```

Gambar 3.7 Data *imputation* menggunakan kode Python

	gender	age	hypertension	heart_disease	avg_glucose_level	bmi	smoking_status	stroke
0	Male	3.0	0	0	95.12	18.0	unknown	0
1	Male	58.0	1	0	87.96	39.2	never smoked	0
2	Female	8.0	0	0	110.89	17.6	unknown	0
3	Female	70.0	0	0	69.04	35.9	formerly smoked	0
4	Male	14.0	0	0	161.28	19.1	unknown	0
...	...	...	...	...	...	...	...	...
43395	Female	10.0	0	0	58.64	20.4	never smoked	0
43396	Female	56.0	0	0	213.61	55.4	formerly smoked	0
43397	Female	82.0	1	0	91.94	28.9	formerly smoked	0
43398	Male	40.0	0	0	99.16	33.2	never smoked	0
43399	Female	82.0	0	0	79.48	20.6	never smoked	0

Gambar 3.8 Data *imputation*

3.4.2.3 Data *Encoding*

Pada tahap ini, fitur kategorikal, seperti *gender* dan *smoking_status* akan dilakukan *encoding*, yaitu dengan mengganti isinya menjadi angka. Fitur *gender*

yang berisikan *male*, *female*, *other* akan diganti menjadi 0,1,2, sendangkan fitur *smoking_status* yang berisi *unknown*, *never smoked*, *formerly smoked*, *smokes*, akan diganti menjadi 0,1,2,3. Hal ini dilakukan agar model dapat *fit* pada saat membuat model Deep Neural Network (DNN). Gambar 3.9 merupakan kode data *encoding* pada Python dan gambar 3.10 merupakan contoh dataset setelah dilakukan data *encoding*.

```
df['gender'].replace(['Male', 'Female', 'Other'],[0, 1, 2], inplace=True)
df['smoking_status'].replace(['unknown','never smoked', 'formerly smoked', 'smokes'],[0, 1, 2,3], inplace=True)
```

Gambar 3.9 *Encoding* menggunakan kode Python

	gender	age	hypertension	heart_disease	avg_glucose_level	bmi	smoking_status	stroke
0	0	3.0	0	0	95.12	18.0	0	0
1	0	58.0	1	0	87.96	39.2	1	0
2	1	8.0	0	0	110.89	17.6	0	0
3	1	70.0	0	0	69.04	35.9	2	0
4	0	14.0	0	0	161.28	19.1	0	0
...	...	...	...	...	...	...	...	...
43395	1	10.0	0	0	58.64	20.4	1	0
43396	1	56.0	0	0	213.61	55.4	2	0
43397	1	82.0	1	0	91.94	28.9	2	0
43398	0	40.0	0	0	99.16	33.2	1	0
43399	1	82.0	0	0	79.48	20.6	1	0

Gambar 3.10 *Encoding* pada fitur *gender* dan *smoking_status*

3.4.2.4 Hapus Data Duplikat

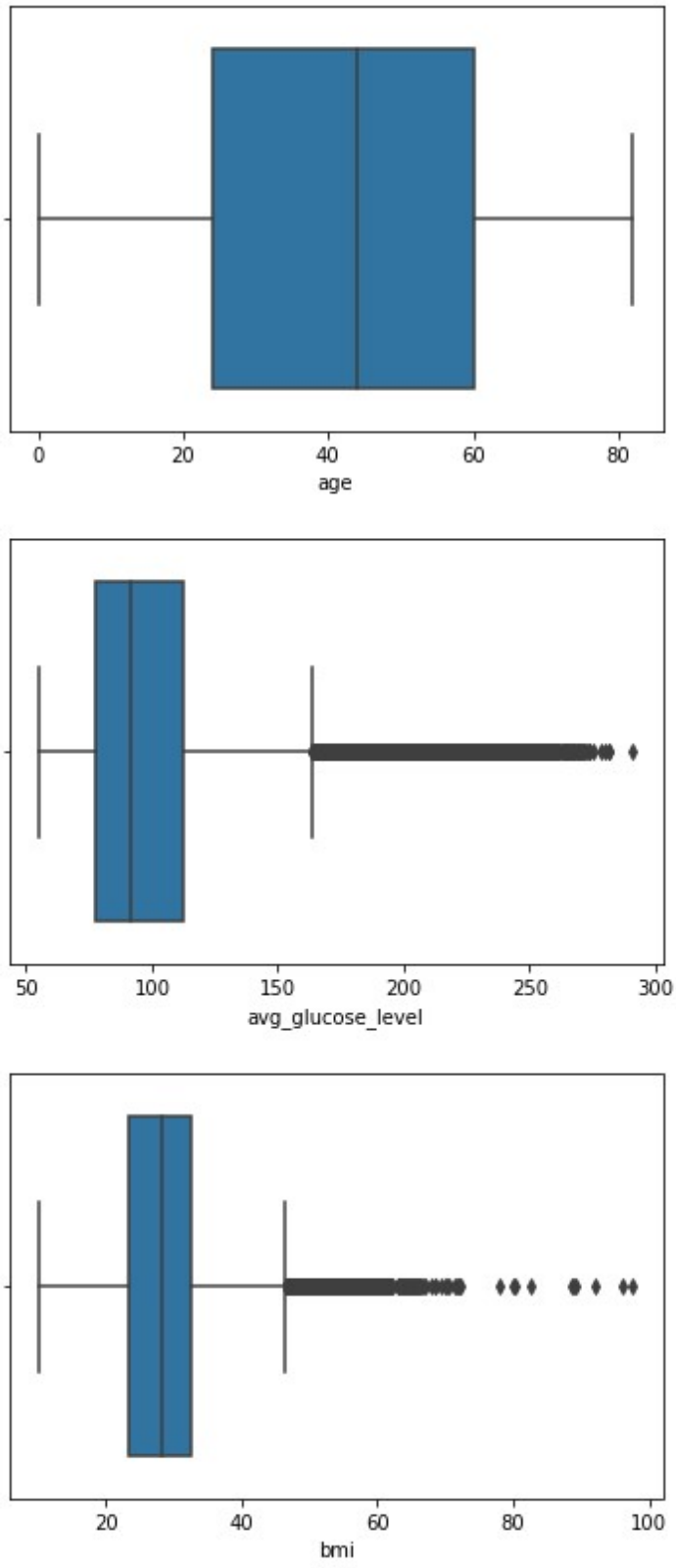
Dalam tahap ini, dilakukan pengecekan terhadap data duplikat. Jika terdapat data duplikat, maka baris data tersebut akan dihapus.

3.4.2.5 *Cleaning Outliers*

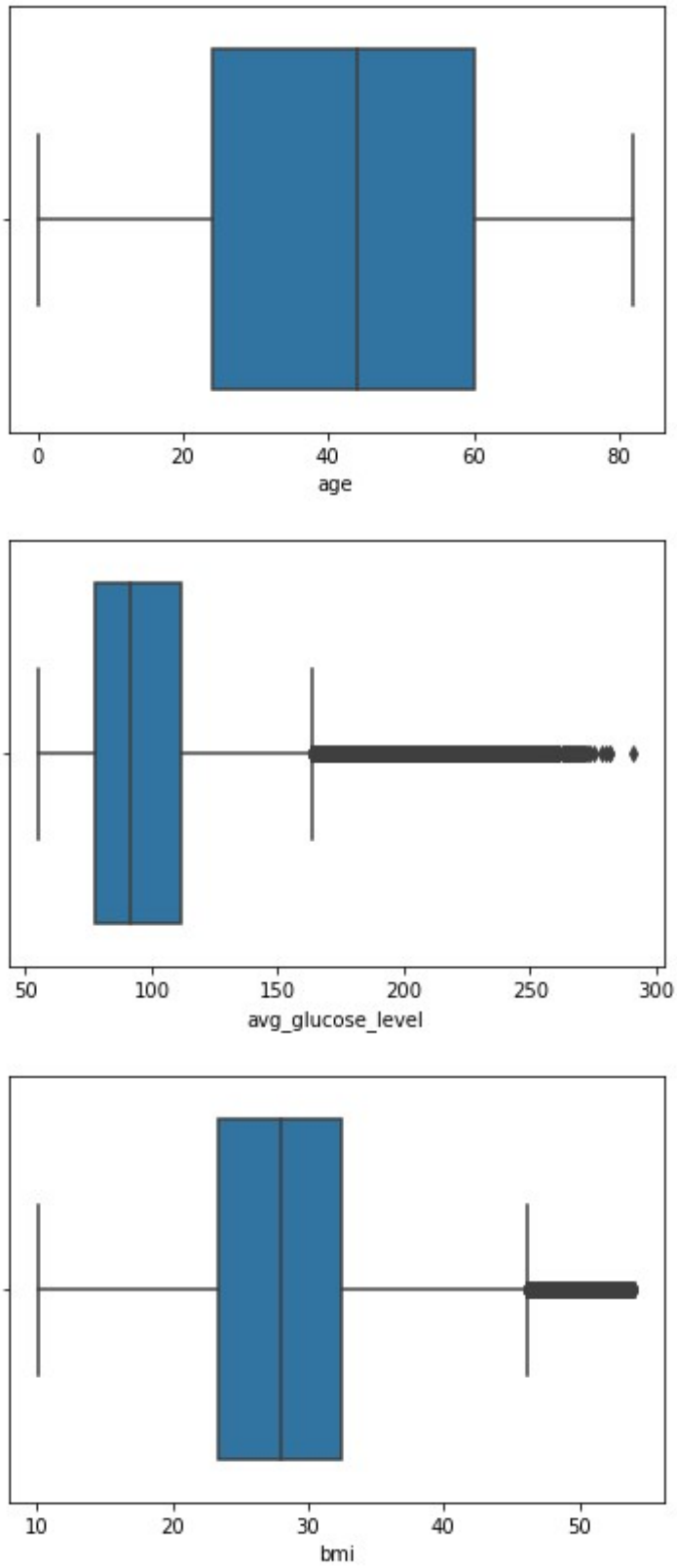
Dataset akan dilakukan pengecekan terhadap *outliers* menggunakan *boxplot*. Fitur yang dicek adalah *age*, *avg_glucose_level*, dan *bmi*. Fitur *age* tidak memiliki *outliers*. *Outliers* pada fitur *avg_glucose_level* tidak akan dihapus, karena memungkinkan memiliki kadar gula darah lebih dari 300 mg/dl [25]. *Outliers* pada fitur *bmi* akan dihapus dan data yang diambil hanya dari data awal sampai 54, dikarenakan seseorang yang memiliki BMI 54 sudah termaksud *extreme obesity* [26]. Gambar 3.11 merupakan kode *cleaning outliers* pada Python, sedangkan gambar 3.12 merupakan *boxplot* sebelum dilakukan data *cleaning* dan gambar 3.13 merupakan *boxplot* sesudah dilakukan data *cleaning*.

```
df = df[(df['bmi'] <= 54 )]
df.describe()
```

Gambar 3.11 *Cleaning outliers* menggunakan kode Python



Gambar 3.12 *Boxplot* sebelum dilakukan data *cleaning*



Gambar 3.13 *Boxplot* sesudah dilakukan data *cleaning*

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

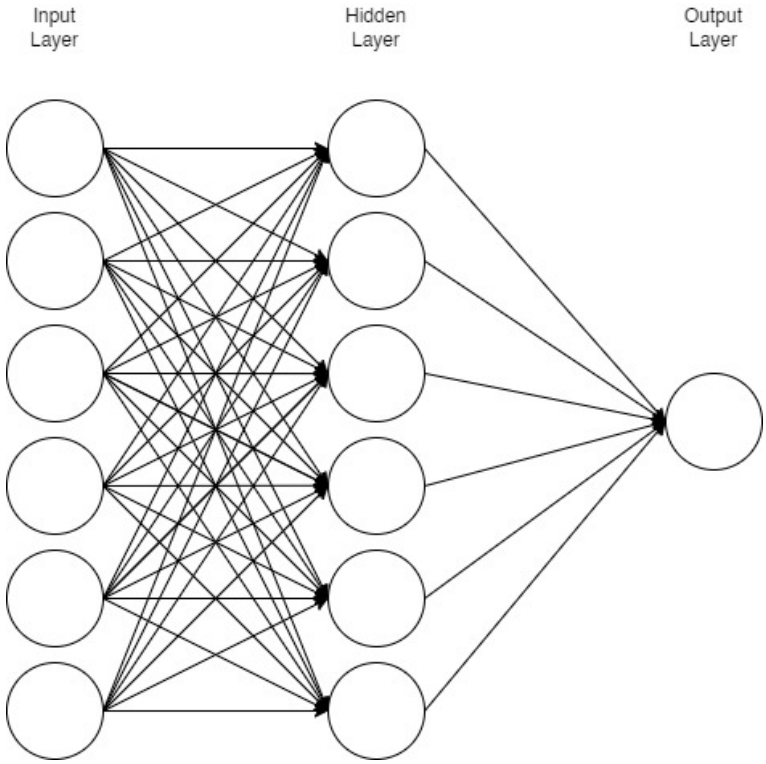
Pada tahap ini dilakukan pembagian *dataset* untuk proses *training* dan *testing* yang akan digunakan untuk membuat model Deep Neural Network. Pembagian *dataset* adalah 80% untuk data *training* dan 20% untuk data *testing*.

3.5 Perhitungan *Deep Neural Network*

Arsitektur yang akan digunakan dalam *neural network* dibagi menjadi 3 *layer*, yaitu:

1. *S*atu buah input layer yang terdiri dari 6 *neuron* yang merepresentasikan 6 fitur dan menggunakan *activation function* tanh.
2. *S*atu buah hidden layer yang terdiri dari 8 *neuron* yang menggunakan *activation function* tanh.
3. Satu buah *output layer* yang terdiri dari 1 *neuron* dan menggunakan *activation function sigmoid*.

Gambar arsitektur Deep Neural Network dapat dilihat pada gambar 3.14.



Gambar 3.14 Arsitektur *Deep Neural Network*

Nilai *weight* didapatkan dengan cara diinisialisasi melalui kode Python dan menghasilkan 62 data secara *random* dengan rentang tertentu.

Inisialisasi Data:

Jumlah *input layer* = 1

Jumlah *neuron* pada *input layer* = 6

Activation function pada *input layer* = tanh

Jumlah *hidden layer* = 1

Jumlah *neuron* pada *hidden layer* = 6

Activation function pada *hidden layer* = tanh

Jumlah *output layer* = 1

Jumlah *neuron* pada *output layer* = 1

Activation function pada *output layer = sigmoid*

Rentang *weight* = -0.4082482904638631 sampai dengan +0.4082482904638631

Bias = 0

Input (weight setiap fitur yang ada):

$x1$ = -0.27574518 $x2$ = -0.3866881 $x3$ = -0.18542552 $x4$ = -0.03321722 $x5$ = -0.18747924 $x6$ = -0.34349378

Hidden layer:

$w_{k1}j_1$ = 0.32069702 $w_{k2}j_1$ = -0.09854238 $w_{k3}j_1$ = 0.32952055 $w_{k4}j_1$ = -0.40533615 $w_{k5}j_1$ = 0.2939386 $w_{k6}j_1$ = -0.17256972

$w_{k1}j_2$ = 0.09631948 $w_{k2}j_2$ = 0.04388243 $w_{k3}j_2$ = 0.12275857 $w_{k4}j_2$ = -0.12201437 $w_{k5}j_2$ = 0.1916928 $w_{k6}j_2$ = -0.2026422

$w_{k1}j_3$ = -0.1264057 $w_{k2}j_3$ = -0.23267549 $w_{k3}j_3$ = -0.16984761 $w_{k4}j_3$ = 0.07741352 $w_{k5}j_3$ = -0.24032459 $w_{k6}j_3$ = 0.28520692

$w_{k1}j_4$ = 0.20277835 $w_{k2}j_4$ = 0.31052999 $w_{k3}j_4$ = -0.14207178 $w_{k4}j_4$ = 0.27065191 $w_{k5}j_4$ = -0.18517663 $w_{k6}j_4$ = 0.08230256

$w_{k1}j_5$ = 0.382188915 $w_{k2}j_5$ = -0.13970395 $w_{k3}j_5$ = -0.25528209 $w_{k4}j_5$ = 0.13248654 $w_{k5}j_5$ = $w_{k6}j_5$ = 0.31288484

$w_{k1}j_6$ = 0.36247341 $w_{k2}j_6$ = -0.03712616 $w_{k3}j_6$ = -0.35457662 $w_{k4}j_6$ = 0.12722402 $w_{k5}j_6$ = 0.27924119 $w_{k6}j_6$ = -0.40418496

Output layer:

$w_{k1}j_7$ = 0.01376106 $w_{k2}j_7$ = -0.34259953 $w_{k3}j_7$ = -0.21056884 $w_{k4}j_7$ = 0.12141414 $w_{k5}j_7$ = -0.060796 $w_{k6}j_7$ = -0.19922706

Berikut ini merupakan perhitungan terhadap *hidden layer* dengan menggunakan rumus 2.1 dan 2.3:

$h1 = ((w_{k1}\,j_1 \times x1) + (w_{k2}\,j_1 \times x2) + (w_{k3}\,j_1 \times x3) + (w_{k4}\,j_1 \times x4) + (w_{k5}\,j_1 \times x5) + (w_{k6}\,j_1 \times x6)) + b$

$h1 = ((0.32069702 \times -0.27574518) + (-0.09854238 \times -0.3866881) + (0.32952055 \times -0.18542552) + (-0.40533615 \times -0.03321722) + (0.2939386 \times -0.18747924) + (-0.17256972 \times -0.34349378)) + 0$

$h1 = 0.02738362964$

$h2 = ((w_{k1}\,j_2 \times x1) + (w_{k2}\,j_2 \times x2) + (w_{k3}\,j_2 \times x3) + (w_{k4}\,j_2 \times x4) + (w_{k5}\,j_2 \times x5) + (w_{k6}\,j_2 \times x6)) + b$

$h2 = ((0.09631948 \times -0.27574518) + (0.04388243 \times -0.3866881) + (0.12275857 \times -0.18542552) + (-0.12201437 \times -0.03321722) + (0.1916928 \times -0.18747924) + (-0.2026422 \times -0.34349378)) + 0$

$h2 = 0.007906679016$

$h3 = ((w_{k1}\,j_3 \times x1) + (w_{k2}\,j_3 \times x2) + (w_{k3}\,j_3 \times x3) + (w_{k4}\,j_3 \times x4) + (w_{k5}\,j_3 \times x5) + (w_{k6}\,j_3 \times x6)) + b$

$h3 = ((-0.1264057 \times -0.27574518) + (-0.23267549 \times -0.3866881) + (-0.16984761 \times -0.18542552) + (0.07741352 \times -0.03321722) + (-0.24032459 \times -0.18747924) + (0.28520692 \times -0.34349378)) + 0$

$h3 = 0.07769713625$

$h4 = ((w_{k1}\,j_4 \times x1) + (w_{k2}\,j_4 \times x2) + (w_{k3}\,j_4 \times x3) + (w_{k4}\,j_4 \times x4) + (w_{k5}\,j_4 \times x5) + (w_{k6}\,j_4 \times x6)) + b$

$h4 = ((0.20277835 \times -0.27574518) + (0.31052999 \times - 0.3866881) + (-0.14207178 \times -0.18542552) + (0.27065191 \times -0.03321722) + (-0.18517663 \times -0.18747924) + (0.08230256 \times -0.34349378)) + 0$

$h4 = -0.2331063547$

$h5 = ((w_{k1}\,j_5 \times x1) + (w_{k2}\,j_5 \times x2) + (w_{k3}\,j_5 \times x3) + (w_{k4}\,j_5 \times x4) + (w_{k5}\,j_5 \times x5) + (w_{k6}\,j_5 \times x6)) + b$

$h5 = ((0.382188915 \times -0.27574518) + (-0.13970395 \times -0.3866881) + (-0.25528209 \times -0.18542552) + (0.13248654 \times -0.03321722) + (-0.35657559 \times -0.18747924) + (0.31288484 \times -0.34349378)) + 0$

$h5 = -0.08866090313$

$h6 = ((w_{k1}\,j_6 \times x1) + (w_{k2}\,j_6 \times x2) + (w_{k3}\,j_6 \times x3) + (w_{k4}\,j_6 \times x4) + (w_{k5}\,j_6 \times x5) + (w_{k6}\,j_6 \times x6)) + b$

$h6 = ((0.36247341 \times -0.27574518) + (-0.03712616 \times -0.3866881) + (-0.35457662 \times -0.18542552) + (0.12722402 \times -0.03321722) + (0.27924119 \times -0.18747924) + (-0.40418496 \times -0.34349378)) + 0$

$h6 = 0.02437631376$

Setelah mendapatkan hasil perhitungan terhadap *hidden layer*, selanjutnya diterapkan *activation function tanh* sesuai rumus 2.5.

$tanh\ (h1) = \frac{e^{h1} - e^{-h1}}{e^{h1} + e^{-h1}}$

$tanh\ (h1) = \frac{e^{0.02738362964} - e^{-0.02738362964}}{e^{0.02738362964} + e^{-0.02738362964}}$

$tanh\ (h1) = 0.027376$

$tanh\ (h2) = \frac{e^{h2} - e^{-h2}}{e^{h2} + e^{-h2}}$

$tanh\ (h2) = \frac{e^{0.007906679016} - e^{-0.007906679016}}{e^{0.007906679016} + e^{-0.007906679016}}$

$tanh\ (h2) = 0.007906$

$tanh\ (h3) = \frac{e^{h3} - e^{-h3}}{e^{h3} + e^{-h3}}$

$tanh\ (h3) = \frac{e^{0.07769713625} - e^{-0.07769713625}}{e^{0.07769713625} + e^{-0.07769713625}}$

$tanh\ (h3) = 0.077541$

$tanh\ (h4) = \frac{e^{h4} - e^{-h4}}{e^{h4} + e^{-h4}}$

$tanh\ (h4) = \frac{e^{-0.2331063547} - e^{--0.2331063547}}{e^{-0.2331063547} + e^{--0.2331063547}}$

$tanh\ (h4) = -0.228973$

$tanh\ (h5) = \frac{e^{h5} - e^{-h5}}{e^{h5} + e^{-h5}}$

$tanh\ (h5) = \frac{e^{-0.08866090313} - e^{--0.08866090313}}{e^{-0.08866090313} + e^{--0.08866090313}}$

$tanh\ (h5) = -0.088429$

$tanh\ (h6) = \frac{e^{h6} - e^{-h6}}{e^{h6} + e^{-h6}}$

$tanh\ (h6) = \frac{e^{0.02437631376} - e^{-0.02437631376}}{e^{0.02437631376} + e^{-0.02437631376}}$

$tanh\ (h6) = -0.024371$

Selanjutnya, dilakukan perhitungan terhadap *output layer* dengan menggunakan rumus 2.1 dan 2.3:

$output = ((w_{k1} j_7 \times \tanh(h1)) + (w_{k2} j_7 \times \tanh(h2)) + (w_{k3} j_7 \times \tanh(h3)) + (w_{k4} j_7 \times \tanh(h4)) + (w_{k5} j_7 \times \tanh(h5)) + (w_{k6} j_7 \times \tanh(h6))) + b$

$output = ((-0.01376106 \times 0.027376) + (-0.34259953 \times 0.007906) + (-0.21056884 \times 0.077541) + (0.12141414 \times -0.228973) + (-0.0607964 \times -0.088429) + (-0.19922706 \times -0.024371) + 0)$

$output = -0.03698206543$

Langkah terakhir, adalah melakukan perhitungan dengan *activation function sigmoid* pada *output layer* dengan menggunakakan rumus 2.4 untuk mendapatkan hasil prediksi antara 0 (tidak stroke) atau 1 (stroke):

$\sigma(z) = = 1/(1+exp(-z))$
$\sigma(-0,03698206543)$ = 1 / (1 + exp(–0,03698206543))
$\sigma(-0,03698206543)$ = 1 / (1 + exp(–0,03698206543))
$\sigma(-0,03698206543)$ = 1 / (1 + 1.03767441)
$\sigma(-0,03698206543)$ = 1 / 2.03767441
$\sigma(-0,03698206543)$ = 0.4907555372

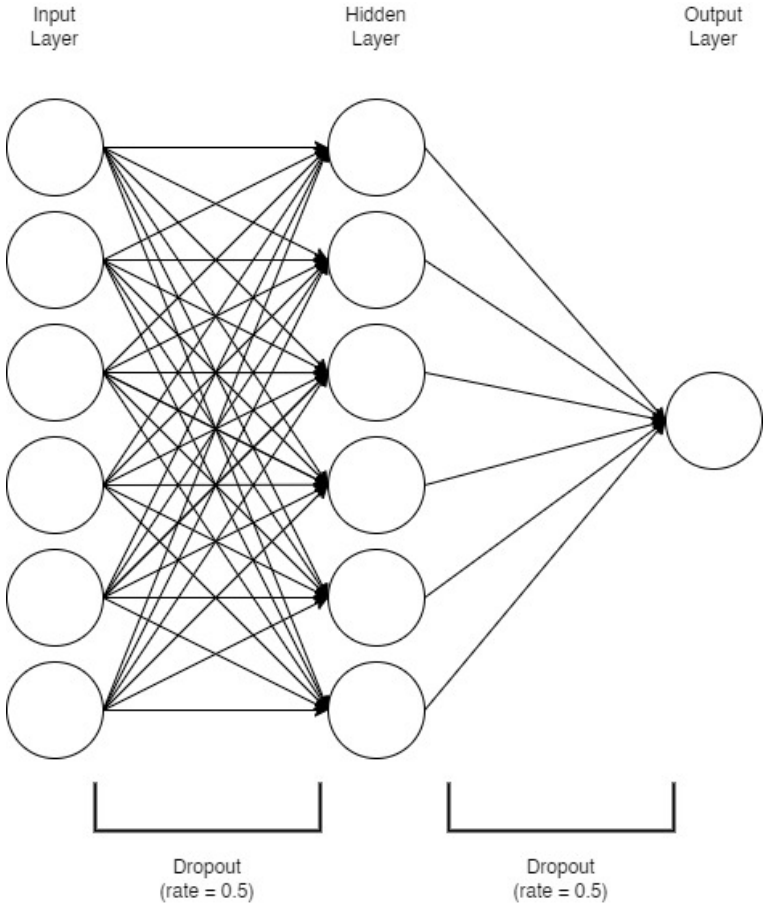
Pada perhitungan di atas, hasil *output neuron* adalah 0.4907555372 yang akan diklasifikasikan ke kelas 0 atau tidak terkena penyakit stroke. Jika hasil *output neuron* lebih kecil dari 0.5 , maka data tersebut masuk ke dalam kelas 0, yaitu tidak terkena penyakit stroke, namun jika hasil *output* menunjukkan angka lebih besar sama dengan 0.5, maka data tersebut masuk ke dalam kelas 1, yang artinya terkena penyakit stroke.

3.6 Perhitungan *Deep Neural Network dengan Dropout*

Arsitektur yang akan digunakan dalam *neural network* dibagi menjadi 3 *layer* yang ditambah dengan *dropout* antara tiap *layer*, yaitu:

1. *S*atu buah input layer yang terdiri dari 6 *neuron* yang merepresentasikan 6 fitur dan menggunakan *activation function* tanh.
2. Di antara *input layer* dan *hidden layer*, diberikan *dropout* dengan *rate* = 0.5.
3. *S*atu buah hidden layer yang terdiri dari 6 *neuron* yang menggunakan *activation function* tanh.
4. Di antara *hidden layer* dan *output layer*, diberikan *dropout* dengan *rate* = 0.5.
5. Satu buah *output layer* yang terdiri dari 1 *neuron* dan menggunakan *activation function sigmoid*.

Gambar arsitektur Deep Neural Network dengan *dropout* dapat dilihat pada gambar 3.15.



Gambar 3.15 Arsitektur *Deep Neural Network* dengan *dropout*

Nilai *weight* didapatkan dengan cara diinisialisasi melalui kode Python dan menghasilkan 62 data secara *random* dengan rentang tertentu.

Inisialisasi Data:

Jumlah *input layer* = 1

Jumlah *neuron* pada *input layer* = 6

Activation function pada *input layer* = tanh

Dropout rate = 0.5

Jumlah *hidden layer* = 1

Jumlah *neuron* pada *hidden layer* = 6

Activation function pada *hidden layer* = tanh

Dropout rate = 0.5

Jumlah *output layer* = 1

Jumlah *neuron* pada *output layer* = 1

Activation function pada *output layer = sigmoid*

Rentang *weight* = -0.4082482904638631 sampai dengan +0.4082482904638631

Bias = 0

Input (weight setiap fitur yang ada):

$x1$ = -0.27574518 $x2$ = -0.3866881 $x3$ = -0.18542552 $x4$ = -0.03321722 $x5$ = -0.18747924 $x6$ = -0.34349378

Hidden layer:

$w_{k1}j_1$ = 0.32069702 $w_{k2}j_1$ = -0.09854238 $w_{k3}j_1$ = 0.32952055 $w_{k4}j_1$ = -0.40533615 $w_{k5}j_1$ = 0.2939386 $w_{k6}j_1$ = -0.17256972

$w_{k1}j_2$ = 0.09631948 $w_{k2}j_2$ = 0.04388243 $w_{k3}j_2$ = 0.12275857 $w_{k4}j_2$ = -0.12201437 $w_{k5}j_2$ = 0.1916928 $w_{k6}j_2$ = -0.2026422

$w_{k1}j_3$ = -0.1264057 $w_{k2}j_3$ = -0.23267549 $w_{k3}j_3$ = -0.16984761 $w_{k4}j_3$ = 0.07741352 $w_{k5}j_3$ = -0.24032459 $w_{k6}j_3$ = 0.28520692

$w_{k1}j_4$ = 0.20277835 $w_{k2}j_4$ = 0.31052999 $w_{k3}j_4$ = -0.14207178 $w_{k4}j_4$ = 0.27065191 $w_{k5}j_4$ = -0.18517663 $w_{k6}j_4$ = 0.08230256

$w_{k1}j_5$ = 0.382188915 $w_{k2}j_5$ = -0.13970395 $w_{k3}j_5$ = -0.25528209 $w_{k4}j_5$ = 0.13248654 $w_{k5}j_5$ = $w_{k6}j_5$ = 0.31288484

$w_{k1}j_6$ = 0.36247341 $w_{k2}j_6$ = -0.03712616 $w_{k3}j_6$ = -0.35457662 $w_{k4}j_6$ = 0.12722402 $w_{k5}j_6$ = 0.27924119 $w_{k6}j_6$ = -0.40418496

Output layer:

$w_{k1}j_7$ = 0.01376106 $w_{k2}j_7$ = -0.34259953 $w_{k3}j_7$ = -0.21056884 $w_{k4}j_7$ = 0.12141414 $w_{k5}j_7$ = -0.060796 $w_{k6}j_7$ = -0.19922706

Berikut ini merupakan perhitungan terhadap *hidden layer* dengan menggunakan rumus 2.1 dan 2.3:

$h1 = ((w_{k1}\,j_1 \times x1) + (w_{k2}\,j_1 \times x2) + (w_{k3}\,j_1 \times x3) + (w_{k4}\,j_1 \times x4) + (w_{k5}\,j_1 \times x5) + (w_{k6}\,j_1 \times x6)) + b$

$h1$ = ((0.32069702 × -0.27574518) + (-0.09854238 × -0.3866881) + (0.32952055 × -0.18542552) + (-0.40533615 × -0.03321722) + (0.2939386 × -0.18747924) + (-0.17256972 × -0.34349378)) + 0

$h1 = 0.02738362964$

$h2 = ((w_{k1}\,j_2 \times x1) + (w_{k2}\,j_2 \times x2) + (w_{k3}\,j_2 \times x3) + (w_{k4}\,j_2 \times x4) + (w_{k5}\,j_2 \times x5) + (w_{k6}\,j_2 \times x6)) + b$

$h2$ = ((0.09631948 × -0.27574518) + (0.04388243 × -0.3866881) + (0.12275857 × -0.18542552) + (-0.12201437 × -0.03321722) + (0.1916928 × -0.18747924) + (-0.2026422 × -0.34349378)) + 0

$h2 = 0.007906679016$

$h3 = ((w_{k1}\,j_3 \times x1) + (w_{k2}\,j_3 \times x2) + (w_{k3}\,j_3 \times x3) + (w_{k4}\,j_3 \times x4) + (w_{k5}\,j_3 \times x5) + (w_{k6}\,j_3 \times x6)) + b$

$h3$ = ((-0.1264057 × -0.27574518) + (-0.23267549 × -0.3866881) + (-0.16984761 × -0.18542552) + (0.07741352 × -0.03321722) + (-0.24032459 × -0.18747924) + (0.28520692 × -0.34349378)) + 0

$h3 = 0.07769713625$

$h4 = ((w_{k1}\,j_4 \times x1) + (w_{k2}\,j_4 \times x2) + (w_{k3}\,j_4 \times x3) + (w_{k4}\,j_4 \times x4) + (w_{k5}\,j_4 \times x5) + (w_{k6}\,j_4 \times x6)) + b$

$h4$ = ((0.20277835 × -0.27574518) + (0.31052999 × - 0.3866881) + (-0.14207178 × -0.18542552) + (0.27065191 × -0.03321722) + (-0.18517663 × -0.18747924) + (0.08230256 × -0.34349378)) + 0

$h4 = -0.2331063547$

$h5 = ((w_{k1}\,j_5 \times x1) + (w_{k2}\,j_5 \times x2) + (w_{k3}\,j_5 \times x3) + (w_{k4}\,j_5 \times x4) + (w_{k5}\,j_5 \times x5) + (w_{k6}\,j_5 \times x6)) + b$

$h5$ = ((0.382188915 × -0.27574518) + (-0.13970395 × -0.3866881) + (-0.25528209 × -0.18542552) + (0.13248654 × -0.03321722) + (-0.35657559 × -0.18747924) + (0.31288484 × -0.34349378)) + 0

$h5 = -0.08866090313$

$h6 = ((w_{k1}\,j_6 \times x1) + (w_{k2}\,j_6 \times x2) + (w_{k3}\,j_6 \times x3) + (w_{k4}\,j_6 \times x4) + (w_{k5}\,j_6 \times x5) + (w_{k6}\,j_6 \times x6)) + b$

$h6$ = ((0.36247341 × -0.27574518) + (-0.03712616 × -0.3866881) + (-0.35457662 × -0.18542552) + (0.12722402 × -0.03321722) + (0.27924119 × -0.18747924) + (-0.40418496 × -0.34349378)) + 0

$h6 = 0.02437631376$

Setelah mendapatkan hasil perhitungan terhadap *hidden layer*, selanjutnya diterapkan *activation function tanh* sesuai rumus 2.5.

$tanh\ (h1) = \frac{e^{h1} - e^{-h1}}{e^{h1} + e^{-h1}}$

$tanh\ (h1) = \frac{e^{0.02738362964} - e^{-0.02738362964}}{e^{0.02738362964} + e^{-0.02738362964}}$

$tanh\ (h1) = 0.027376$

$tanh\ (h2) = \frac{e^{h2} - e^{-h2}}{e^{h2} + e^{-h2}}$

$tanh\ (h2) = \frac{e^{0.007906679016} - e^{-0.007906679016}}{e^{0.007906679016} + e^{-0.007906679016}}$

$tanh\ (h2) = 0.007906$

$tanh\ (h3) = \frac{e^{h3} - e^{-h3}}{e^{h3} + e^{-h3}}$

$tanh\ (h3) = \frac{e^{0.07769713625} - e^{-0.07769713625}}{e^{0.07769713625} + e^{-0.07769713625}}$

$tanh\ (h3) = 0.077541$

$tanh\ (h4) = \frac{e^{h4} - e^{-h4}}{e^{h4} + e^{-h4}}$

$tanh\ (h4) = \frac{e^{-0.2331063547} - e^{--0.2331063547}}{e^{-0.2331063547} + e^{--0.2331063547}}$

$tanh\ (h4) = -0.228973$

$tanh\ (h5) = \frac{e^{h5} - e^{-h5}}{e^{h5} + e^{-h5}}$

$tanh\ (h5) = \frac{e^{-0.08866090313} - e^{--0.08866090313}}{e^{-0.08866090313} + e^{--0.08866090313}}$

$tanh\ (h5) = -0.088429$

$tanh\ (h6) = \frac{e^{h6} - e^{-h6}}{e^{h6} + e^{-h6}}$

$tanh\ (h6) = \frac{e^{0.02437631376} - e^{-0.02437631376}}{e^{0.02437631376} + e^{-0.02437631376}}$

$tanh\ (h6) = -0.024371$

Setelah mendapatkan hasil perhitungan *activation function tanh*, dilakukan perhitungan *dropout* sesuai rumus 2.8:

$dropout_1$ = rate * tanh($h1$)
$dropout_1$ = 0.5 * 0.027376
$dropout_1$ = 0.013688

$dropout_2$ = rate * tanh($h2$)
$dropout_2$ = 0.5 * 0.007906
$dropout_2$ = 0.003953

$dropout_3$ = rate * tanh($h3$)
$dropout_3$ = 0.5 * 0.077541
$dropout_3$ = 0.0387705

$dropout_4$ = rate * tanh($h4$)
$dropout_4$ = 0.5 * -0.228973
$dropout_4$ = -0.1144865

$dropout_5$ = rate * tanh($h5$)
$dropout_5$ = 0.5 * -0.088429
$dropout_5$ = -0.0442145

$dropout_6$ = rate * tanh($h6$)
$dropout_6$ = 0.5 * -0.024371
$dropout_6$ = -0.0121855

Selanjutnya, dilakukan perhitungan terhadap *output layer* dengan menggunakan rumus 2.1 dan 2.3:

$output = ((w_{k1} j_7 \times dropout_1) + (w_{k2} j_7 \times dropout_2) + (w_{k3} j_7 \times dropout_3) + (w_{k4} j_7 \times dropout_4) + (w_{k5} j_7 \times dropout_5) + (w_{k6} j_7 \times dropout_6)) + b$

$output$ = ((-0.01376106 × 0.013688) + (-0.34259953 × 0.003953) + (-0.21056884 × 0.0387705) + (0.12141414 × -0.1144865) + (-0.0607964 × -0.0442145) + (-0.19922706 × -0.0121855) + 0)

$output$ = -0.01849103271

Langkah terakhir, adalah melakukan perhitungan dengan *activation function sigmoid* pada *output layer* dengan menggunakakan rumus 2.4 untuk

mendapatkan hasil prediksi antara 0 (tidak stroke) atau 1 (stroke):

$\sigma(z) = = 1/(1+exp(-z))$
$\sigma(-0.01849103271)$ = 1 / (1 + exp(–0.01849103271))
$\sigma(-0.01849103271)$ = 1 / (1 + 1.03767441)
$\sigma(-0.01849103271)$ = 1 / 2.03767441
$\sigma(-0.01849103271)$ = 0.4907555372

Pada perhitungan di atas, hasil *output neuron* adalah 0.4907555372 yang akan diklasifikasikan ke kelas 0 atau tidak terkena penyakit stroke. Jika hasil *output neuron* lebih kecil dari 0.5 , maka data tersebut masuk ke dalam kelas 0, yaitu tidak terkena penyakit stroke, namun jika hasil *output* menunjukkan angka lebih besar sama dengan 0.5, maka data tersebut masuk ke dalam kelas 1, yang artinya terkena penyakit stroke.

3.7 Perhitungan Cost untuk Cost-Sensitive Learning

Sesuai dengan rumus yang sudah dijelaskan pada persamaan 2.11, maka hasil *output* dari perhitungan *Deep Neural Network* dengan *dropout* akan digunakan untuk mencari *cost*.

$O_i^*(x) = \eta \sum_{j=1}^{M} O_i(x)C(i,j)$
$O_i^*(x) = \eta(0.4907555372 * (42336/781))$
$O_i^*(x) = \eta 0.4907555372 * 54.207426$
$O_i^*(x) = \eta 26.8531325$
$O_i^*(x) = 1$

Pada perhitungan di atas, dikarenakan *output neural network* yang dipakai hanya 1 buah, maka hasil *cost* dari kesalahan klasifikasi jika dinormalisasi pasti akan menghasilkan angka 1. Jika *output neural network* lebih dari 1, maka hasil normalisasi akan lebih terlihat. Hasil *cost* ini akan dibandingkan dengan *output* dari *weight neural network* lainnya dan akan diambil nilai yang terbesar sebagai nilai *cost-sensitive learning*.

DAFTAR REFERENSI

[1] Kementerian Kesehatan Republik Indonesia, ”Infodatin Pusat Data dan Informasi Kementerian Kesehatan RI Stroke Don’t Be The One”, Kementerian Kesehatan Republik Indonesia, 2019. [Online]. Available: https://pusdatin.kemkes.go.id/resources/download/pusdatin/infodatin/infodatin-stroke-dont-be-the-one.pdf. [Accessed: Oct 9, 2021].

[2] G. Sailasya and G. L. A. Kumari, ”Analyzing the Performance of Stroke Prediction using ML Classification Algorithms,” *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 6, pp. 539-545, 2021.

[3] M. S. Azam, M. Habibullah ,and H. K. Rana, ”Performance Analysis of Various Machine Learning Approaches in Stroke Prediction,” *International Journal of Computer Applications*, vol. 175, no. 21, pp. 11-15, 2020.

[4] Maya B S and Asha T, ”Predictive Model for Brain Stroke in CT using Deep Neural Network”, *International Journal of Recent Technology and Engineering (IJRTE)*, vol. 9, no. 1, pp. 2011-2017, 2020.

[5] C. Y. Hung, W. C. Chen, P. T. Lai, C. H. Lin, and C. C. Lee, ”Comparing Deep Neural Network and Other Machine Learning Algorithms for Stroke Prediction in a Large-Scale Population-Based Electronic Medical Claims Database.” *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp 3110-3113, 2017.

[6] J. Choi, S. Y. Seo, P. J. Kim, Y. S. Kim, S. H. Lee, J. H. Sohn, D. K. Kim, J. J. Lee and C. Kim, ”Prediction of Hemorrhagic Transformation after Ischemic Stroke Using Machine Learning,” *Journal of Personalized Medicine*, vol. 11, no. 9, pp. 1-11, 2021.

[7] N. Someeh, M. A. Jafarabadi, S. M. Shamshirgaran, F. Farzipoor, ”The outcome in patients with brain stroke: A deep learning neural network modeling,” *Journal of Research in Medical Sciences*, vol. 25, no. 1, pp. 1-7, 2020.

[8] S. Cheon, J. Kim, and J. Lim, ”The Use of Deep Learning to Predict Stroke Patient Mortality,” *International Journal of Environmental Research and Public Health*, vol. 16, no. 11, 2019.

[9] B. Krawczyk and M. Wo´zniak, ”Cost-Sensitive Neural Network with ROC-Based Moving Threshold for Imbalanced Classification,” pp. 45-52, 2015.

[10] C. Garbin, X. Zhu, and O. Marques, ”Dropout vs. batch normalization: an empirical study of their impact to deep learning,” *Multimedia Tools and Applications*, vol. 79, no. 19, pp 12777-12815, 2020.

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, ”Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1930-1958, 2014.

[12] C. Chugh, ”Acute Ischemic Stroke: Management Approach” *Indian Journal of Critical Care Medicine*, vol. 23, pp. S140–S146, 2019.

[13] A. K. Boehme, C. Esenwa, and M. S.V. Elkind, ”Stroke Risk Factors, Genetics, and Prevention” *Circulation Research*, vol. 120, no. 3 pp.472-495, 2017.

[14] J. Grus, *Data Science from Scratch*, O'Reilly Media, Inc, 2015.

[15] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow 2nd Edition*, O'Reilly Media, Inc, 2019.

[16] S. Haykin, *Neural Networks and Learning Machines Third Edition*, Pearson, 2009.

[17] J. Brownlee. ”Step-By-Step Framework for Imbalanced Classification Projects”. *Machine Learning Mastery*. 2020. [Online]. Available: https://machinelearningmastery.com/framework-for-imbalanced-classification-projects/. [Accessed: Jan 30, 2022].

[18] J. Brownlee. ”Cost-Sensitive Learning for Imbalanced Classification”. *Machine Learning Mastery*. 2020. [Online]. Available: https://machinelearningmastery.com/cost-sensitive-learning-for-imbalanced-classification/. [Accessed: Jan 30, 2022].

[19] A. Fernández, S. García, M. Galar, R. C. Prati, B Krawczyk, and F. Herrera, ”Learning from Imbalanced Data Sets,” *Springer*, 2014.

[20] K. Munir, H. Elahi, A. Ayub, F. Frezza, A. Rizzi, ”Cancer Diagnosis Using Deep Learning: A Bibliographic Review,” *Cancers*, vol. 11, no. 9, pp. 1235, 2019.

[21] Y. Ma, H. He., *Imbalanced Learning: Foundations, Algorithms, and Applications 1st Edition*, Wiley, 2013.

[22] J. Brownlee. ”Tour of Evaluation Metrics for Imbalanced Classification”. *Machine Learning Mastery*. 2020. [Online]. Available: https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/. [Accessed: Jan. 30, 2022].

[23] S. Tiwari, 2019. Cerebral Stroke Prediction-Imbalanced Dataset. [Online]. Available: https://www.kaggle.com/shashwatwork/cerebral-stroke-predictionimbalaced-dataset [Accessed: Oct 30, 2021].

[24] A. Ashiquzzaman, A. K. Tushar, M. R. Islam, D. S. K. Im, J. H. Park, D. S. Lim, and J. Kim, ”Reduction of Overfitting in Diabetes Prediction Using Deep Learning Neural Network,” *IT Convergence and Security*, pp. 35-43, 2017.

[25] dr. Tania Savitri. *Hello Sehat*. ”Awas, Ini Akibatnya Jika Gula Darah Anda Terlalu Tinggi”. 2022. [Online]. Available at: https://hellosehat.com/diabetes/akibat-gula-darah-tinggi/. [Accessed: Feb. 10, 2022].

[26] National Heart, Lung, and Blood Institute. ”Body Mass Index Table”. [Online]. Available at: https://www.nhlbi.nih.gov/health/educational/lose_wt/BMI/bmi_tbl.pdf. [Accessed: Feb. 10, 2022].