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

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

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*, dan *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 *imbalanced*.

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 *imbalanced* 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 *imbalanced* 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 *testing*.
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 *dropout*.
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 imbalanced 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

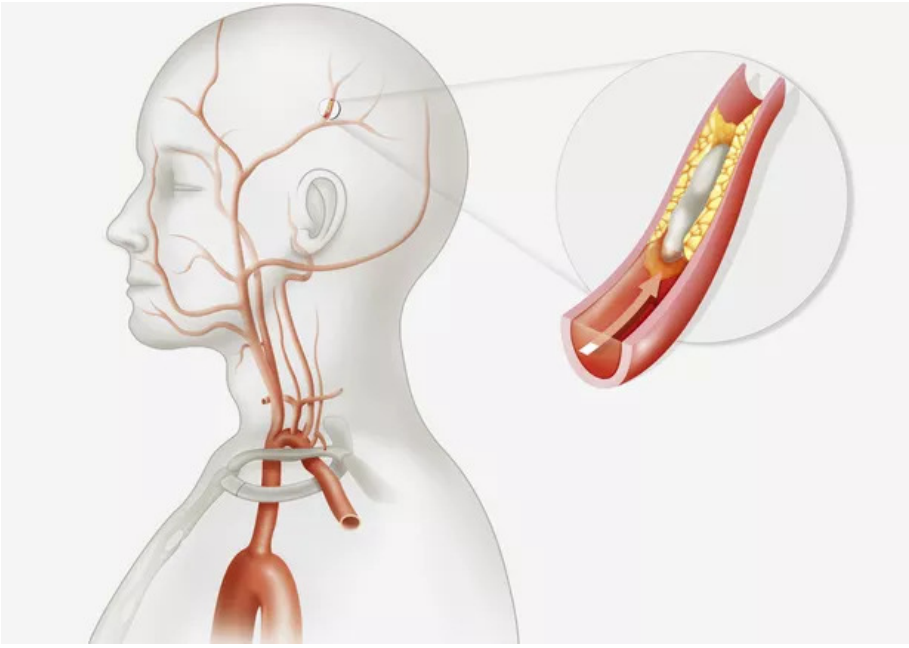
2.1 Tinjauan Pustaka

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]:

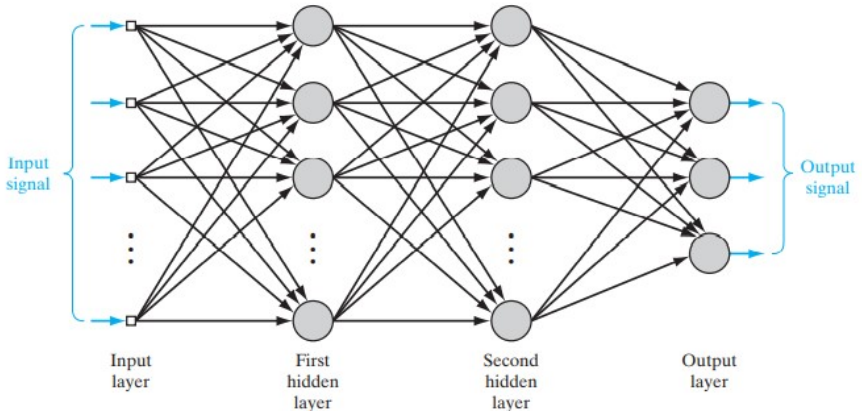
1. Faktor kesehatan, seperti tekanan hipertensi (lebih dari 140/90), mengidap penyakit jantung (gagal jantung, penyakit jantung bawaan, infeksi jantung, atau aritmia), obesitas, kolestrol tinggi, mengidap penyakit diabetes, *sleep apnea*, dan pernah mengalami TIA (stroke ringan) atau serangan jantung 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]

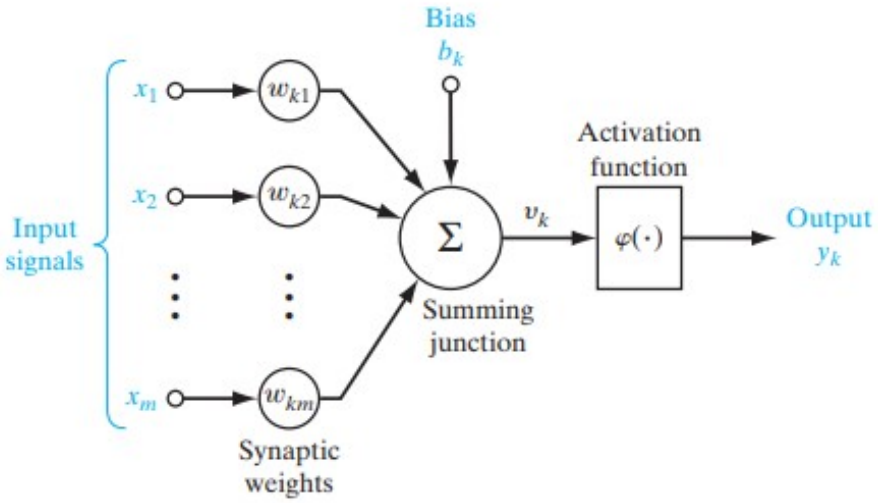


Gambar 2.2 *Neural Network* [16]

Gambar 3.16 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*), dan *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*

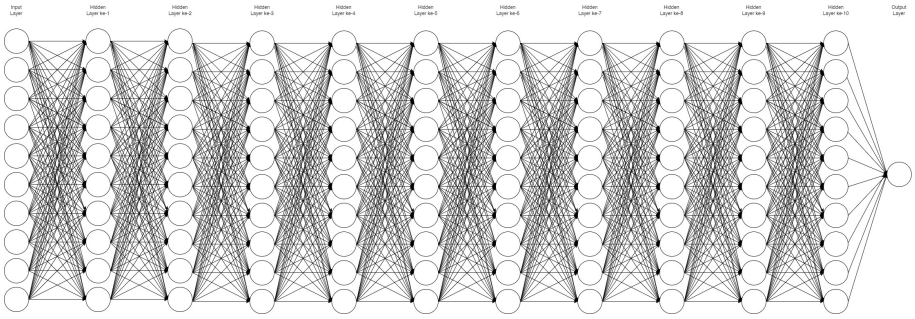
x : *input*

b_k : bias

φ : *activation function*

Perbedaan *Neural Network* dengan *Deep Neural Network* adalah pada jumlah *hidden layer*-nya. Dalam Buku [15] dan Penelitian [17] disebutkan bahwa

jika jumlah *hidden layer* lebih dari 2, sudah termaksud *Deep Neural Network*, dan dalam Penelitian [17], dijelaskan bahwa *Neural Network* dengan lebih dari 1 *hidden layer* akan menjadi kompleks. Gambar 2.4 di bawah ini merupakan contoh gambar arsitektur Deep Neural Network yang akan digunakan dalam penelitian ini.



Gambar 2.4 Contoh arsitektur DNN yang akan digunakan

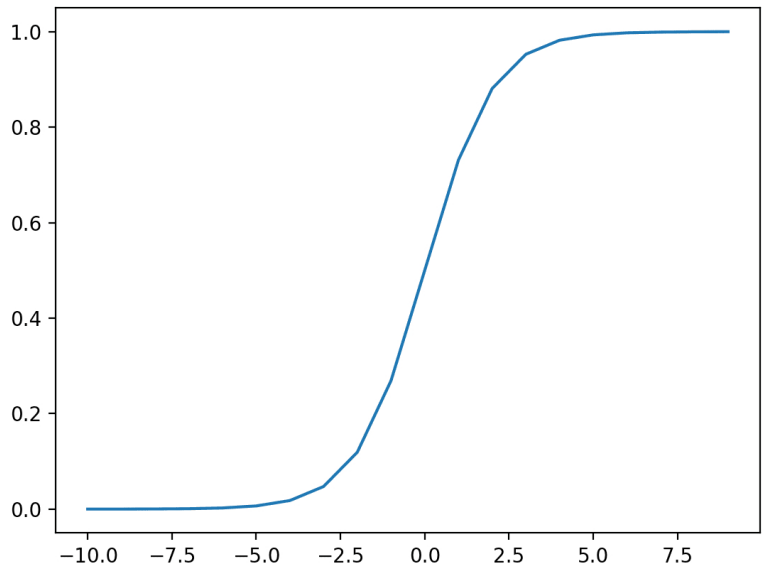
2.1.2.1 Activation Function

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 2.5.

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

Keterangan :
z : input
σ : fungsi *sigmoid*

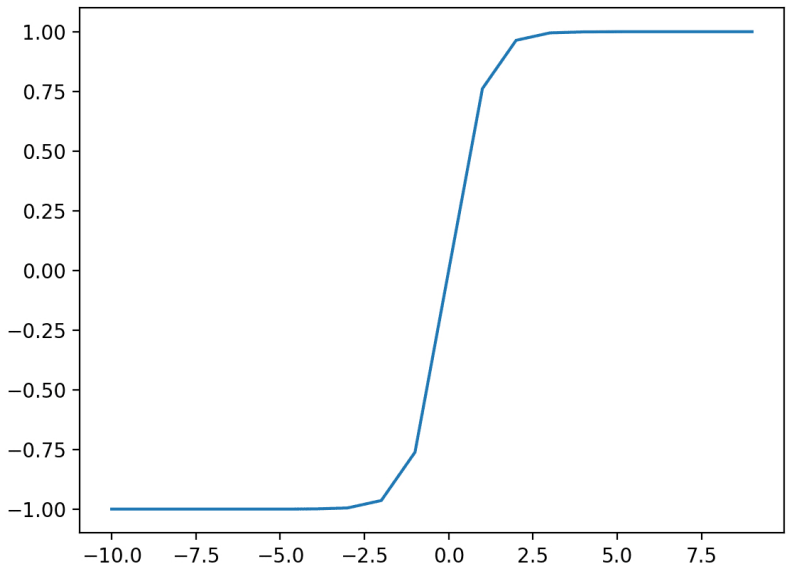


Gambar 2.5 Fungsi *sigmoid*

2. *Hyperbolic tangent* (tanh), yang memiliki rentang antara -1 sampai 1. Rumus dan kurva *tanh* dapat dilihat pada Persamaan 2.5 dan Gambar 2.6

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

Keterangan :
x : input



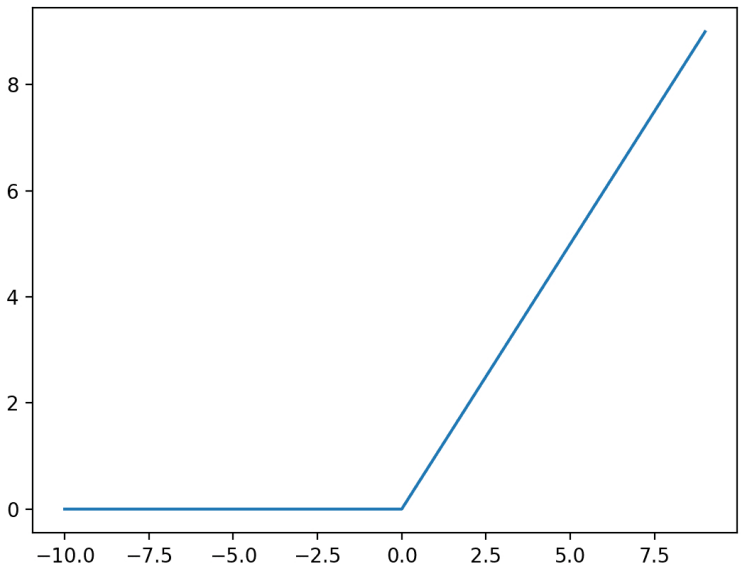
Gambar 2.6 Fungsi *tanh*

‘

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 2.7

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

Keterangan :
z : input
σ : fungsi *ReLU*



Gambar 2.7 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, dalam setiap fase *training*, semua 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 2.7, 2.8, 2.9, dan 2.10 berikut.

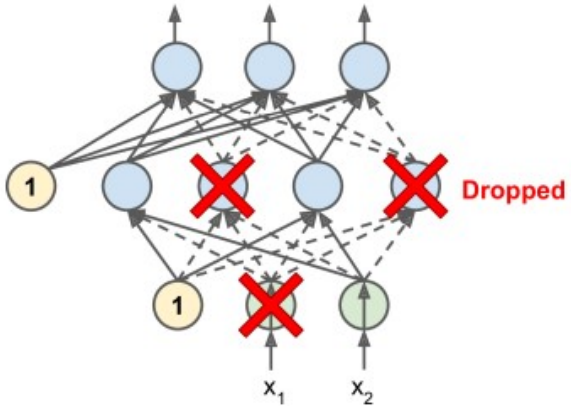
$$r_j^l \sim Bernoulli(p) \tag{2.7}$$

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

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

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

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 2.8 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 dengan 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 *Imbalanced Class*

Bagian ini akan menjelaskan berbagai macam teknik untuk mengatasi *imbalanced 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 [18]:

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 [20]:

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]. Gambar 2.9 adalah gambar *cost matrix*.

	Actual Negative	Actual Positive
Predicted Negative	C(0,0), TN	C(0,1), FN
Predicted Positive	C(1,0), FP	C(1,1), TP

Gambar 2.9 *Cost matrix*

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
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*

Probability Tuning dibagi 2 cara, yaitu [20]:

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*

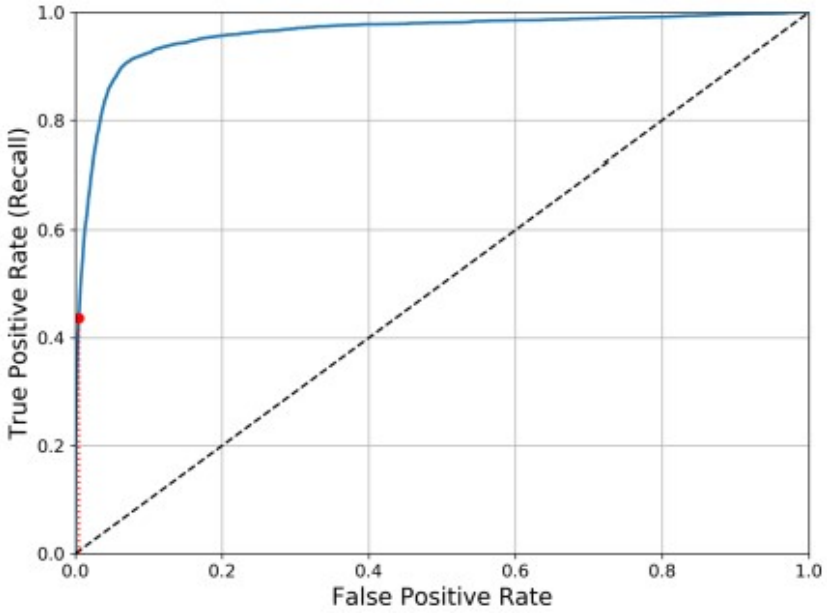
Receiver Operating Characteristic (ROC) *curve* adalah metode untuk melakukan pengukuran, yang tujuannya melakukan *plotting true positive rate* (TPR) atau *recall* atau *sensitivity* dengan *false positive rate* (FPR). FPR sendiri adalah rasio negatif yang salah diklasifikasikan sebagai kelas positif. 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 2.10 Contoh 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 [21]. 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 [22]. 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* [23].

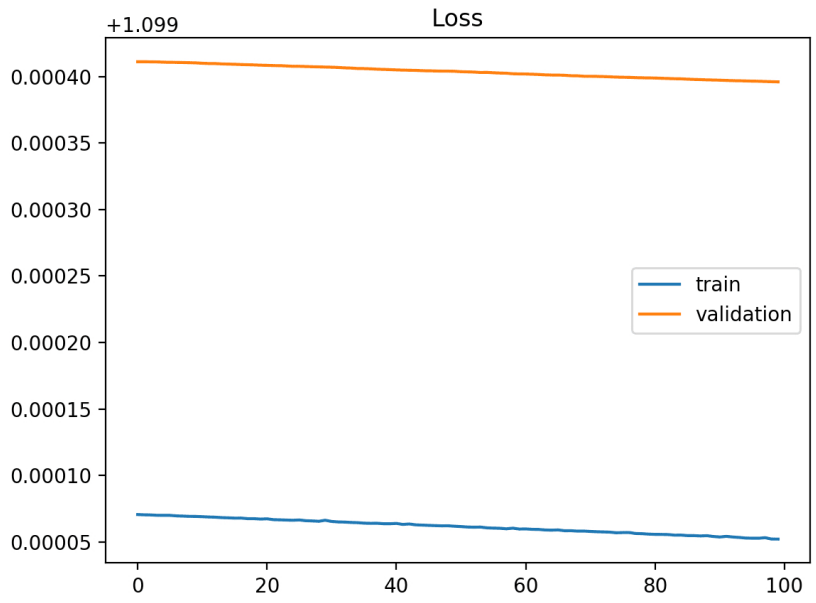
2.1.5 Learning Curves

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

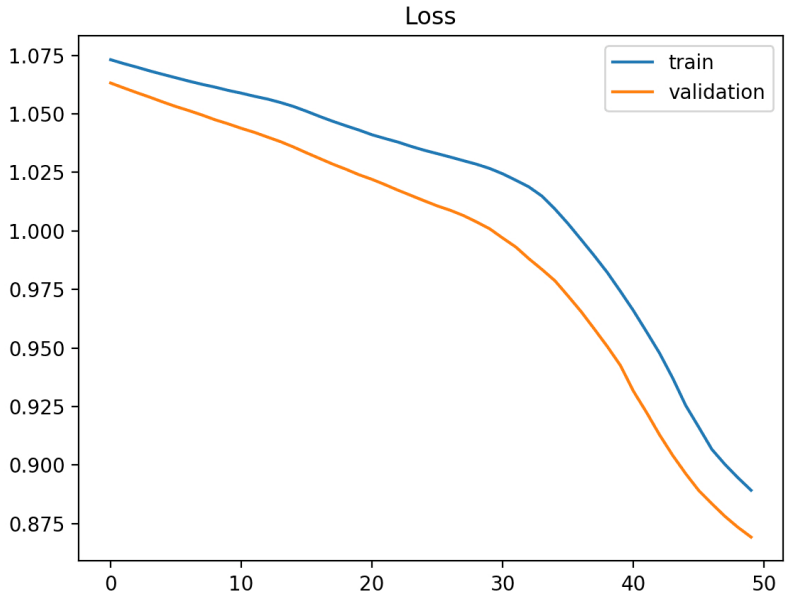
1. *Train learning curve*, yaitu *learning curves* yang dihitung dari *dataset training* yang berfungsi untuk menggambarkan seberapa baik model belajar.
2. *Validation learning curve*, yaitu *learning curves* yang dihitung dari *dataset validation* yang berfungsi untuk menggambarkan seberapa baik model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

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

1. *Underfit*, yaitu model tidak dapat mempelajari *dataset training*. Gambar 2.11 dan 2.12 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 2.11 *Learning curves* yang mengalami *underfitting*

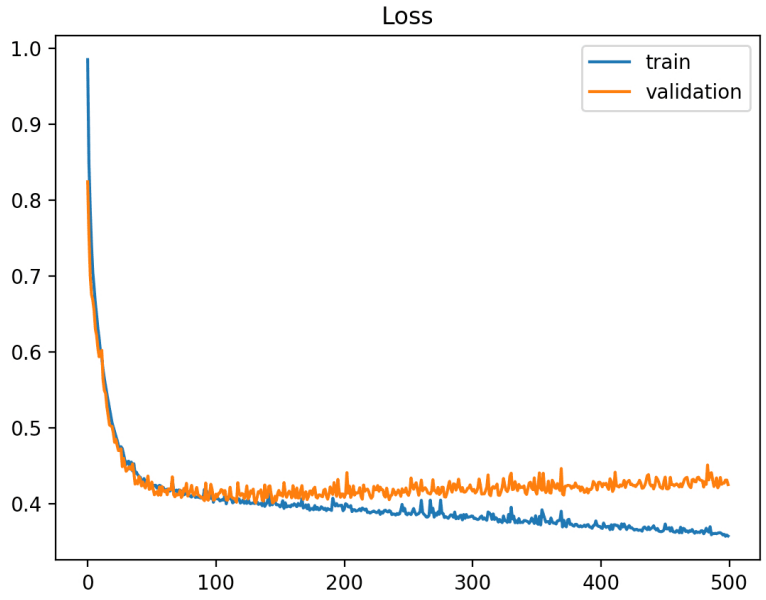


Gambar 2.12 *Learning curves* yang mengalami *underfitting*

Gambar 2.11 di atas menjelaskan bahwa model yang dibuat tidak dapat mempelajari data sama sekali, sedangkan pada Gambar 2.12 menjelaskan

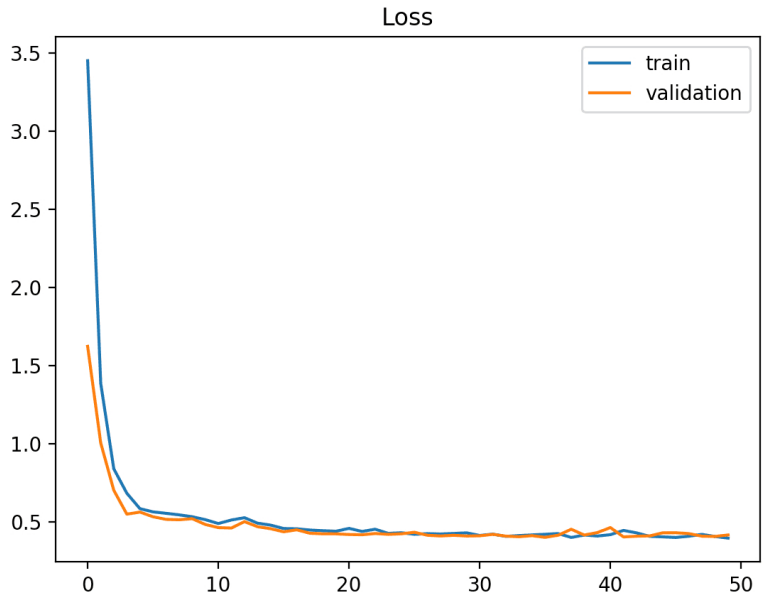
bahwa model sebenarnya mampu belajar lebih lanjut dan hal ini dapat terjadi karena proses *training* yang sudah dihentikan sebelum waktunya.

2. *Overfit*, yaitu model yang mempelajari *dataset training* dengan terlalu baik, termaksud *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 2.13 adalah contoh *learning curves* yang mengalami *overfitting*.



Gambar 2.13 *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 2.14 adalah contoh *learning curves* yang bersifat *good fit*.



Gambar 2.14 *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* Numpy

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.

2.1.6.6 Keras

Keras adalah *library* yang digunakan untuk melakukan pemrosesan *deep learning*.

Tabel 2.6 Daftar metode yang digunakan dari *library* Keras

No	Metode	Masukan	Luaran	Keterangan
1	model.add	hidden_dim: int, activation_function: string, input_dim: int	-	Inisialisasi jumlah layer dan *activation function* yang akan dipakai.
2	model.compile	learning_rate: float, loss: string, metrics: string	-	Inisialisasi *learning rate*, dan jenis pengukuran apa yang akan dipakai.
3	model.fit	x_train: float[], y_train: float[], epoch: int	-	Menjalankan proses *neural network* sesuai inisialisasi dan konfigurasi yang sudah dilakukan sebelumnya.
4	model.evaluate	x_test: float[], y_test: float[]	-	Mengevaluasi hasil proses *neural network* yang sudah dijalankan sebelumnya terhadap data *testing*.

2.2 Tinjauan Studi

Pada Tabel 2.7 diberikan penjelasan mengenai studi terkait dalam penelitian:

Tabel 2.7 Tinjauan Studi

No	Judul	Rumusan Masalah	Metode	Hasil
1	S. Dev, H. Wang, C.S. Nwosu, N. Jain, B. Veeravalli, and D. John, ”A Predictive Analytics Approach For Stroke Prediction Using Machine Learning and Neural Networks”, *Healthcare Analytics*, 2022. [24]	Apakah dengan menggunakan PCA dapat mencari variabel yang paling berperan penting dalam kasus stroke?	1. *Neural Network* + *Principal Component Analysis* 2. *Random undersampling*	Model yang dibuat dengan Neural Network tanpa PCA meraih akurasi dan recall sebesar 77% dan 74%, sedangkan model yang dibuat dengan *feature extraction* PCA dan Neural Network meraih akurasi dan *recall* 75% dan 68%, dengan variabel yang paling berpengaruh adalah *gender*, *age*, *hypertension*, *ever_married*, *work_type*, dan *avg_glucose_level*.

2	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]	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, 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. [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-Imbalanced Dataset* yang didapatkan dari Mendeley Data dan diterbitkan di situs Kaggle [26]. Dataset 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. HealthData.gov adalah sebuah situs web pemerintah Amerika Serikat yang dikelola oleh *U.S. Department of Health & Human Services*. Dataset ini bersifat *imbalance* dengan data orang yang menderita stroke sebesar 783 dan data orang yang tidak menderita penyakit stroke sebesar 42.617 data. Oleh karena itu, perlu penanganan khusus untuk mengatasi *imbalance class*, yaitu dengan menggunakan teknik *cost-sensitive learning* dan *probability tuning*.

BAB 3 ANALISIS DAN PERANCANGAN SISTEM

3.1 Analisis Masalah

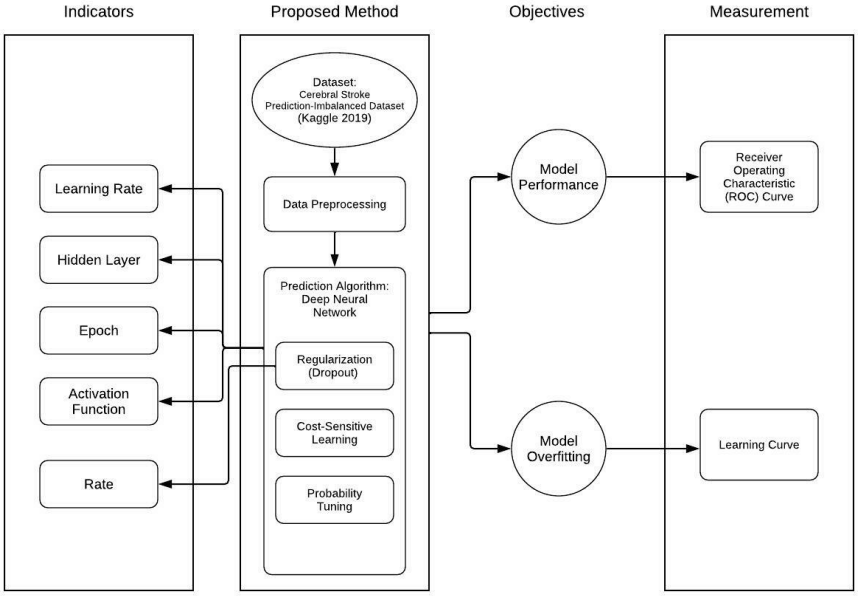
Seperti yang sudah dijelaskan pada Bab 1, banyak penelitian sebelumnya yang sudah menggunakan beberapa metode *machine learning* untuk prediksi stroke, namun metode tersebut tentu saja memiliki banyak kelemahan yang dapat diatasi oleh metode *deep learning*, khususnya *Deep Neural Network* (DNN). *Dataset* 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 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, tetapi diperlukan *epoch* yang besar, otomatis waktunya akan semakin lama. Jika terlalu besar, model tidak dapat konvergen. *Learning rate* yang akan digunakan dalam penelitian ini adalah 0.1 dan 0.01, karena dalam Penelitian [7], *learning rate* yang digunakan sebesar 0.1 dan mendapatkan hasil terbaik, sedangkan pada Buku [15] dijelaskan bahwa learning rate yang *terlalu kecil* dapat konvergen namun membutuhkan waktu yang lama. Jika *learning rate* terlalu besar, model tidak dapat konvergen. Oleh sebab itu, digunakan *learning rate* yang lebih kecil, yaitu 0.01.
   (b) *Hidden layer*, berfungsi untuk pola-pola yang tidak terlihat di *neural network*. Semakin banyak, maka akan semakin lambat, tetapi bisa menemukan pola yang kompleks. *Hidden layer* yang akan digunakan dalam penelitian ini adalah 5, 10, dan 20. Dalam Penelitian [4] menggunakan *hidden layer* hanya sebesar 5 dan mendapatkan akurasi yang cukup memuaskan, yaitu 86.42%, sedangkan dalam Penelitian [7] menggunakan beragam *hidden layer*, mulai dari 10 sampai 50, namun penulis akan mencoba *hidden layer* 10 dan 20, dikarenakan hasilnya sudah

sangat memuaskan, yaitu pada *hiddden layer* 10 mendapatkan ROC tertinggi 0.97% dan akurasi 99.5% sendangkan pada *hidden layer* 20, model mendapatkan ROC tertinggi 99% dan akurasi 99.3%

(c) *Epoch*, merupakan iterasi 1 siklus program selesai dijalankan. Semakin besar semakin bisa meningkatkan akurasi, namun akan semakin lama, dan dapat menyebabkan *overfitting*. *Epoch* yang akan digunakan dalam penelitian ini adalah sebesar 10, 50, dan 100, karena pada Penelitian [8] memakai *epoch* sebesar 50 dan mendapatkan akurasi 84.03%, sedangkan pada Penelitian [7] menggunakan beragam *epoch*, yaitu dari 100 hingga 1000, tetapi hasil dari *epoch* 100 sendiri sudah baik, yaitu sudah mencapai 93.9%. Disisi lain, penulis ingin membuktikan apakah model tidak mendapat akurasi yang baik jika *epoch* diperkecil. Oleh sebab itu, digunakan *epoch* 10.

(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*. *Activation function* yang akan digunakan dalam penelitian ini adalah ReLu dan Tanh pada *input* dan *hidden layer*, dan sigmoid pada *output layer*. Ketiga *activation function* ini digunakan karena dalam Buku [16], disebutkan bahwa ketiga *activation function* ini merupakan *activation function* yang populer dipakai dan dalam Penelitian [7], *activation function* yang digunakan adalah ReLu, Tanh, dan sigmoid. Semua percobaannya menghasilkan ROC curve paling kecil 90% dan *sensitivity* atau *recall* paling kecil adalah 83.8.%

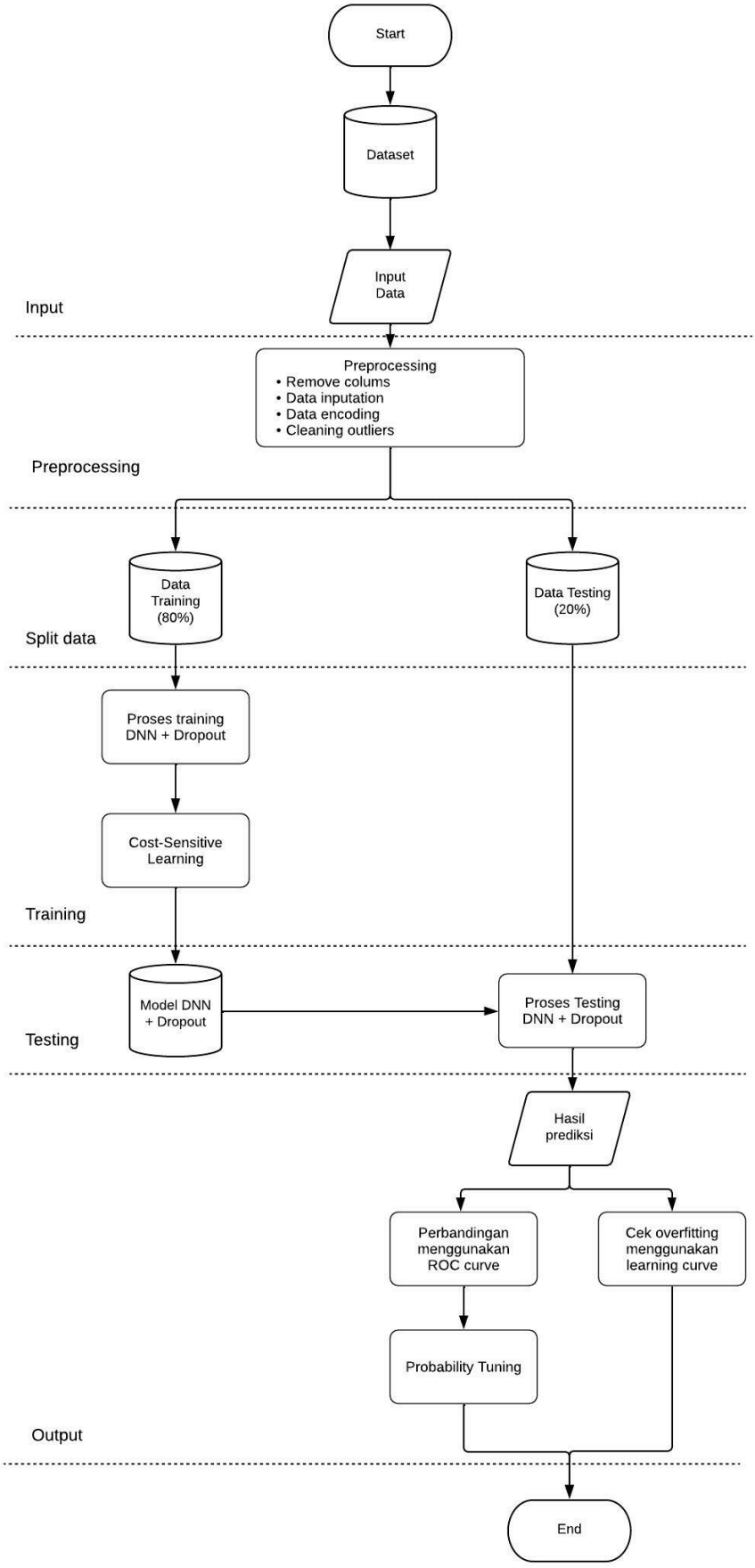
(e) *Rate*, merupakan probabilitas mempertahankan unit. Probabilitas = 1 artinya tidak ada *dropout*, dan semakin kecil nilai probabilitasnya, semakin banyak *neuron* yang di*dropout*. Semakin kecil nilai probabilitas, artinya semakin membutuhkan banyak *neuron* yang otomatis akan memperlambat *training* dan hasilnya akan cenderung *underfitting*. *Rate* yang akan digunakan dalam penelitian ini adalah 0.1 di antara *input layer* dan *hidden layer*, 0.5, dan 0.8 di antara *hidden layer* dan *output layer*.

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 imputation* 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 dan model *overfitting*.
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.

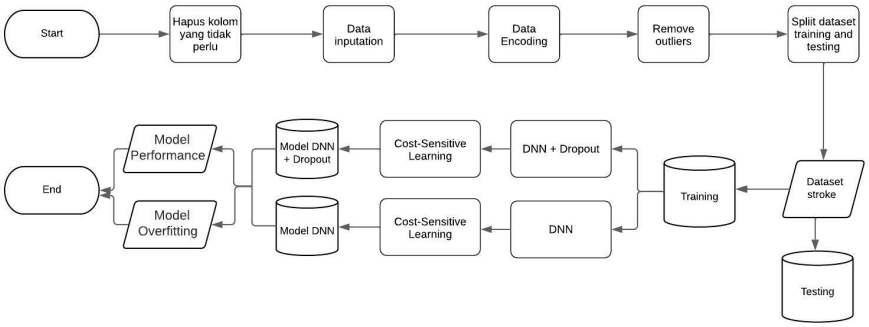

Start
Dataset
Input Data
Input
Preprocessing
• Remove colums
• Data inputation
• Data encoding
• Cleaning outliers
Preprocessing
Data Training (80%)
Data Testing (20%)
Split data
Proses training DNN + Dropout
Cost-Sensitive Learning
Training
Model DNN + Dropout
Proses Testing DNN + Dropout
Testing
Hasil 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*

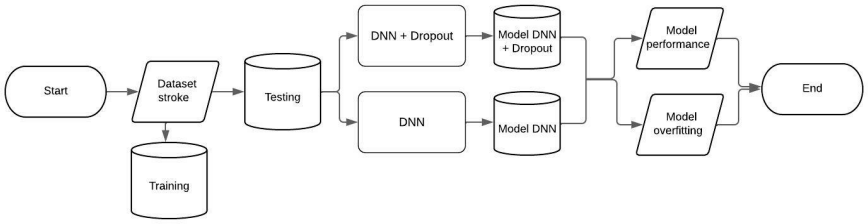
1. Dataset yang digunakan terdiri dari 43.400 data yang memiliki 12 variabel, yaitu *id*, *gender*, *age*, *hypertension*, *heart_disease*, *ever_married*, *work_type*, *residence_type*, *avg_glucose_level*, *bmi*, *smoking_status*, dan *stroke*.
2. Selanjutnya, akan dilakukan penghapusan terhadap variabel *id*.
3. *Data imputation* dilakukan menggunakan *mean imputation* 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. Membagi *dataset* sebesar 80% untuk proses *training Deep Neural Network* (DNN).
7. Akan menghasilkan 2 model, yaitu model DNN biasa yang akan dibandingkan dengan model DNN yang ditambahkan *dropout*.
8. Dikarenakan kelas stroke dan tidak terkena stroke tidak seimbang, maka akan

dilakukan teknik *cost-sensitive learning* dan *probability tuning*.

9. Terakhir, akan diukur performa dari masing-masing model menggunakan ROC *curve* dan dibandingkan menggunakan AUC. Pengukuran mengenai *overfitting* akan dilakukan menggunakan *learning curves*.

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 10 fitur, yaitu *gender*, *age*, *hypertension*, *heart_disease*, *ever_married*, *work_type*, *residence_type*, *avg_glucose_level*, *bmi*, dan *smoking_status*.
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 dan model *overfitting* diukur menggunakan *learning curves*.

3.4 Analisis Manual

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

3.4.1 Dataset

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* karena *id* hanya identitas angka pasien, yang tidak ada hubungannya dengan prediksi penyakit stroke. Gambar 3.5 merupakan kode penghapusan kolom pada Python dan Gambar 3.6 merupakan contoh *dataset* setelah dilakukan penghapusan beberapa kolom.

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

Gambar 3.5 Penghapusan kolom menggunakan kode Python

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Male	3.0	0	0	No	children	Rural	95.12	18.0	NaN	0
1	Male	58.0	1	0	Yes	Private	Urban	87.96	39.2	never smoked	0
2	Female	8.0	0	0	No	Private	Urban	110.89	17.6	NaN	0
3	Female	70.0	0	0	Yes	Private	Rural	69.04	35.9	formerly smoked	0
4	Male	14.0	0	0	No	Never_worked	Rural	161.28	19.1	NaN	0
...	...	...	...	...	...	...	...	...	...	...	...
43395	Female	10.0	0	0	No	children	Urban	58.64	20.4	never smoked	0
43396	Female	56.0	0	0	Yes	Govt_job	Urban	213.61	55.4	formerly smoked	0
43397	Female	82.0	1	0	Yes	Private	Urban	91.94	28.9	formerly smoked	0
43398	Male	40.0	0	0	Yes	Private	Urban	99.16	33.2	never smoked	0
43399	Female	82.0	0	0	Yes	Private	Urban	79.48	20.6	never smoked	0

43400 rows × 11 columns

Gambar 3.6 Penghapusan kolom *id*

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
ever_married           0
work_type              0
Residence_type         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
ever_married       0
work_type          0
Residence_type     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	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	Male	3.0	0	0	No	children	Rural	95.12	18.0	unknown	0
1	Male	58.0	1	0	Yes	Private	Urban	87.96	39.2	never smoked	0
2	Female	8.0	0	0	No	Private	Urban	110.89	17.6	unknown	0
3	Female	70.0	0	0	Yes	Private	Rural	69.04	35.9	formerly smoked	0
4	Male	14.0	0	0	No	Never_worked	Rural	161.28	19.1	unknown	0
...	...	...	...	...	...	...	...	...	...	...	...
43395	Female	10.0	0	0	No	children	Urban	58.64	20.4	never smoked	0
43396	Female	56.0	0	0	Yes	Govt_job	Urban	213.61	55.4	formerly smoked	0
43397	Female	82.0	1	0	Yes	Private	Urban	91.94	28.9	formerly smoked	0
43398	Male	40.0	0	0	Yes	Private	Urban	99.16	33.2	never smoked	0
43399	Female	82.0	0	0	Yes	Private	Urban	79.48	20.6	never smoked	0

43400 rows × 11 columns

Gambar 3.8 Data *imputation*

3.4.2.3 Data *Encoding*

Pada tahap ini, fitur kategorikal, seperti *gender*, *ever_married*, *work_type*, *residence_type*, dan *smoking_status* akan dilakukan *encoding*, yaitu dengan mengganti isinya menjadi angka, dengan rincian sebagai berikut:

1. Fitur *gender* yang berisi *male*, *female*, dan *other* diganti menjadi 0, 1, dan 2.
2. Fitur *ever_married* yang berisi *no* dan *yes* diganti menjadi 0 dan 1.
3. Fitur *work_type* yang berisi *never_worked*, *children*, *govt_job*, *private*, dan *self-employed* diganti menjadi 0, 1, 2, 3, dan 4.
4. Fitur *residence_type* yang berisi *rural* dan *urban* diganti menjadi 0 dan 1.
5. Fitur *smoking_status* yang berisi *unknown*, *never smoked*, *formerly smoked*,

smokes diganti menjadi 0, 1, 2, dan 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['ever_married'].replace(['No', 'Yes'],[0, 1], inplace=True)
df['work_type'].replace(['Never_worked', 'children', 'Govt_job', 'Private', 'self-employed'],[0, 1, 2, 3, 4], inplace=True)
df['Residence_type'].replace(['Rural', 'Urban'],[0, 1], 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	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
0	0	3.0	0	0	0	1	0	95.12	18.0	0	0
1	0	58.0	1	0	1	3	1	87.96	39.2	1	0
2	1	8.0	0	0	0	3	1	110.89	17.6	0	0
3	1	70.0	0	0	1	3	0	69.04	35.9	2	0
4	0	14.0	0	0	0	0	0	161.28	19.1	0	0
...	...	...	...	...	...	...	...	...	...	...	...
43395	1	10.0	0	0	0	1	1	58.64	20.4	1	0
43396	1	56.0	0	0	1	2	1	213.61	55.4	2	0
43397	1	82.0	1	0	1	3	1	91.94	28.9	2	0
43398	0	40.0	0	0	1	3	1	99.16	33.2	1	0
43399	1	82.0	0	0	1	3	1	79.48	20.6	1	0

43400 rows × 11 columns

Gambar 3.10 *Encoding* pada fitur *gender*, *ever_married*, *work_type*, *residence_type*, dan *smoking_status*

3.4.2.4 Hapus Data Duplikat

Dalam tahap ini, dilakukan pengecekan terhadap data duplikat. Setelah dilakukan pengecekan, *dataset* ini tidak terdapat data duplikat. Gambar 3.11 merupakan kode pengecekan data duplikat pada Python dan Gambar 3.12 merupakan hasil pengecekan data duplikat.

```
df[df.duplicated()]
```

Gambar 3.11 Cek data duplikat menggunakan kode Python

gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke

Gambar 3.12 Hasil dari pengecekan data duplikat

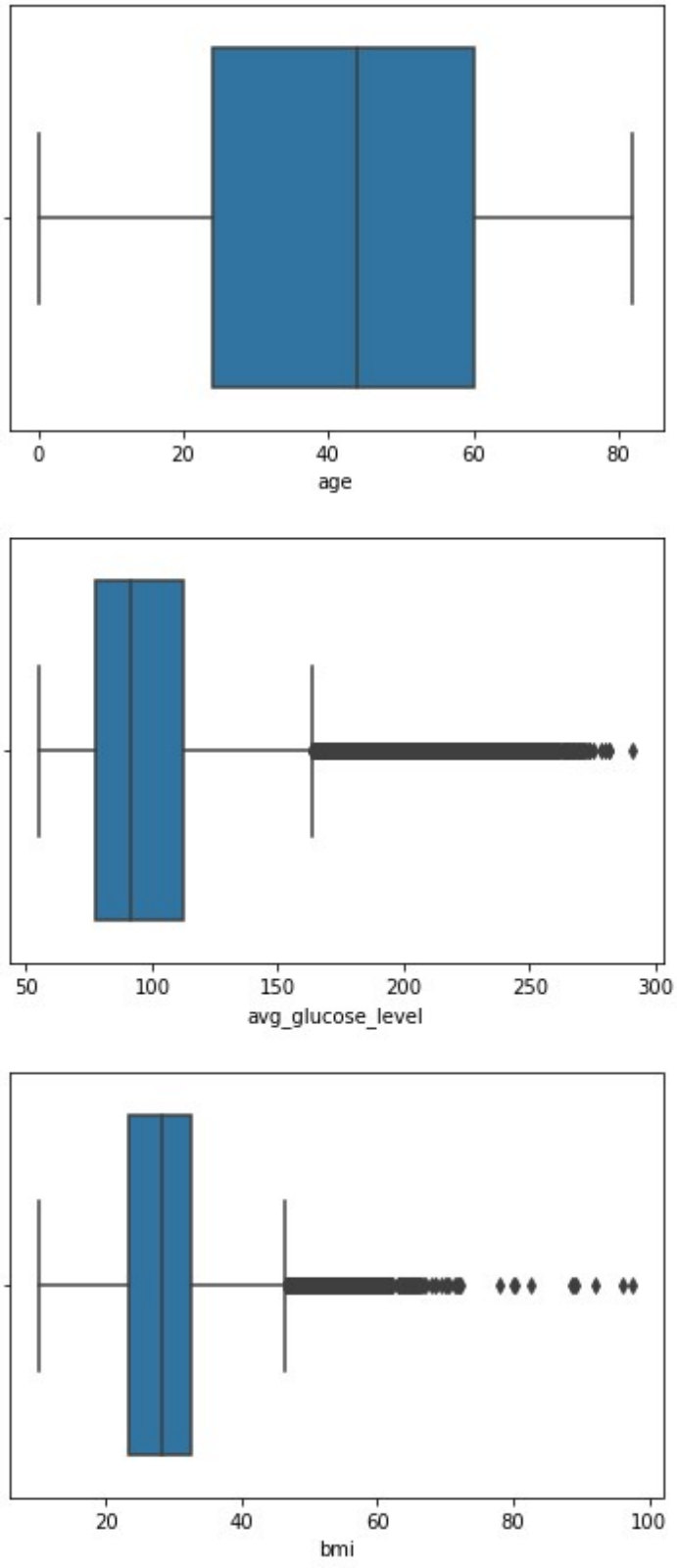
3.4.2.5 *Cleaning Outliers*

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 [27]. *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* [28] sehingga BMI lebih dari 54 sudah tidak bisa dipertanggungjawabkan

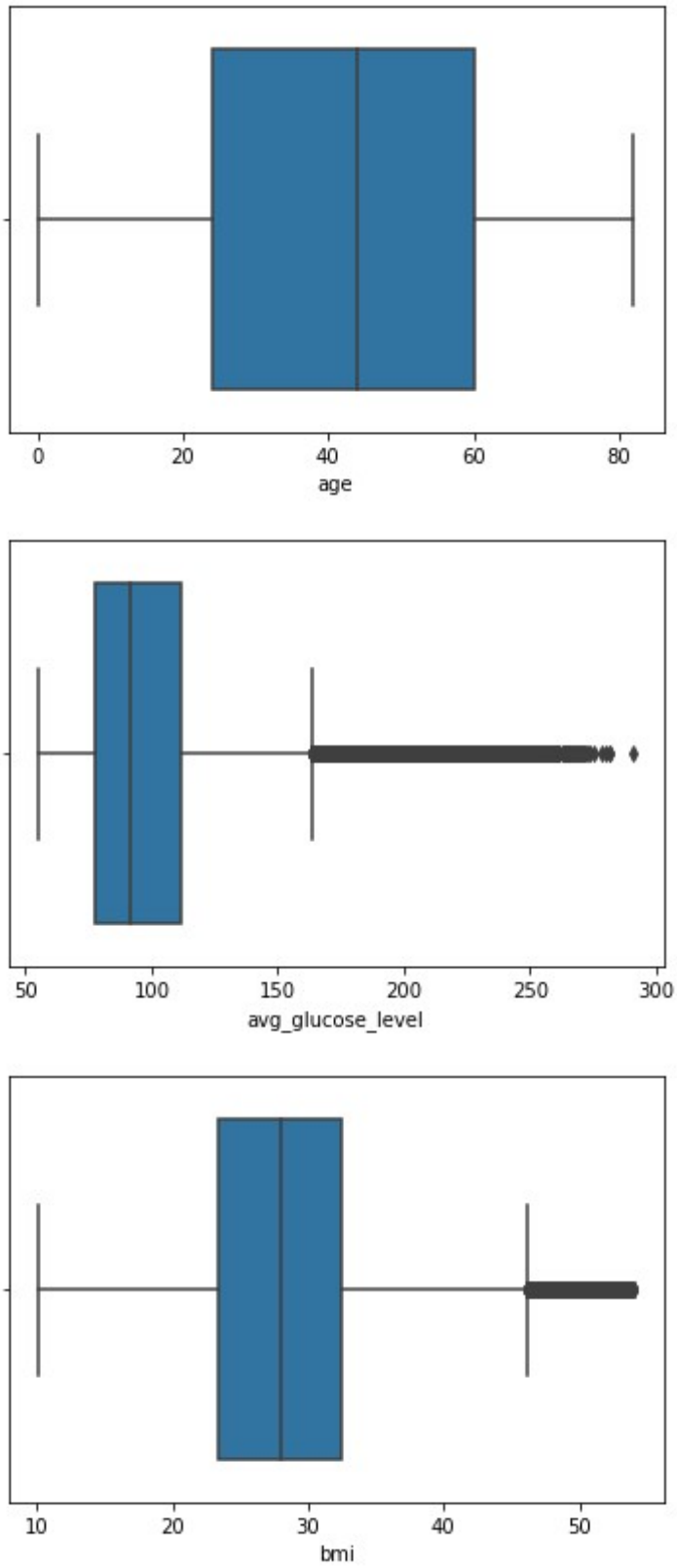
kebenarannya. Gambar 3.13 merupakan kode *cleaning outliers* pada Python, sedangkan Gambar 3.14 merupakan *boxplot* sebelum dilakukan data *cleaning* dan Gambar 3.15 merupakan *boxplot* sesudah dilakukan data *cleaning*.

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

Gambar 3.13 *Cleaning outliers* menggunakan kode Python



Gambar 3.14 *Boxplot* sebelum dilakukan data *cleaning*



Gambar 3.15 *Boxplot* sesudah dilakukan data *cleaning*

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

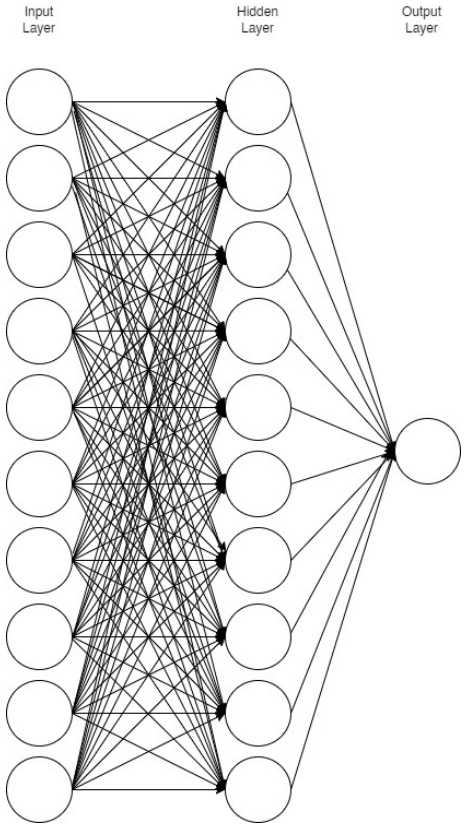
Pada tahap ini dilakukan pembagian *dataset* untuk proses *training* dan *testing* yang akan digunakan untuk membuat model *Deep Neural Network*. Pembagian *dataset* adalah 80% untuk data *training* dan 20% untuk data *testing*.

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 10 *neuron* yang merepresentasikan 10 fitur dan menggunakan *activation function* tanh.
2. *S*atu buah hidden layer yang terdiri dari 10 *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 3.16.



Gambar 3.16 Arsitektur *Deep Neural Network*

Nilai *weight* didapatkan dengan cara diinisialisasi melalui kode Python dan menghasilkan 120 data secara *random* dengan rentang tertentu.

Inisialisasi Data:
Jumlah *input layer* = 1
Jumlah *neuron* pada *input layer* = 10
Activation function pada *input layer* = tanh

Jumlah *hidden layer* = 1
Jumlah *neuron* pada *hidden layer* = 10
Activation function pada *hidden layer* = tanh

Jumlah *output layer* = 1
Jumlah *neuron* pada *output layer* = 1
Activation function pada *output layer* = *sigmoid*

Rentang *weight* = -0.31622776601683794 sampai dengan 0.31622776601683794

Bias = 0

Input (weight setiap fitur yang ada):

$x1 = -0.02835948$ $x2 = -0.08527658$ $x3 = 0.26398902$ $x4 = -0.02807259$ $x5 = 0.04415979$ $x6 = -0.15166705$ $x7 = -0.18228751$ $x8 = 0.3134241$ $x9 = -0.00786456$ $x10 = 0.17467744$

Hidden layer:

$w_{k1}j_1 = -0.08154228$ $w_{k2}j_1 = 0.30040546$ $w_{k3}j_1 = 0.24854657$ $w_{k4}j_1 = 0.01691013$ $w_{k5}j_1 = -0.18283772$ $w_{k6}j_1 = 0.15720913$ $w_{k7}j_1 = 0.08201578$ $w_{k8}j_1 = -0.19598779$ $w_{k9}j_1 = 0.26268099$ $w_{k10}j_1 = 0.13676267$

$w_{k1}j_2 = 0.21293817$ $w_{k2}j_2 = -0.25220019$ $w_{k3}j_2 = 0.29390853$ $w_{k4}j_2 = -0.18997839$ $w_{k5}j_2 = -0.14565171$ $w_{k6}j_2 = 0.19445793$ $w_{k7}j_2 = -0.17048571$ $w_{k8}j_2 = -0.13914967$ $w_{k9}j_2 = -0.11090133$ $w_{k10}j_2 = 0.09844929$

$w_{k1}j_3 = -0.22548086$ $w_{k2}j_3 = 0.14510369$ $w_{k3}j_3 = 0.22841273$ $w_{k4}j_3 = -0.22168199$ $w_{k5}j_3 = 0.05429651$ $w_{k6}j_3 = -0.22866022$ $w_{k7}j_3 = -0.24450572$ $w_{k8}j_3 = -0.24502611$ $w_{k9}j_3 = -0.30133026$ $w_{k10}j_3 = -0.306759$

$w_{k1}j_4 = 0.03463625$ $w_{k2}j_4 = -0.00885326$ $w_{k3}j_4 = -0.22394442$ $w_{k4}j_4 = -0.06561425$ $w_{k5}j_4 = 0.07484731$ $w_{k6}j_4 = 0.01486269$ $w_{k7}j_4 = -0.10394584$ $w_{k8}j_4 = 0.17709923$ $w_{k9}j_4 = 0.07621706$ $w_{k10}j_4 = -0.09844928$

$w_{k1}j_5 = -0.12021486$ $w_{k2}j_5 = -0.07829361$ $w_{k3}j_5 = -0.12997544$ $w_{k4}j_5 = 0.21279237$ $w_{k5}j_5 = 0.20977342$ $w_{k6}j_5 = -0.22495591$ $w_{k7}j_5 = -0.00190682$ $w_{k8}j_5 = -0.22943178$ $w_{k9}j_5 = 0.27366045$ $w_{k10}j_5 = -0.20595517$

$w_{k1}j_6 = -0.00716274$ $w_{k2}j_6 = -0.05448304$ $w_{k3}j_6 = 0.06476339$ $w_{k4}j_6 = 0.03532464$ $w_{k5}j_6 = 0.09245855$ $w_{k6}j_6 = 0.1695172$ $w_{k7}j_6 = 0.23680192$ $w_{k8}j_6 = 0.04477918$ $w_{k9}j_6 = 0.24950235$ $w_{k10}j_6 = -0.30254995$

$w_{k1}j_7 = -0.14745197$ $w_{k2}j_7 = -0.08891625$ $w_{k3}j_7 = 0.22152022$ $w_{k4}j_7 = -0.0214877$ $w_{k5}j_7 = -0.12194666$ $w_{k6}j_7 = -0.05346982$ $w_{k7}j_7 = 0.26132289$ $w_{k8}j_7 = 0.26464311$ $w_{k9}j_7 = 0.29086091$ $w_{k10}j_7 = 0.07891517$

$w_{k1}j_8 = 0.30138627$ $w_{k2}j_8 = -0.17017802$ $w_{k3}j_8 = 0.0968307$ $w_{k4}j_8 = 0.26289768$ $w_{k5}j_8 = 0.01635267$ $w_{k6}j_8 = -0.06321122$ $w_{k7}j_8 = 0.1968099$ $w_{k8}j_8 = -0.14343715$ $w_{k9}j_8 = -0.01907697$ $w_{k10}j_8 = -0.24814027$

$w_{k1}j_9 = 0.30255215$ $w_{k2}j_9 = -0.08199351$ $w_{k3}j_9 = 0.17765137$ $w_{k4}j_9 = 0.07950517$ $w_{k5}j_9 = 0.15693231$ $w_{k6}j_9 = -0.06841941$ $w_{k7}j_9 = -0.01554272$ $w_{k8}j_9 = 0.21543019$ $w_{k9}j_9 = -0.19608282$ $w_{k10}j_9 = 0.27772592$

$w_{k1}j_{10} = -0.27114336$ $w_{k2}j_{10} = -0.12175201$ $w_{k3}j_{10} = 0.03013474$ $w_{k4}j_{10} = -0.10994942$ $w_{k5}j_{10} = 0.10416485$ $w_{k6}j_{10} = 0.21983722$ $w_{k7}j_{10} = 0.296758$ $w_{k8}j_{10} = 0.10953304$ $w_{k9}j_{10} = -0.06145034$ $w_{k10}j_{10} = 0.27154544$

Output layer:

$w_{k1}j_{11} = -0.14147132$ $w_{k2}j_{11} = 0.05554158$ $w_{k3}j_{11} = -0.27054289$ $w_{k4}j_{11} = -0.03277796$ $w_{k5}j_{11} = -0.10409342$ $w_{k6}j_{11} = -0.24334135$ $w_{k7}j_{11} = -0.06665223$ $w_{k8}j_{11} = 0.01262219$ $w_{k9}j_{11} = 0.18286435$ $w_{k10}j_{11} = 0.16453867$

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

$h1 = ((w_{k1}j_1 * x1) + (w_{k2}j_1 * x2) + (w_{k3}j_1 * x3) + (w_{k4}j_1 * x4) + (w_{k5}j_1 * x5) + (w_{k6}j_1 * x6) + (w_{k7}j_1 * x7) + (w_{k8}j_1 * x8) + (w_{k9}j_1 * x9) + (w_{k10}j_1 * x10)) + b$

$h1$ = ((-0.08154228 * -0.02835948) + (-0.02835948 * -0.08527658) + (0.24854657 * 0.26398902) + (0.01691013 * -0.02807259) + (-0.18283772 * 0.04415979) + (0.15720913 * -0.15166705) + (0.08201578 * -0.18228751) + (-0.19598779 * 0.3134241) + (0.26268099 * -0.00786456) + (0.13676267 * 0.17467744)) + 0

$h1$ = -0.016602036215524193

$h2 = ((w_{k1}j_2 * x1) + (w_{k2}j_2 * x2) + (w_{k3}j_2 * x3) + (w_{k4}j_2 * x4) + (w_{k5}j_2 * x5) + (w_{k6}j_2 * x6) + (w_{k7}j_2 * x7) + (w_{k8}j_2 * x8) + (w_{k9}j_2 * x9) + (w_{k10}j_2 * x10)) + b$

$h2$ = ((0.21293817 * -0.02835948) + (-0.25220019 * -0.08527658) + (0.29390853 * 0.26398902) + (-0.18997839 * -0.02807259) + (-0.14565171 * 0.04415979) + (0.19445793 * -0.15166705) + (-0.17048571 * -0.18228751) + (-0.13914967 * 0.3134241) + (-0.11090133 * -0.00786456) + (0.09844929 * 0.17467744)) + 0

$h2$ = 0.06799857023423941

$h3 = ((w_{k1}j_3 * x1) + (w_{k2}j_3 * x2) + (w_{k3}j_3 * x3) + (w_{k4}j_3 * x4) + (w_{k5}j_3 * x5) + (w_{k6}j_3 * x6) + (w_{k7}j_3 * x7) + (w_{k8}j_3 * x8) + (w_{k9}j_3 * x9) + (w_{k10}j_3 * x10)) + b$

$h3$ = ((-0.22548086 * -0.02835948) + (0.14510369 * -0.08527658) + (0.22841273 * 0.26398902) + (-0.22168199 * -0.02807259) + (0.05429651 * 0.04415979) + (-0.22866022 * -0.15166705) + (-0.24450572 * -0.18228751) + (0.24502611 * 0.3134241) + (-0.30133026 * -0.00786456) + (-0.306759 * 0.17467744)) + 0

$h3$ = 0.16777353734936898

$h4 = ((w_{k1}j_4 * x1) + (w_{k2}j_4 * x2) + (w_{k3}j_4 * x3) + (w_{k4}j_4 * x4) + (w_{k5}j_4 * x5) + (w_{k6}j_4 * x6) + (w_{k7}j_4 * x7) + (w_{k8}j_4 * x8) + (w_{k9}j_4 * x9) + (w_{k10}j_4 * x10)) + b$

$h4$ = ((0.03463625 * -0.02835948) + (-0.00885326 * -0.08527658) + (-0.22394442 * 0.26398902) + (-0.06561425 * -0.02807259) + (0.07484731 * 0.04415979) + (0.01486269 * -0.15166705) + (-0.10394584 * -0.18228751) + (0.17709923 * 0.3134241) + (0.07621706 * -0.00786456) + (0.09844928 * 0.17467744)) + 0

$h4$ = 0.03459951448869128

$h5 = ((w_{k1}j_5 * x1) + (w_{k2}j_5 * x2) + (w_{k3}j_5 * x3) + (w_{k4}j_5 * x4) + (w_{k5}j_5 * x5) + (w_{k6}j_5 * x6) + (w_{k7}j_5 * x7) + (w_{k8}j_5 * x8) + (w_{k9}j_5 * x9) + (w_{k10}j_5 * x10)) + b$

$h5$ = ((-0.12021486 * -0.02835948) + (-0.07829361 * -0.08527658) + (-0.12997544 * 0.26398902) + (0.21279237 * -0.02807259) + (0.20977342 * 0.04415979) + (-0.22495591 * -0.15166705) + (-0.00190682 * -0.18228751) + (-0.22943178 * 0.3134241) + (0.27366045 * -0.00786456) + (-0.20595517 * 0.17467744)) + 0

$h5$ = -0.09650773091582979

$h6 = ((w_{k1}j_6 * x1) + (w_{k2}j_6 * x2) + (w_{k3}j_6 * x3) + (w_{k4}j_6 * x4) + (w_{k5}j_6 * x5) + (w_{k6}j_6 * x6) + (w_{k7}j_6 * x7) + (w_{k8}j_6 * x8) + (w_{k9}j_6 * x9) + (w_{k10}j_6 * x10)) + b$

$h6 = ((-0.00716274 * -0.02835948) + (-0.05448304 * -0.08527658) + (0.06476339 * 0.26398902) + (0.03532464 * -0.02807259) + (0.09245855 * 0.04415979) + (0.1695172 * -0.15166705) + (0.23680192 * -0.18228751) + (0.04477918 * 0.3134241) + (0.24950235 * -0.00786456) + (-0.30254995 * 0.17467744)) + 0$

$h6 = -0.08461482998284209$

$h7 = ((w_{k1}j_7 * x1) + (w_{k2}j_7 * x2) + (w_{k3}j_7 * x3) + (w_{k4}j_7 * x4) + (w_{k5}j_7 * x5) + (w_{k6}j_7 * x6) + (w_{k7}j_7 * x7) + (w_{k8}j_7 * x8) + (w_{k9}j_7 * x9) + (w_{k10}j_7 * x10)) + b$

$h7 = ((-0.14745197 * -0.02835948) + (-0.08891625 * -0.08527658) + (0.22152022 * 0.26398902) + (0.0214877 * -0.02807259) + (-0.12194666 * 0.04415979) + (-0.05346982 * -0.15166705) + (0.26132289 * -0.18228751) + (0.26464311 * 0.3134241) + (0.29086091 * -0.00786456) + (0.07891517 * 0.17467744)) + 0$

$h7 = 0.1191711327063339$

$h8 = ((w_{k1}j_8 * x1) + (w_{k2}j_8 * x2) + (w_{k3}j_8 * x3) + (w_{k4}j_8 * x4) + (w_{k5}j_8 * x5) + (w_{k6}j_8 * x6) + (w_{k7}j_8 * x7) + (w_{k8}j_8 * x8) + (w_{k9}j_8 * x9) + (w_{k10}j_8 * x10)) + b$

$h8 = ((0.30138627 * -0.02835948) + (-0.17017802 * -0.08527658) + (0.0968307 * 0.26398902) + (0.26289768 * -0.02807259) + (0.01635267 * 0.04415979) + (-0.06321122 * -0.15166705) + (0.1968099 * -0.18228751) + (-0.14343715 * 0.3134241) + (-0.01907697 * -0.00786456) + (-0.24814027 * 0.17467744)) + 0$

$h8 = -0.0895708672147945$

$h9 = ((w_{k1}j_9 * x1) + (w_{k2}j_9 * x2) + (w_{k3}j_9 * x3) + (w_{k4}j_9 * x4) + (w_{k5}j_9 * x5) + (w_{k6}j_9 * x6) + (w_{k7}j_9 * x7) + (w_{k8}j_9 * x8) + (w_{k9}j_9 * x9) + (w_{k10}j_9 * x10)) + b$

$h9 = ((0.30255215 * -0.02835948) + (-0.08199351 * -0.08527658) + (0.17765137 * 0.26398902) + (0.07950517 * -0.02807259) + (0.15693231 * 0.04415979) + (-0.06841941 * -0.15166705) + (-0.01554272 * -0.18228751) + (-0.21543019 * 0.3134241) + (-0.19608282 * -0.00786456) + (0.27772592 * 0.17467744)) + 0$

$h9 = 0.0457518555709885$

$h10 = ((w_{k1}j_{10} * x1) + (w_{k2}j_{10} * x2) + (w_{k3}j_{10} * x3) + (w_{k4}j_{10} * x4) + (w_{k5}j_{10} * x5) + (w_{k6}j_{10} * x6) + (w_{k7}j_{10} * x7) + (w_{k8}j_{10} * x8) + (w_{k9}j_{10} * x9) + (w_{k10}j_{10} * x10)) + b$

$h10 = ((-0.27114336 * -0.02835948) + (-0.12175201 * -0.08527658) + (0.03013474 * 0.26398902) + (-0.10994942 * -0.02807259) + (0.10416485 * 0.04415979) + (0.21983722 * -0.15166705) + (0.296758 * -0.18228751) + (0.10953304 * 0.3134241) + (-0.06145034 * -0.00786456) + (0.27154544 * 0.17467744)) + 0$

$h10 = 0.028522880227219695$

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

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

$tanh\ (h1) = \frac{e^{-0.016602036215524193} - e^{--0.016602036215524193}}{e^{-0.016602036215524193} + e^{--0.016602036215524193}}$

$tanh\ (h1) = -0.0166005$

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

$tanh\ (h2) = \frac{e^{0.06799857023423941} - e^{-0.06799857023423941}}{e^{0.06799857023423941} + e^{-0.06799857023423941}}$

$tanh\ (h2) = 0.067894$

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

$tanh\ (h3) = \frac{e^{0.16777353734936898} - e^{-0.16777353734936898}}{e^{0.16777353734936898} + e^{-0.16777353734936898}}$

$tanh\ (h3) = 0.166217$

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

$tanh\ (h4) = \frac{e^{0.03459951448869128} - e^{-0.03459951448869128}}{e^{0.03459951448869128} + e^{-0.03459951448869128}}$

$tanh\ (h4) = 0.0345857$

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

$tanh\ (h5) = \frac{e^{-0.09650773091582979} - e^{--0.09650773091582979}}{e^{-0.09650773091582979} + e^{--0.09650773091582979}}$

$tanh\ (h5) = -0.0962092$

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

$tanh\ (h6) = \frac{e^{-0.08461482998284209} - e^{--0.08461482998284209}}{e^{-0.08461482998284209} + e^{--0.08461482998284209}}$

$tanh\ (h6) = -0.0844135$

$tanh\ (h7) = \frac{e^{h7} - e^{-h7}}{e^{h7} + e^{-h7}}$

$tanh\ (h7) = \frac{e^{0.1191711327063339} - e^{-0.1191711327063339}}{e^{0.1191711327063339} + e^{-0.1191711327063339}}$

$tanh\ (h7) = 0.11861$

$tanh\ (h8) = \frac{e^{h8} - e^{-h8}}{e^{h8} + e^{-h8}}$

$tanh\ (h8) = \frac{e^{-0.0895708672147945} - e^{--0.0895708672147945}}{e^{-0.0895708672147945} + e^{--0.0895708672147945}}$

$tanh\ (h8) = -0.0893321$

$tanh\ (h9) = \frac{e^{h9} - e^{-h9}}{e^{h9} + e^{-h9}}$

$tanh\ (h9) = \frac{e^{0.0457518555709885} - e^{-0.0457518555709885}}{e^{0.0457518555709885} + e^{-0.0457518555709885}}$

$tanh\ (h9) = 0.04572$

$tanh\ (h10) = \frac{e^{h10} - e^{-h10}}{e^{h10} + e^{-h10}}$

$tanh\ (h10) = \frac{e^{0.028522880227219695} - e^{-0.028522880227219695}}{e^{0.028522880227219695} + e^{-0.028522880227219695}}$

$tanh\ (h10) = 0.0285151$

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

$output$ = (($w_{k1} j_7$ * tanh($h1$)) + ($w_{k2} j_7$ * tanh($h2$)) + ($w_{k3} j_7$ * tanh($h3$)) + ($w_{k4} j_7$ * tanh($h4$)) + ($w_{k5} j_7$ * tanh($h5$)) + ($w_{k6} j_7$ * tanh($h6$)) + ($w_{k7} j_7$ * tanh($h7$)) + ($w_{k8} j_7$ * tanh($h8$)) + ($w_{k9} j_7$ * tanh($h9$)) + ($w_{k10} j_7$ * tanh($h10$))) + b

$output$ = ((-0.14147132 * -0.0166005) + (0.05554158 * 0.067894) + (-0.27054289 * 0.166217) + (-0.03277796 * 0.0345857) + (-0.10409342 * -0.0962092) + (-0.24334135 * -0.0844135) + (-0.06665223 * 0.11861) + (0.01262219 * -0.0893321) + (0.18286435 * 0.04572) + (0.16453867 * 0.0285151) + 0)

$output$ = -0.005407794875114999

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):

$\sigma(z) = = 1/(1 + exp(-z))$

$\sigma(-0.005407794875114999)$ = 1 / (1 + exp(- -0.005407794875114999))

$\sigma(-0.005407794875114999)$ = 1 / (1 + 1.005422443391307)

$\sigma(-0.005407794875114999)$ = 1 / 2.005422443391307

$\sigma(-0.005407794875114999)$ = 0.4986480546

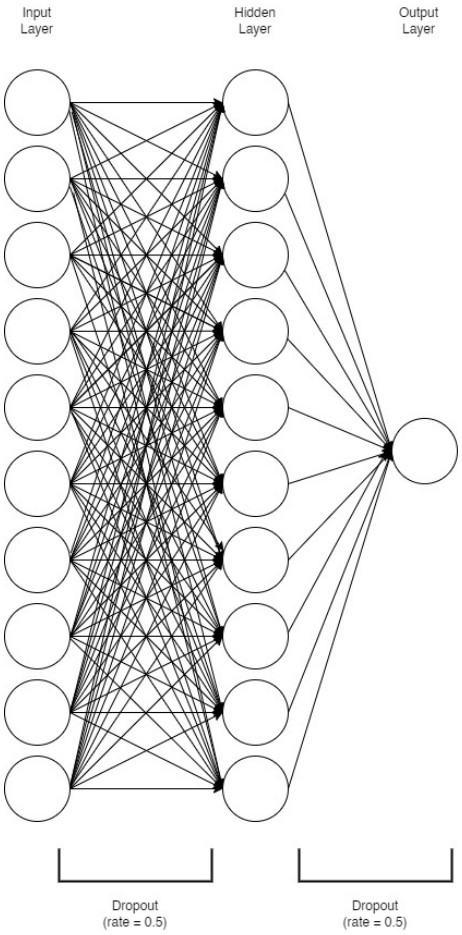
Pada perhitungan di atas, hasil *output neuron* adalah 0.4986480546 yang akan diklasifikasikan ke kelas 0 atau tidak terkena penyakit stroke. Jika hasil *output neuron* lebih kecil dari 0.5 , maka data tersebut masuk ke dalam kelas 0, yaitu tidak terkena penyakit stroke, namun jika hasil *output* menunjukkan 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 10 *neuron* yang merepresentasikan 10 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 10 *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 3.17.



Gambar 3.17 Arsitektur *Deep Neural Network* dengan *dropout*

Nilai *weight* didapatkan dengan cara diinisialisasi melalui kode Python dan menghasilkan 120 data secara *random* dengan rentang tertentu.

Inisialisasi Data:
Jumlah *input layer* = 1
Jumlah *neuron* pada *input layer* = 10
Activation function pada *input layer* = tanh

Dropout rate = 0.5

Jumlah *hidden layer* = 1
Jumlah *neuron* pada *hidden layer* = 10
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* = -0.31622776601683794 sampai dengan 0.31622776601683794

Bias = 0

Input (weight setiap fitur yang ada):

$x1 = -0.02835948$ $x2 = -0.08527658$ $x3 = 0.26398902$ $x4 = -0.02807259$ $x5 = 0.04415979$ $x6 = -0.15166705$ $x7 = -0.18228751$ $x8 = 0.3134241$ $x9 = -0.00786456$ $x10 = 0.17467744$

Hidden layer:

$w_{k1}j_1 = -0.08154228$ $w_{k2}j_1 = 0.30040546$ $w_{k3}j_1 = 0.24854657$ $w_{k4}j_1 = 0.01691013$ $w_{k5}j_1 = -0.18283772$ $w_{k6}j_1 = 0.15720913$ $w_{k7}j_1 = 0.08201578$ $w_{k8}j_1 = -0.19598779$ $w_{k9}j_1 = 0.26268099$ $w_{k10}j_1 = 0.13676267$

$w_{k1}j_2 = 0.21293817$ $w_{k2}j_2 = -0.25220019$ $w_{k3}j_2 = 0.29390853$ $w_{k4}j_2 = -0.18997839$ $w_{k5}j_2 = -0.14565171$ $w_{k6}j_2 = 0.19445793$ $w_{k7}j_2 = -0.17048571$ $w_{k8}j_2 = -0.13914967$ $w_{k9}j_2 = -0.11090133$ $w_{k10}j_2 = 0.09844929$

$w_{k1}j_3 = -0.22548086$ $w_{k2}j_3 = 0.14510369$ $w_{k3}j_3 = 0.22841273$ $w_{k4}j_3 = -0.22168199$ $w_{k5}j_3 = 0.05429651$ $w_{k6}j_3 = -0.22866022$ $w_{k7}j_3 = -0.24450572$ $w_{k8}j_3 = -0.24502611$ $w_{k9}j_3 = -0.30133026$ $w_{k10}j_3 = -0.306759$

$w_{k1}j_4 = 0.03463625$ $w_{k2}j_4 = -0.00885326$ $w_{k3}j_4 = -0.22394442$ $w_{k4}j_4 = -0.06561425$ $w_{k5}j_4 = 0.07484731$ $w_{k6}j_4 = 0.01486269$ $w_{k7}j_4 = -0.10394584$ $w_{k8}j_4 = 0.17709923$ $w_{k9}j_4 = 0.07621706$ $w_{k10}j_4 = -0.09844928$

$w_{k1}j_5 = -0.12021486$ $w_{k2}j_5 = -0.07829361$ $w_{k3}j_5 = -0.12997544$ $w_{k4}j_5 = 0.21279237$ $w_{k5}j_5 = 0.20977342$ $w_{k6}j_5 = -0.22495591$ $w_{k7}j_5 = -0.00190682$ $w_{k8}j_5 = -0.22943178$ $w_{k9}j_5 = 0.27366045$ $w_{k10}j_5 = -0.20595517$

$w_{k1}j_6 = -0.00716274$ $w_{k2}j_6 = -0.05448304$ $w_{k3}j_6 = 0.06476339$ $w_{k4}j_6 = 0.03532464$ $w_{k5}j_6 = 0.09245855$ $w_{k6}j_6 = 0.1695172$ $w_{k7}j_6 = 0.23680192$ $w_{k8}j_6 = 0.04477918$ $w_{k9}j_6 = 0.24950235$ $w_{k10}j_6 = -0.30254995$

$w_{k1}j_7 = -0.14745197$ $w_{k2}j_7 = -0.08891625$ $w_{k3}j_7 = 0.22152022$ $w_{k4}j_7 = -0.0214877$ $w_{k5}j_7 = -0.12194666$ $w_{k6}j_7 = -0.05346982$ $w_{k7}j_7 = 0.26132289$ $w_{k8}j_7 = 0.26464311$ $w_{k9}j_7 = 0.29086091$ $w_{k10}j_7 = 0.07891517$

$w_{k1}j_8 = 0.30138627$ $w_{k2}j_8 = -0.17017802$ $w_{k3}j_8 = 0.0968307$ $w_{k4}j_8 = 0.26289768$ $w_{k5}j_8 = 0.01635267$ $w_{k6}j_8 = -0.06321122$ $w_{k7}j_8 = 0.1968099$ $w_{k8}j_8 = -0.14343715$ $w_{k9}j_8 = -0.01907697$ $w_{k10}j_8 = -0.24814027$

$w_{k1}j_9 = 0.30255215$ $w_{k2}j_9 = -0.08199351$ $w_{k3}j_9 = 0.17765137$ $w_{k4}j_9 = 0.07950517$ $w_{k5}j_9 = 0.15693231$ $w_{k6}j_9 = -0.06841941$ $w_{k7}j_9 = -0.01554272$ $w_{k8}j_9 = 0.21543019$ $w_{k9}j_9 = -0.19608282$ $w_{k10}j_9 = 0.27772592$

$w_{k1}j_{10} = -0.27114336$ $w_{k2}j_{10} = -0.12175201$ $w_{k3}j_{10} = 0.03013474$ $w_{k4}j_{10} = -0.10994942$ $w_{k5}j_{10} = 0.10416485$ $w_{k6}j_{10} = 0.21983722$ $w_{k7}j_{10} = 0.296758$ $w_{k8}j_{10} = 0.10953304$ $w_{k9}j_{10} = -0.06145034$ $w_{k10}j_{10} = 0.27154544$

Output layer:

$w_{k1}j_{11} = -0.14147132$ $w_{k2}j_{11} = 0.05554158$ $w_{k3}j_{11} = -0.27054289$ $w_{k4}j_{11} = -0.03277796$ $w_{k5}j_{11} = -0.10409342$ $w_{k6}j_{11} = -0.24334135$ $w_{k7}j_{11} = -0.06665223$ $w_{k8}j_{11} = 0.01262219$ $w_{k9}j_{11} = 0.18286435$ $w_{k10}j_{11} = 0.16453867$

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

$h1 = ((w_{k1}j_1 * x1) + (w_{k2}j_1 * x2) + (w_{k3}j_1 * x3) + (w_{k4}j_1 * x4) + (w_{k5}j_1 * x5) + (w_{k6}j_1 * x6) + (w_{k7}j_1 * x7) + (w_{k8}j_1 * x8) + (w_{k9}j_1 * x9) + (w_{k10}j_1 * x10)) + b$

$h1$ = ((-0.08154228 * -0.02835948) + (-0.02835948 * -0.08527658) + (0.24854657 * 0.26398902) + (0.01691013 * -0.02807259) + (-0.18283772 * 0.04415979) + (0.15720913 * -0.15166705) + (0.08201578 * -0.18228751) + (-0.19598779 * 0.3134241) + (0.26268099 * -0.00786456) + (0.13676267 * 0.17467744)) + 0

$h1$ = -0.016602036215524193

$h2 = ((w_{k1}j_2 * x1) + (w_{k2}j_2 * x2) + (w_{k3}j_2 * x3) + (w_{k4}j_2 * x4) + (w_{k5}j_2 * x5) + (w_{k6}j_2 * x6) + (w_{k7}j_2 * x7) + (w_{k8}j_2 * x8) + (w_{k9}j_2 * x9) + (w_{k10}j_2 * x10)) + b$

$h2$ = ((0.21293817 * -0.02835948) + (-0.25220019 * -0.08527658) + (0.29390853 * 0.26398902) + (-0.18997839 * -0.02807259) + (-0.14565171 * 0.04415979) + (0.19445793 * -0.15166705) + (-0.17048571 * -0.18228751) + (-0.13914967 * 0.3134241) + (-0.11090133 * -0.00786456) + (0.09844929 * 0.17467744)) + 0

$h2$ = 0.06799857023423941

$h3 = ((w_{k1}j_3 * x1) + (w_{k2}j_3 * x2) + (w_{k3}j_3 * x3) + (w_{k4}j_3 * x4) + (w_{k5}j_3 * x5) + (w_{k6}j_3 * x6) + (w_{k7}j_3 * x7) + (w_{k8}j_3 * x8) + (w_{k9}j_3 * x9) + (w_{k10}j_3 * x10)) + b$

$h3$ = ((-0.22548086 * -0.02835948) + (0.14510369 * -0.08527658) + (0.22841273 * 0.26398902) + (-0.22168199 * -0.02807259) + (0.05429651 * 0.04415979) + (-0.22866022 * -0.15166705) + (-0.24450572 * -0.18228751) + (0.24502611 * 0.3134241) + (-0.30133026 * -0.00786456) + (-0.306759 * 0.17467744)) + 0

$h3$ = 0.16777353734936898

$h4 = ((w_{k1}j_4 * x1) + (w_{k2}j_4 * x2) + (w_{k3}j_4 * x3) + (w_{k4}j_4 * x4) + (w_{k5}j_4 * x5) + (w_{k6}j_4 * x6) + (w_{k7}j_4 * x7) + (w_{k8}j_4 * x8) + (w_{k9}j_4 * x9) + (w_{k10}j_4 * x10)) + b$

$h4$ = ((0.03463625 * -0.02835948) + (-0.00885326 * -0.08527658) + (-0.22394442 * 0.26398902) + (-0.06561425 * -0.02807259) + (0.07484731 * 0.04415979) + (0.01486269 * -0.15166705) + (-0.10394584 * -0.18228751) + (0.17709923 * 0.3134241) + (0.07621706 * -0.00786456) + (0.09844928 * 0.17467744)) + 0

$h4$ = 0.03459951448869128

$h5 = ((w_{k1}j_5 * x1) + (w_{k2}j_5 * x2) + (w_{k3}j_5 * x3) + (w_{k4}j_5 * x4) + (w_{k5}j_5 * x5) + (w_{k6}j_5 * x6) + (w_{k7}j_5 * x7) + (w_{k8}j_5 * x8) + (w_{k9}j_5 * x9) + (w_{k10}j_5 * x10)) + b$

$h5$ = ((-0.12021486 * -0.02835948) + (-0.07829361 * -0.08527658) + (-0.12997544 * 0.26398902) + (0.21279237 * -0.02807259) + (0.20977342 * 0.04415979) + (-0.22495591 * -0.15166705) + (-0.00190682 * -0.18228751) + (-0.22943178 * 0.3134241) + (0.27366045 * -0.00786456) + (-0.20595517 * 0.17467744)) + 0

$h5$ = -0.09650773091582979

$h6 = ((w_{k1}j_6 * x1) + (w_{k2}j_6 * x2) + (w_{k3}j_6 * x3) + (w_{k4}j_6 * x4) + (w_{k5}j_6 * x5) + (w_{k6}j_6 * x6) + (w_{k7}j_6 * x7) + (w_{k8}j_6 * x8) + (w_{k9}j_6 * x9) + (w_{k10}j_6 * x10)) + b$

$h6 = ((-0.00716274 * -0.02835948) + (-0.05448304 * -0.08527658) + (0.06476339 * 0.26398902) + (0.03532464 * -0.02807259) + (0.09245855 * 0.04415979) + (0.1695172 * -0.15166705) + (0.23680192 * -0.18228751) + (0.04477918 * 0.3134241) + (0.24950235 * -0.00786456) + (-0.30254995 * 0.17467744)) + 0$

$h6 = -0.08461482998284209$

$h7 = ((w_{k1}j_7 * x1) + (w_{k2}j_7 * x2) + (w_{k3}j_7 * x3) + (w_{k4}j_7 * x4) + (w_{k5}j_7 * x5) + (w_{k6}j_7 * x6) + (w_{k7}j_7 * x7) + (w_{k8}j_7 * x8) + (w_{k9}j_7 * x9) + (w_{k10}j_7 * x10)) + b$

$h7 = ((-0.14745197 * -0.02835948) + (-0.08891625 * -0.08527658) + (0.22152022 * 0.26398902) + (0.0214877 * -0.02807259) + (-0.12194666 * 0.04415979) + (-0.05346982 * -0.15166705) + (0.26132289 * -0.18228751) + (0.26464311 * 0.3134241) + (0.29086091 * -0.00786456) + (0.07891517 * 0.17467744)) + 0$

$h7 = 0.1191711327063339$

$h8 = ((w_{k1}j_8 * x1) + (w_{k2}j_8 * x2) + (w_{k3}j_8 * x3) + (w_{k4}j_8 * x4) + (w_{k5}j_8 * x5) + (w_{k6}j_8 * x6) + (w_{k7}j_8 * x7) + (w_{k8}j_8 * x8) + (w_{k9}j_8 * x9) + (w_{k10}j_8 * x10)) + b$

$h8 = ((0.30138627 * -0.02835948) + (-0.17017802 * -0.08527658) + (0.0968307 * 0.26398902) + (0.26289768 * -0.02807259) + (0.01635267 * 0.04415979) + (-0.06321122 * -0.15166705) + (0.1968099 * -0.18228751) + (-0.14343715 * 0.3134241) + (-0.01907697 * -0.00786456) + (-0.24814027 * 0.17467744)) + 0$

$h8 = -0.0895708672147945$

$h9 = ((w_{k1}j_9 * x1) + (w_{k2}j_9 * x2) + (w_{k3}j_9 * x3) + (w_{k4}j_9 * x4) + (w_{k5}j_9 * x5) + (w_{k6}j_9 * x6) + (w_{k7}j_9 * x7) + (w_{k8}j_9 * x8) + (w_{k9}j_9 * x9) + (w_{k10}j_9 * x10)) + b$

$h9 = ((0.30255215 * -0.02835948) + (-0.08199351 * -0.08527658) + (0.17765137 * 0.26398902) + (0.07950517 * -0.02807259) + (0.15693231 * 0.04415979) + (-0.06841941 * -0.15166705) + (-0.01554272 * -0.18228751) + (-0.21543019 * 0.3134241) + (-0.19608282 * -0.00786456) + (0.27772592 * 0.17467744)) + 0$

$h9 = 0.0457518555709885$

$h10 = ((w_{k1}j_{10} * x1) + (w_{k2}j_{10} * x2) + (w_{k3}j_{10} * x3) + (w_{k4}j_{10} * x4) + (w_{k5}j_{10} * x5) + (w_{k6}j_{10} * x6) + (w_{k7}j_{10} * x7) + (w_{k8}j_{10} * x8) + (w_{k9}j_{10} * x9) + (w_{k10}j_{10} * x10)) + b$

$h10 = ((-0.27114336 * -0.02835948) + (-0.12175201 * -0.08527658) + (0.03013474 * 0.26398902) + (-0.10994942 * -0.02807259) + (0.10416485 * 0.04415979) + (0.21983722 * -0.15166705) + (0.296758 * -0.18228751) + (0.10953304 * 0.3134241) + (-0.06145034 * -0.00786456) + (0.27154544 * 0.17467744)) + 0$

$h10 = 0.028522880227219695$

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

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

$tanh\ (h1) = \frac{e^{-0.016602036215524193} - e^{--0.016602036215524193}}{e^{-0.016602036215524193} + e^{--0.016602036215524193}}$

$tanh\ (h1) = -0.0166005$

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

$tanh\ (h2) = \frac{e^{0.06799857023423941} - e^{-0.06799857023423941}}{e^{0.06799857023423941} + e^{-0.06799857023423941}}$

$tanh\ (h2) = 0.067894$

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

$tanh\ (h3) = \frac{e^{0.16777353734936898} - e^{-0.16777353734936898}}{e^{0.16777353734936898} + e^{-0.16777353734936898}}$

$tanh\ (h3) = 0.166217$

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

$tanh\ (h4) = \frac{e^{0.03459951448869128} - e^{-0.03459951448869128}}{e^{0.03459951448869128} + e^{-0.03459951448869128}}$

$tanh\ (h4) = 0.0345857$

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

$tanh\ (h5) = \frac{e^{-0.09650773091582979} - e^{--0.09650773091582979}}{e^{-0.09650773091582979} + e^{--0.09650773091582979}}$

$tanh\ (h5) = -0.0962092$

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

$tanh\ (h6) = \frac{e^{-0.08461482998284209} - e^{--0.08461482998284209}}{e^{-0.08461482998284209} + e^{--0.08461482998284209}}$

$tanh\ (h6) = -0.0844135$

$tanh\ (h7) = \frac{e^{h7} - e^{-h7}}{e^{h7} + e^{-h7}}$

$tanh\ (h7) = \frac{e^{0.1191711327063339} - e^{-0.1191711327063339}}{e^{0.1191711327063339} + e^{-0.1191711327063339}}$

$tanh\ (h7) = 0.11861$

$tanh\ (h8) = \frac{e^{h8} - e^{-h8}}{e^{h8} + e^{-h8}}$

$tanh\ (h8) = \frac{e^{-0.0895708672147945} - e^{--0.0895708672147945}}{e^{-0.0895708672147945} + e^{--0.0895708672147945}}$

$tanh\ (h8) = -0.0893321$

$tanh\ (h9) = \frac{e^{h9} - e^{-h9}}{e^{h9} + e^{-h9}}$

$tanh\ (h9) = \frac{e^{0.0457518555709885} - e^{-0.0457518555709885}}{e^{0.0457518555709885} + e^{-0.0457518555709885}}$

$tanh\ (h9) = 0.04572$

$tanh\ (h10) = \frac{e^{h10} - e^{-h10}}{e^{h10} + e^{-h10}}$

$tanh\ (h10) = \frac{e^{0.028522880227219695} - e^{-0.028522880227219695}}{e^{0.028522880227219695} + e^{-0.028522880227219695}}$

$tanh\ (h10) = 0.0285151$

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

$dropout_1$ = rate * tanh($h1$)
$dropout_1$ = 0.5 * -0.0166005
$dropout_1$ = -0.00830025

$dropout_2$ = rate * tanh($h2$)
$dropout_2$ = 0.5 * 0.067894
$dropout_2$ = 0.033947

$dropout_3$ = rate * tanh($h3$)
$dropout_3$ = 0.5 * 0.166217
$dropout_3$ = 0.0831085

$dropout_4$ = rate * tanh($h4$)
$dropout_4$ = 0.5 * 0.0345857
$dropout_4$ = 0.01729285

$dropout_5$ = rate * tanh($h5$)
$dropout_5$ = 0.5 * -0.0962092
$dropout_5$ = -0.0481046

$dropout_6$ = rate * tanh($h6$)
$dropout_6$ = 0.5 * -0.0844135
$dropout_6$ = -0.04220675

$dropout_7$ = rate * tanh($h7$)
$dropout_7$ = 0.5 * 0.11861
$dropout_7$ = 0.059305

$dropout_8$ = rate * tanh($h8$)
$dropout_8$ = 0.5 * -0.0893321
$dropout_8$ = -0.04466605

$dropout_9$ = rate * tanh($h9$)
$dropout_9$ = 0.5 * -0.04572
$dropout_9$ = -0.02286

$dropout_{10}$ = rate * tanh($h10$)
$dropout_{10}$ = 0.5 * -0.0285151
$dropout_{10}$ = -0.01425755

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

$output = ((w_{k1} j_7 * dropout_1) + (w_{k2} j_7 * dropout_2) + (w_{k3} j_7 * dropout_3) + (w_{k4} j_7 * dropout_4) + (w_{k5} j_7 * dropout_5) + (w_{k6} j_7 * dropout_6) + (w_{k7} j_7 * dropout_7) + (w_{k8} j_7 * dropout_8) + (w_{k9} j_7 * dropout_9) + (w_{k10} j_7 * dropout_{10})) + b$

$output$ = ((-0.14147132 * -0.00830025) + (0.05554158 * 0.033947) + (-0.27054289 * 0.0831085) + (-0.03277796 * 0.01729285) + (-0.10409342 * -0.0481046) + (-0.24334135 * -0.04220675) + (-0.06665223 * 0.059305) + (0.01262219 * -0.04466605) + (0.18286435 * -0.02286) + (0.16453867 * -0.01425755) + 0)

$output$ = -0.015756292148474498

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):

$\sigma(z) = = 1/(1+exp(-z))$
$\sigma(-0.015756292148474498)$ = 1 / (1 + exp(- -0.015756292148474498))
$\sigma(-0.015756292148474498)$ = 1 / (1 + 1.015881077)
$\sigma(-0.015756292148474498)$ = 1 / 2.015881077
$\sigma(-0.015756292148474498)$ = 0.4960610085

Pada perhitungan di atas, hasil *output neuron* adalah 0.4960610085 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*.

$O_i^*(x) = \eta \sum_{j=1}^{M} O_i(x)C(i,j)$
$O_i^*(x) = \eta(0.4960610085 * (42336/781))$
$O_i^*(x) = \eta 0.4960610085 * 54.207426$
$O_i^*(x) = \eta 26.89019059$
$O_i^*(x) = 1$

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*.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Dalam bab ini akan dijelaskan 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 yang digunakan dalam proses pembuatan sistem, baik dari perangkat keras maupun perangkat lunak yang digunakan.

4.1.1 Spesifikasi Perangkat Keras

Spesifikasi perangkat keras yang digunakan untuk pembuatan sistem prediksi penyakit stroke adalah sebagai berikut:

1. Laptop ASUS Vivobook Flip 14 TP410F.
2. *Processor* Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 1.99GHz.
3. HDD sebesar 1TB dan SSD sebesar 128GB.
4. RAM sebesar 16GB.

4.1.2 Lingkungan Perangkat Lunak

Lingkungan perangkat lunak yang digunakan untuk pembuatan sistem prediksi penyakit stroke adalah sebagai berikut:

1. Sistem operasi Windows 11 Home Single Language.
2. Jupyter Notebook 6.0.3.
3. Python 3.8.3. (Jupyter Notebook)
4. Google Colab.

4.2 Daftar *Class* dan *Method*

Pada bab ini akan dijelaskan mengenai *class* dan *method* yang akan digunakan dalam pembuatan sistem prediksi penyakit stroke.

4.2.1 *Class DeepNeuralNetwork*

Tabel 4.1 Daftar *method* pada *class DeepNeuralNetwork*

No.	*Method*	Masukan	Luaran	Keterangan

1.	__init__	x_train: float[], x_test: float[], y_train: int[], y_test: int[], units: int, hidden_layer: int	-	Fungsi yang bertujuan inisialisasi variabel yang akan digunakan, serta inisialisasi *weight* dengan nilai *random* dan *bias* dengan nilai 0.
2.	forward_prop	hidden_layer: int	float[]	Fungsi yang bertujuan melakukan perhitungan sesuai rumus *forward propagation* dan hasilnya ditampung dalam *array*.
3.	backward_prop	hidden_layer: int, learning_rate: int	-	Fungsi yang bertujuan melakukan *backward propagation* yang berfungsi untuk *update weight* dan *bias* agar *error* yang didapat model menjadi lebih kecil dan hasil prediksi menjadi lebih baik.
4.	train	x_train: float[], y_train: int[], learning_rate: int	-	Fungsi yang bertujuan melakukan *training* dengan cara memanggil fungsi lain yaitu forward_prop dan backward_prop.
5.	hitung_akurasi	x_train: float[], y_train: int[]	-	Fungsi yang bertujuan melakukan perhitungan akurasi model *training* pada setiap *epoch*.

6.	akurasi_testing	x_test: float[], y_test: int[]	-	Fungsi yang bertujuan melakukan perhitungan akurasi model *testing*, ROC AUC, TPR, FPR, G-Mean, serta menggambarkan grafik ROC *curve*.

4.2.2 *Class DeepNeuralNetworkDropout*

Tabel 4.2 Daftar *method* pada *class DeepNeuralNetworkDropout*

No.	***Method***	**Masukan**	**Luaran**	**Keterangan**
1.	__init__	x_train: float[], x_test: float[], y_train: int[], y_test: int[], units: int, hidden_layer: int, rate: float	-	Fungsi yang bertujuan inisialisasi variabel yang akan digunakan, serta inisialisasi *weight* dengan nilai *random* dan *bias* dengan nilai 0.
2.	forward_prop	hidden_layer: int, rate: float	float[]	Fungsi yang bertujuan melakukan perhitungan sesuai rumus *forward propagation* dan hasilnya ditampung dalam *array*.
3.	backward_prop	hidden_layer: int, learning_rate: int, rate: float	-	Fungsi yang bertujuan melakukan *backward propagation* yang berfungsi untuk *update weight* dan *bias* agar *error* yang didapat model menjadi lebih kecil dan hasil prediksi menjadi lebih baik.

4.	train	x_train: float[], y_train: int[], learning_rate: int	-	Fungsi yang bertujuan melakukan *training* dengan cara memanggil fungsi lain yaitu forward_prop dan backward_prop.
5.	hitung_akurasi	x_train: float[], y_train: int[]	-	Fungsi yang bertujuan melakukan perhitungan akurasi model *training* pada setiap *epoch*.
6.	akurasi_testing	x_test: float[], y_test: int[]	-	Fungsi yang bertujuan melakukan perhitungan akurasi model *testing*, ROC AUC, TPR, FPR, G-Mean, serta menggambarkan grafik ROC *curve*.

4.2.3 *Fungsi sigmoid*

Tabel 4.3 Penjelasan fungsi *sigmoid*

No.	***Method***	**Masukan**	**Luaran**	**Keterangan**
1.	sigmoid	t: float[]	float[]	Fungsi yang bertujuan melakukan perhitungan sesuai rumus sigmoid.

4.2.4 *Fungsi sigmoid_derivative*

Tabel 4.4 Penjelasan fungsi *sigmoid_derivative*

No.	***Method***	**Masukan**	**Luaran**	**Keterangan**
1.	sigmoid_derivative	t: float[]	float[]	Fungsi yang bertujuan melakukan perhitungan sesuai turunan rumus sigmoid.

4.2.5 *Fungsi tanh*

Tabel 4.5 Penjelasan fungsi *tanh*

No.	*Method*	Masukan	Luaran	Keterangan
1.	tanh	t: float[]	float[]	Fungsi yang bertujuan melakukan perhitungan sesuai rumus tanh.

4.2.6 *Fungsi tanh_derivative*

Tabel 4.6 Penjelasan fungsi *tanh_derivative*

No.	*Method*	Masukan	Luaran	Keterangan
1.	tanh_derivative	t: float[]	float[]	Fungsi yang bertujuan melakukan perhitungan sesuai turunan rumus tanh.

4.2.7 *Fungsi relu*

Tabel 4.7 Penjelasan fungsi *relu*

No.	*Method*	Masukan	Luaran	Keterangan
1.	relu	t: float[]	float[]	Fungsi yang bertujuan melakukan perhitungan sesuai rumus ReLU.

4.2.8 *Fungsi relu_derivative*

Tabel 4.8 Penjelasan fungsi *relu_derivative*

No.	*Method*	Masukan	Luaran	Keterangan
1.	relu_derivative	t: float[]	float[]	Fungsi yang bertujuan melakukan perhitungan sesuai turunan rumus ReLU.

4.3 Implementasi Perangkat Lunak

Pada bab ini akan dijelaskan mengenai cara implementasi sistem prediksi penyakit stroke menggunakan *Deep Neural Network*, *dropout*, *cost-sensitive learning*, dan *probability tuning*.

4.3.1 *Implementasi Preprocessing*

Seperti yang sudah dijelaskan pada Bab 3, sebelum dilakukan pembuatan model dengan *Deep Neural Network*, dilakukan *preprocessing* terlebih dahulu. Pada penelitian ini, *preprocessing data* yang dilakukan meliputi penghapusan beberapa kolom, data *imputation*, data *encoding*, pengecekan data duplikat, dan *cleaning outliers*. Pada *class preprocessing* terdapat beberapa *method* yang digunakan dari *library Pandas* yaitu *read_csv*, *drop*, *isnull*, *sum*, *fillna*, *mean*, *replace*, *duplicated*, *describe*, dan *value_counts*. *Method-method* tersebut berfungsi untuk membaca *dataset*, menghapus kolom yang tidak diperlukan, mengecek data *null*, mengecek data duplikat, *cleaning outliers*, dan menghitung jumlah data dari masing-masing kelas.

4.3.2 Implementasi *Deep Neural Network*

Pada bagian ini, dilakukan pembuatan model dengan menggunakan arsitektur *Deep Neural Network* (DNN) menggunakan *dataset* yang telah dilakukan *preprocessing* sebelumnya. Arsitektur DNN yang akan digunakan dalam penelitian ini digambarkan pada Gambar 2.4. Dalam penelitian ini, terdapat beberapa hyperparameter yang akan digunakan, yaitu *learning rate* sebesar 0.1 dan 0.01. *hidden layer* sebesar 10, 20, dan 30, *epoch* sebesar 50, 200, dan 400, *activation function* yang dipakai pada *input layer* dan *hidden layer* adalah ReLu dan tanh, sedangkan pada output layer selalu memakai *sigmoid*.

4.3.2.1 Implementasi *Training Deep Neural Network*

Proses *training Deep Neural Network* adalah sebagai berikut.

1. Inisialisasi jumlah *hidden layer*, *learning rate*, *epoch*, dan *activation function* yang akan digunakan.
2. Melakukan perhitungan menggunakan data *input* dari hasil *preprocessing* menggunakan algoritme *forward propagation*.
3. Melakukan proses perhitungan *error* menggunakan rumus MSE.
4. Melakukan *update weight* menggunakan algoritme *backpropagation*.

4.3.2.2 Implementasi *Testing Deep Neural Network*

Proses *testing Deep Neural Network* adalah sebagai berikut.

1. Melakukan perhitungan dengan algoritme *forward propagation*.
2. Menghitung nilai *true positive*, *true negative*, *false positive*, *false negative* untuk dimasukkan dalam ROC *curve*
3. Melihat *overfitting* model dengan menggunakan *learning curve*.

4.3.3 Implementasi *Dropout*

Pada bagian ini, dilakukan pembuatan model dengan menggunakan arsitektur *Deep Neural Network* (DNN) ditambah dengan *dropout*. Dalam *dropout*, terdapat *hyperparameter* tambahan yang bernama *rate*. *Rate* akan digunakan sebesar 0.1 di antara *input layer* dengan *hidden layer*, sedangkan 0.5 dan 0.8 di antara *hidden layer* dan *output layer*.

4.3.3.1 Implementasi *Training Deep Neural Network* dengan *Dropout*

Proses *training Deep Neural Network* adalah sebagai berikut.

1. Inisialisasi jumlah *hidden layer*, *learning rate*, *epoch*, *activation function*, dan *rate* yang akan digunakan.
2. Melakukan perhitungan menggunakan data *input* dari hasil *preprocessing* menggunakan algoritme *forward propagation* dan rumus *dropout*.
3. Melakukan proses perhitungan *error (loss)* menggunakan rumus MSE.
4. Melakukan *update weight* menggunakan algoritme *backpropagation*.

4.3.3.2 Implementasi *Testing Deep Neural Network* dengan *Dropout*

Proses *testing Deep Neural Network* adalah sebagai berikut.

1. Melakukan perhitungan dengan algoritme *forward propagation*.
2. Menghitung nilai *true positive*, *true negative*, *false positive*, *false negative* untuk dimasukkan dalam ROC *curve*
3. Melihat *overfitting* model dengan menggunakan *learning curve*.

4.4 Pengujian

Pada penelitian ini, pengujian yang dilakukan adalah membandingkan algoritme *Deep Neural Network* (DNN) dengan algoritme *Deep Neural Network* yang ditambahkan *regularization dropout* untuk melakukan prediksi penyakit stroke. Hasil yang diharapkan dalam penelitian ini adalah model DNN yang ditambahkan *dropout* memiliki *overfitting* lebih rendah dan nilai ROC *curve*-nya lebih tinggi daripada model yang hanya menggunakan metode DNN saja.

Dalam penelitian ini, penulis akan menggunakan banyak kombinasi *hyperparameter* agar dapat menemukan kombinasi yang tepat. *Hyperparameter* yang digunakan dalam DNN adalah *learning rate*, *hidden layer*, *epoch*, dan *activation function*, sedangkan dalam *dropout* menggunakan *rate*.

4.4.1 Skenario Pengujian *Deep Neural Network*

Pengujian pertama akan dilakukan hanya menggunakan algoritme *Deep Neural Network* dengan menggunakan 4 jenis *hyperparameter* dengan jumlah kombinasi sebesar 36 kombinasi. Kombinasi *hyperparameter* digambarkan dalam Gambar 4.1 di bawah ini.

DNN				
	Learning Rate	Hidden Layer	Epoch	Activation Function
	0.1	5	10	ReLU
	0.01	10	50	Tanh
		20	100	
Total	2	3	3	2
Total pengujian	= 2 x 3 x 3 x 2			
	36			

Gambar 4.1 Kombinasi *Deep Neural Network*

Pengujian selanjutnya yaitu menguji algoritme *Deep Neural Network* dengan *dropout* dengan menggunakan 5 jenis *hyperparameter*. Kombinasi yang akan digunakan dalam pengujian ini adalah 4 hasil kombinasi terbaik dari percobaan sebelumnya, lalu ditambahkan dengan *rate*, sehinggal dalam pengujian ini terdapat sebesar 8 kombinasi.

4.4.2 Pengujian *Deep Neural Network*

Pada bagian ini, akan dijelaskan mengenai hasil pengujian dari *Deep Neural Network* sesuai dengan *hyperparameter* yang sudah dijelaskan sebelumnya.

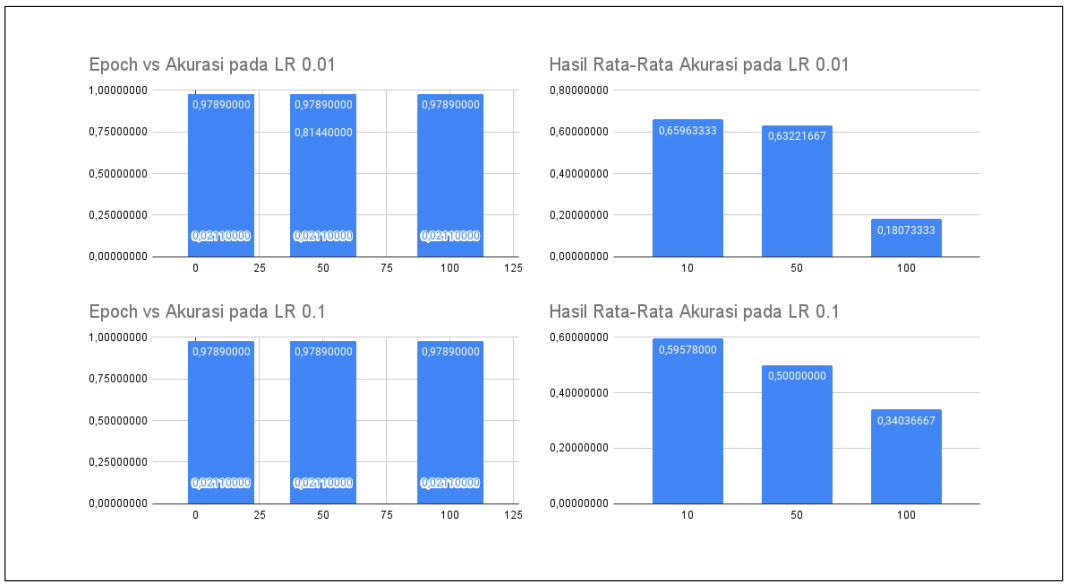
Tabel 4.9 Hasil pengujian *Deep Neural Network*

No.	*Learning Rate*	*Hidden Layer*	*Epoch*	*Activation Function*	*ROC*	*Akurasi*
1.	0.01	5	10	tanh	nan	0.0211
2.	0.01	5	10	relu	nan	0.9789
3.	0.01	5	50	tanh	nan	0.0211
4.	0.01	5	50	relu	0.5189	0.8144
5.	0.01	5	100	tanh	nan	0.0211
6.	0.01	5	100	relu	nan	0.0211
7.	0.01	10	10	tanh	nan	0.9789

8.	0.01	10	10	relu	nan	0.0211
9.	0.01	10	50	tanh	nan	0.0211
10.	0.01	10	50	relu	nan	0.9789
11.	0.01	10	100	tanh	nan	0.0211
12.	0.01	10	100	relu	nan	0.9789
13.	0.01	20	10	tanh	nan	0.9789
14.	0.01	20	10	relu	nan	0.9789
15.	0.01	20	50	tanh	nan	0.9789
16.	0.01	20	50	relu	nan	0.9789
17.	0.01	20	100	tanh	nan	0.0211
18.	0.01	20	100	relu	nan	0.0211
19.	0.1	5	10	tanh	nan	0.0211
20.	0.1	5	10	relu	nan	0.9789
21.	0.1	5	50	tanh	nan	0.9789
22.	0.1	5	50	relu	nan	0.9789
23.	0.1	5	100	tanh	nan	0.9789
24.	0.1	5	100	relu	nan	0.0211
25.	0.1	10	10	tanh	nan	0.0211
26.	0.1	10	10	relu	nan	0.9789
27.	0.1	10	50	tanh	nan	0.9789
28.	0.1	10	50	relu	nan	0.0211
29.	0.1	10	100	tanh	nan	0.9789
30.	0.1	10	100	relu	nan	0.0211
31.	0.1	20	10	tanh	nan	0.0211
32.	0.1	20	10	relu	nan	0.9789
33.	0.1	20	50	tanh	nan	0.0211
34.	0.1	20	50	relu	nan	0.0211
35.	0.1	20	100	tanh	nan	0.0211
36.	0.1	20	100	relu	nan	0.0211

Pada pengujian menggunakan metode *Deep Neural Network*, yang

ditambah *cost-sensitive learning*, hasil menunjukkan bahwa akurasi paling tinggi adalah 97.89% dan paling rendah adalah 2.11%. Pengujian menunjukkan tidak ada perubahan walaupun *learning rate*, *hidden layer*, *epoch*, dan *activation function* diganti. Teknik *probability tuning* tidak dapat digunakan dalam pengujian kali ini, dikarenakan setiap pengujian terdapat kasus TPR atau FPR yang berisi nan, sehingga tidak bisa dilakukan perhitungan G-Mean dan ROC. Ringkasan hasil pengujian yang sudah digambarkan dengan grafik, dapat dilihat pada gambar di bawah ini.

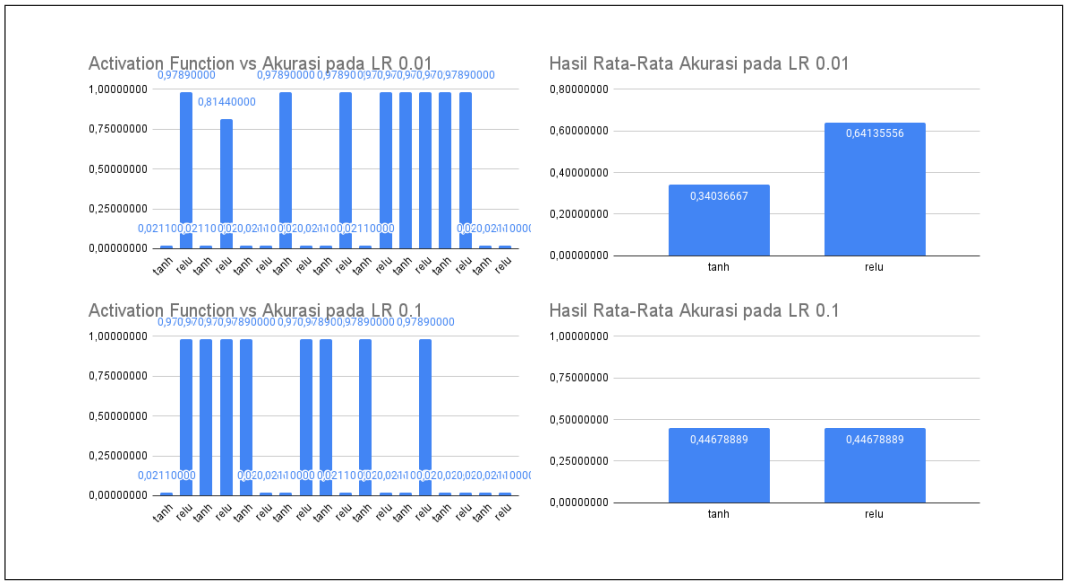


Gambar 4.2 Perbandingan akurasi berdasarkan jumlah *epoch* dan *learning rate*

Pada gambar 4.2, baik dengan *learning rate* 0.01 maupun 0.1, hasil rata-rata pengujian menunjukkan bahwa semakin besar *epoch*, akurasi akan semakin kecil. Hal ini sesuai dengan teori yang ada, yaitu semakin besar *epoch* dapat meningkatkan akurasi, namun dapat juga menyebabkan *overfitting* yang ditunjukkan dengan akurasi yang turun secara drastis saat pengujian dengan menggunakan *learning rate* 0.01 dan *epoch* 100.

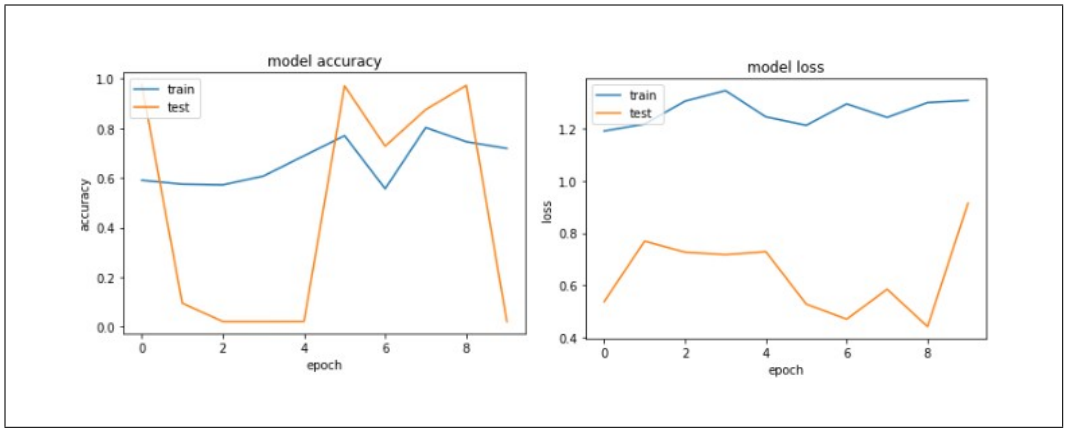
Gambar 4.3 Perbandingan akurasi berdasarkan jumlah *epoch* dan *learning rate*

Pada gambar 4.3, dengan *learning rate* 0.01, hasil rata-rata menunjukkan bahwa semakin besar *hidden layer*, maka akurasi semakin meningkat. Tetapi hal ini berbeda dengan *learning rate* 0.1, yang menunjukkan bahwa semakin besar *hidden layer*, maka akurasinya semakin menurun. Hal ini dapat terjadi karena *learning rate* yang dipakai terlalu besar sehingga model tidak dapat konvergen. Hal ini juga sesuai dengan teori yang ada, bahwa hidden layer dapat mencari pola-pola yang tidak terlihat dalam data, namun dapat menyebabkan *overfitting* yang dibuktikan dengan turunnya akurasi pada *learning rate* 0.1.



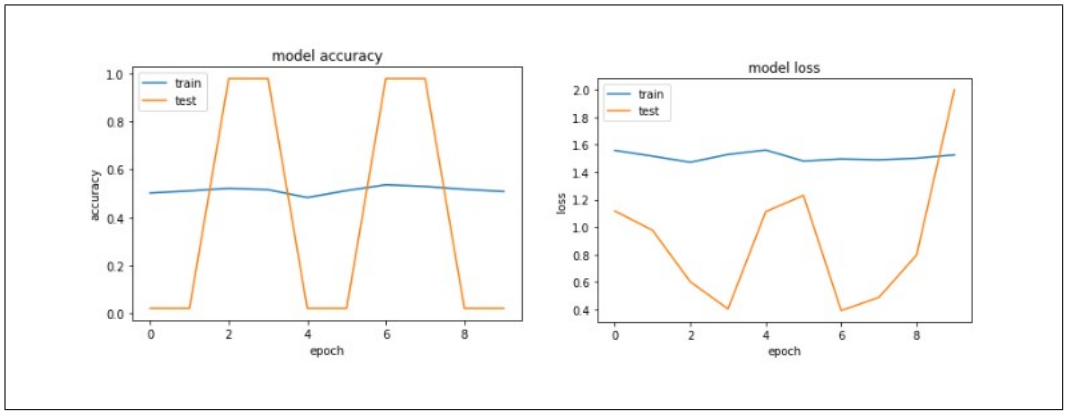
Gambar 4.4 Perbandingan akurasi berdasarkan jumlah *activation function* dan *learning rate*

Pada gambar 4.4, dengan *learning rate* 0.01, hasil rata-rata menunjukkan bahwa model lebih cocok menggunaka *activation function* ReLu, dikarenakan rata-rata akurasinya yang lebih besar, yaitu 64.13%. Namun hal ini tidak terjadi pada penggunaan *learning rate* 0.1, dikarenakan baik penggunakan *activation function* tanh maupun ReLu tidak memberikan perubahan nilai akurasi, yaitu sebesar 44.67%.



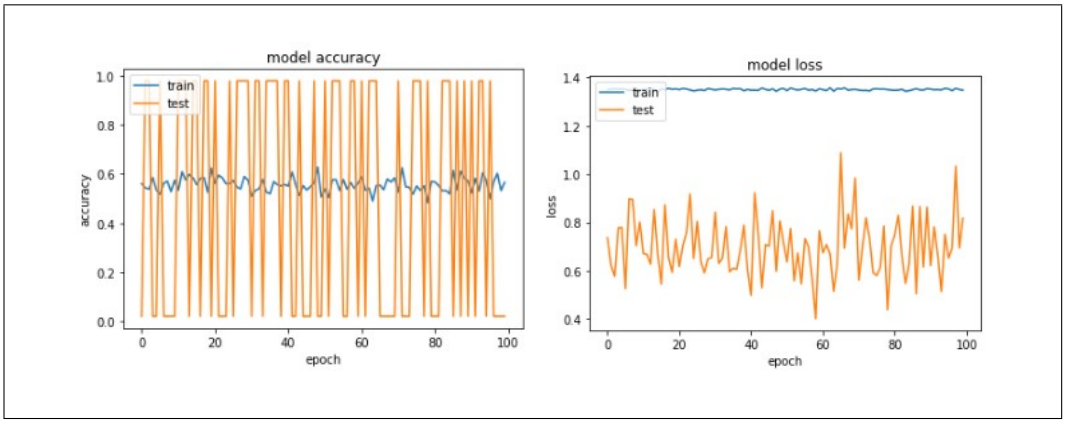
Gambar 4.5 *Learning curve* model 1

Gambar 4.5 merupakan hasil dari kombinasi *hyperparameter learning rate* 0.01, *hidden layer* 5, *epoch* sebesar 10, dan *activation function* tanh. Grafik *model accuracy* bagian training menunjukkan bahwa pada saat model dilakukan *training*, akurasi berada di 60% lalu lambat laun meningkat saat berada di *epoch* 5, namun turun saat berada di *epoch* 6 dan meningkat kembali. Dapat dilihat bahwa hasil dari grafik *training* cenderung stabil. Sementara pada grafik *test*, grafik sangat tidak stabil, dikarenakan akurasi yang turun lalu naik secara signifikan. Pada grafik *model loss*, grafik *train* dan *test* saling berjauhan. Dapat disimpulkan model mengalami *underfitting*.



Gambar 4.6 *Learning curve* model 2

Gambar 4.6 merupakan hasil dari kombinasi *hyperparameter learning rate* 0.1, *hidden layer* 5, *epoch* sebesar 10, dan *activation function* tanh. Grafik *model accuracy* bagian training menunjukkan bahwa pada saat model dilakukan *training*, akurasi berada di 50% lalu cenderung stabil sampai *epoch* 10. Sementara pada grafik *test*, grafik sangat tidak stabil, dikarenakan akurasi yang naik lalu turun secara signifikan. Pada grafik *model loss*, grafik *train* dan *test* saling berjauhan dan pada garis *test*, *loss* tiba-tiba meningkat secara drastis pada *epoch* 8 sampai 10. Dapat disimpulkan model mengalami *underfitting*.



Gambar 4.7 *Learning curve* model 3

Gambar 4.7 merupakan hasil dari kombinasi *hyperparameter learning rate* 0.1, *hidden layer* 20, *epoch* sebesar 100, dan *activation function* relu. Grafik *model accuracy* bagian *training*, grafik masih cenderung stabil, tidak banyak perubahan akurasi yang terjadi. Sementara pada grafik *test* sangat tidak stabil, dikarenakan grafik naik turun sangat signifikan secara terus menerus. Pada grafik *model loss*, grafik *train* dan *test* saling berjauhan dan pada garis *test*, *loss* mengalami perubahan naik turun secara terus menerus. Dapat disimpulkan model mengalami *underfitting*.

4.4.3 Pengujian *Deep Neural Network* dengan *Dropout*

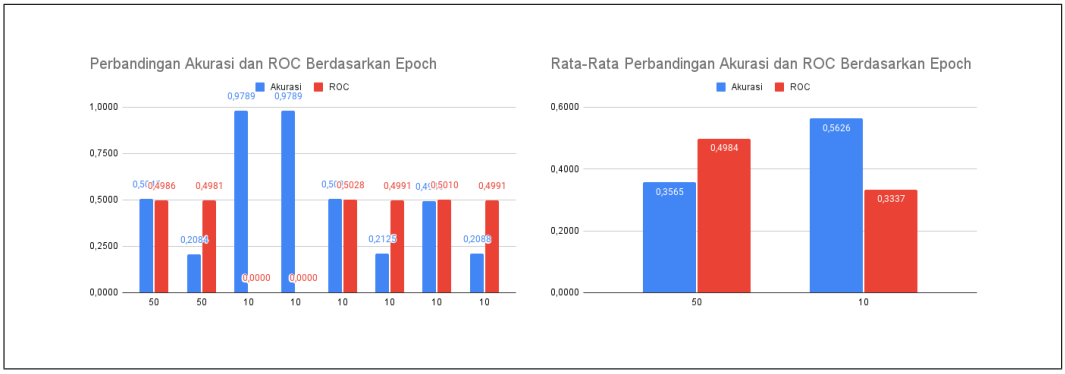
Pada bagian ini, akan dijelaskan mengenai hasil pengujian dari *Deep Neural Network* dengan *dropout* dengan memilih 4 hasil terbaik dari percobaan sebelumnya.

Tabel 4.10 Hasil pengujian *Deep Neural Network* dengan *Dropout*

No.	*Learning Rate*	*Hidden Layer*	*Epoch*	*Activation Function*	*Rate*	*ROC*	*Akurasi*
1.	0.01	20	50	relu	0.5	0.4986	0.5045
2.	0.01	20	50	relu	0.8	0.4981	0.2084

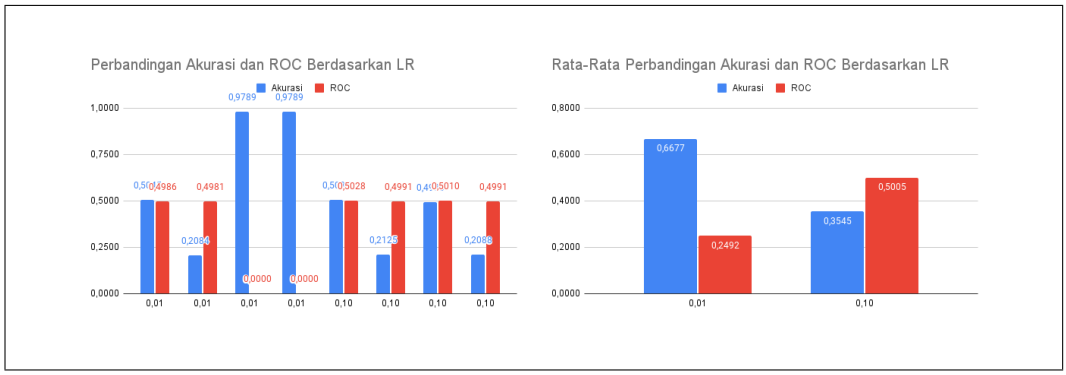
3.	0.01	20	10	tanh	0.5	nan	0.9789
4.	0.01	20	10	tanh	0.8	nan	0.9789
5.	0.1	20	10	relu	0.5	0.5028	0.5037
6.	0.1	20	10	relu	0.8	0.4991	0.2125
7.	0.1	10	10	relu	0.5	0.5010	0.4930
8.	0.1	10	10	relu	0.8	0.4991	0.2088

Pada pengujian menggunakan metode *Deep Neural Network*, yang ditambah *cost-sensitive learning*, hasil menunjukkan bahwa akurasi paling tinggi adalah 97.89% dan paling rendah adalah 20.84%. Namun, akurasi 97.89% tersebut memiliki ROC dan G-Mean nan, sehingga teknik *probability tuning* tidak bisa dilakukan dan dalam *confusion matrix*, model tersebut memiliki kecenderungan *overfitting*. Jadi dapat dikatakan, dalam kasus ini, *activation function* tanh tidak cocok digunakan.



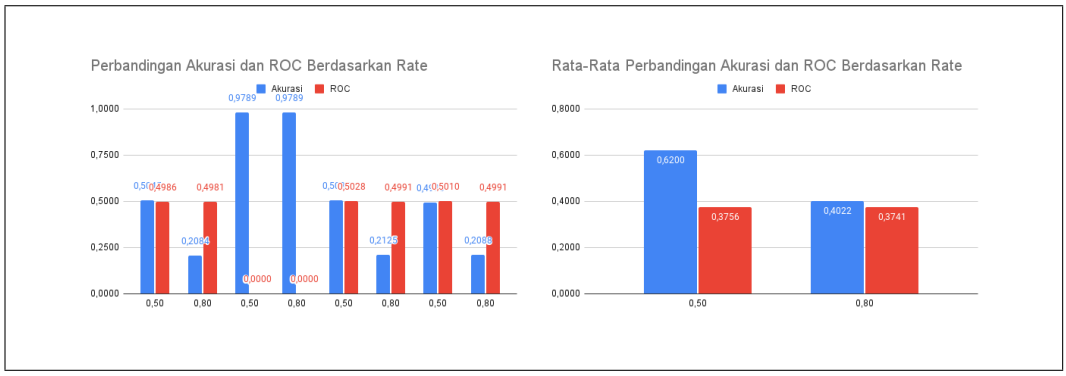
Gambar 4.8 Perbandingan akurasi dan ROC berdasarkan jumlah *epoch*

Pada Gambar 4.8 dapat dilihat bahwa rata-rata perbandingan akurasi dan ROC berdasarkan epoch memiliki perbedaan yang cukup signifikan. Saat *epoch* 10, model mendapatkan akurasi sebesar 56.26% dan ROC sebesar 33.37%, sedangkan saat *epoch* diubah menjadi 50, akurasi turun menjadi hanya sebesar 35.65%, namun ROC mengalami kenaikkan menjadi 49.84%. Hal ini sesuai dengan teori yang ada, bahwa semakin besar *epoch*, semakin besar pula kemungkinan model *overfitting*.



Gambar 4.9 Perbandingan akurasi dan ROC berdasarkan jumlah *learning rate*

Pada Gambar 4.8 dapat dilihat bahwa rata-rata perbandingan akurasi dan ROC berdasarkan *learning rate* memiliki perbedaan yang signifikan. Saat *learning rate* 0.01, model mendapatkan akurasi sebesar 66.77% dan ROC hanya sebesar 24.92%, sedangkan saat *learning rate* 50, akurasi turun cukup drastis menjadi sebesar 35.45%, namun ROC mengalami kenaikkan menjadi 50.045%. Dengan grafik ini bisa diambil kesimpulan bahwa semakin tinggi *learning rate*, tidak membuat akurasinya menjadi lebih baik, namun ROC berubah menjadi lebih baik.



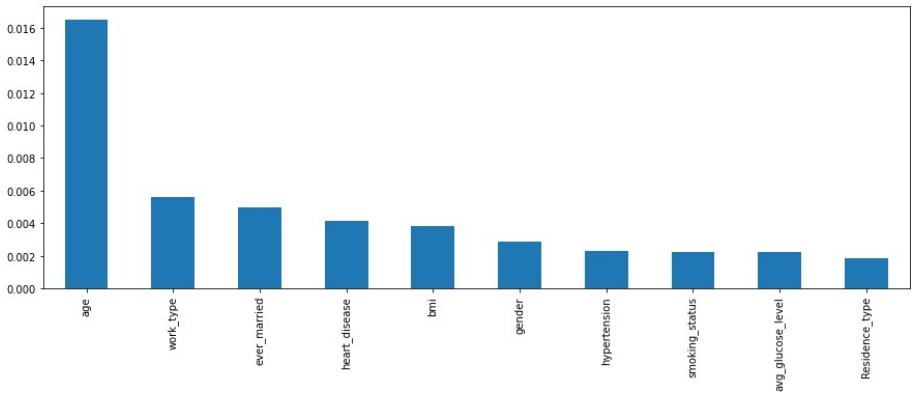
Gambar 4.10 Perbandingan akurasi dan ROC berdasarkan jumlah *rate*

Gambar 4.10 dapat dilihat bahwa rata-rata perbandingan akurasi dan ROC berdasarkan *rate* memiliki perbedaan akurasi yang cukup signifikan. Pada saat *rate* 0.5, model mendapatkan akurasi sebesar 66% dan ROC hanya sebesar 37.56%, sedangkan saat *rate* diperbesar menjadi 0.8, akurasi turun cukup drastis menjadi 40.22%, dan ROC tidak mengalami perubahan yang berarti. Dengan grafik ini bisa diambil kesimpulan bahwa semakin tinggi *rate*, maka akurasi akan semakin rendah dikarenakan semakin banyak *neuron* yang dimatikan sehingga model tidak dapat melakukan prediksi dengan baik.

4.4.4 Pengujian Tambahan dengan *Feature Selection Information Gain*

Pada tabel 4.15 yang menggunakan algoritme DNN menghasilkan akurasi yang sangat *overfitting* dan *underfitting*. Hasil akurasi hanya ada 3 variasi angka, yaitu

2.11%, 81.44%, dan 97.89%. Pada tabel 4.16 yang menggunakan algoritme DNN dan *dropout* menghasilkan ROC dan akurasi yang buruk. Hasil akurasi tertinggi adalah 97.89%, namun hasil dari *confusion matrix* model tersebut menunjukkan bahwa model sangat *overfitting*. Model lain tidak *overfitting*, namun memberikan hasil yang sangat buruk. Akurasi tertinggi model hanya 50.37% dan ROC tertinggi hanya 50.28%. Dalam kasus prediksi stroke, yang hanya memprediksi 2 kemungkinan (sakit stroke atau tidak sakit stroke), peluang 50% tentu sama saja dengan menebak. Hal ini mungkin saja terjadi dikarenakan fitur yang dipakai terlalu banyak dan tidak relevan untuk prediksi stroke. Oleh karena itu, akan dilakukan *feature selection* dengan metode *information gain*.



Gambar 4.11 Hasil variabel yang paling berpengaruh dari *information gain*

Diagram batang 4.11 di atas menunjukkan bahwa fitur *age* adalah fitur yang sangat berpengaruh terhadap prediksi stroke. Fitur selanjutnya masih memiliki pengaruh, namun sudah tidak signifikan terhadap prediksi stroke. Berdasarkan diagram di atas maka hanya akan diambil 5 fitur terbaik saja, dikarenakan fitur selanjutnya sudah semakin tidak signifikan.

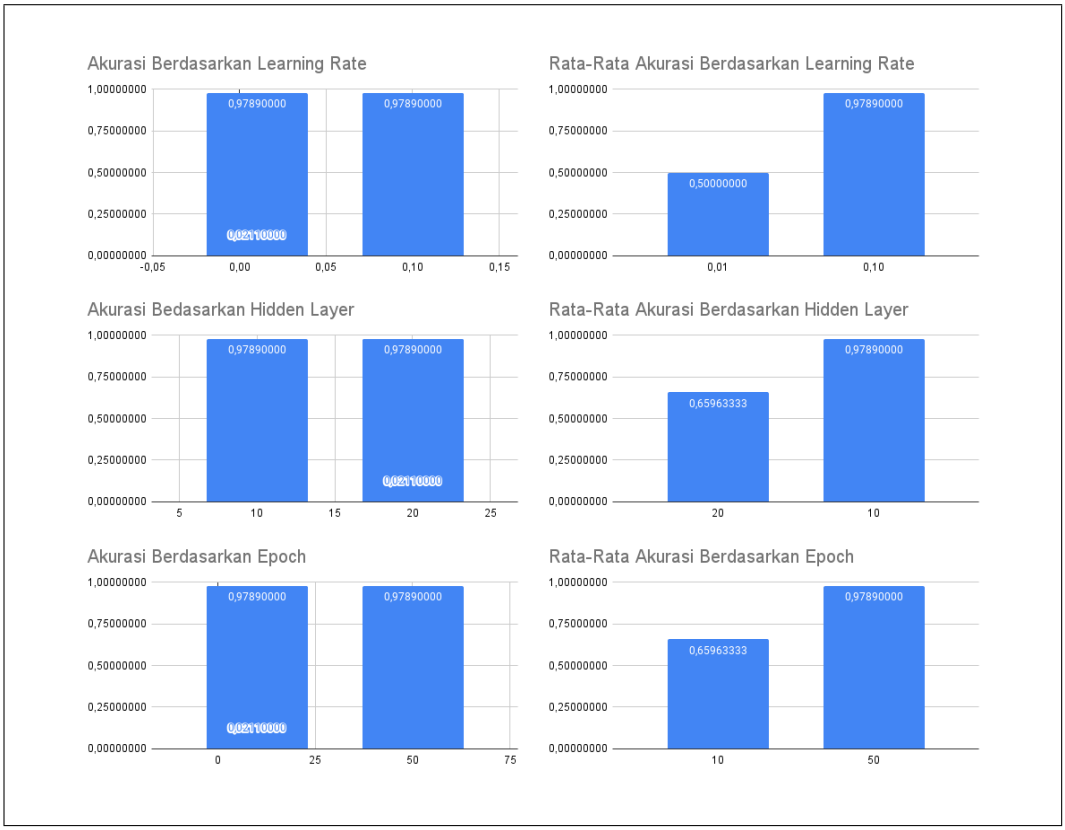
4.4.4.1 Pengujian *Deep Neural Network* dengan *Feature Selection Information Gain*

Setelah dilakukan pengambilan 5 fitur terbaik, maka dilakukan kembali pengujian dengan memilih 4 kombinasi terunik dari model sebelumnya (tanpa menggunakan *information gain*)jika dilihat dari *learning curve*.

Tabel 4.11 Hasil pengujian *Deep Neural Network*

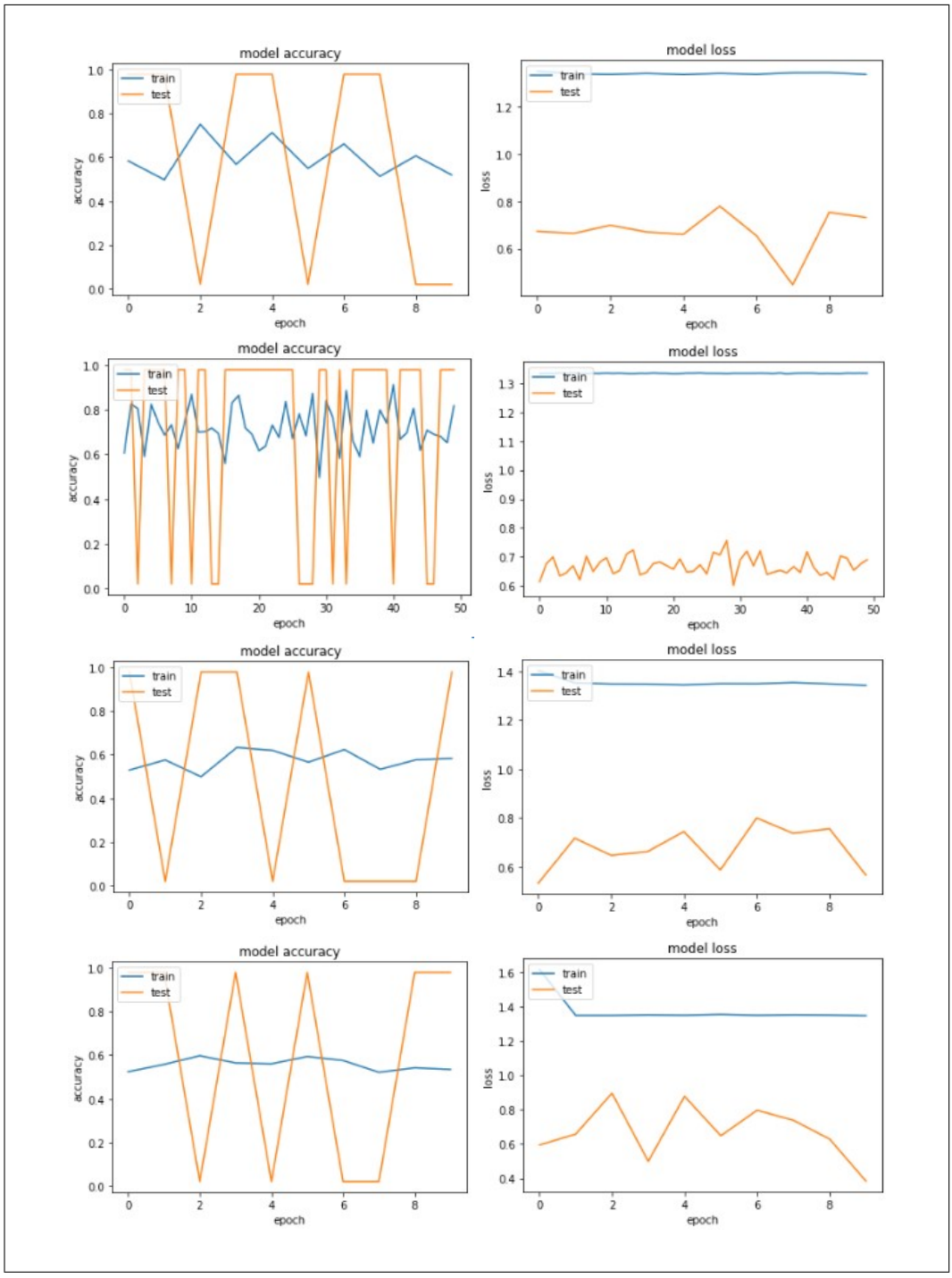
No.	***Learning Rate***	***Hidden Layer***	***Epoch***	***Activation Function***	***ROC***	***Akurasi***
1.	0.01	20	10	tanh	nan	0.0211

2.	0.01	20	50	relu	nan	0.9789
3.	0.1	10	10	relu	nan	0.9789
4.	0.1	20	10	relu	nan	0.9789



Gambar 4.12 Akurasi berdasarkan jumlah *learning rate*, *hidden layer*, dan *epoch*

Pada Gambar 4.12 dapat dilihat bahwa jika hanya mengambil 5 fitur terbaik, maka *learning rate* 0.1, *hidden layer* 10, dan *epoch* sebesar 50 menghasilkan akurasi yang lebih tinggi. Namun, hasil model tersebut merupakan model yang mengalami *overfitting*. Untuk lebih jelasnya, dapat dilihat pada Gambar 4.13 *learning curve* di bawah ini.



Gambar 4.13 Perbandingan *Learning curve* untuk setiap percobaan

Gambar 4.13 menjelaskan bahwa, semua model mengalami overfitting, yang dibuktikan dari garis *train* dan *test* pada grafik *model accuracy* yang tidak berhimpitan atau saling menjauh. Dapat dikatakan, model tidak cocok menggunakan kombinasi *hypermeter* di atas. Oleh sebab itu, maka akan dilakukan pengujian dengan *hidden layer* yang lebih kecil untuk mencegah *overfitting* pada model.

Jika hasil antara Tabel 4.15 dengan Tabel 4.11 dibandingkan, maka hasil akurasi hanya di kombinasi pertama yang memiliki perubahan drastis, sedangkan ROC nya tidak mengalami perubahan, namun kombinasi lainnya memiliki nilai yang sama. Perbandingan antara hasil pengujian model sebelum dan sesudah ditambah *feature selection* dapat dilihat pada Tabel 4.12 di bawah ini.

Tabel 4.12 Hasil pengujian *Deep Neural Network* dengan *information gain*

No.	***Akurasi Sebelum***	***Akurasi Sesudah***	***ROC Sebelum***	***ROC Sesudah***	***Keterangan***
1.	0.9789	0.0211	nan	nan	Akurasi meningkat, ROC tidak mengalami perubahan.
2.	0.9789	0.9789	nan	nan	Akurasi dan ROC tidak mengalami perubahan.
3.	0.9789	0.9789	nan	nan	Akurasi dan ROC tidak mengalami perubahan.
4.	0.9789	0.9789	nan	nan	Akurasi dan ROC tidak mengalami perubahan.

4.4.4.2 Pengujian *Deep Neural Network* dan *Dropout* dengan *Feature Selection Information Gain*

Setelah dilakukan pengambilan 5 fitur terbaik, maka dilakukan kembali pengujian dengan memilih 3 kombinasi terbaik dari model *Deep Neural Network* dengan *dropout*.

Tabel 4.13 Hasil pengujian *Deep Neural Network* dan *Dropout* dengan *information gain*

No.	***Learning Rate***	***Hidden Layer***	***Epoch***	***Activation Function***	***Rate***	***ROC***	***Akurasi***
1.	0.01	20	50	relu	0.5	0.5004	0.5053
2.	0.01	20	50	relu	0.8	0.5017	0.2106
3.	0.01	20	50	relu	0.5	0.4972	0.4986

Jika hasil antara Tabel 4.16 dengan Tabel 4.13 dibandingkan, maka perbedaan angkanya akurasi dan ROC tidak signifikan. Pada kombinasi pertama, akurasi model setelah ditambah *feature selection information gain* meningkat. Pada kombinasi kedua, akurasi dan ROC meningkat, sedangkan pada kombinasi ketiga,

akurasi dan ROC menurun. Perbandingan antara hasil pengujian model sebelum dan sesudah ditambah *feature selection* dapat dilihat pada Tabel 4.14 di bawah ini.

Tabel 4.14 Hasil pengujian *Deep Neural Network* dan *Dropout* dengan *information gain*

No.	*Akurasi Sebelum*	*Akurasi Sesudah*	*ROC Sebelum*	*ROC Sesudah*	*Keterangan*
1.	0.5045	0.5053	0.4986	0.5004	Akurasi dan ROC meningkat
2.	0.2084	0.2106	0.4981	0.5017	Akurasi dan ROC meningkat
3.	0.5037	0.4986	0.5028	0.4972	Akurasi dan ROC menurun

Hasil pengujian model yang sudah dilakukan *feature selection* mengalami perubahan, namun sangat tidak signifikan. Hal ini mungkin saja terjadi dikarenakan penggunaan *hidden layer* yang terlalu banyak yang mengakibatkan model mengalami *overfitting*. Oleh sebab itu, akan dilakukan kembali pengujian dengan *hidden layer* yang lebih kecil untuk menghindari *overfitting*.

4.4.5 Pengujian tambahan dengan *Hidden Layer* yang Lebih Kecil

Pengujian tambahan dilakukan agar mendapatkan akurasi yang lebih baik dengan cara menguji menggunakan *hidden layer* yang lebih kecil untuk menghindari overfitting.

4.4.5.1 Pengujian Deep Neural Network

Kombinasi pengujian di bawah ini merupakan kombinasi pengujian untuk algoritme *Deep Neural Network*, yang ditambah dengan *cost sensitive learning* dan *probability tuning*.

DNN				
	Learning Rate	**Hidden Layer**	**Epoch**	**Activation Function**
	0.1	2	10	ReLU
	0.01	3	50	Tanh
		4	100	
Total	**2**	**3**	**3**	**2**
Total pengujian	= 2 x 3 x 3 x 2			
	36			

Gambar 4.14 Pengujian tambahan dengan menggunakan *hidden layer* yang lebih kecil

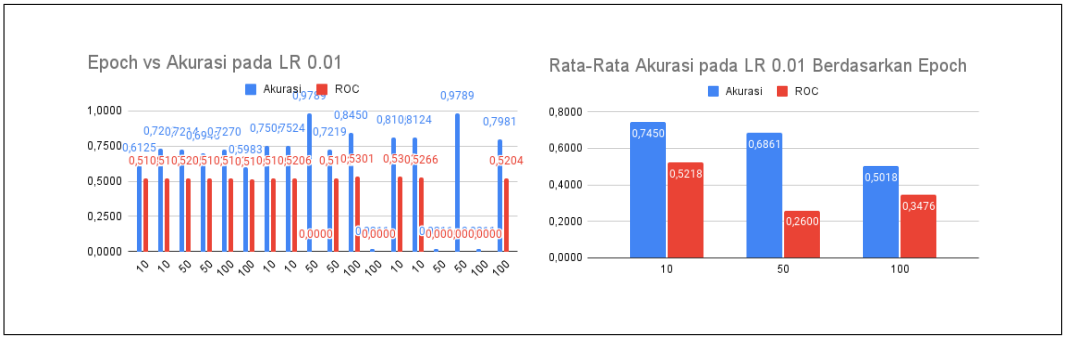
Pengujian ini menggunakan *hidden layer* 2, 3, dan 4. *Hidden layer* 2 digunakan karena dalam buku [15], *hidden layer* 2 sudah termaksud *Deep Neural Network*. Hidden layer 3 digunakan karena dalam Penelitian [29], model mendapatkan MSE sebesar 0.2596. Hasil nilai *error* yang kecil dalam Penelitian [29] tentu sudah sangat baik. Hidden layer 4 digunakan karena pada Penelitian [8], model mendapatkan akurasi 83.48%. Untuk hasil pengujian dapat dilihat pada Tabel 4.15 di bawah ini.

Tabel 4.15 Hasil pengujian *Deep Neural Network*

No.	***Learning Rate***	***Hidden Layer***	***Epoch***	***Activation Function***	***ROC***	***Akurasi***
1.	0.01	2	10	tanh	0.5168	0.6125
2.	0.01	2	10	relu	0.5177	0.7292
3.	0.01	2	50	tanh	0.5225	0.7214
4.	0.01	2	50	relu	0.5183	0.6945
5.	0.01	2	100	tanh	0.5193	0.7270
6.	0.01	2	100	relu	0.5156	0.5983
7.	0.01	3	10	tanh	0.5186	0.7506
8.	0.01	3	10	relu	0.5206	0.7524
9.	0.01	3	50	tanh	0	0.9789
10.	0.01	3	50	relu	0.5192	0.7219
11.	0.01	3	100	tanh	**0.5301**	**0.8450**
12.	0.01	3	100	relu	nan	0.0211
13.	0.01	4	10	tanh	**0.5303**	**0.8130**
14.	0.01	4	10	relu	**0.5266**	**0.8124**
15.	0.01	4	50	tanh	nan	0.0211
16.	0.01	4	50	relu	nan	0.9789
17.	0.01	4	100	tanh	nan	0.0211
18.	0.01	4	100	relu	0.5204	**0.7981**
19.	0.1	2	10	tanh	nan	0.0211
20.	0.1	2	10	relu	nan	0.0211
21.	0.1	2	50	tanh	nan	0.0211

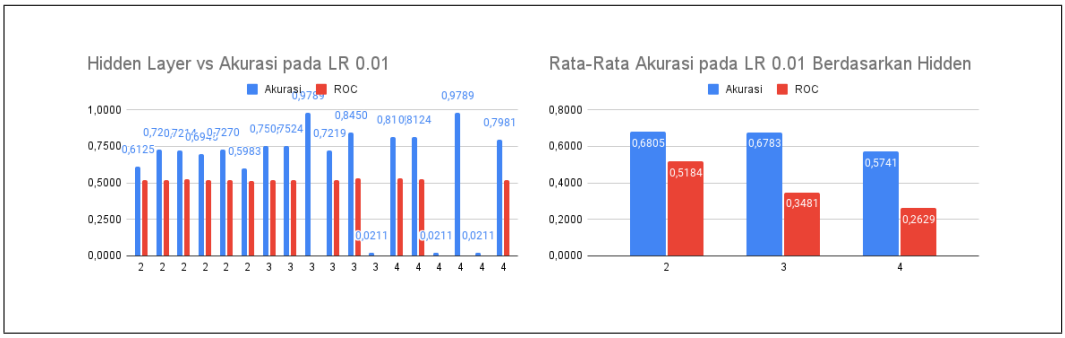
22.	0.1	2	50	relu	nan	0.0211
23.	0.1	2	100	tanh	nan	0.0211
24.	0.1	2	100	relu	nan	0.0211
25.	0.1	3	10	tanh	nan	0.0211
26.	0.1	3	10	relu	nan	0.0211
27.	0.1	3	50	tanh	nan	0.0211
28.	0.1	3	50	relu	nan	0.0211
29.	0.1	3	100	tanh	nan	0.9789
30.	0.1	3	100	relu	nan	0.0211
31.	0.1	4	10	tanh	nan	0.0211
32.	0.1	4	10	relu	nan	0.9789
33.	0.1	4	50	tanh	nan	0.0211
34.	0.1	4	50	relu	nan	0.0211
35.	0.1	4	100	tanh	nan	0.0211
36.	0.1	4	100	relu	nan	0.0211

Pada pengujian menggunakan metode *Deep Neural Network*, yang ditambah *cost-sensitive learning*, hasil menunjukkan bahwa pada saat *learning rate* 0.01 akurasi paling tinggi adalah 97.89%, namun akurasi 97.89% merupakan hasil model yang *overfitting*. Akurasi model yang tinggi namun tidak mengalami overfitting adalah kombinasi ke-11, yaitu dengan akurasi 84.50%, sedangkan akurasi paling rendah 2.11% diraih oleh 3 kombinasi. Hal ini dikarenakan *epoch* yang terlalu besar pada *hidden layer* yang tergolong cukup besar, sehingga model tidak bisa melakukan prediksi dengan benar. Pada *learning rate* 0.1, model tidak dapat melakukan prediksi dengan baik. Hal ini dikarenakan *learning rate* yang terlalu besar yang menyebabkan model tidak dapat konvergen. Rangkuman hasil pengujian dapat dilihat dalam bentuk grafik di bawah ini.



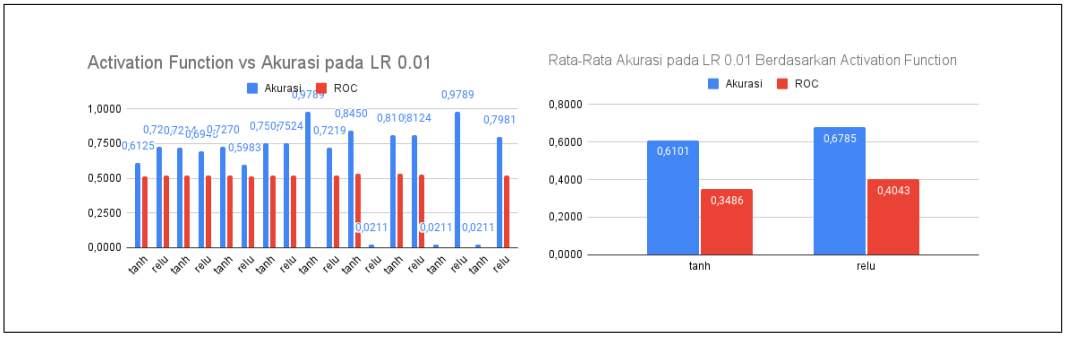
Gambar 4.15 Akurasi dan ROC dari pengujian tambahan berdasarkan jumlah *epoch*

Pada Gambar 4.15, dapat disimpulkan bahwa dalam kasus ini, rata-rata akurasi tertinggi didapat dengan *epoch* sebesar 10. Hal ini terjadi karena terdapat 2 kombinasi model dengan *epoch* 10 yang memiliki akurasi tertinggi. Semakin besar *epoch*, maka akurasi semakin menurun, namun berbeda halnya dengan ROC, yaitu rata-rata ROC tertinggi terdapat pada *epoch* 10 dan rata-rata ROC terendah terdapat pada *epoch* 50. Oleh karena itu, tidak ada pola khusus dari *epoch* yang menentukan kapan ROC akan tinggi dan rendah.



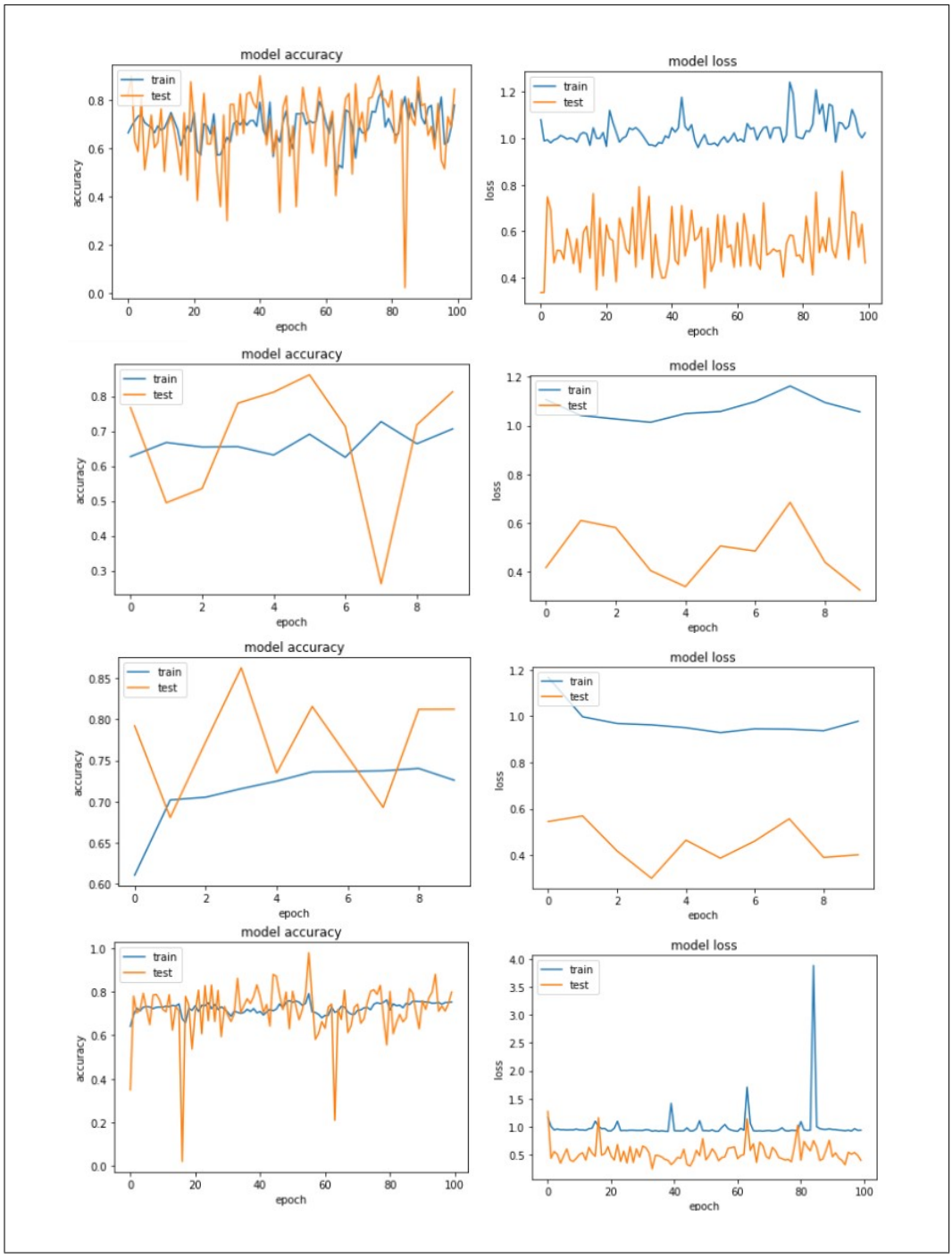
Gambar 4.16 Akurasi dan ROC dari pengujian tambahan berdasarkan jumlah *hidden layer*

Pada Gambar 4.16, dapat disimpulkan bahwa dalam kasus ini, rata-rata akurasi tertinggi didapat dengan *hidden layer* 2. Hal ini terjadi dikarenakan, walaupun *hidden layer* 4 memiliki 3 kombinasi model terbaik, namun dalam *hidden layer* 4 juga terdapat model yang *overfitting* dan *underfitting*, sementara dalam pengujian *hidden layer* 2, walaupun akurasi tidak terlalu tinggi, namun cenderung stabil. Mengenai ROC, dapat disimpulkan bahwa semakin tinggi *hidden layer*, maka ROC akan semakin rendah.



Gambar 4.17 Akurasi dan ROC dari pengujian tambahan berdasarkan jumlah *learning rate*

Pada Gambar 4.17, dapat disimpulkan bahwa dalam kasus ini, rata-rata akurasi dan ROC tertinggi didapat dengan menggunakan *activation function* relu. Namun, jika dilihat dari grafik sebelah kiri, baik tanh maupun relu, pernah mengalami *overfitting* dan *underfitting* pada model. Oleh sebab itu, belum dapat ditarik kesimpulan *activation function* mana yang lebih cocok dalam kasus prediksi penyakit stroke.



Gambar 4.18 Perbandingan *Learning curve* untuk setiap percobaan

Pada Gambar 4.18 dapat dilihat bahwa dari 4 kombinasi terbaik, terdapat 2 model yang cenderung mengalami overfitting, yaitu kombinasi dari gambar kedua (*learning rate* 0.01, *hidden layer* 4, *epoch* 10, dan *activation function* tanh) dan kombinasi dari gambar ketiga (*learning rate* 0.01, *hidden layer* 4, *epoch* 10, dan *activation function* relu). Oleh karena itu, dapat disimpulkan bahwa *hidden layer* 4 masih terlalu besar untuk kasus prediksi penyakit stroke.

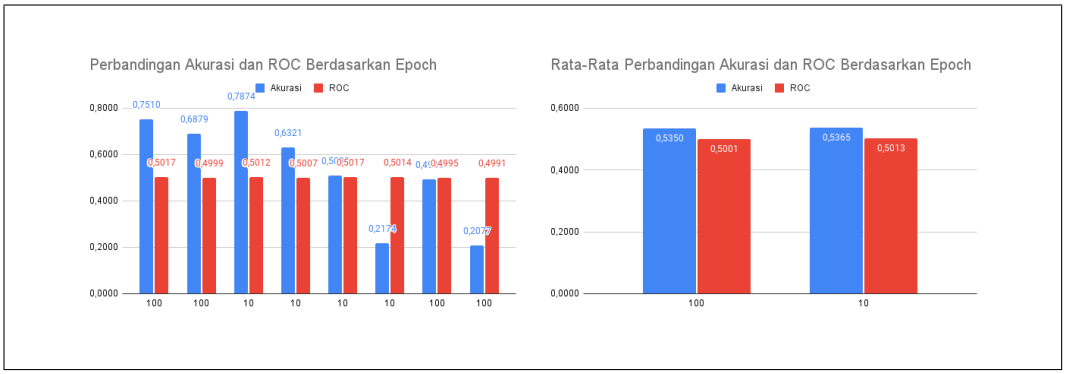
4.4.5.2 Pengujian Deep Neural Network dan Dropout

Pengujian *Deep Neural Network* dengan *dropout* akan menggunakan 4 hasil kombinasi terbaik pada pengujian sebelumnya. Hasil pengujian dapat dilihat pada tabel di bawah ini.

Tabel 4.16 Hasil pengujian *Deep Neural Network* dengan *Dropout*

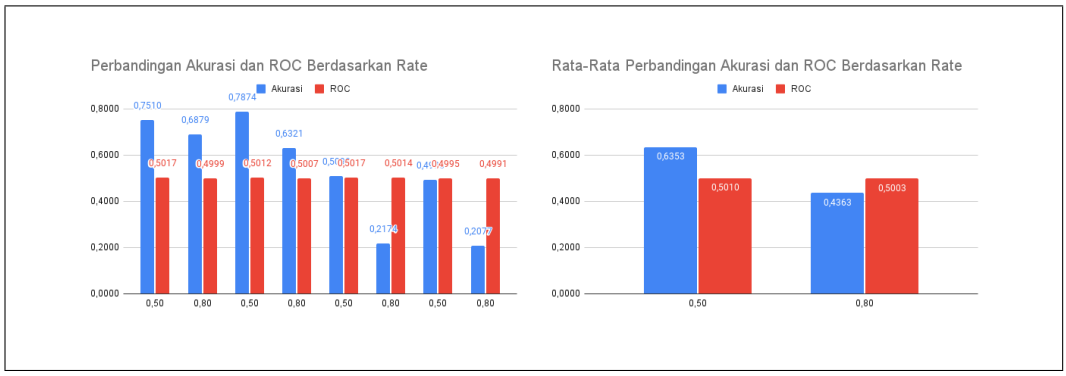
No.	*Learning Rate*	*Hidden Layer*	*Epoch*	*Activation Function*	*Rate*	*ROC*	*Akurasi*
1.	0.01	3	100	tanh	0.5	**0.5017**	**0.7510**
2.	0.01	3	100	tanh	0.8	0.4999	**0.6879**
3.	0.01	4	10	tanh	0.5	**0.5012**	**0.7874**
4.	0.01	4	10	tanh	0.8	0.5007	0.6321
5.	0.01	4	10	relu	0.5	**0.5017**	0.5090
6.	0.01	4	10	relu	0.8	0.5014	0.2174
7.	0.01	4	100	relu	0.5	0.4995	0.4986
8.	0.01	4	100	relu	0.8	0.4991	0.2077

Pada pengujian menggunakan metode *Deep Neural Network* dengan *dropout*, yang ditambah *cost-sensitive learning* dan *probability tuning*, hasil menunjukkan bahwa akurasi paling tinggi adalah 78.74% dan paling rendah adalah 20.77%. Untuk akurasi 3 model tertinggi diraih oleh *activation function* tanh, dan semua model yang menggunakan *activation function* relu memiliki hasil akurasi yang buruk, terutama saat menggunakan *rate* 0.8. Dapat disimpulkan, dalam kasus prediksi penyakit stroke dan untuk penggunaan hidden layer yang kecil, *activation function* relu tidak cocok digunakan.



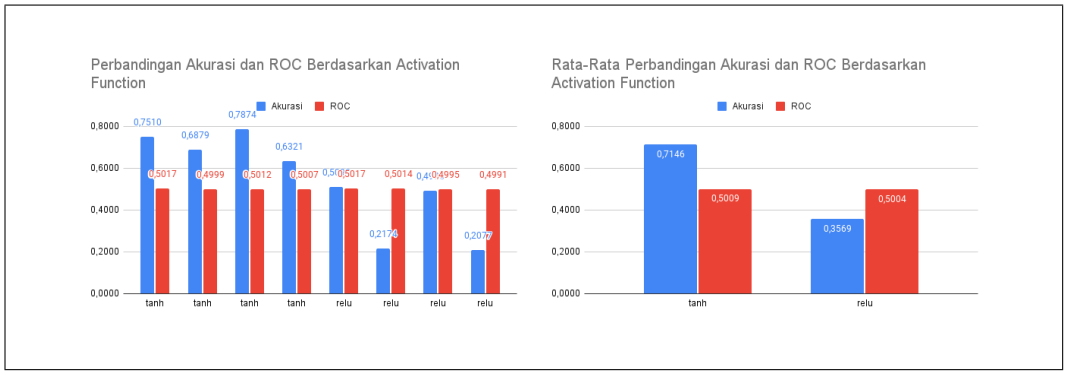
Gambar 4.19 Perbandingan akurasi dan ROC berdasarkan jumlah *epoch*

Pada Gambar 4.19 dapat dilihat bahwa rata-rata perbandingan akurasi dan ROC lebih tinggi *epoch* 10, namun tidak memiliki perbedaan yang cukup signifikan. Hal ini dikarenakan baik epoch 10 maupun epoch 100, mendapat akurasi yang tinggi dan rendah, sehingga tidak ada pola khusus dari *epoch* untuk menentukan *epoch* mana yang paling cocok untuk prediksi penyakit stroke.



Gambar 4.20 Perbandingan akurasi dan ROC berdasarkan jumlah *rate*

Pada Gambar 4.20 dapat dilihat bahwa rata-rata perbandingan akurasi dan ROC berdasarkan *rate* memiliki perbedaan akurasi yang cukup signifikan. Pada saat *rate* 0.5, model mendapatkan akurasi sebesar 63.53% dan ROC hanya sebesar 50.10%, sedangkan saat *rate* diperbesar menjadi 0.8, akurasi turun menjadi 43.63%, dan ROC tidak mengalami perubahan yang berarti. Dari grafik ini dapat diambil kesimpulan bahwa semakin tinggi *rate*, maka akurasi akan semakin rendah dikarenakan semakin banyak *neuron* yang dimatikan sehingga model tidak dapat melakukan prediksi dengan baik.



Gambar 4.21 Perbandingan akurasi dan ROC berdasarkan jumlah *rate*

Gambar 4.21 menjelaskan bahwa rata-rata perbandingan akurasi dan ROC berdasarkan *activation function* memiliki perbedaan yagng signifikan, terutama pada akurasi. Pada saat model menggunakan activation function tanh, rata-rata akurasi cukup baik, yaitu 71.46% dan ROC 50.09%, namun pada saat model

menggunakaan *activation function* relu, akurasi model turun drastis menjadi 35.69% dan ROC tidak mengalami perubahan berarti, yaitu 50.04%. Dari grafik ini dapat disimpulkan bahwa *activation function* relu tidak cocok digunakan dalam kasus prediksi penyakit stroke yang menggunakan *hidden layer* kecil.

4.4.6 Pembahasan Pengujian

Dari semua hasil pengujian yang telah dilakukan, sangat sulit untuk mendapatkan akurasi sampai 90% yang tidak mengalami *overfitting*, dikarenakan penelitian ini menggunakan dataset penyakit stroke, yang mana penyakit stroke disebabkan dari banyak faktor, yang memiliki kemungkinan bahwa orang tersebut sudah menderita stroke di luar faktor yang tercatat di dataset. Oleh karena itu, sangat sulit jika hanya melakukan prediksi dari melihat 1 dataset karena tidak cukup fitur untuk melakukan prediksi.

Tabel 4.17 di bawah ini merangkum 7 hasil terbaik dari keseluruhan pengujian, yang terdiri dari 4 pengujian algoritme *Deep Neural Network* dan 3 pengujian *Deep Neural Network* dengan *dropout*.

Tabel 4.17 Hasil pengujian *Deep Neural Network* dengan *Dropout*

No.	***Metode***	***Learning Rate***	***Hidden Layer***	***Epoch***	***Activation Function***	***Rate***	***ROC***	***Akurasi***
1.	DNN	0.01	3	100	tanh	-	0.5301	**0.8450**
2.	DNN	0.01	4	10	tanh	-	**0.5303**	0.8130
3.	DNN	0.01	4	10	relu	-	0.5266	0.8124
4.	DNN	0.01	4	100	relu	-	0.5204	0.7981
5.	DNN+dropout	0.01	3	100	tanh	0.5	0.5017	0.7510
6.	DNN+dropout	0.01	3	100	tanh	0.8	0.4999	0.6879
7.	DNN+dropout	0.01	4	10	tanh	0.5	0.5012	0.7874

Berdasarkan Tabel 4.17, dapat disimpulkan bahwa *hidden layer* yang terlalu besar dapat memuat hasil prediksi buruk. Hal ini dapat diakibatkan oleh 2 faktor, seperti faktor teknis yang disebutkan dalam penelitian [30], bahwa overfitting dapat disebabkan oleh jumlah *hidden layer* yang terlalu besar dan faktor keterbatasan *hardware* yang tidak mampu melakukan proses perhitungan angka angka yang terlampau besar. Prediksi stroke dengan dataset ini juga tidak cocok menggunakan *learning rate* 0.1. Hal ini dikarenakan *learning rate* 0.1 terlalu besar sehingga model tidak dapat konvergen.

Pengujian model menggunakan teknik *feature selection information gain* pada DNN tidak mempengaruhi hasil, sedangkan pada DNN dengan *dropout* hampir tidak berpengaruh. Hal ini dapat diakibatkan oleh jumlah hidden layer sebelumnya yang masih sangat besar, sedangkan information gain berguna untuk mengurangi fitur yang kurang relevan pada prediksi model, sehingga titik permasalahnnya bukan pada jumlah fitur namun pada jumlah hidden layer yang terlalu besar.

Pengujian model menggunakan *activation function* tanh dinilai cocok untuk kasus yang memiliki *hidden layer* kecil, sedangkan *activation function* relu dinilai cocok untuk kasus yang memiliki *hidden layer* besar. Namun, jika menggunakan *dropout*, *activation function* relu menjadi tidak cocok digunakan.

Hasil terbaik model *Deep Neural Network* diraih oleh model yang menggunakan *hidden layer* berjumlah 4, sedangkan hasil terbaik model *Deep Neural Network* dengan *dropout* diraih oleh model yang menggunakan *hidden layer* berjumlah 3. Hal ini dapat diakibatkan sifat *dropout* yang mematikan *neuron* sehingga *neuron* yang hidup akan berkurang.

BAB 5 KESIMPULAN DAN SARAN

Pada bab ini dibahas mengenai kesimpulan penelitian dan pengujian. Selain itu, terdapat juga saran yang dapat dipertimbangkan untuk penelitian di masa mendatang.

5.1 Kesimpulan

Kesimpulan dari metode *Deep Neural Network*, *cost sensitive learning*, *probability tuning*, dan *dropout* pada prediksi penyakit stroke adalah:

1. Kombinasi *hyperparameter* terbaik dalam penelitian ini adalah *learning rate* 0.01, *hidden layer* 3, *epoch* 100, dan *activation function* tanh.
2. *Hidden layer* yang terlalu besar dapat membuat hasil prediksi menjadi *overfitting*.
3. Metode cost sensitive learning dan probability tuning serta penambahan metode *feature selection information gain* tidak terlalu membantu menyelesaikan masalah *overfitting*.
4. Hasil akurasi dan ROC model dinilai lebih baik tanpa menggunakan *dropout*.
5. Sangat sulit untuk mendapatkan akurasi lebih dari 90% yang tidak mengalami overfitting, dikarenakan penyakit stroke dapat disebabkan dari berbagai faktor yang mungkin saja tidak tercatat di dataset yang dipakai.

5.2 Saran

Saran dari penulis untuk pengembangan model prediksi penyakit stroke di masa mendatang adalah:

1. Menggunakan beberapa dataset dengan fitur yang berbeda-beda, sehingga semakin banyak faktor penyebab stroke yang didapatkan oleh model.
2. Menggunakan metode *handling imbalance class* selain *cost sensitive learning* dan *probability tuning* yang dapat memberikan dampak lebih signifikan.
3. Pada metode *dropout*, dapat menentukan nilai *rate* yang berbeda agar dapat memberikan hasil yang lebih optimal.
4. Menggunakan RAM yang lebih besar atau Google Colab Pro untuk mengatasi keterbatasan *hardware*.

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] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, 2018.

[18] 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].

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

[20] 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].

[21] 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.

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

[23] 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].

[24] S. Dev, H. Wang, C.S. Nwosu, N. Jain, B. Veeravalli, and D. John, "A

Predictive Analytics Approach For Stroke Prediction Using Machine Learning and Neural Networks", *Healthcare Analytics*, 2022.

[25] 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.

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

[27] A. E. Kitabchi, G. E. Umpierrez, J. M. Miles, J. N. Fisher, "Hyperglycemic Crises in Adult Patients With Diabetes," *Diabetes Care*, vol. 32 no. 7, pp 1335-1343, 2019.

[28] 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].

[29] P. Chantamit-o-pas and M. Goyal, "Prediction of Stroke Using Deep Learning Model," *International Conference on Neural Information Processing*, pp. 774–781, 2017.

[30] M. Uzair and N. Jamil, "Effects of Hidden Layers on the Efficiency of Neural Networks," *2020 IEEE 23rd International Multitopic Conference (INMIC)*, pp. 1–6, 2020.