HYPERPARAMETER OPTIMIZATION PADA ARSITEKTUR CNN UNTUK DETEKSI KELAINAN DETAK JANTUNG

TUGAS AKHIR

KEVIN CHENG

NIM : 312210013

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI DAN DESAIN
UNIVERSITAS MA CHUNG MALANG

2025

LEMBAR PENGESAHAN

TUGAS AKHIR

HYPERPARAMETER OPTIMIZATION PADA ARSITEKTUR CNN UNTUK DETEKSI KELAINAN DETAK JANTUNG

Oleh:

KEVIN CHENG

NIM : 312210013

dari:

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI DAN DESAIN

UNIVERSITAS MA CHUNG

Telah dinyatakan lulus dalam melaksanakan Tugas Akhir sebagai syarat kelulusan dan berhak mendapatkan gelar Sarjana Komputer

Dosen Pembimbing I,	Dosen Pembimbing II,
	
Windra Swastika, S.Kom., MT., Ph.D.	**Hendry Setiawan, ST., M.Kom**
NIP. 20070039	**NIP. 20100006**

Dekan Fakultas Teknologi dan Desain,

Prof. Dr.Eng. Romy Budhi, ST., MT., M.Pd.

NIP. 20070035

PERNYATAAN KEASLIAN SKRIPSI

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Skripsi saya dengan "**HYPERPARAMETER OPTIMIZATION PADA ARSITEKTUR CNN UNTUK DETEKSI KELAINAN DETAK JANTUNG**" adalah benar benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diizinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka. Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Malang, 9 Januari 2026

Kevin Cheng

NIM. 312210013

HYPERPARAMETER OPTIMIZATION PADA ARSITEKTUR CNN UNTUK DETEKSI KELAINAN DETAK JANTUNG

Kevin Cheng

Universitas Ma Chung

Abstrak

Penyakit kardiovaskular (CVD) merupakan penyebab kematian terbesar di dunia, sehingga deteksi dini melalui analisis suara jantung (*phonocardiogram*/PCG) menjadi sangat penting. *Convolutional Neural Network* (CNN) telah terbukti efektif dalam klasifikasi PCG, namun performanya sangat bergantung pada konfigurasi *hyperparameter* yang seringkali ditentukan secara manual (*trial and error*). Penelitian ini bertujuan untuk mengoptimalkan kinerja arsitektur CNN dalam mendeteksi kelainan detak jantung dengan membandingkan empat metode *Hyperparameter Optimization* (HPO): *Grid Search, Random Search, Bayesian Optimization,* dan *Genetic Algorithm*. Menggunakan dataset *PhysioNet/CinC Challenge 2016*, penelitian ini mengevaluasi dampak optimasi terhadap *F1-Score* dan efisiensi komputasi. Hasil eksperimen menunjukkan bahwa secara nominal, Random Search mencatatkan performa tertinggi dengan F1-Score Macro sebesar 0,889 pada data uji. Meskipun uji statistik *One-Way ANOVA* menunjukkan tidak ada perbedaan kinerja yang signifikan secara statistik di antara keempat metode (p=0,442), Genetic Algorithm terbukti sebagai metode paling unggul secara menyeluruh karena efisiensi komputasinya yang superior, mampu mencapai konvergensi optimal hanya dalam waktu 2,5 jam dibandingkan *Grid Search* yang membutuhkan 23,8 jam. Model hasil optimasi ini juga menunjukkan ketahanan (*robustness*) tinggi dalam menangani ketidakseimbangan data dengan capaian nilai AUC sebesar 0,97 untuk deteksi kelas *unhealthy*.

Kata Kunci : *Convolutional Neural Network, Genetic Algorithm, Hyperparameter Optimization, Phonocardiogram*

HYPERPARAMETER OPTIMIZATION PADA ARSITEKTUR CNN UNTUK DETEKSI KELAINAN DETAK JANTUNG

Kevin Cheng

Universitas Ma Chung

Abstract

Cardiovascular disease (CVD) is the leading cause of death globally, making early detection through heart sound analysis (phonocardiogram/PCG)[1]. While Convolutional Neural Networks (CNN) have proven effective for PCG classification, their performance heavily depends on hyperparameter configurations that are often determined manually through trial and error. This study aims to optimize CNN architecture performance for detecting heartbeat abnormalities by comparing four Hyperparameter Optimization (HPO) methods: Grid Search, Random Search, Bayesian Optimization, and Genetic Algorithm. Utilizing the PhysioNet/CinC Challenge 2016 dataset, this research evaluates the impact of optimization on F1-Score and computational efficiency. Experimental results indicate that nominally, Random Search recorded the highest performance with a Macro F1-Score of 0.889 on the test set[5]. Although One-Way ANOVA statistical testing showed no statistically significant performance difference among the four methods (p=0.442), the Genetic Algorithm proved to be the most superior method overall due to its superior computational efficiency, reaching optimal convergence in just 2.5 hours compared to Grid Search which required 23.8 hours. These optimized models also demonstrated high robustness in handling data imbalance, achieving an AUC of 0.97 for the unhealthy class detection.

Key Word : *Convolutional Neural Network, Genetic Algorithm, Hyperparameter Optimization, Phonocardiogram*

KATA PENGANTAR

Puji dan syukur ke hadirat Tuhan Yang Maha Esa atas rahmat dan karunianya, sehingga penyusunan laporan tugas akhir yang berjudul "HYPERPARAMETER OPTIMIZATION PADA ARSITEKTUR CNN UNTUK DETEKSI KELAINAN DETAK JANTUNG" dapat terselesaikan dengan baik.

Dalam penyusunan tugas akhir penulis telah mendapatkan banyak bantuan dari berbagai pihak. Pada kesempatan ini penulis mengucapkan terima kasih kepada:

1. Tuhan Yang Maha Esa atas berkat dan rahmat-Nya tugas akhir dapat diselesaikan dengan baik.
2. Kedua orangtua dan keluarga lainnya yang memberikan dukungan berupa doa dan restu selama melaksanakan dan pengerjaan tugas akhir.
3. Bapak Prof. Dr.Eng. Romy Budhi, ST., MT., M.Pd. selaku Dekan Fakultas Teknologi dan Desain.
4. Bapak Hendry Setiawan, ST., M.Kom selaku Kepala Program Studi Teknik Informatika.
5. Windra Swastika, S.Kom., MT., Ph.D. selaku dosen pembimbing I selama penyelesaian tugas akhir yang telah meluangkan waktu, memberikan arahan, bimbingan, dan motivasi yang tak ternilai.
6. Bapak/Ibu dosen Program Studi Teknik Informatika yang telah memberikan ilmu selama masa studi saya di Universitas Ma Chung.
7. Rekan-Rekan Mahasiswa Tekik Informatika Univeristas Ma Chung Angkatan 2022.

Adapun dalam penyusunan laporan ini penulis menyadari kekurangan dalam laporan tugas akhir ini dan menyambut baik kritik serta saran yang membangun. Besar harapan penulis agar laporan ini memberikan ilmu dan manfaat bagi semua .

Malang, 9 Januari 2026

Kevin Cheng

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

Jantung adalah organ vital yang bekerja tanpa henti sepanjang hidup, memompa darah 60–100 kali per menit untuk menjaga distribusi oksigen dan nutrisi ke seluruh tubuh. Gangguan pada fungsi jantung dapat berdampak serius pada organ lain dan mengancam nyawa apabila tidak ditangani secara cepat dan tepat.Penyakit kardiovaskular (*cardiovascular disease*/*CVD*) adalah penyebab kematian terbesar di dunia. Menurut data terbaru dari *Global Burden of Disease Study*, *CVD* menyebabkan sekitar 19,05 juta kematian pada tahun 2020, merepresentasikan 32% dari total kematian global (Roth et al., 2020). Di Indonesia, prevalensi penyakit jantung menunjukkan tren yang mengkhawatirkan. Data Survei Kesehatan Indonesia 2023 mencatat prevalensi penyakit jantung sebesar 0,85% (*GoodStats*, 2023), sementara laporan WHO (2021) menunjukkan bahwa *CVD* berkontribusi terhadap 37% kematian di Indonesia.

Banyak penderita tidak mendeteksi kelainan jantung pada tahap awal karena tidak merasakan gejala umum seperti nyeri dada, sesak napas, atau keringat dingin. Padahal, deteksi dini sangat penting untuk meningkatkan prognosis pasien dan mencegah komplikasi serius (Lloyd-Jones et al., 2022). Metode konvensional seperti auskultasi manual menggunakan stetoskop sangat bergantung pada keahlian dan pengalaman tenaga medis, sehingga rentan terhadap variabilitas diagnostik dan dapat menghasilkan hasil yang tidak konsisten, terutama pada fasilitas kesehatan dengan keterbatasan sumber daya (Nishimura et al., 2021).

Perkembangan teknologi *artificial intelligence* (*AI*) dan *deep learning* memberikan peluang besar dalam deteksi dini kelainan jantung melalui analisis suara jantung (*phonocardiogram*/*PCG*). Suara jantung mengandung informasi diagnostik penting yang dapat dimanfaatkan untuk mendeteksi *murmur*, kelainan katup, dan gangguan irama (Deng & Bentley, 2021).Beberapa penelitian terkini telah menunjukkan potensi analisis audio detak jantung sebagai metode *screening* awal yang efektif dan *cost-effective*. Gharehbaghi et al. (2021)

mengimplementasikan sistem *screening* berbasis *PCG* di *setting* komunitas untuk identifikasi dini penyakit kardiovaskular, menunjukkan *cost-effectiveness* sebagai alternatif *echocardiography* untuk *screening* populasi besar.

Convolutional Neural Network (*CNN*) terbukti efektif untuk menganalisis data audio dan sinyal biomedis dalam penelitian-penelitian terkini. Zhang et al. (2022) mengembangkan pendekatan berbasis *scaled spectrogram* dan *partial transfer learning* untuk klasifikasi suara jantung, mencapai akurasi 98,2%. Humayun et al. (2020) juga membuktikan bahwa kombinasi *CNN* dengan fitur *Mel-Frequency Cepstral Coefficients* (*MFCC*) dapat meningkatkan performa klasifikasi dengan akurasi mencapai 93,5%.

Meskipun arsitektur *CNN* telah banyak digunakan dalam klasifikasi suara jantung dan menunjukkan hasil yang menjanjikan sebagai alat *screening*, performa model sangat bergantung pada pemilihan *hyperparameter* yang tepat. *Hyperparameter* seperti *learning rate*, *batch size*, jumlah *layer*, jumlah *kernel*, *dropout rate*, dan *optimizer* memiliki pengaruh signifikan terhadap akurasi, kecepatan konvergensi, dan kemampuan generalisasi model (Yang & Shami, 2020). Dalam praktiknya, pemilihan *hyperparameter* selama ini sering dilakukan secara manual melalui pendekatan *trial and error*, di mana peneliti melakukan eksperimen berulang kali dengan mencoba berbagai kombinasi nilai *hyperparameter* hingga memperoleh hasil evaluasi yang memuaskan. Tanpa optimasi *hyperparameter* yang sistematis, model dapat mengalami *underfitting* atau *overfitting* yang mengurangi efektivitas sistem deteksi (Goodfellow et al., 2016). Baghel et al. (2020) menunjukkan bahwa optimasi *batch size* dan *dropout rate* berkontribusi signifikan terhadap peningkatan *F1-score* dalam deteksi kelainan jantung, mengindikasikan pentingnya konfigurasi *hyperparameter* yang tepat.

Berbagai metode optimasi *hyperparameter* telah dikembangkan untuk mengatasi keterbatasan pendekatan manual. Metode konvensional seperti *grid search* dan *random search* memberikan pendekatan yang sistematis namun sering kali memerlukan *computational cost* yang tinggi (Probst et al., 2020). Metode yang lebih canggih seperti *Bayesian optimization*, *genetic algorithm*, dan *particle swarm*

optimization menawarkan efisiensi yang lebih tinggi dalam mengeksplorasi ruang pencarian *hyperparameter* (Paleyes et al., 2021).

Meskipun berbagai metode optimasi *hyperparameter* telah tersedia, belum ada penelitian komprehensif yang secara sistematis membandingkan efektivitas dan efisiensi berbagai metode tersebut dalam konteks klasifikasi detak jantung menggunakan arsitektur *CNN* dengan fitur *MFCC*. Sebagian besar penelitian terdahulu fokus pada pengembangan arsitektur model atau teknik ekstraksi fitur, namun kurang memberikan perhatian pada aspek optimasi *hyperparameter* yang sebenarnya memiliki kontribusi signifikan terhadap performa akhir sistem (Waring et al., 2020).

1.2 Identifikasi Masalah

Berdasarkan latar belakang yang telah dipaparkan, dapat diidentifikasi beberapa masalah utama yang menjadi dasar penelitian ini, yaitu:

- Meskipun telah ada beberapa penelitian mengenai sistem berbasis *deep learning* untuk analisis audio detak jantung sebagai metode *screening* awal, performa model CNN sangat bergantung pada pemilihan *hyperparameter* yang tepat, namun proses optimasi *hyperparameter* sering kali dilakukan secara manual atau *trial-and-error* yang tidak efisien.

1.3 Batasan Masalah

Untuk membatasi ruang lingkup penelitian agar lebih fokus dan terarah, maka ditetapkan batasan masalah sebagai berikut:

1. Dataset yang digunakan adalah *PhysioNet/CinC Challenge 2016* yang berisi rekaman audio detak jantung dengan klasifikasi normal dan abnormal.
2. Metode *deep learning* yang digunakan terbatas pada arsitektur *Convolutional Neural Network (CNN) 2D* untuk pemrosesan sinyal audio.

3. *Preprocessing* audio meliputi *resampling*, normalisasi, dan ekstraksi fitur MFCC (*Mel-Frequency Cepstral Coefficients*).
4. *Hyperparameter* yang dioptimasi meliputi *learning rate, batch size,* jumlah kernel per layer, *kernel size*, *dense unit*, *dropout rate,* dan *optimizer*
5. Metode optimasi *hyperparameter* yang dibandingkan mencakup *grid search, random search, Bayesian optimization,* dan *Genetic Algorithm*
6. Evaluasi performa model menggunakan metrik *recall, F1-score*, dan *computational efficiency* (waktu training dan inference).
7. Implementasi sistem dilakukan menggunakan bahasa pemrograman *Python* dengan framework *TensorFlow/Keras* dan *library* optimasi seperti *Keras Tuner*

1.4 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, maka rumusan masalah dalam penelitian ini adalah:

1. Bagaimana pengaruh optimasi *hyperparameter* terhadap performa model CNN dalam klasifikasi detak jantung?
2. Metode optimasi *hyperparameter* manakah yang paling efektif untuk meningkatkan *F1-Score* dan *recall* model CNN dalam mendeteksi kelainan jantung?

1.5 Tujuan Penelitian

Berdasarkan rumusan masalah yang telah ditetapkan, maka tujuan penelitian ini adalah:

1. Menganalisis pengaruh optimasi *hyperparameter* terhadap performa model CNN dalam klasifikasi detak jantung.

2. Mengidentifikasi metode optimasi *hyperparameter* yang paling efektif dalam meningkatkan *F1-Score* dan *recall* model CNN untuk deteksi kelainan jantung.

1.6 Manfaat Penelitian

Manfaat bagi Universitas

- Menjadi referensi akademik bagi penelitian lanjutan yang berhubungan dengan klasifikasi sinyal jantung, optimasi *hyperparameter*, maupun aplikasi *machine learning* di bidang kesehatan.
- Meningkatkan reputasi universitas dalam pengembangan teknologi berbasis AI yang relevan dengan kebutuhan dunia medis dan kesehatan Masyarakat.
- Memberikan peluang kolaborasi riset dengan institusi kesehatan atau rumah sakit terkait penerapan hasil penelitian dalam sistem deteksi dini kelainan jantung.
- Memperkaya portfolio penelitian universitas dalam domain *deep learning optimization* dan medical AI.

Manfaat bagi Mahasiswa

- Memberikan pengalaman praktis dalam merancang, mengimplementasikan, dan mengevaluasi sistem klasifikasi berbasis *deep learning* dengan pendekatan optimasi yang sistematis.
- Membekali mahasiswa dengan pemahaman mendalam tentang pengaruh *hyperparameter* terhadap performa model dan teknik-teknik optimasi modern dalam *deep learning*.
- Memberikan keterampilan analisis data dan riset terapan yang dapat diimplementasikan di dunia kerja, khususnya dalam bidang *machine learning engineering*.

1.7 Luaran

Penelitian ini diharapkan menghasilkan beberapa luaran sebagai berikut:

1. Model *deep learning* optimal berbasis arsitektur CNN dengan konfigurasi hyperparameter yang telah dioptimasi secara sistematis untuk klasifikasi detak jantung normal dan abnormal.
2. Artikel ilmiah yang membahas metode, eksperimen, serta hasil penelitian, sehingga dapat dipublikasikan pada forum akademik nasional maupun internasional.
3. Dokumentasi teknis berupa laporan tugas akhir yang dapat dijadikan referensi bagi penelitian dan pengembangan selanjutnya.

BAB II
TINJAUAN PUSTAKA

2.1 Penyakit Kardiovaskular

Penyakit kardiovaskular (*cardiovascular disease*/*CVD*) merupakan kelompok penyakit yang melibatkan jantung dan pembuluh darah, termasuk penyakit jantung koroner, penyakit serebrovaskular, penyakit jantung rematik, dan kondisi lainnya (World Health Organization, 2021). *CVD* menjadi penyebab utama kematian secara global. Menurut data terbaru dari *Global Burden of Disease Study* 2019, *CVD* menyebabkan sekitar 19,05 juta kematian pada tahun 2020, merepresentasikan 32% dari seluruh kematian di dunia (Roth et al., 2020).

Kelainan jantung dapat bermanifestasi dalam berbagai bentuk, termasuk kelainan katup jantung, penyakit jantung kongenital, kardiomiopati, dan aritmia (Virani et al., 2020). Deteksi dini kelainan jantung sangat krusial karena dapat mencegah progresivitas penyakit, mengurangi risiko komplikasi, dan meningkatkan kualitas hidup pasien (Lloyd-Jones et al., 2022). Penelitian terkini menunjukkan bahwa *early detection* dan *appropriate management* dapat mengurangi mortalitas *CVD* hingga 30–40% (Mensah et al., 2023).

2.2 Phonocardiogram (PCG)

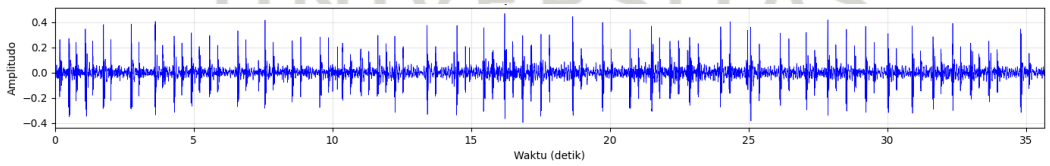
Phonocardiogram (*PCG*) adalah rekaman digital dari suara jantung yang dihasilkan oleh aktivitas mekanik jantung, termasuk pembukaan dan penutupan katup jantung, aliran darah *turbulent*, dan getaran dinding jantung (Messner et al., 2023). *PCG* merepresentasikan vibrasi yang dihasilkan oleh jantung dalam bentuk sinyal audio yang dapat dianalisis secara kuantitatif untuk keperluan diagnostik dan *screening*.

Suara jantung normal terdiri dari dua komponen utama: S1 dan S2. S1 (*first heart sound*) dihasilkan oleh penutupan katup atrioventrikular (*mitral* dan *trikuspid*) pada awal sistol ventrikel, sedangkan S2 (*second heart sound*)

dihasilkan oleh penutupan katup *semilunar* (*aorta* dan *pulmonal*) pada akhir sistol ventrikel (Nishimura et al., 2021). Pada kondisi patologis, dapat muncul suara tambahan seperti S3, S4, *murmur*, *clicks*, atau *rubs* yang mengindikasikan kelainan jantung spesifik seperti *heart failure*, *valvular disease*, atau *pericardial disease* (Otto et al., 2020).

Murmur jantung merupakan suara tambahan yang dihasilkan oleh aliran darah *turbulent* melalui struktur jantung. *Murmur* dapat diklasifikasikan sebagai *innocent* (*benign*) atau *pathological* berdasarkan karakteristik akustiknya (Frank et al., 2022). Karakteristik penting *murmur* meliputi *timing* (*systolic, diastolic, continuous*), intensitas (*grade* 1–6 berdasarkan *Levine scale*), *pitch* (*high, medium, low*), *quality* (*blowing, harsh, rumbling*), lokasi auskultasi terbaik, dan pola radiasi ke area anatomis lainnya (Bonow et al., 2021).

Analisis *PCG* memiliki beberapa keunggulan dibandingkan auskultasi manual, termasuk objektivitas, *repeatability*, kemampuan untuk analisis kuantitatif, dan potensi untuk *screening* massal (Deng & Bentley, 2021). Dengan perkembangan teknologi *digital stethoscope* dan algoritma pemrosesan sinyal berbasis *artificial intelligence*, analisis *PCG* otomatis menjadi semakin *feasible* untuk aplikasi klinis, terutama untuk *telemedicine* dan *point-of-care screening*.



Gambar 2.1 Visualisasi *Phonocardiogram*

2.3 Artificial Intelligence (AI)

Artificial Intelligence (*AI*) adalah cabang ilmu komputer yang bertujuan menciptakan sistem yang dapat melakukan tugas-tugas yang normalnya memerlukan kecerdasan manusia (Russell & Norvig, 2020). *AI* mencakup berbagai kemampuan kognitif seperti *learning, reasoning, problem-solving, perception,* dan *language understanding*. Secara fundamental, *AI* berupaya untuk membuat mesin yang dapat berpikir dan bertindak secara rasional dalam berbagai

situasi. Konsep *AI* pertama kali diperkenalkan oleh McCarthy et al. (1956) dalam *Dartmouth Conference*, di mana mereka mendefinisikan *AI* sebagai *"the science and engineering of making intelligent machines"*. Russell dan Norvig (2020) mengklasifikasikan definisi *AI* menjadi empat kategori: *systems that think like humans, systems that think rationally, systems that act like humans,* dan *systems that act rationally*.

2.4 Deep Learning

Deep learning adalah cabang dari *machine learning* yang menggunakan *neural networks* dengan beberapa *hidden layers* (*deep neural networks*) untuk mempelajari representasi data yang kompleks (LeCun et al., 2015). Keunggulan utama *deep learning* adalah kemampuannya untuk secara otomatis mengekstraksi fitur dari data mentah tanpa memerlukan *feature engineering* manual yang ekstensif (Goodfellow et al., 2016). Istilah *"deep"* merujuk pada jumlah *layers* dalam *network* yang memungkinkan model mempelajari fitur pada berbagai tingkat abstraksi, di mana *layer* bawah mendeteksi pola sederhana dan *layer* atas mengombinasikannya menjadi representasi yang lebih kompleks. *Deep learning* telah mencapai dampak transformasional dalam berbagai domain dengan kinerja yang sangat tinggi, terutama sejak kemajuan dalam *large-scale models* dan sumber daya komputasi (Sejnowski, 2020).

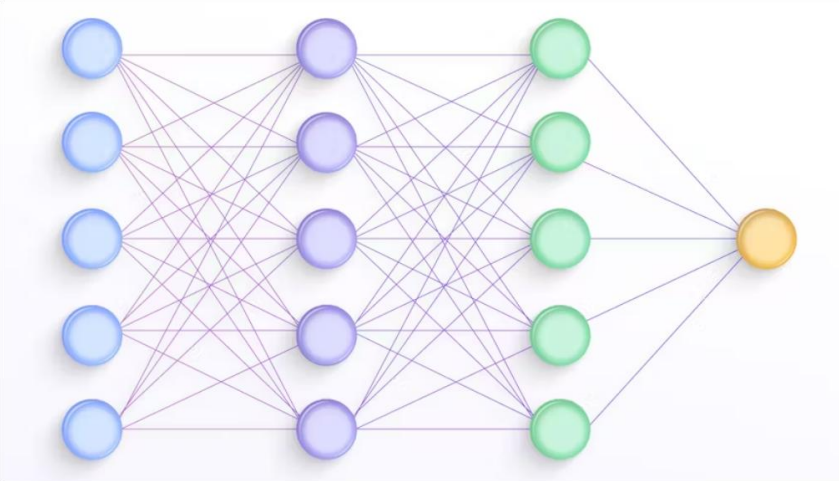
Prinsip fundamental *deep learning* didasarkan pada pembelajaran hierarkis (*hierarchical learning*) dan representasi terdistribusi (*distributed representations*). *Neural networks* dalam *deep learning* terdiri dari *interconnected layers* dari neuron buatan yang melakukan transformasi non-linear terhadap data masukan. Setiap *layer* menerima input dari *layer* sebelumnya, melakukan penjumlahan berbobot dan aktivasi non-linear, kemudian meneruskan output ke *layer* berikutnya. Proses pelatihan menggunakan algoritma *backpropagation* untuk mengoptimalkan bobot berdasarkan *loss function* yang mengukur perbedaan antara prediksi model dan label sebenarnya. Optimisasi dilakukan melalui variasi *gradient descent* seperti *Stochastic Gradient Descent (SGD)*, *Adam*, atau *AdamW* yang menyesuaikan parameter secara iteratif untuk meminimalkan *loss* (Loshchilov & Hutter, 2019). Kemajuan terbaru dalam teknik

optimisasi, metode regularisasi, dan inovasi arsitektur telah meningkatkan stabilitas pelatihan serta kinerja model secara signifikan (Zhang et al., 2021).

2.5 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) adalah jenis *deep neural network* yang dirancang khusus untuk pemrosesan data dengan struktur menyerupai grid, seperti gambar dan sinyal deret waktu (*time-series*) (LeCun et al., 1998). CNN telah merevolusi bidang *computer vision* dan semakin banyak diaplikasikan untuk pemrosesan sinyal audio serta data biomedis (Krizhevsky et al., 2012). Arsitektur CNN terinspirasi dari organisasi *visual cortex* pada mamalia, di mana neuron merespons rangsangan hanya pada wilayah terbatas yang disebut *receptive field* (Hubel & Wiesel, 1962). CNN mengimplementasikan konsep ini melalui konektivitas lokal (*local connectivity*) dan pembagian bobot (*weight sharing*), yang membuat *network* efisien dalam mendeteksi pola pada berbagai lokasi dalam data masukan.

Prinsip fundamental CNN didasarkan pada tiga konsep utama (Goodfellow et al., 2016). Pertama, konektivitas lokal (*local connectivity*), di mana setiap neuron hanya terhubung ke wilayah lokal dari masukan, sehingga jumlah parameter berkurang drastis dan *network* dapat fokus pada pola lokal. Kedua, pembagian parameter (*parameter sharing*), yaitu penggunaan kernel yang sama pada berbagai posisi dalam masukan, membuat CNN bersifat *translation equivariant* serta efisien dalam penggunaan parameter. Ketiga, pembelajaran fitur hierarkis (*hierarchical feature learning*), di mana CNN secara otomatis mempelajari representasi hierarkis melalui penumpukan beberapa *layers*, dengan *layer* awal mendeteksi fitur sederhana dan *layer* lebih dalam mempelajari fitur semantik tingkat tinggi (LeCun et al., 2015).



Gambar 2.2 Visualisasi Convolutional Neural Network

2.5.1 CNN 2D

CNN 2D (*Two-Dimensional Convolutional Neural Network*) adalah arsitektur *deep learning* yang dirancang untuk memproses data dengan struktur *grid* dua dimensi (LeCun et al., 2015). Dalam klasifikasi audio, CNN 2D beroperasi pada representasi visual dari sinyal suara seperti *spectrogram, mel-spectrogram,* atau *MFCC*, di mana sumbu horizontal merepresentasikan waktu dan sumbu vertikal merepresentasikan frekuensi (Hershey et al., 2017).

Keunggulan CNN 2D dalam pemrosesan sinyal audio mencakup beberapa aspek. Dari sisi ekstraksi fitur time-frequency, CNN 2D mampu menangkap pola kompleks secara simultan pada domain waktu dan frekuensi, yang penting untuk membedakan karakteristik suara jantung seperti S1, S2, dan murmur (Potes et al., 2016). Dari sisi representasi hierarchical, CNN 2D membangun fitur bertingkat dari low-level features seperti edges pada layer awal, hingga high-level features seperti pola akustik kompleks pada layer lebih dalam (Zeiler & Fergus, 2014).

2.5.2 Komponen Arsitektur CNN

Arsitektur CNN terdiri dari beberapa komponen fundamental yang bekerja bersama untuk feature extraction dan classification:

2.5.2.1 Convolutional Layer

Convolutional layer adalah komponen inti dari CNN yang melakukan operasi konvolusi antara sinyal masukan (*input signal*) dan kernel yang dapat dipelajari (*learnable kernels* atau *kernels*) untuk mengekstraksi fitur (LeCun et al., 2015). Pada *CNN 1D*, operasi konvolusi dapat dinyatakan sebagai:

$$y[i] = k\sum x[i + k] \cdot w[k] + b \quad (2\text{-}1)$$

di mana x adalah sinyal masukan, w adalah bobot kernel (*kernel weights*), bbb adalah *bias term*, dan y adalah peta fitur keluaran (*output feature map*). *Convolutional layer* berfungsi untuk mendeteksi pola lokal (*local patterns*) dalam data masukan, seperti tepi (*edges*), tekstur (*textures*), atau dalam konteks audio, pola suara spesifik serta karakteristik temporal. Lapisan ini menggunakan konsep pembagian bobot (*weight sharing*), di mana kernel yang sama diaplikasikan ke seluruh bagian masukan, sehingga secara signifikan mengurangi jumlah parameter dan memungkinkan deteksi pola pada berbagai posisi dalam data.

2.5.2.2 Activation Function

Activation function memperkenalkan *non-linearity* ke dalam *neural network*, memungkinkan model untuk mempelajari hubungan yang kompleks dan non-linear dalam data (Nair & Hinton, 2010). Tanpa *activation function*, *neural network* hanya mampu mempelajari transformasi linear meskipun memiliki kedalaman berlapis.

Beberapa jenis *activation function* yang umum digunakan dalam *deep learning*:

- **ReLU (Rectified Linear Unit)**

ReLU (Rectified Linear Unit) adalah *activation function* paling populer dalam *deep learning* modern. ReLU memberikan beberapa keunggulan: kesederhanaan komputasi karena hanya melibatkan operasi ambang (*thresholding operation*), mengurangi masalah *vanishing gradient* yang memfasilitasi pelatihan *deep networks*, serta mendorong terjadinya *sparsity* dengan menghasilkan aktivasi nol

untuk input negatif (Agarap, 2018). Namun, ReLU dapat mengalami masalah *dying ReLU* di mana neuron menjadi tidak aktif secara permanen jika menerima nilai negatif yang besar selama pelatihan.

- **Leaky ReLU**

Leaky ReLU adalah varian dari ReLU yang mengatasi masalah *dying ReLU* dengan memberikan kemiringan kecil (*small negative slope*) untuk input negatif, di mana α adalah konstanta kecil (biasanya 0.01). Leaky ReLU mempertahankan keunggulan ReLU sekaligus memungkinkan aliran gradien kecil pada nilai negatif, sehingga mencegah neuron menjadi benar-benar tidak aktif (Maas et al., 2013).

- **Sigmoid**

Sigmoid adalah *activation function* klasik yang memetakan input ke rentang (0, 1). Fungsi ini sering digunakan pada *output layer* untuk klasifikasi biner karena keluarannya dapat diinterpretasikan sebagai probabilitas. Namun, sigmoid mengalami masalah serius *vanishing gradient* untuk input yang sangat besar atau sangat kecil, sehingga kurang cocok digunakan pada *hidden layer* dalam *deep networks* (Goodfellow et al., 2016).

- **Softmax**

Softmax adalah *activation function* yang biasanya digunakan pada *output layer* untuk klasifikasi multi-kelas, dengan mengubah skor mentah (*logits*) menjadi distribusi probabilitas. Softmax memastikan bahwa nilai keluaran berjumlah total 1 dan setiap nilai berada di antara 0 dan 1, sehingga dapat diinterpretasikan sebagai probabilitas kelas.

Pemilihan *activation function* dapat berdampak signifikan pada performa model, dinamika pelatihan, dan kecepatan konvergensi (Dubey et al., 2022). Dalam praktiknya, ReLU dan variannya paling umum digunakan untuk *hidden layer* karena efisiensi komputasi dan efektivitasnya, sedangkan sigmoid

digunakan untuk keluaran klasifikasi biner dan softmax untuk keluaran klasifikasi multi-kelas.

2.5.2.3 Pooling Layer

Pooling layer melakukan operasi *downsampling* untuk mengurangi dimensi spasial dari *feature maps* sambil tetap mempertahankan informasi penting (Scherer et al., 2010). *Max pooling*, jenis *pooling* yang paling umum, mengambil nilai maksimum dari setiap wilayah lokal:

$$y[i] = max(x[i \cdot s : i \cdot s + k]) \tag{2-2}$$

di mana s adalah *stride* dan k adalah ukuran *pooling window*. Fungsi *pooling layer* adalah mengurangi dimensi dan biaya komputasi, memberikan sifat *translation invariance* yang membuat model lebih *robust* terhadap pergeseran kecil pada input, serta mencegah *overfitting* dengan mengurangi jumlah parameter. Selain itu, *pooling* membantu mengekstraksi fitur dominan yang tetap relevan meskipun berada pada posisi berbeda dalam sinyal input.

2.5.2.4 Batch Normalization

Batch Normalization adalah teknik yang menormalisasi aktivasi dari setiap *layer* untuk setiap *mini-batch*, sehingga secara signifikan meningkatkan dinamika pelatihan (Ioffe & Szegedy, 2015). Operasinya dapat dinyatakan sebagai:

$$x\hat{} = \frac{x - \mu_{batch}}{\sqrt{\sigma^2{}_{batch} + \in}} \tag{2-3}$$

$$y = \gamma \cdot x\hat{} + \beta \tag{2-4}$$

di mana μ_{batch} dan $\sigma^2{}_{batch}$ adalah *mean* dan *variance* dari *mini-batch*, $\in$adalah konstanta kecil untuk stabilitas numerik, serta γ dan β adalah parameter yang dapat dipelajari.

Batch normalization berfungsi untuk mempercepat pelatihan dengan memungkinkan *learning rate* yang lebih tinggi, mengurangi *internal covariate shift* sehingga pelatihan lebih stabil, berperan sebagai bentuk *regularization* yang dapat mengurangi kebutuhan *dropout* pada beberapa kasus, serta meningkatkan aliran gradien di dalam *network* sehingga memfasilitasi pelatihan arsitektur yang lebih dalam.

2.5.2.5 Dropout

Dropout adalah teknik *regularization* yang secara acak menonaktifkan (mengatur ke nol) sebagian neuron selama pelatihan dengan probabilitas ppp (*dropout rate*) (Srivastava et al., 2014). Fungsinya adalah mencegah *overfitting* dengan memaksa *network* untuk tidak bergantung pada neuron tertentu, mendorong pembelajaran representasi yang lebih *robust* dan terdistribusi sehingga mampu melakukan generalisasi lebih baik, serta memberikan efek seperti *model averaging* karena pelatihan secara efektif menciptakan *ensemble* dari berbagai sub-*network*. *Dropout* hanya aktif selama pelatihan dan dinonaktifkan pada saat inferensi, di mana semua neuron digunakan, tetapi output diskalakan sesuai *dropout rate* untuk menjaga nilai ekspektasi.

2.5.2.6 Flatten Layer

Flatten layer mengonversi *feature maps* multi-dimensi dari *convolutional* dan *pooling layers* menjadi vektor satu dimensi (Goodfellow et al., 2016). *Layer* ini berfungsi sebagai lapisan transisi antara bagian *convolutional* dari *network* (yang bekerja pada data spasial/temporal) dan bagian *fully connected* (yang membutuhkan input satu dimensi). Operasi *flatten* hanya melakukan *reshaping* data tanpa mempelajari parameter apa pun, dengan tetap mempertahankan semua informasi dari *feature maps* dalam susunan linear yang sesuai untuk *dense layers*.

2.5.2.7 Fully Connected Layer

Fully connected layer (*dense layer*) menghubungkan setiap neuron dengan semua neuron pada *layer* sebelumnya, melakukan integrasi global terhadap fitur-

fitur yang diekstraksi oleh *convolutional layers* (Goodfellow et al., 2016). Operasinya dapat dinyatakan sebagai:

$$y = W \cdot x + b \tag{2-5}$$

di mana W adalah matriks bobot, x adalah vektor input, dan b adalah vektor bias.

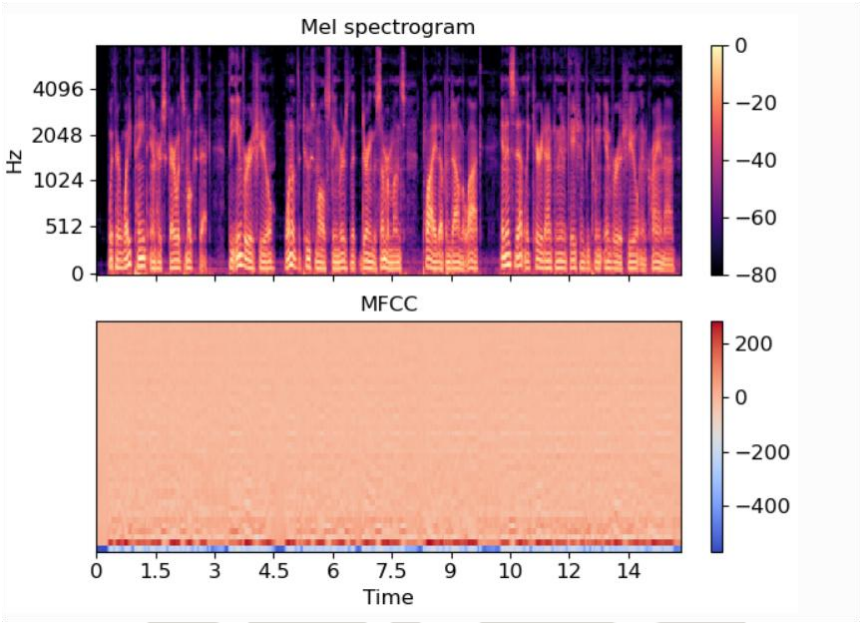
Fully connected layers berfungsi untuk mengombinasikan semua fitur yang diekstraksi dalam membuat keputusan klasifikasi akhir, mempelajari kombinasi non-linear yang kompleks dari fitur-fitur tersebut, serta memetakan dari *feature space* ke *output space* (label kelas). *Layer* ini umumnya ditempatkan mendekati bagian keluaran *network* untuk melakukan penalaran tingkat tinggi dan pengambilan keputusan.

2.6 Mel-Frequency Cepstral Coefficients (MFCC)

*Mel-Frequency Cepstral Coefficients (*MFCC) adalah representasi fitur audio yang paling banyak digunakan dalam *speech recognition* dan *audio classification* (Davis & Mermelstein, 1980). MFCC mengekstraksi fitur yang merefleksikan karakteristik spektral sinyal audio dengan cara yang sesuai dengan persepsi auditory system manusia.

Nama "mel" berasal dari *mel scale*, yang merupakan *perceptual scale of pitches* yang dikembangkan oleh Stevens et al. (1937). *Mel scale* didasarkan pada pengamatan bahwa telinga manusia tidak mempersepsikan frekuensi secara linear.

Gambar 2.3 memperlihatkan representasi visual dari fitur audio menggunakan *Mel spectrogram* (atas) dan *Mel-Frequency Cepstral Coefficients* (MFCC) (bawah)



Gambar 2.3 Mel spectrogram dan Mel-Frequency Cepstral Coefficients (MFCC)

2.7 Hyperparameter

Hyperparameter adalah parameter konfigurasi yang nilainya ditentukan sebelum proses training dimulai dan tidak dipelajari dari data training (*Goodfellow et al., 2016*). Berbeda dengan parameter model (seperti bobot dan bias) yang dioptimasi selama training melalui backpropagation, hyperparameter harus ditetapkan secara eksplisit dan memiliki pengaruh signifikan terhadap performa model. Pemilihan hyperparameter yang tepat merupakan faktor krusial dalam pengembangan model *deep learning* yang efektif, karena dapat mempengaruhi kecepatan konvergensi, akurasi akhir, dan kemampuan generalisasi (*Yang & Shami, 2020*).

2.7.1 Learning Rate

Learning rate adalah salah satu hyperparameter paling kritis yang menentukan seberapa besar pembaruan bobot dalam setiap iterasi (*Bengio, 2012*). *Learning rate* yang terlalu besar dapat menyebabkan optimisasi menyimpang atau berosilasi, sedangkan *learning rate* yang terlalu kecil membuat konvergensi sangat lambat dan dapat terjebak di *local minima*. Penelitian terkini menunjukkan bahwa *adaptive learning rate schedules*, seperti *cosine annealing* dan *learning*

rate warmup, dapat meningkatkan stabilitas training dan performa akhir model (*Loshchilov & Hutter, 2017*). *Smith et al. (2021)* mendemonstrasikan bahwa *learning rate* yang optimal sering bergantung pada ukuran *batch*, dengan *batch* yang lebih besar membutuhkan *learning rate* yang lebih tinggi secara proporsional untuk menjaga dinamika training yang efektif.

2.7.2 **Batch Size**

Batch size mempengaruhi kecepatan pelatihan, kebutuhan memori, dan kemampuan model untuk menggeneralisasi data baru (*Keskar et al., 2017*). Ukuran *batch* yang kecil menghasilkan estimasi gradien yang bervariasi (*noisy*) namun dapat meningkatkan kemampuan generalisasi model. Sebaliknya, ukuran *batch* yang besar memberikan estimasi gradien yang lebih stabil tetapi memerlukan memori yang lebih besar dan cenderung menghasilkan model yang kurang mampu menggeneralisasi data baru dengan baik.

2.7.3 **Network Architecture**

Jumlah *layer* dan unit per *layer* menentukan kapasitas representasi model dalam mempelajari pola data (*Hanin & Seluk, 2018*). *Network* yang terlalu dangkal mungkin tidak dapat menangkap pola yang kompleks (*underfitting*), sedangkan *network* yang terlalu dalam dapat mengalami *overfitting* dan sulit untuk dilatih. *Tan dan Le (2021)* mengembangkan *EfficientNetV2* yang menunjukkan pentingnya *systematic architecture scaling*, bahwa kedalaman, lebar, dan resolusi sebaiknya diskalakan secara bersama-sama, bukan secara terpisah. Penelitian modern juga menekankan pentingnya komponen arsitektural seperti *skip connections* dan *attention mechanisms* dalam memungkinkan training yang efektif pada *very deep networks* (*Liu et al., 2022*).

2.7.4 **Dropout Rate**

Dropout rate menentukan proporsi neuron yang dinonaktifkan secara acak selama proses pelatihan (*Srivastava et al., 2014*). *Dropout rate* yang terlalu rendah memberikan regularisasi yang tidak memadai sehingga model cenderung *overfit*, sedangkan *dropout rate* yang terlalu tinggi dapat mengurangi kapasitas model

secara signifikan sehingga performanya menurun. Penelitian oleh *Labach et al. (2019)* mengeksplorasi strategi *adaptive dropout* di mana *dropout rate* disesuaikan secara dinamis selama training berdasarkan performa validasi. Selain itu, teknik regularisasi alternatif seperti *DropBlock* dan *Cutout* menunjukkan hasil yang menjanjikan untuk domain tertentu seperti *computer vision* dan *audio processing* (*Ghiasi et al., 2018; DeVries & Taylor, 2017*).

2.7.5 **Optimizer Selection**

Pemilihan *optimizer* dapat memberikan dampak signifikan pada performa dan dinamika training model (*Ruder, 2016*). *Stochastic Gradient Descent (SGD)* dengan momentum adalah *optimizer* klasik yang tangguh tetapi memerlukan penyetelan yang cermat. *Adam optimizer* (*Kingma & Ba, 2015*) mengadaptasi *learning rate* untuk setiap parameter dan sering kali konvergen lebih cepat, tetapi kadang menghasilkan generalisasi yang lebih buruk pada beberapa tugas. Perkembangan terbaru termasuk *AdamW* yang memisahkan *weight decay* dari pembaruan gradien (*Loshchilov & Hutter, 2019*), dan *LAMB optimizer* yang memungkinkan pelatihan *batch* besar secara efektif (*You et al., 2020*). *Liu et al. (2020)* melakukan analisis komprehensif yang menunjukkan bahwa pilihan *optimizer* yang optimal bergantung pada kompleksitas tugas, arsitektur model, dan sumber daya komputasi yang tersedia. *Choi et al. (2020)* menunjukkan bahwa pendekatan hibrid yang menggabungkan keunggulan dari beberapa *optimizer* dapat mencapai performa yang lebih baik di berbagai jenis tugas.

2.8 Hyperparameter Optimization (HPO)

Hyperparameter optimization (HPO) adalah proses sistematis untuk menemukan kombinasi *hyperparameter* yang menghasilkan performa model terbaik (Feurer & Hutter, 2019). Performa model deep learning sangat bergantung pada konfigurasi *hyperparameter* yang optimal, di mana setiap *hyperparameter* berinteraksi secara kompleks dan mempengaruhi hasil akhir model (Bergstra & Bengio, 2012). Sebagai analogi, hyperparameter dapat diibaratkan sebagai "resep" dalam pembuatan kue, di mana setiap bahan (*learning rate, batch size, dropout rate*, dll.) harus dalam takaran yang tepat agar menghasilkan produk yang sempurna.

Manual tuning atau *trial-and-error* merupakan pendekatan konvensional yang tidak efisien dan sering kali tidak menghasilkan konfigurasi optimal, terutama ketika berhadapan dengan *high-dimensional hyperparameter space* di mana banyak *hyperparameter* saling berinteraksi (Claesen & De Moor, 2015). Kompleksitas ini mendasari munculnya metode HPO yang lebih sistematis dan otomatis untuk mengeksplorasi *hyperparameter space* secara efisien.

Dalam *hyperparameter optimization*, *objective function* yang ingin dioptimalkan adalah (Yang & Shami, 2020):

$$\lambda * = \ argmin\ L(A\_\lambda, D\_train, D\_valid) \quad (2\text{-}6)$$
$$\lambda \in \Lambda$$

dimana:

- λ adalah konfigurasi *hyperparameter*
- Λ adalah *hyperparameter search space*
- A_λ adalah algoritma dengan *hyperparameter* λ
- D_train adalah *training data*
- D_valid adalah *validation data*
- L adalah loss atau *error metric*

Tujuannya adalah menemukan λ* yang meminimalkan *validation error* dan menghindari *overfiting*.

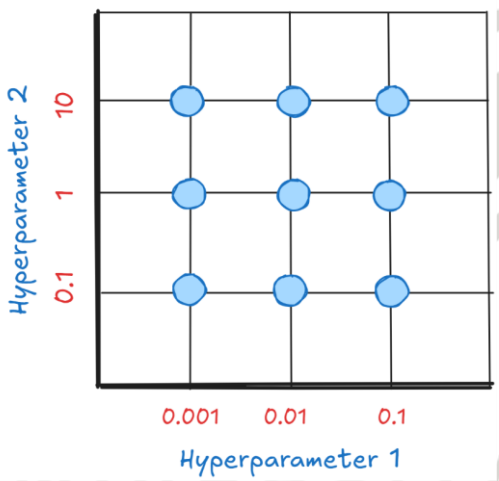
2.8.1 Grid Search

Grid search merupakan metode HPO yang paling sederhana dan *straightforward* dengan pendekatan *brute-force* (Bergstra & Bengio, 2012). Metode ini bekerja dengan mendefinisikan grid dari nilai-nilai diskrit untuk setiap *hyperparameter*, kemudian secara exhaustive mencoba setiap kombinasi yang mungkin dalam grid tersebut (Liashchynskyi & Liashchynskyi, 2019).

Kelebihan *grid search* adalah kesederhanaannya dan *guarantee* bahwa kombinasi terbaik dalam *grid* yang didefinisikan akan ditemukan (Mantovani et al.,

2015). Metode ini juga mudah untuk diparalelisasi karena setiap evaluasi independen satu sama lain. Namun, kekurangan utama *grid search* adalah *computational cost* yang sangat tinggi, terutama ketika jumlah *hyperparameter* dan *range* nilainya besar (Bergstra & Bengio, 2012). Kompleksitas waktu *grid search* meningkat secara eksponensial dengan jumlah *hyperparameter* (*curse of dimensionality*), sehingga menjadi kurang efektif untuk *high-dimensional search spaces* (Claesen & De Moor, 2015).

Gambar 2.4 menunjukkan ilustrasi *Grid Search* dalam pencarian *hyperparameter*, dengan pendekatan *brute-force* yang mengevaluasi seluruh kombinasi dari nilai *hyperparameter* yang telah ditentukan.



Gambar 2.4 Ilustrasi *Grid Search.*

2.8.1.1 Reduce Grid

Strategi *reduced grid* dibuat melalui pendekatan *intelligent sampling* yang mempertahankan *diversity* dalam ruang pencarian *hyperparameter*. *Keras Tuner GridSearch* secara otomatis melakukan *sampling* dengan prioritas pada kombinasi yang memiliki probabilitas tinggi menghasilkan performa optimal berdasarkan interaksi antar *hyperparameter*. Pendekatan ini mengikuti prinsip yang dijelaskan oleh *Bergstra dan Bengio (2012)* bahwa tidak semua dimensi *hyperparameter* memiliki pengaruh yang sama terhadap performa model, sehingga *sampling* dapat difokuskan pada *region* yang lebih *promising*. *Reduced grid* dibentuk dengan memastikan setiap nilai dari *hyperparameter* penting (seperti jumlah *kernels*, *kernel*

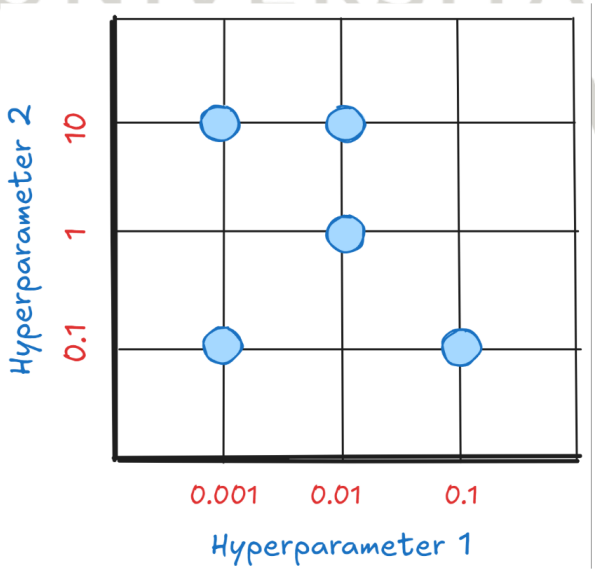
size, dan *learning rate*) terrepresentasi dengan baik dalam kombinasi yang dievaluasi.

2.8.2 Random Search

Random search merupakan alternatif yang lebih efisien dibandingkan *grid search*, di mana metode ini secara *random* mengambil sampel kombinasi *hyperparameter* dari distribusi yang telah ditentukan (Bergstra & Bengio, 2012). Alih-alih mencoba setiap kombinasi secara sistematis, *random search* mengeksplorasi *hyperparameter space* dengan *sampling* acak.

Bergstra dan Bengio (2012) dalam penelitian mereka menunjukkan bahwa *random search* secara signifikan lebih efisien daripada *grid search* untuk *hyperparameter optimization*. Keunggulan utama *random search* adalah probabilitas yang lebih tinggi untuk menemukan kombinasi yang baik dalam budget komputasi yang sama, terutama ketika beberapa *hyperparameter* lebih penting daripada yang lain. *Random search* tidak membuang *resources* untuk mencoba kombinasi yang sistematis tetapi kurang penting, sehingga dapat mengeksplorasi lebih banyak nilai untuk *hyperparameter* yang *critical* (Bergstra & Bengio, 2012).

Gambar 2.5 adalah representasi dari metode *Random Search*, di mana kombinasi nilai hyperparameter dipilih secara acak dari ruang pencarian.



Gambar 2.5 Ilustrasi *Random Search*

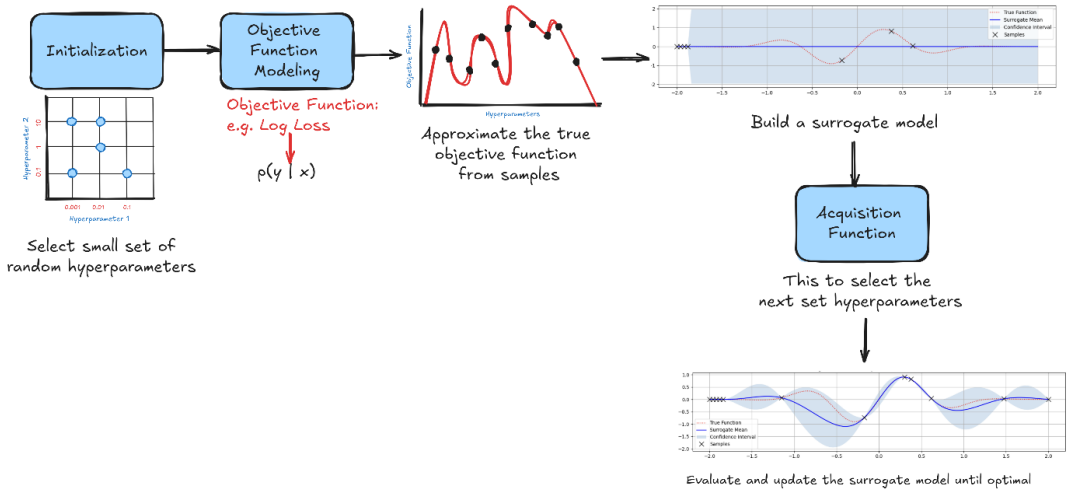
2.8.3 Bayesian Optimization

Bayesian optimization adalah metode HPO yang lebih *sophisticated* dan *intelligent*, yang membangun model probabilistik dari fungsi objektif (performa model) terhadap *hyperparameter* yang diuji (Shahriari et al., 2016). Metode ini menggunakan *prior knowledge* dari eksperimen sebelumnya untuk memandu pencarian *hyperparameter* selanjutnya, sehingga dapat menemukan konfigurasi optimal dengan jumlah evaluasi yang lebih sedikit (Snoek et al., 2012).

Bayesian optimization bekerja dengan membangun *surrogate model*, biasanya *Gaussian Process* (GP), yang memodelkan distribusi probabilitas dari fungsi objektif (Mockus, 1975). Model ini kemudian digunakan untuk menghitung *acquisition function* yang menentukan *hyperparameter* mana yang paling menjanjikan untuk dievaluasi selanjutnya (Brochu et al., 2010). Proses ini melibatkan *trade-off* antara *exploitation* (mengeksplorasi region yang sudah diketahui menghasilkan performa baik) dan *exploration* (mencari region baru yang belum pernah diuji) (Shahriari et al., 2016).

Acquisition functions yang umum digunakan meliputi *Expected Improvement* (EI), *Probability of Improvement* (PI), dan *Upper Confidence Bound* (UCB), yang masing-masing memiliki karakteristik *exploration-exploitation* yang berbeda (Snoek et al., 2012). Keunggulan utama *Bayesian optimization* adalah efisiensi komputasinya yang *superior*, karena metode ini secara intelligent memilih kombinasi *hyperparameter* yang paling menjanjikan dan menghindari evaluasi pada *region* yang *unlikely* menghasilkan *improvement* (Frazier, 2018).

Gambar 2.6 adalah representasi dari metode *Bayesian Optimization*, di mana pemilihan *hyperparameter* dilakukan dengan membangun *surrogate model* untuk memodelkan fungsi objektif dan menggunakan *acquisition function* untuk menentukan kombinasi *hyperparameter* berikutnya.



Gambar 2.6 Ilustrasi Bayesian Optimization

2.8.4 Genetic Algorithm

Genetic algorithm (GA) adalah metode HPO(*Hyperparameter Optimization*) yang terinspirasi dari proses evolusi biologis, menggunakan mekanisme seperti *selection, crossover*, dan *mutation* untuk mengeksplorasi *hyperparameter space* (Holland, 1992). Metode ini memaintain populasi dari kandidat solusi (kombinasi hyperparameter) dan secara iteratif mengevolusi populasi tersebut menuju solusi yang lebih baik (Goldberg & Holland, 1988).

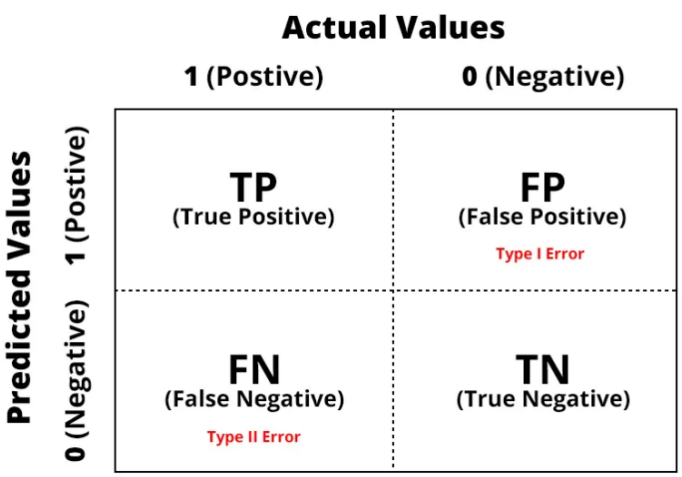
Proses GA dimulai dengan inisialisasi populasi *random*, kemudian mengevaluasi *fitness* (performa) setiap individu dalam populasi (Lorenzo et al., 2017). Individu dengan *fitness* tinggi memiliki probabilitas lebih besar untuk diseleksi sebagai *parents* untuk generasi berikutnya. *Crossover* operation menggabungkan *hyperparameter* dari dua *parents* untuk menghasilkan *offspring*, sedangkan *mutation operation* memperkenalkan variasi random untuk maintain diversity dan menghindari premature convergence (Eiben & Smith, 2015).

Keunggulan *genetic algorithm* adalah kemampuannya untuk mengeksplorasi *non-convex* dan *multimodal search spaces*, serta tidak memerlukan *gradient information* (Lorenzo et al., 2017). GA juga *naturally parallelizable* karena evaluasi *fitness* dalam satu generasi dapat dilakukan secara independen. Namun, kekurangan GA termasuk memerlukan tuning dari GA parameters itu sendiri (*population size,*

mutation rate, crossover rate) dan dapat memerlukan banyak evaluasi untuk *converge*, terutama untuk high-dimensional spaces (Eiben & Smith, 2015).

2.9 Confusion Matrix

Confusion Matrix adalah tabel yang menggambarkan performa model klasifikasi dengan membandingkan prediksi model terhadap label aktual (Sokolova & Lapalme, 2009). Tabel ini terdiri dari empat komponen utama: *True Positives* (TP) yaitu data positif yang diprediksi benar sebagai positif, *True Negatives* (TN) yaitu data negatif yang diprediksi benar sebagai negatif, *False Positives* (FP) yaitu data negatif yang salah diprediksi sebagai positif, dan *False Negatives* (FN) yaitu data positif yang salah diprediksi sebagai negatif (Powers, 2011). Confusion matrix menjadi dasar untuk menghitung berbagai metrik evaluasi lainnya.



Gambar 2.7 Confusion Matrix

2.9.1 Accuracy

Accuracy adalah proporsi total prediksi yang benar dari keseluruhan prediksi (Powers, 2011). Metrik ini mengukur seberapa akurat model dapat mengklasifikasikan data dengan benar. Nilai accuracy dapat diperoleh dengan persamaan:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2-7}$$

dimana TP = *True Positives*, TN = *True Negatives*, FP = *False Positives*, dan FN = *False Negatives*.

2.9.2 Precision

Precision merupakan proporsi prediksi positif yang benar dari total prediksi positif (Fawcett, 2006). *Precision* berfokus pada meminimalisir false positive dan mengukur ketepatan model ketika memprediksi kelas positif (Davis & Goadrich, 2006). Nilai *precision* dapat diperoleh dengan persamaan:

$$precision = \frac{TP}{TP + FP} \tag{2-8}$$

2.9.3 Recall (Sensitivity)

Recall atau *sensitivity* mengukur kemampuan model untuk mendeteksi semua sampel positif (Hossin & Sulaiman, 2015). Metrik ini penting dalam konteks medis dimana mendeteksi semua kasus positif lebih krusial daripada menghindari false positives (Japkowicz & Shah, 2011). Nilai *recall* dapat diperoleh dengan persamaan:

$$recall = \frac{TP}{TP + FN} \tag{2-9}$$

dimana TP = *True Positives* dan FN = *False Negatives*.

2.9.4 F1-Score

F1-Score adalah *harmonic mean* dari *precision* dan *recall* yang memberikan keseimbangan antara kedua metrik tersebut (Powers, 2011). *F1-Score* berguna ketika terdapat *trade-off* antara *precision* dan *recall*, serta penting untuk dataset dengan *class imbalance*. Nilai *F1-Score* dapat diperoleh dengan persamaan:

$$F1 - Score = 2 \times \frac{precision \times recall}{precision + recall} \tag{2-10}$$

dimana *precision* dan *recall* adalah nilai dari masing-masing metrik yang telah dihitung sebelumnya.

2.10 Python

Python merupakan bahasa pemrograman yang banyak digunakan dalam pengembangan aplikasi *machine learning* dan *deep learning* karena sintaks yang mudah dibaca, library yang komprehensif, serta dukungan komunitas yang luas (Chollet, 2021). *Python* memiliki keunggulan fleksibilitas yang memungkinkan peneliti menguji berbagai algoritma dan arsitektur deep learning, yang sangat penting dalam domain medical signal processing karena akurasi dan performa menjadi prioritas utama (Géron, 2022).

2.10.1 Librosa

Librosa merupakan *library Python* yang dirancang khusus untuk analisis dan pemrosesan sinyal audio (McFee et al., 2015). Dalam penelitian ini, Librosa digunakan untuk *loading* dan *preprocessing audio files, resampling audio signals* ke *uniform sampling rate*, ekstraksi MFCC feature sebagai *input* untuk model *deep learning*, dan visualisasi *audio signals* dan spectrograms untuk exploratory analysis. Menurut McFee et al. (2015), Librosa menyediakan implementasi yang *reliable* dan *well-tested* dari berbagai *audio processing algorithms.*

2.10.2 TensorFlow dan Keras

TensorFlow dengan API *Keras* merupakan kerangka kerja *deep learning* yang dikembangkan oleh Google (Abadi et al., 2016; Chollet, 2021). *TensorFlow* dan *Keras* digunakan untuk membangun arsitektur CNN, melatih model dengan berbagai konfigurasi *hyperparameter*, membuat fungsi pemantau untuk proses optimasi *hyperparameter*, mengevaluasi kinerja model dengan metrik bawaan, serta menyimpan dan memuat model yang sudah dilatih untuk keperluan perbandingan. Menurut Chollet (2021), *Keras* memiliki keunggulan dalam mendukung pengembangan prototipe secara cepat serta memudahkan penerapan arsitektur yang kompleks untuk analisis sinyal biomedis.

2.10.3 Scikit-learn

Scikit-learn menyediakan implementasi algoritma *machine learning* dan berbagai alat evaluasi yang komprehensif (Pedregosa et al., 2011). *Library* ini digunakan untuk membagi dataset menjadi data latih, validasi, dan uji dengan

teknik stratifikasi, mengevaluasi model menggunakan metrik seperti akurasi, presisi, *recall*, dan *F1-score*, membuat *confusion matrix* untuk analisis kinerja yang lebih detail, melakukan *cross-validation* untuk estimasi kinerja yang lebih andal.Serta melakukan praproses data seperti penskalaan dan normalisasi. Menurut Géron (2022), Scikit-learn memiliki antarmuka yang konsisten dan mudah digunakan untuk berbagai tugas machine learning.

2.10.4 NumPy

NumPy merupakan *library* fundamental untuk komputasi ilmiah di *Python* yang menyediakan dukungan terhadap *array* dan matriks multidimensi, beserta fungsi matematika tingkat tinggi (Harris et al., 2020). Dalam penelitian ini, *NumPy* digunakan untuk representasi dan manipulasi audio data dalam bentuk array multidimensi, operasi matematika untuk normalisasi dan *preprocessing*, array reshaping untuk menyiapkan input ke model CNN, *statistical calculations* untuk data analysis, dan performance-critical operations dalam feature extraction pipeline.

2.10.5 Pandas

Pandas menyediakan struktur data dan berbagai alat untuk menganalisis data secara efektif (McKinney, 2010). Dalam penelitian ini, *Pandas* digunakan untuk mengelola metadata dataset (seperti lokasi file, label, dan durasi), melakukan analisis eksploratif untuk memahami karakteristik data, mengatur konfigurasi *hyperparameter* beserta hasilnya, membuat ringkasan statistik dari hasil eksperimen, serta mengorganisasi metrik evaluasi guna membandingkan berbagai metode secara lebih efektif.

2.10.6 Matplotlib dan Seaborn

Matplotlib dan *Seaborn* merupakan *library* visualisasi dalam *Python* yang sangat penting dalam analisis data dan *machine learning* (Hunter, 2007; Waskom, 2021). *Library* visualisasi ini digunakan untuk *plotting waveforms* dari audio signals, *visualizing spectrograms* dan MFCC features, *training history plots* (*loss* dan *accuracy curves*), *confusion matrices* dengan *heatmaps*, *convergence plots* untuk *optimization* methods, dan *comparative bar plots* untuk method *comparison*.

Menurut Géron (2022), visualisasi yang efektif sangat penting untuk memahami behavior model dan mengkomunikasikan hasil penelitian.

2.10.7 KaggleHub

KaggleHub adalah official *Python library* untuk mengakses *Kaggle datasets* dan models secara programmatic (Kaggle, 2024). *Library* ini menyediakan interface sederhana untuk *downloading datasets*, *automatic authentication* menggunakan *Kaggle* API *credentials*, *intelligent caching* untuk menghindari *redundant downloads*, dan *support* untuk dataset *versioning*. Dalam penelitian ini, *KaggleHub* digunakan untuk *downloading dataset* secara efisien, managing dataset versions untuk *reproducibility*, dan *automating* dataset *preparation pipeline.*

2.10.8 Keras Tuner

Keras Tuner adalah pustaka untuk penyesuaian *hyperparameter* yang terintegrasi dengan TensorFlow/Keras, serta menyediakan antarmuka yang berskala dan mudah digunakan untuk mengoptimalkan konfigurasi model (O'Malley et al., 2019). Keras Tuner mengimplementasikan berbagai algoritma optimasi *hyperparameter* mutakhir, termasuk *Random Search*, *Bayesian Optimization*, dan *Hyperband*. Dalam penelitian ini, Keras Tuner digunakan untuk menerapkan dan membandingkan berbagai metode penyesuaian *hyperparameter*, mendefinisikan ruang pencarian *hyperparameter*, mengelola percobaan optimasi beserta hasilnya, serta melakukan integrasi dengan TensorBoard untuk visualisasi. Menurut O'Malley et al. (2019), Keras Tuner merupakan kerangka kerja yang fleksibel dan kuat sehingga secara signifikan menyederhanakan proses penyesuaian *hyperparameter*.

2.11 Penelitian Terdahulu

Penelitian yang pertama adalah penelitian yang dilakukan oleh Maknickas dan Maknickas pada tahun 2017 dengan judul *"Recognition of Normal-Abnormal Phonocardiographic Signals using Deep Convolutional Neural Networks and Mel-Frequency Spectral Coefficients"*. Penelitian ini menerapkan Convolutional Neural Network (CNN) untuk mengklasifikasikan suara jantung normal dan abnormal dengan akurasi tinggi mencapai 86.5% pada dataset PhysioNet/CinC Challenge

2016. Arsitektur yang digunakan terdiri dari empat convolutional layers dengan kernel sizes [32, 64, 128, 256], masing-masing diikuti oleh ReLU activation dan max pooling. Penelitian mereka menunjukkan bahwa CNN mampu secara otomatis mengekstraksi fitur relevan dari sinyal audio tanpa memerlukan feature engineering manual yang ekstensif.

Penelitian terdahulu yang kedua adalah penelitian yang dilakukan oleh Nogueira et al. pada tahun 2019 dengan judul *"Classifying Heart Sounds using Images of Motifs, MFCC and Temporal Features"*. Penelitian ini membuktikan bahwa kombinasi CNN dengan fitur *Mel-Frequency Cepstral Coefficients* (MFCC) dan *temporal features* dapat meningkatkan performa klasifikasi suara jantung. Mereka menggunakan pendekatan multi-modal yang menggabungkan *images of motifs*, MFCC, dan *temporal features* untuk klasifikasi. Sistem mereka mencapai akurasi 88.9%, precision 87.1%, dan recall 91.2% pada PhysioNet dataset. Penelitian ini mendemonstrasikan bahwa kombinasi multiple representations dapat capture complementary information dari PCG signals dan meningkatkan performa klasifikasi secara signifikan.

Penelitian yang ketiga adalah penelitian yang dilakukan oleh Baghel et al. pada tahun 2020 dengan judul *"Automatic Diagnosis of Multiple Cardiac Diseases from PCG Signals using Convolutional Neural Network"*. Penelitian mereka menunjukkan bahwa optimasi *batch size* dan *dropout rate* berkontribusi signifikan terhadap peningkatan F1-score dalam deteksi kelainan jantung. Mereka mengembangkan multi-channel CNN untuk otomatis diagnosis dari *multiple cardiac diseases* (normal, aortic stenosis, mitral regurgitation, mitral stenosis, dan mitral valve prolapse) dengan overall accuracy 94.3%. Hasil penelitian menunjukkan peningkatan performa yang signifikan dengan konfigurasi *hyperparameter* yang tepat, dimana optimasi *dropout rate* dari 0.3 ke 0.5 meningkatkan *F1-score* sebesar 3.2%.

Penelitian yang keempat adalah penelitian yang dilakukan oleh Bergstra dan Bengio pada tahun 2012 dengan judul *"Random Search for Hyper-parameter Optimization"*. Penelitian seminal ini membandingkan grid search dan random search untuk optimasi hyperparameter dalam machine learning. Hasil penelitian

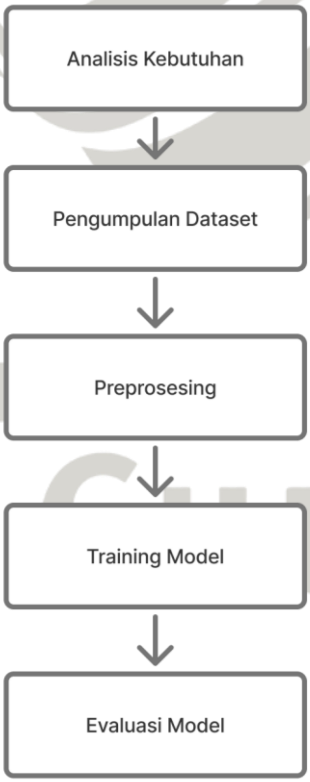
menunjukkan bahwa random search secara signifikan lebih efisien dibandingkan grid search dalam menemukan konfigurasi optimal, terutama ketika beberapa hyperparameter lebih berpengaruh dibandingkan yang lain. Mereka memberikan justifikasi teoretis bahwa jika *objective function* hanya bergantung pada proyeksi berdimensi rendah dari ruang hyperparameter, maka random search akan menghasilkan lebih banyak variasi nilai pada dimensi yang penting.

Penelitian yang kelima adalah penelitian yang dilakukan oleh Snoek et al. pada tahun 2012 dengan judul *"Practical Bayesian Optimization of Machine Learning Algorithms"*. Penelitian ini mengembangkan metode optimasi Bayesian dengan menggunakan *Gaussian Processes* untuk penyesuaian hyperparameter pada algoritma machine learning. Mereka membuktikan bahwa optimasi Bayesian mampu menemukan konfigurasi hyperparameter yang optimal dengan jumlah evaluasi yang jauh lebih sedikit dibandingkan metode konvensional. Eksperimen pada berbagai dataset, termasuk klasifikasi pada CIFAR-10, menunjukkan bahwa optimasi Bayesian dapat mencapai hasil setara dengan *state-of-the-art* hanya dengan sekitar sepuluh kali lebih sedikit evaluasi fungsi dibandingkan *random search*. Penelitian ini juga memperkenalkan *Expected Improvement* sebagai fungsi akuisisi yang efektif untuk memandu proses pencarian.

Penelitian yang keenam dan menjadi landasan arsitektur dalam penelitian ini adalah penelitian yang dilakukan oleh Rubin et al. pada tahun 2016 dengan judul "*Classifying Heart Sound Recordings using Deep Convolutional Neural Networks and Mel-Frequency Cepstral Coefficients*". Penelitian ini meraih peringkat ke-8 dari 48 tim pada *PhysioNet/CinC Challenge* 2016 dengan *overall score* 84.8%, *sensitivity* 76.5%, dan *specificity* 93.1%. Arsitektur yang digunakan terdiri dari dua convolutional layers dengan 64 kernels dan kernel sizes (2×20) dan (2×10), diikuti oleh max pooling layers, serta dua fully connected layers dengan 1024 dan 512 units. Arsitektur ini dijadikan kerangka awal dalam penelitian ini karena strukturnya yang sederhana namun efektif, terdokumentasi dengan baik, dan telah terbukti memberikan hasil yang konsisten pada dataset *PhysioNet 2016*. Dengan menggunakan arsitektur ini sebagai fondasi, penelitian ini akan fokus pada eksplorasi dan perbandingan berbagai metode optimasi *hyperparameter* untuk menemukan konfigurasi optimal yang dapat meningkatkan performa klasifikasi

BAB III
ANALISA DAN PERANCANGAN MODEL

Penelitian ini didasarkan pada metode klasifikasi sinyal biomedis dengan pendekatan *deep learning*. Objek penelitian berupa rekaman audio phonocardiogram (PCG) yang merepresentasikan suara jantung manusia dalam format file .wav. Uji coba dilakukan dengan arsitektur Convolutional Neural Network (CNN) 1D yang dioptimasi menggunakan empat metode hyperparameter tuning, yaitu Grid Search, Random Search, Bayesian Optimization, dan Genetic Algorithm. Proses penelitian dibagi menjadi beberapa tahapan sebagaimana ditunjukkan pada Gambar 3.1.



Gambar 3.1 Tahap Penelitian

3.1 Analisis Kebutuhan

Pada tahap ini dilakukan identifikasi terhadap kebutuhan yang diperlukan dalam pelaksanaan penelitian agar dapat berjalan secara sistematis dan optimal. Analisis kebutuhan mencakup perangkat lunak (*software*), perangkat keras (*hardware*), serta literatur pendukung.

Software yang digunakan antara lain *Python 3.8* atau lebih tinggi sebagai bahasa pemrograman utama, serta *library* Python seperti *TensorFlow/Keras* untuk pembangunan dan pelatihan model deep learning, *Librosa* untuk pemrosesan sinyal audio dan ekstraksi fitur, *NumPy* untuk pengolahan data numerik dan operasi array, *Pandas* untuk manajemen data dan analisis, *Scikit-learn* untuk splitting data dan perhitungan metrik evaluasi, *Matplotlib* dan *Seaborn* untuk visualisasi data dan hasil eksperimen, serta *Keras Tuner* untuk implementasi berbagai metode hyperparameter optimization. Selain itu, digunakan juga *KaggleHub* untuk mengunduh dataset secara programmatic dari platform Kaggle.

Untuk lingkungan pengembangan, digunakan *Integrated Development Environment (IDE)* berupa *Visual Studio Code* yang menyediakan fitur untuk eksperimen interaktif dan visualisasi hasil secara real-time. *TensorBoard* juga digunakan untuk monitoring proses training dan visualisasi grafik performa model.

Perangkat keras yang digunakan dalam penelitian ini adalah satu unit komputer workstation yang berlokasi di Laboratorium Aiditech. Perangkat ini dilengkapi dengan GPU NVIDIA GeForce RTX 3060 yang memiliki VRAM sebesar 12GB, prosesor dengan kecepatan 4.64 GHz, serta RAM sebesar 16GB. Spesifikasi *hardware* ini dipilih untuk mendukung proses training model deep learning yang memerlukan komputasi intensif, khususnya dalam proses hyperparameter optimization yang melibatkan ratusan *trials* evaluasi model.

3.2 Pengumpulan Dataset

Dataset dalam penelitian ini diperoleh dari platform *Kaggle* dengan nama *Heart Sound Database*. Dataset ini merupakan koleksi rekaman *phonocardiogram* yang berasal dari *PhysioNet/Computing in Cardiology Challenge 2016*, sebuah kompetisi ilmiah yang berfokus pada klasifikasi suara jantung normal dan abnormal.

PhysioNet Challenge 2016 merupakan kompetisi internasional yang bertujuan mengembangkan algoritma untuk *automatic classification* dari rekaman *phonocardiogram*. Dataset yang dikumpulkan dalam *challenge* ini telah melalui proses verifikasi oleh tenaga medis profesional dan *quality control* yang ketat, sehingga memiliki *ground truth labels* yang reliable. Dataset ini telah digunakan secara luas dalam penelitian-penelitian terdahulu terkait klasifikasi *PCG* menggunakan *machine learning* dan *deep learning*, menjadikannya *benchmark standard* untuk evaluasi performa model.

Data dikumpulkan dalam format audio *.wav (Waveform Audio File Format)* yang merupakan format standar untuk penyimpanan data audio tanpa kompresi *lossy*. Proses pengambilan dilakukan menggunakan berbagai jenis *digital stethoscopes* di berbagai lokasi anatomis (*mitral, tricuspid, aortic,* dan *pulmonic areas*) pada subjek dengan berbagai kondisi kardiovaskular. Keberagaman dalam perangkat *recording* dan lokasi *auskultasi* ini memberikan variabilitas yang *realistic*, membuat model yang dilatih pada dataset ini lebih *robust* dan *applicable* untuk kondisi klinis yang sebenarnya.

Dataset ini mencakup dua kategori utama, yaitu suara jantung normal (*healthy/normal*) dan suara jantung abnormal (*unhealthy/abnormal*). Kategori abnormal mencakup berbagai jenis kelainan kardiovaskular seperti *heart murmurs* yang disebabkan oleh *valvular disease*, *arrhythmias* atau *irregular heart rhythms*, dan kondisi patologis lainnya yang terdeteksi dari karakteristik suara jantung yang abnormal. Klasifikasi ini dilakukan berdasarkan *auskultasi* oleh *cardiologists* dan dikonfirmasi dengan *diagnostic tests* seperti *echocardiography*.

3.3 *Preprocessing* Data

Setelah *dataset* yang merupakan data mentah terkumpul, dilakukan serangkaian tahapan *preprocessing* untuk mempersiapkan data dalam format yang sesuai untuk pelatihan model CNN. Metodologi *preprocessing* yang digunakan dalam penelitian ini mengikuti pendekatan yang telah divalidasi oleh Rubin et al. (2016) yang terbukti efektif pada PhysioNet/CinC Challenge 2016.

Gambar 3.2 Alur *Preprocessing* Data

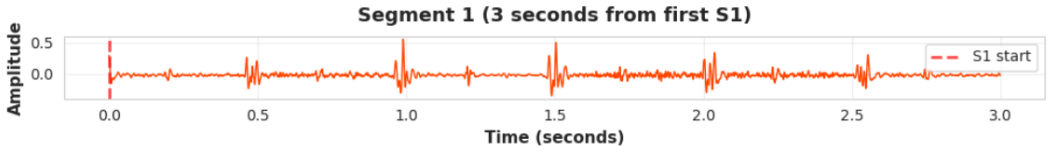
Tahapan pertama adalah *audio loading* dan *resampling***.** Rekaman audio PCG dari PhysioNet/CinC Challenge 2016 dataset memiliki *sampling rate* yang bervariasi. Seluruh rekaman di-*resample* ke *sampling rate uniform* 2000 Hz menggunakan *librosa resampling function*. Pemilihan *sampling rate* 2000 Hz didasarkan pada dua pertimbangan utama yaitu komponen frekuensi penting dari suara jantung normal dan abnormal berada di bawah 1000 Hz, sehingga berdasarkan *Nyquist-Shannon sampling theorem*, sampling rate 2000 Hz memadai untuk merepresentasikan seluruh informasi frekuensi yang relevan tanpa kehilangan informasi atau aliasing.

Tahapan kedua adalah *heart sound segmentation* menggunakan *algoritma Springer*. Segmentasi otomatis dilakukan untuk mengidentifikasi lokasi fundamental *heart sounds* (S1 dan S2) serta *systolic* dan *diastolic intervals* dalam setiap rekaman. *Algoritma Springer* menggunakan *Hidden Semi-Markov Model*

(HSMM) yang telah dilatih untuk mengenali empat *state* dalam *cardiac cycle*: S1, systole, S2, dan diastole. Algoritma bekerja dengan mengekstraksi *envelope* dari sinyal PCG menggunakan *Hilbert transform*, kemudian mengaplikasikan HSMM untuk melakukan *state decoding* berdasarkan durasi dan amplitudo karakteristik dari setiap suara detak jantung. Output dari proses ini adalah anotasi yang menandai *onset time* dari setiap S1 dan S2 dalam rekaman, yang akan digunakan sebagai *reference points* untuk ekstraksi segmen.

Gambar 3.3 Visualisasi Segmentasi Audio

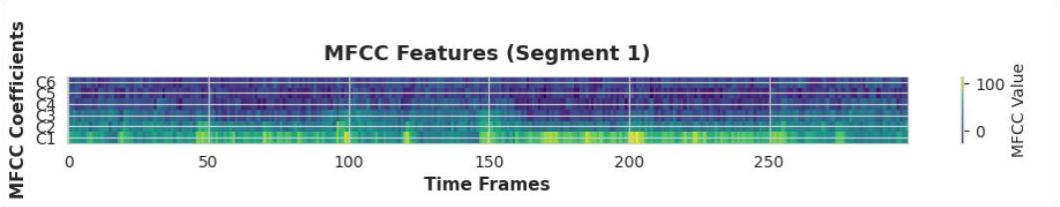
Tahapan ketiga adalah *segment extraction***.** Setiap rekaman dibagi menjadi beberapa *overlapping segments* dengan durasi *fixed* 3 detik. Setiap segmen diekstraksi dimulai tepat pada *onset* S1 yang telah diidentifikasi oleh *algoritma Springer*. Dengan *sampling rate* 2000 Hz, setiap segmen 3 detik menghasilkan *array* dengan panjang 6,000 *samples*. Pemilihan durasi 3 detik didasarkan pada analisis bahwa durasi ini cukup panjang untuk menangkap beberapa *cycle* detak jantung pada *heart rate* normal berkisar 60-100 BPM, durasi 3 detik akan mencakup sekitar 3-5 *cycle* detak jantung. Segmentasi dilakukan secara *overlapping* dimana setiap S1 yang terdeteksi menjadi starting point untuk satu segmen, sehingga satu rekaman panjang dapat menghasilkan beberapa *training sample.*



Gambar 3.4 Visualisasi Ekstraksi Segmentasi

Tahapan keempat adalah MFCC *feature extraction***.** Proses ekstraksi MFCC dilakukan dalam beberapa sub-tahap pertama, *Short-Time Fourier Transform* (STFT) diterapkan pada setiap segmen dengan *frame size* 25 ms dan *hop length* 10

ms, menghasilkan 300 *time frames* untuk segmen 3 detik. *Window function* yang digunakan adalah *Hamming window* untuk mengurangi *spectral leakage*. Kedua, *power spectrum* dari setiap *frame* dikonversi ke *mel scale* menggunakan 40 *triangular mel kernelbanks* yang didistribusikan secara logaritmik antara 0 Hz hingga 1000 Hz. Ketiga, logarithm diterapkan pada *mel-spectrum* untuk merepresentasikan presepsi suara manusia yang bersifat logarithmic. Keempat, *Discrete Cosine Transform* (DCT) diaplikasikan untuk mengkompresi informasi dan menghasilkan 13 MFCC coefficients. Mengikuti Rubin et al., hanya 6 koefisien MFCC pertama (MFCC 1 sampai MFCC 6) yang digunakan, mengabaikan *zeroth coefficient* (yang merepresentasikan total energy) dan *higher-order coefficients* (7-12). Pemilihan ini didasarkan pada observasi bahwa *lower-order coefficients* menangkap *broad spectral envelope* yang paling relevan untuk klasifikasi heart sounds, sementara *higher-order coefficients* cenderung menangkap *fine spectral details* dan *noise* yang kurang informatif dan dapat menyebabkan *overfitting*. Hasil akhir dari tahap ini adalah representasi 2D dengan dimensi 6×300 untuk setiap segmen, dimana sumbu pertama merepresentasikan 6 MFCC *coefficients* dan sumbu kedua merepresentasikan 300 time frames, membentuk "*heat map*" yang dapat diperlakukan sebagai *grayscale* image untuk input ke CNN.



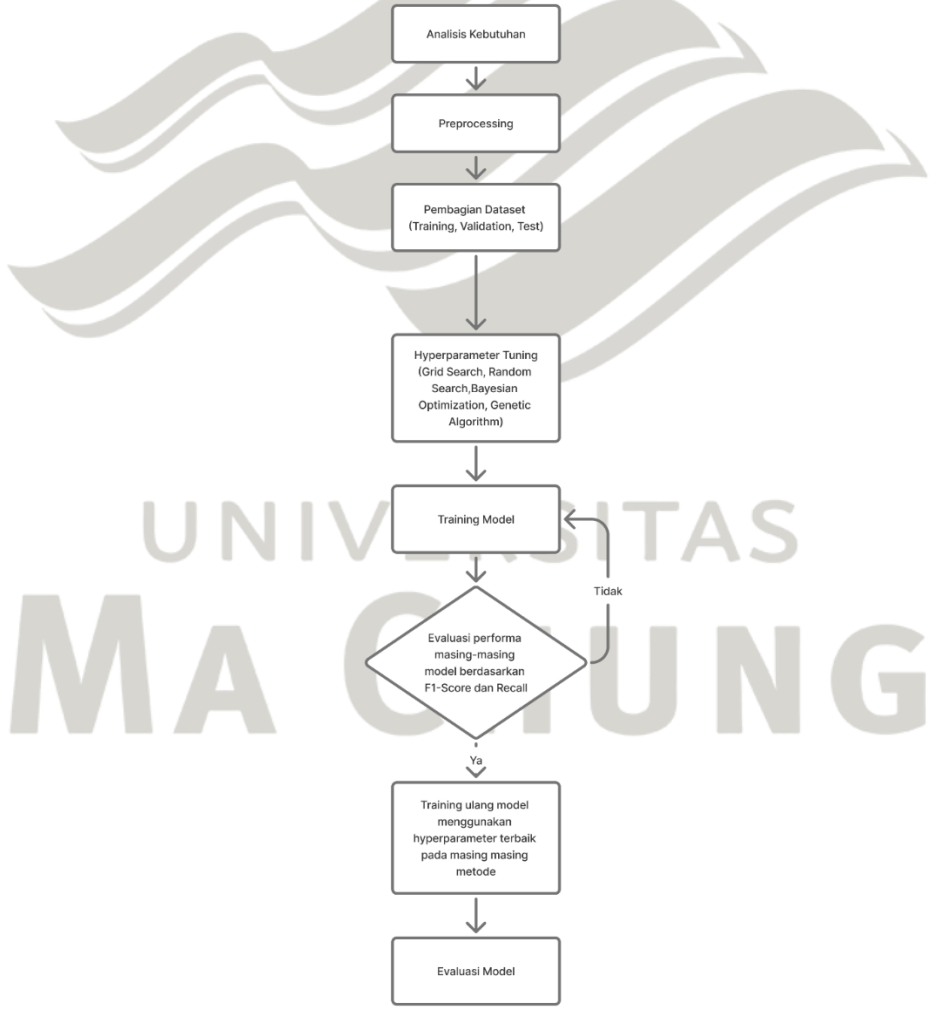
Gambar 3.5 Visualisasi Ekstraksi Fitur MFCC

Tahapan kelima adalah standarisasi**.** *Z-score normalization* diterapkan pada MFCC *features* untuk memastikan bahwa setiap *coefficient* memiliki *mean* nol dan *standard deviation* satu. Normalisasi dilakukan secara independent untuk setiap MFCC *coefficient* dengan menghitung *mean* dan *standard deviation* pada semua rekaman *frame* dan semua segmen dalam *training set*, kemudian mentransformasi setiap nilai menggunakan formula $z = (x - \mu) / \sigma$.

3.4 Training Model

Penelitian ini menggunakan arsitektur *CNN 2D (Two-Dimensional Convolutional Neural Network)* dengan pendekatan optimisasi *hyperparameter*, yaitu membandingkan kinerja dari empat metode optimisasi berbeda dalam menemukan konfigurasi terbaik untuk klasifikasi fonokardiogram. Keempat metode yang digunakan adalah *Grid Search*, *Random Search*, *Bayesian Optimization*, dan *Genetic Algorithm*.

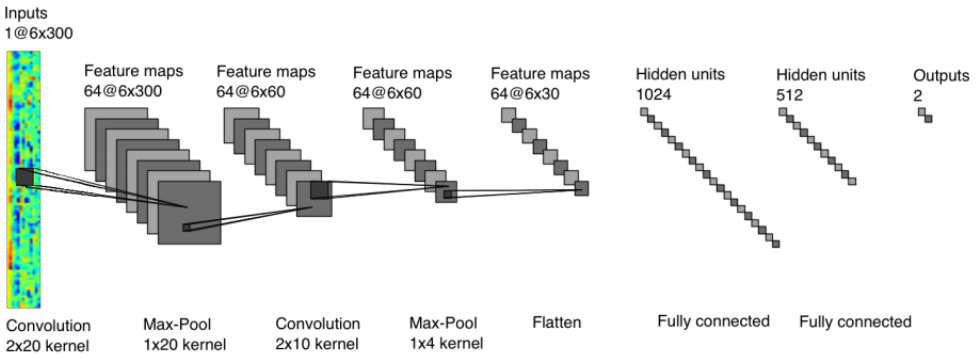
Berikut adalah *diagram flow* dalam Training model:



Gambar 3.6 Diagram Flow Training model

3.4.1 Arsitektur CNN 2D

Arsitektur CNN 2D yang digunakan mengadaptasi pendekatan Rubin et al. yang meraih peringkat ke-8 dalam *PhysioNet/Computing in Cardiology Challenge* 2016. Berikut adalah visualisasi arsitekturnya.



Gambar 3.7 Visualisasi arsitektur Rubin et al.

Arsitektur ini menerima input berupa MFCC *heat map* berukuran 6×300 dan menghasilkan klasifikasi biner untuk menentukan apakah suara jantung normal atau abnormal.

Struktur jaringan terdiri dari *dua convolutional layer* yang masing-masing diikuti oleh *max-pooling layer*. *Convolutional layer* pertama menggunakan 64 *kernel* dengan *kernel* 2×20 dan *same padding*, kemudian dilanjutkan dengan *max-pooling* 1×20 dengan *stride* 5 yang menghasilkan 64 *feature maps* berukuran 6×60. *Convolutional layer* kedua menerapkan 64 *kernel* dengan *kernel* 2×10 dan *same padding*, diikuti *max-pooling* 1×4 dengan *stride* 2 yang mereduksi setiap *feature map* menjadi ukuran 6×30.

Setelah tahap konvolusi, dilakukan operasi *flattening* yang mengubah 64 *feature maps* (6×30) menjadi vektor satu dimensi berukuran 11.520. Vektor ini kemudian diproses melalui dua *fully connected layer* dengan 1.024 dan 512 *hidden units* secara berurutan, sebelum akhirnya menghasilkan *output* klasifikasi biner

3.4.2 Hyperparameter Search Space

Tabel 3.1 Hyperparameter Search Scope

Kategori	*Hyperparameter*	*Range/Option*
Architecture Parameters	*Kernel Conv Layer 1*	[64, 80, 96]
	Kernel Conv Layer 2	[48, 64, 80]
	Kernel Size Conv Layer 1	[(2,20), (3,20), (2,25), (3,25)]
	Kernel Size Conv Layer 2	[(2,10), (3,10), (2,12), (3,12)]
	Dense Unit Layer 1	[768, 1024, 1536]
	Dense Unit Layer 2	[384, 512, 768]
Regularization Parameters	*Drop Out Rate Layer 1*	[0.2, 0.4, 0.6, 0.8]
	Drop Out Rate Layer 2	[0.2, 0.4, 0.6, 0.8]
Training Parameters	*Learning Rate*	[0.0001, 0.001, 0.01]
	Batch Size	[64, 128, 256]
	Optimizer	[Adam, Nadam, Adamax]
Total Search Space	3x3x4x4x3x3x4x4x3x3x3	559.872

Pemilihan jumlah kernel untuk setiap blok konvolusi mengikuti prinsip *progressive feature abstraction* yang telah mapan dalam computer vision oleh Krizhevsky dkk. (2012) dan Simonyan & Zisserman (2015), kemudian diadaptasi untuk pemrosesan sinyal audio oleh Piczak (2015) dalam penelitian environmental sound classification menggunakan CNN. Rentang untuk blok konvolusi pertama dipilih antara 64 hingga 96 kernel sebagai titik awal untuk ekstraksi fitur tingkat rendah dari MFCC *heat map*. Hershey dkk. (2017) dalam penelitian CNN untuk *audio event detection* menunjukkan bahwa 64-96 kernel optimal untuk lapisan awal dalam pemrosesan representasi spektral audio. Rentang untuk blok konvolusi kedua dipilih antara 48 hingga 80 kernel. Berbeda dari arsitektur *vision* yang umumnya meningkatkan jumlah kernel secara progresif, penelitian Pons dkk. (2017) tentang *end-to-end learning for music audio* menunjukkan bahwa untuk input 2D *time-*

frequency representation, jumlah kernel pada lapisan kedua tidak selalu harus lebih besar dari lapisan pertama, karena *pooling operation* sudah mengurangi dimensi spatial dan meningkatkan *receptive field*.

Pemilihan rentang untuk ukuran *kernel* didasarkan pada karakteristik *time-frequency* dari representasi MFCC yang telah dianalisis dalam penelitian Zhang dkk. (2017) tentang *deep learning for environmental sound classification*. Dengan input shape (6, 300) dimana sumbu pertama merepresentasikan 6 MFCC *coefficients* dan sumbu kedua merepresentasikan 300 *time frames*, ukuran *kernel* harus disesuaikan dengan struktur data ini. Dimensi pertama *kernel* dirancang untuk menangkap korelasi antar MFCC *coefficients* yang berdekatan. Lee dkk. (2009) dalam penelitian *Convolutional Deep Belief Networks* menunjukkan bahwa *kernel* kecil pada dimensi *frequency* (2-3) efektif untuk menangkap *local frequency patterns* dalam *spektrogram*. Dimensi kedua *kernel* berkaitan dengan resolusi temporal yang disesuaikan dengan karakteristik *cardiac cycle*.

Setelah *flattening operation* dari *output convolutional block* terakhir, dimensi *flatten* adalah sekitar 11,520-14,400 features tergantung konfigurasi kernel. Chollet (2017) merekomendasikan bahwa *fully connected layer* pertama setelah *convolutional blocks* sebaiknya memiliki kapasitas 5-15% dari *input dimension* untuk mencegah information *bottleneck* sambil memberikan *dimensionality reduction*. Rentang 768-1536 units (sekitar 5-13% dari ~11,520 *input features*) mengikuti prinsip ini, dengan Potes dkk. (2016) dalam solusi pemenang PhysioNet/CinC Challenge 2016 menggunakan 1024 units untuk *heart sound classification. Dense layer* kedua dengan rentang 384-768 units (sekitar 50% dari dense layer pertama) memberikan *progressive dimensionality reduction* menuju *output layer*.

Rentang *learning rate* dipilih mengikuti eksplorasi skala logaritmik: 10^{-4}, 10^{-3}, 10^{-2}, sebagaimana direkomendasikan oleh Bergstra & Bengio (2012) yang menunjukkan bahwa pengambilan sampel *log-uniform* lebih efektif untuk *hyperparameter learning rate*. Rentang ini mencakup dari *learning rate* konservatif (0,0001) yang menjamin konvergensi stabil namun berpotensi lambat, hingga

learning rate agresif (0,01) yang memberikan konvergensi lebih cepat tetapi dengan risiko melampaui atau ketidakstabilan. Kingma & Ba (2015) dalam makalah tentang *optimizer Adam* merekomendasikan 0,001 sebagai *learning rate default* yang baik, dan rentang yang dipilih mencakup nilai *default* ini sambil memungkinkan eksplorasi ke kedua arah. *Learning rate* yang terlalu kecil dapat menyebabkan konvergensi lambat dan kemungkinan terjebak di minima lokal, sedangkan *learning rate* yang terlalu besar dapat menyebabkan divergensi atau osilasi di sekitar optima. Rentang 0,0001-0,01 menyediakan ruang eksplorasi yang wajar untuk menemukan *learning rate* optimal untuk tugas spesifik.

Rentang ukuran *batch* dipilih sebagai pangkat 2 (16, 32, 64, 128) untuk efisiensi komputasi GPU dan penyelarasan memori. Masters & Luschi (2018) dalam penelitian tentang pelatihan *batch* kecil menunjukkan bahwa *batch* kecil (16-32) memberikan generalisasi yang lebih baik karena gradien yang bising bertindak sebagai regularisasi implisit, membantu model menghindari minima tajam yang generalisasinya buruk. Sebaliknya, Keskar dkk. (2017) menunjukkan bahwa *batch* besar (64-128) memungkinkan pelatihan lebih cepat karena pemanfaatan GPU yang lebih baik dan estimasi gradien yang lebih stabil, tetapi dapat konvergen ke minima tajam. Rentang 16-128 memungkinkan metode optimisasi untuk mengeksplorasi pertukaran antara kemampuan generalisasi (menguntungkan *batch* lebih kecil) dan efisiensi komputasi (menguntungkan *batch* lebih besar).

Inklusi tiga jenis optimizer (Adam, Adamax, Nadam) memberikan cakupan komprehensif dari strategi optimisasi berbasis adaptif yang berbeda. Optimizer Adam (Kingma & Ba, 2015) menggunakan *learning rate* adaptif dengan menggabungkan keunggulan dari RMSprop dan momentum, menjadikannya pilihan *default* yang baik dan tangguh di berbagai tugas. Adamax, yang juga diperkenalkan oleh Kingma & Ba (2015), merupakan varian dari Adam yang berbasis pada norma tak hingga (L_∞). Metode ini dirancang untuk memberikan stabilitas yang lebih tinggi pada data dengan pembaruan parameter yang jarang (*sparse*) atau *noisy*, serta sering kali lebih *robust* terhadap perubahan skala gradien dibandingkan Adam standar. Sementara itu, Nadam (Dozat, 2016) menggabungkan mekanisme Adam dengan *Nesterov Accelerated Gradient* (NAG).

Integrasi momentum Nesterov ini memungkinkan optimizer untuk memperhitungkan arah langkah selanjutnya sebelum menghitung gradien, yang secara teoritis dan praktis sering menghasilkan konvergensi yang lebih cepat dan kemampuan generalisasi yang lebih baik. Ruder (2016) dalam survei komprehensif tentang metode optimisasi menunjukkan bahwa tidak ada optimizer tunggal terbaik untuk semua tugas, dan kinerja bergantung pada karakteristik dari masalah spesifik. Oleh karena itu, cakupan dari varian metode adaptif modern ini (Adam, Adamax, Nadam) dipilih untuk memaksimalkan peluang menemukan konfigurasi konvergensi terbaik.

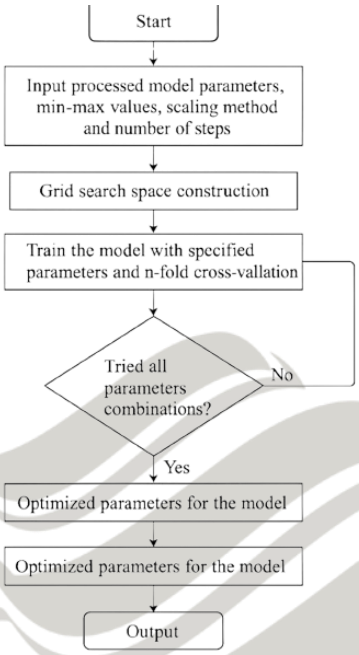
Kombinasi total teoritis dari semua *hyperparameter* adalah 3 × 3 × 4 × 4 × 3 ×3 × 4 × 4 × 3 × 3 × 3 = 559.872 kombinasi

3.4.3 Metode Grid Search

Grid search merupakan metode HPO yang paling sederhana dan *straightforward* dengan pendekatan *brute-force* (Bergstra & Bengio, 2012). Metode ini bekerja dengan mendefinisikan grid dari nilai-nilai diskrit untuk setiap *hyperparameter*, kemudian secara exhaustive mencoba setiap kombinasi yang mungkin dalam grid tersebut (Liashchynskyi & Liashchynskyi, 2019).

Kelebihan *grid search* adalah kesederhanaannya dan *guarantee* bahwa kombinasi terbaik dalam *grid* yang didefinisikan akan ditemukan (Mantovani et al., 2015). Metode ini juga mudah untuk diparalelisasi karena setiap evaluasi independen satu sama lain. Namun, kekurangan utama *grid search* adalah *computational cost* yang sangat tinggi, terutama ketika jumlah *hyperparameter* dan *range* nilainya besar (Bergstra & Bengio, 2012). Kompleksitas waktu *grid search* meningkat secara eksponensial dengan jumlah *hyperparameter* (*curse of dimensionality*), sehingga menjadi kurang efektif untuk *high-dimensional search spaces* (Claesen & De Moor, 2015).

Berikut adalah *diagram flow* dari metode *grid search*:



Gambar 3.8 Diagram Flow Grid Search

3.4.3.1 Parameter dan Strategi Optimization

Table 3.2 Parameter dan Strategi *Optimization Grid Search*

Komponen	**Nilai/Strategi**
Search Strategy	*Reduced grid* dengan *selective parameters*
Grid Size	*Maximum* 200 *evaluations*
Selection Method	*Cartesian product*
Evaluation Metric	*F1-Score*
Loss Function	*Binary Crossentropy*
Epochs	100 (maksimum)
Early Stopping	*Monitoring: val_loss, Patience: 15, Restore best: True*

Table 3.2 Lanjutan Parameter dan Strategi *Optimization Grid Search*

Komponen	**Nilai/Strategi**
LR Scheduler	*ReduceLROnPlateau (Factor: 0.2, Patience: 7, Min LR: 1×10^{-5})*
Checkpointing	*ModelCheckpoint (validation accuracy tertinggi)*
Validation Split	20% dari *training data (stratified)*
Random Seed	42 (*reproducibility*)

3.4.3.2 Framework Implementasi

Framework hyperparameter optimization menggunakan *Keras Tuner* 1.3.5 dengan *GridSearch module*. *Training environment* menggunakan *TensorFlow* 2.13.0 dengan *Keras API*. *Development* dilakukan di *Visual Studio Code* dengan *workstation local*.

3.4.3.3 Computional Complexity

Kompleksitas komputasi *Grid Search* dianalisis berdasarkan jumlah evaluasi, waktu *training*, dan kebutuhan *resource*. Dari 559.872 kombinasi teoritis yang memerlukan waktu kurang lebih 11,7 tahun, maka dilakukan pengurangan menjadi 200 *trials* menggunakan metode *Reduce Grid* dengan waktu rata-rata 11 menit per *trial* menggunakan *GPU NVIDIA RTX 3060*. Total waktu komputasi adalah 36,7 jam (±1,5 hari) untuk eksekusi *sequential*, yang dapat dipercepat hingga 9 jam dengan 4 *GPUs parallel*. Kebutuhan memori per *trial* mencapai 500 MB dengan *peak usage*, sementara *GPU* 12 GB *VRAM* yang tersedia mampu menjalankan 4–5 *trials parallel*. Total *storage requirement* untuk seluruh proses adalah 1–2 GB.

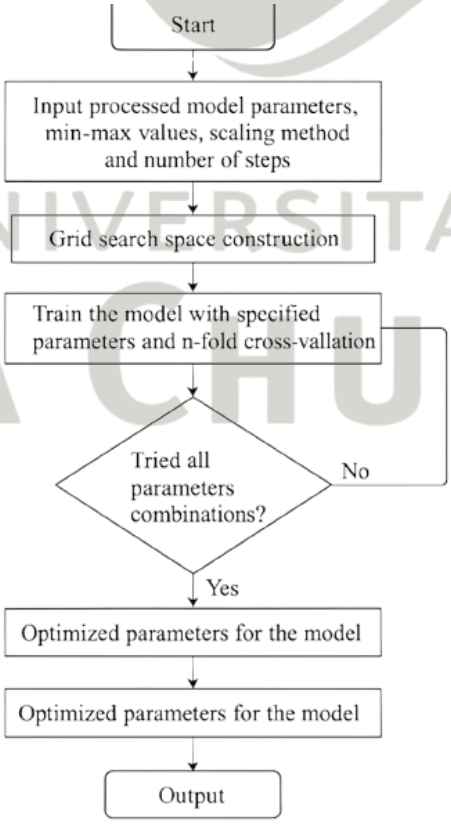
3.4.4 Metode Random Search

Random search merupakan alternatif yang lebih efisien dibandingkan *grid search*, di mana metode ini secara *random* mengambil sampel kombinasi *hyperparameter* dari distribusi yang telah ditentukan (Bergstra & Bengio, 2012). Alih-alih mencoba setiap kombinasi secara sistematis, *random search* mengeksplorasi *hyperparameter space* dengan *sampling* acak.

Bergstra dan Bengio (2012) dalam penelitian mereka menunjukkan bahwa *random search* secara signifikan lebih efisien daripada *grid search* untuk *hyperparameter optimization*. Keunggulan utama *random search* adalah probabilitas yang lebih tinggi untuk menemukan kombinasi yang baik dalam budget komputasi yang sama, terutama ketika beberapa *hyperparameter* lebih penting daripada yang lain. *Random search* tidak membuang *resources* untuk mencoba kombinasi yang sistematis tetapi kurang penting, sehingga dapat mengeksplorasi lebih banyak nilai untuk *hyperparameter* yang *critical* (Bergstra & Bengio, 2012).

Pada *Random Search* menggunakan sampling method *Uniform Sampling* dan *Uniform Sampling* Digunakan untuk *categorical* dan *continuous parameters* yang tidak memiliki skala preferensi tertentu, seperti *batch size*, *optimizer*, *dense units*, *learning rate* dan *dropout rate*. Setiap nilai dalam rentang yang ditentukan memiliki peluang sama untuk dipilih

Berikut adalah *Diagram Flow* dari metode *Random Search*:



Gambar 3.9 Diagram flow Random Search

3.4.4.1 Parameter dan Strategi Optimization

Table 3.3 Parameter dan Strategi *Optimization Random Search*

Komponen	**Nilai/Strategi**
Search Strategy	*Random sampling* dari *distributions*
Number of Iterations	200 *random evaluations*
Sampling Method	*Uniform* untuk *categorical*
Evaluation Metric	*F1-Score*
Loss Function	*Binary Crossentropy*
Epochs	100 (maksimum)
Early Stopping	*Monitoring: val_loss, Patience: 15, Restore best: True*
LR Scheduler	*ReduceLROnPlateau (Factor: 0.2, Patience: 7, Min LR: 1×10^{-5})*
Checkpointing	*ModelCheckpoint (validation accuracy tertinggi)*
Validation Split	20% dari *training data (stratified)*
Random Seed	42 (*reproducibility*)

3.4.4.2 Framework Implementasi

Framework hyperparameter optimization menggunakan *Keras Tuner* 1.3.5 dengan *RandomSearch module*. *Training environment* menggunakan *TensorFlow* 2.13.0 dengan *Keras* API. *Development* dilakukan di *Visual Studio Code* dengan *workstation lokal* .

3.4.4.3 Computational Complexity

Kompleksitas komputasi *Random Search* menggunakan *fixed iterations* sebanyak 200 *trials* dengan waktu rata-rata 10 menit per *trial* pada *GPU NVIDIA*

RTX 3060. Total waktu komputasi adalah 16,7 jam (±0,7 hari) untuk eksekusi *sequential*. Kebutuhan *resource* identik dengan *Grid Search*, dengan *peak memory* 500 MB per *trial* dan total *storage requirement* 500 MB hingga 1 GB.

3.4.5 Metode Bayesian Optimization

Bayesian optimization adalah metode HPO yang lebih *sophisticated* dan *intelligent*, yang membangun model probabilistik dari fungsi objektif (performa model) terhadap *hyperparameter* yang diuji (Shahriari et al., 2016). Metode ini menggunakan *prior knowledge* dari eksperimen sebelumnya untuk memandu pencarian *hyperparameter* selanjutnya, sehingga dapat menemukan konfigurasi optimal dengan jumlah evaluasi yang lebih sedikit (Snoek et al., 2012).

Bayesian optimization bekerja dengan membangun *surrogate model*, biasanya *Gaussian Process* (GP), yang memodelkan distribusi probabilitas dari fungsi objektif (Mockus, 1975). Model ini kemudian digunakan untuk menghitung *acquisition function* yang menentukan *hyperparameter* mana yang paling menjanjikan untuk dievaluasi selanjutnya (Brochu et al., 2010). Proses ini melibatkan *trade-off* antara *exploitation* (mengeksplorasi region yang sudah diketahui menghasilkan performa baik) dan *exploration* (mencari region baru yang belum pernah diuji) (Shahriari et al., 2016).

Acquisition functions yang umum digunakan meliputi *Expected Improvement* (EI), *Probability of Improvement* (PI), dan *Upper Confidence Bound* (UCB), yang masing-masing memiliki karakteristik *exploration-exploitation* yang berbeda (Snoek et al., 2012). Keunggulan utama *Bayesian optimization* adalah efisiensi komputasinya yang *superior*, karena metode ini secara intelligent memilih kombinasi *hyperparameter* yang paling menjanjikan dan menghindari evaluasi pada *region* yang *unlikely* menghasilkan *improvement* (Frazier, 2018).

Pada *Bayesian Optimization* menggunakan *Expected Improvemen*t (EI) yang merupakan salah satu *acquisition function* yang paling populer dan efektif dalam *Bayesian optimization* (Jones et al., 1998). EI mengukur *expected value* dari improvement yang dapat diperoleh dari mengevaluasi *hyperparameter* tertentu dibandingkan dengan *best observed value* saat ini. Metode ini secara natural

menyeimbangkan *exploration* dan *exploitation* dengan mempertimbangkan baik mean prediction maupun uncertainty dari surrogate model.

Expected Improvement didefinisikan sebagai ekspektasi dari *improvement* di atas nilai terbaik yang telah diamati sejauh ini, yang dinotasikan sebagai $f(x^+)$, di mana x^+ adalah konfigurasi hyperparameter terbaik saat ini. Untuk setiap kandidat hyperparameter x, EI dihitung dengan formula:

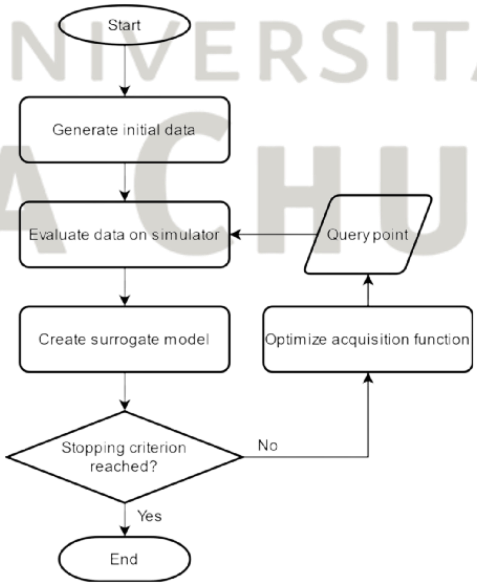
$$EI(x) = E[max(0, f(x) - f(x^+))]$$

Dengan asumsi *Gaussian Process* sebagai *surrogate model*, yang memberikan distribusi normal untuk setiap prediksi dengan mean μ(x) dan standard deviation σ(x), formula EI dapat dihitung secara closed-form:

$$EI(x) = (\mu(x) - f(x^+) - \xi) \cdot \Phi(Z) + \sigma(x) \cdot \varphi(Z)$$

Yang dimana $(\mu(x) - f(x^+) - \xi) \cdot \Phi(Z)$ merepresentasikan expected improvement berdasarkan mean prediction. Term ini mendorong algoritma untuk memilih hyperparameter di region yang diprediksi memiliki performa tinggi berdasarkan model saat ini.

Berikut adalah *Diagram Flow* dari metode *Bayesian Optimization*:



Gambar 3.10 Diagram Flow Bayessian Optimization

3.4.5.1 Parameter dan Strategi Optimization

Tabel 3.4 Parameter dan Strategi *Bayessian Optimization*

Komponen	Nilai / Strategi
Search Strategy	*Sequential model-based optimization*
Surrogate Model	*Gaussian Process* (kernel: Matern v=2.5)
Acquisition Function	*Expected Improvement (EI)*
Initial Points	10 *random initializations*
Guided Iterations	190 *sequential evaluations*
Total Iterations	200 *evaluations*
Evaluation Metric	*F1-Score*
Loss Function	*Binary Crossentropy*
Epochs	100 (maksimum)
Early Stopping	*Monitoring: val_loss, Patience: 15, Restore best: True*
LR Scheduler	*ReduceLROnPlateau (Factor: 0.2, Patience: 7, Min LR: 1×10^{-5})*
Checkpointing	*ModelCheckpoint (validation accuracy tertinggi)*
Validation Split	20% dari *training data (stratified)*

3.4.5.2 Framework Implementasi

Framework hyperparameter optimization menggunakan *Keras Tuner* 1.3.5 dengan *BayesianOptimization module*. *Training environment* menggunakan *TensorFlow* 2.13.0 dengan *Keras API. Development* dilakukan di *Visual Studio Code* dengan *workstation local*.

3.4.5.3 Computational Complexity

Kompleksitas komputasi pada *Bayesian Optimization* terdiri dari 50 *trials* (5 *random* + 45 *guided*) dengan waktu rata-rata sekitar 10 menit per *trial*. Total waktu komputasi mencapai ±8,3 jam untuk eksekusi *sequential*, dan dapat dipercepat menjadi sekitar 7–8 jam dengan *parallel execution*, meskipun percepatan terbatas karena sifat algoritma yang bersifat sekuensial. Overhead komputasi dari operasi *Gaussian Process (GP)* relatif kecil, dengan waktu tambahan sekitar 10–15 detik per iterasi (sekitar 1,5% dari total waktu per *trial*). Kebutuhan memori mencakup 50–100 MB untuk model GP dan sekitar 500 MB per *trial*, dengan total *storage* 300–500 MB.

3.4.6 Metode Genetic Algorithm

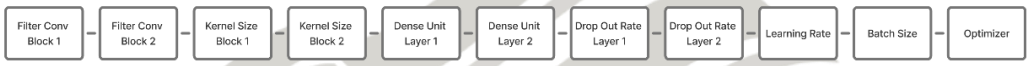
Genetic algorithm (GA) adalah metode HPO yang terinspirasi dari proses evolusi biologis, menggunakan mekanisme seperti *selection, crossover*, dan *mutation* untuk mengeksplorasi *hyperparameter space* (Holland, 1992). Metode ini memaintain populasi dari kandidat solusi (kombinasi hyperparameter) dan secara iteratif mengevolusi populasi tersebut menuju solusi yang lebih baik (Goldberg & Holland, 1988).

Proses GA dimulai dengan inisialisasi populasi *random*, kemudian mengevaluasi *fitness* (performa) setiap individu dalam populasi (Lorenzo et al., 2017). Individu dengan *fitness* tinggi memiliki probabilitas lebih besar untuk diseleksi sebagai *parents* untuk generasi berikutnya. *Crossover* operation menggabungkan *hyperparameter* dari dua *parents* untuk menghasilkan *offspring*, sedangkan *mutation operation* memperkenalkan variasi random untuk maintain diversity dan menghindari premature convergence (Eiben & Smith, 2015).

Keunggulan *genetic algorithm* adalah kemampuannya untuk mengeksplorasi *non-convex* dan *multimodal search spaces*, serta tidak memerlukan *gradient information* (Lorenzo et al., 2017). GA juga *naturally parallelizable* karena evaluasi *fitness* dalam satu generasi dapat dilakukan secara independen. Namun, kekurangan GA termasuk memerlukan tuning dari GA parameters itu sendiri (*population size, mutation rate, crossover rate*) dan dapat memerlukan banyak

evaluasi untuk *converge*, terutama untuk high-dimensional spaces (Eiben & Smith, 2015).

Dalam *Genetic Algorithm*, setiap konfigurasi *hyperparameter* (*individual*) direpresentasikan sebagai *chromosome* yang mengenkode seluruh nilai *hyperparameter*. Pemilihan *encoding scheme* sangat penting untuk memastikan efektivitas operasi genetika. *Chromosome* merupakan susunan terurut dari *genes*, di mana setiap *gene* merepresentasikan satu nilai *hyperparameter*. Pada penelitian ini, dengan total 14 *hyperparameter*, struktur *chromosome* dibentuk sesuai kombinasi nilai dari masing-masing parameter tersebut.



Gambar 3.11 Representasi *Chromosome* dalam *Genetic Algorithm*

3.4.6.1 Parameter dan Strategi Optimization

Tabel 3.5 Parameter dan Strategi *Genetic Algorithm*

Komponen	Nilai / Strategi
Population Size	20 individu per generasi
Number of Generations	10 generasi
Total Evaluations	200 (20×10)
Selection Method	*Tournament selection* (ukuran turnamen: 3)
Crossover	*Single-point crossover (rate: 0.8)*
Mutation	*Uniform mutation (rate: 0.1)*
Elitism	Individu terbaik dipertahankan di setiap generasi
Evaluation Metric	*F1-Score*
Loss Function	*Binary Crossentropy*
Epochs	100 (maksimal)

Tabel 3.5 Lanjutan Parameter dan Strategi *Genetic Algorithm*

Komponen	**Nilai / Strategi**
Early Stopping	*Monitoring: val_loss, Patience: 15, Restore best: True*
LR Scheduler	*ReduceLROnPlateau (Factor: 0.2, Patience: 7, Min LR: 1×10^{-5})*
Checkpointing	*ModelCheckpoint (validation accuracy tertinggi)*
Validation Split	20% dari data pelatihan (stratified)

3.4.6.2 Framework Implementasi

Framework hyperparameter optimization menggunakan *custom implementation* untuk *Genetic Algorithm*. *Training environment* menggunakan *TensorFlow* 2.13.0 dengan *Keras* API. *Development* dilakukan di *Visual Studio Code* dengan *workstation lokal*. *Library* pendukung: DEAP (*Distributed Evolutionary Algorithms in Python*) atau implementasi manual untuk genetic operations.

3.4.6.3 Computational Complexity

Kompleksitas komputasi pada *Genetic Algorithm* ditentukan oleh ukuran populasi dan jumlah generasi, yaitu 20 × 10 sehingga menghasilkan total 200 *evaluations*. Setiap *trial* memerlukan waktu sekitar 10 menit, sehingga total waktu komputasi mencapai ±33,3 jam (sekitar 1,4 hari) untuk eksekusi *sequential*. Dengan *parallel execution*, seluruh 20 individu dalam satu generasi dapat dievaluasi secara bersamaan, menghasilkan waktu komputasi sekitar 8,3 jam dengan 4 GPU atau 16,7 jam dengan 2 GPU. Kebutuhan memori relatif ringan, dengan penyimpanan populasi sekitar 1 KB per konfigurasi, memori per *trial* sebesar 500 MB, dan total *storage* sekitar 1 GB.

3.5 Evaluasi Model

Setelah seluruh metode *hyperparameter optimization* selesai melakukan *exploration* dan menemukan konfigurasi optimal masing-masing, dilakukan proses

evaluasi untuk mengukur performa model CNN 2D dalam mengklasifikasikan *phonocardiogram*. Evaluasi dilakukan pada dua level: pertama, evaluasi *best configuration* dari masing-masing metode HPO (*Grid Search, Random Search, Bayesian Optimization,* dan *Genetic Algorithm*) pada *test set* untuk mengukur *generalization performance*; kedua, *comparison* antar metode HPO berdasarkan *computational efficiency* dan *optimization effectiveness*.

Evaluasi ini menggunakan sejumlah metrik yang umum digunakan dalam bidang klasifikasi biner, antara lain *accuracy, precision, recall, F1-score* dan *confusion matrix*. *Accuracy* menunjukkan sejauh mana model mampu mengklasifikasikan audio dengan benar secara keseluruhan. *Precision* dan *recall* memberikan gambaran mengenai ketepatan dan kelengkapan klasifikasi untuk setiap kelas, sedangkan *F1-score* digunakan untuk menyeimbangkan *precision* dan *recall*. Sementara itu, *confusion matrix* digunakan untuk menganalisis distribusi prediksi model terhadap label sebenarnya, sehingga memudahkan dalam mengidentifikasi pola kesalahan klasifikasi seperti *false positives* (normal diprediksi abnormal) dan *false negatives* (abnormal diprediksi normal).

3.5.1 Evaluasi Per Metode

Evaluasi per metode dilakukan untuk mengukur performa final dari *best configuration* yang ditemukan oleh setiap metode HPO. Setiap metode *optimization* (*Grid Search, Random Search, Bayesian Optimization*, dan *Genetic Algorithm*) menghasilkan satu *best hyperparameter configuration berdasarkan validation accuracy* tertinggi selama proses *optimization*.

Best configuration dari setiap metode HPO dievaluasi pada *test set* yang telah disisihkan sebelumnya dan tidak pernah digunakan selama proses *training* maupun *validation*. Protocol evaluasi mengikuti langkah-langkah berikut:

1. *Model Retraining: Best hyperparameter configuration* di-retrain menggunakan kombinasi *training* dan *validation* set (80% dari total data) untuk memaksimalkan learning dari available data.

2. *Final Evaluation*: Model yang telah di-retrain dievaluasi pada *test set* (20% dari total data) yang *completely unseen* selama proses *optimization*.

3. *Metrics Calculation* : Dihitung *performance metrics* dan *confusion matrix.*

3.5.2 Analisis Komparatif Antar Metode

Analisis komparatif dilakukan untuk membandingkan empat metode HPO berdasarkan multiple criteria yang mencakup *effectiveness, efficiency, stability,* dan *scalability*.

3.5.2.1 Efektivitas Optimization

Efektivitas diukur berdasarkan kualitas *best solution* yang ditemukan oleh setiap metode:

- *Best F1-Score Achieved*: *Validation F1-Score* tertinggi yang dicapai selama proses *optimization*
- *Test Set Performance* : *Generalization performance* pada *test set*

3.5.2.2 Efisiensi Komputasi

Efisiensi diukur berdasarkan computational resources yang dibutuhkan untuk mencapai performa tertentu yaitu:

- *Time to Convergence* : *Wall-clock time* hingga mencapai 95% dari *best accuracy*
- *Number of Trials* : Jumlah evaluasi yang diperlukan hingga mencapai *convergence*
- *Computational Cost* : Total GPU *hours* untuk *complete optimization*
- *Time per Trial* : Rata-rata waktu per *hyperparameter* evaluation

3.5.2.3 Stabilitas Performance

Stabilitas mengukur konsistensi performance across multiple runs dengan configurations yang berbeda. Pengujian dilakukan dengan melatih ulang

(*retraining*) konfigurasi terbaik dari masing-masing metode optimasi sebanyak 5 kali pengulangan (runs) menggunakan inisialisasi random seed yang berbeda (42, 123, 456, 789, 1024):

- *Variance Across Runs*: Standard deviation dari test *F1-Score* pada 5 repeated runs
- *Coefficient of Variation* (CV): CV = (σ / μ) × 100%
- *Range*: Perbedaan antara *best* dan *worst performance*

3.5.2.4 Skalabilitas

Skalabilitas dianalisis dengan mengevaluasi efektivitas hyperparameter terbaik masing-masing HPO pada dataset *sizes* yang berbeda 25%, 50%, 100%.

3.5.3 Statistical Significance Testing

Untuk memastikan bahwa perbedaan kinerja antar metode Hyperparameter Optimization (HPO) benar-benar signifikan secara statistik dan bukan disebabkan oleh variasi acak (*random variation*), dilakukan pengujian signifikansi (hypothesis testing) terhadap hasil akurasi dari masing-masing metode.

Langkah ini penting untuk menentukan apakah perbedaan performa antar metode HPO benar-benar berarti (*statistically significant*) atau hanya muncul akibat faktor acak dalam proses pelatihan model.

Setiap metode HPO dijalankan sebanyak lima kali dengan kondisi yang dikontrol secara hati-hati untuk memastikan hasil yang dapat dibandingkan (*comparability*). *Random seeds* yang berbeda digunakan pada setiap pengulangan dengan nilai 42, 123, 456, 789, dan 1024 untuk menjamin *reproducibility* sekaligus mengeksplorasi variasi yang disebabkan oleh *random initialization*. Semua metode menggunakan ruang pencarian (*search space*) dan rentang *hyperparameter* yang identik untuk memastikan bahwa perbedaan kinerja benar-benar berasal dari efektivitas metode optimasi itu sendiri, bukan karena perbedaan dalam ruang eksplorasi (*exploration space*).

Proses *training*, *validation*, dan *testing* dijaga agar tetap identik di semua metode menggunakan *stratified sampling* dengan *random state* yang sama, sehingga setiap metode dievaluasi pada data yang persis sama. Protokol evaluasi dan perhitungan metrik juga dijaga konsisten untuk memastikan perbandingan yang adil (*fair comparison*).

3.5.3.1 Uji Normalitas

Langkah awal dalam pengujian signifikansi adalah melakukan uji normalitas terhadap data hasil *F1-Score* dari setiap metode HPO. Uji ini bertujuan untuk mengetahui apakah data berdistribusi normal atau tidak, karena hasil ini akan menentukan jenis uji hipotesis yang digunakan selanjutnya.

Uji normalitas dilakukan menggunakan *Shapiro–Wilk* dengan hipotesis (Royston, 1982; Laerd Statistics, 2024):

H0 : Data berdistribusi normal

H1 : Data berdistribusi tidak normal

Apabila data berdistribusi normal, analisis dilanjutkan dengan uji *One-Way ANOVA*. Sebaliknya, jika data tidak berdistribusi normal, digunakan uji statistik *Kruskal Wallis Test.*

3.5.3.2 One-Way ANOVA

One-Way ANOVA merupakan uji statistik parametrik yang digunakan untuk membandingkan rata-rata dari tiga atau lebih kelompok independen (Field, 2013).

Berikut adalah hipotesis untuk uji ANOVA :

- H0 : Tidak terdapat perbedaan signifikan antara hasil akurasi keempat metode HPO.
- H1 : Terdapat perbedaan signifikan antara hasil akurasi keempat metode HPO.

Apabila uji ANOVA menolak hipotesis nol , hal ini hanya menunjukkan bahwa ada perbedaan di antara metode, namun tidak secara spesifik menunjukkan metode mana yang berbeda satu sama lain. maka akan dilanjutkan uji *post-hoc* untuk mengetahui pasangan metode mana yang berbeda secara signifikan (Laerd Statistics, 2024).

3.5.3.3 Kruskal-Wallis Test

Kruskal-Wallis test merupakan uji statistik non-parametrik yang digunakan sebagai alternatif dari *One-Way ANOVA* ketika asumsi normalitas tidak terpenuhi *(*Kruskal & Wallis, 1952).

Berikut adalah hipotesis untuk *Kruskal-Wallis test:*

- H0: Tidak terdapat perbedaan signifikan antara hasil akurasi keempat metode HPO.
- H1: Terdapat perbedaan signifikan antara hasil akurasi keempat metode HPO.

Apabila *Kruskal Wallis test* menolak hipotesis nol, hal ini hanya menunjukkan bahwa ada perbedaan di antara metode, namun tidak secara spesifik menunjukkan metode mana yang berbeda satu sama lain. Untuk mengidentifikasi secara spesifik pasangan metode yang berbeda secara signifikan, dilakukan uji *post-hoc*

3.5.3.4 Post-hoc Test

Apabila hasil uji *One-Way ANOVA* / *Kruskal-Wallis* menunjukkan penolakan terhadap hipotesis nol (H0), yang berarti terdapat perbedaan yang signifikan secara statistik antara setidaknya dua kelompok metode, maka analisis akan dilanjutkan dengan uji lanjut atau *Post-hoc Test*. Pengujian ini bertujuan untuk mengidentifikasi secara spesifik pasangan metode mana yang memiliki perbedaan performa yang nyata.

Uji ini akan mengelompokkan metode-metode optimasi ke dalam *subset* yang homogen untuk melihat metode mana yang memiliki kinerja setara dan metode

mana yang berbeda secara signifikan. Hipotesis yang diuji dalam perbandingan berpasangan ini adalah:

- H0: $\mu_i = \mu_j$ (Tidak terdapat perbedaan rata-rata kinerja yang signifikan antara metode i dan metode j).

- H1: $\mu_i \neq \mu_j$ (Terdapat perbedaan rata-rata kinerja yang signifikan antara metode i dan metode j)

BAB IV
HASIL DAN PEMBAHASAN

4.1 Data Penelitian dan Hasil *Preprocessing* Data

Data yang digunakan dalam penelitian ini bersumber dari PhysioNet/CinC Challenge 2016 yang diakses melalui repositori Kaggle. *Dataset* ini terdiri dari rekaman audio *phonocardiogram* (PCG) yang diklasifikasikan ke dalam dua kelas utama Normal dan Abnormal. Sebelum digunakan untuk pelatihan model CNN, data mentah telah melalui serangkaian tahapan *Preprocessing* untuk memastikan kualitas dan keseragaman input.

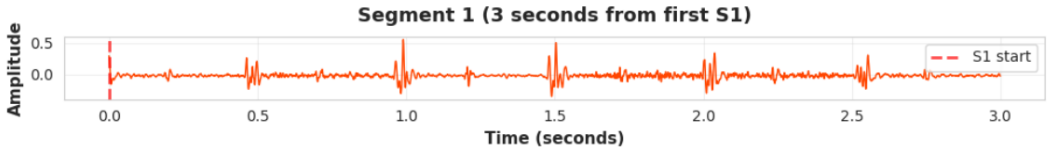
4.1.1 Hasil Preprocessing Data

Langkah pertama yang dilakukan adalah penyeragaman frekuensi sampling (*resampling*) seluruh data audio menjadi 2000 Hz. Berdasarkan hasil observasi spektral, komponen frekuensi utama dari suara jantung normal dan abnormal terkonsentrasi di bawah 1000 Hz. Oleh karena itu, *sampling rate* 2000 Hz terbukti memadai untuk mempertahankan integritas informasi sinyal tanpa mengalami *aliasing*, sesuai dengan Teorema Nyquist-Shannon.

Selanjutnya, algoritma Springer diterapkan untuk mendeteksi lokasi S1 (suara jantung pertama) dan S2 (suara jantung kedua). Algoritma ini menggunakan *Hidden Semi-Markov Model* (HSMM) untuk memetakan probabilitas *state* siklus jantung.

Gambar 4.1 Visualisasi Algoritma Springer dalam deteksi Lokasi S1 dan S2

Visualisasi hasil segmentasi dapat dilihat pada Gambar 4.1. Garis vertikal merah menandakan *onset* S1 dan garis hijau menandakan S2. Hasil ini menunjukkan bahwa algoritma berhasil mengidentifikasi batas-batas siklus jantung secara otomatis meskipun terdapat variasi amplitudo pada sinyal asli.



Gambar 4.2 Hasil pemotongan sinyal pada S1

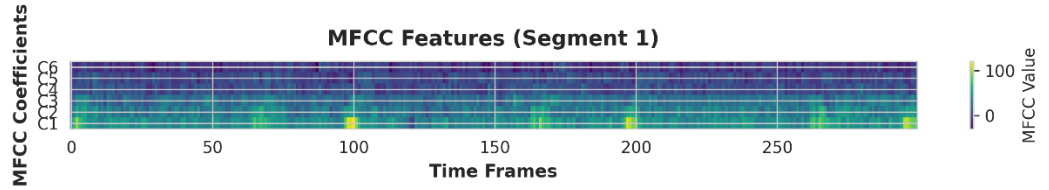
Berdasarkan titik deteksi S1 tersebut, sinyal dipotong menjadi segmen-segmen *overlapping* dengan durasi tetap 3 detik (6.000 *samples*). Pemilihan durasi ini terbukti efektif mencakup rata-rata 3 hingga 5 siklus detak jantung (*cardiac cycles*) pada rentang detak jantung normal (60–100 BPM), sehingga memberikan konteks temporal yang cukup bagi model CNN untuk membedakan pola normal dan abnormal.

Berikut adalah perbandingan jumlah data sebelum dan sesudah dilakukan proses segmentasi:

Tabel 4.1 Tabel perbandingan jumlah data

Kategori Data	Jumlah File Asli	Jumlah Segmentasi Hasil
Normal	2.725	53.652
Abnormal	816	14.585

Setelah segmentasi, setiap potongan sinyal 3 detik dikonversi dari domain waktu ke domain frekuensi menggunakan ekstraksi fitur *Mel-Frequency Cepstral Coefficients* (MFCC). Hasil ekstraksi ini menghasilkan matriks fitur berdimensi 6 × 300. Dimensi ini membentuk representasi visual berupa *heatmap* yang berfungsi sebagai citra input bagi arsitektur CNN 2D.



Gambar 4.3 Visualisasi Ekstraksi Fitur MFCC

Gambar 4.3 memperlihatkan visualisasi fitur MFCC setelah melalui proses standarisasi (*Z-score normalization*). Pada visualisasi ini, sumbu Y merepresentasikan 6 koefisien MFCC yang menangkap *spectral envelope*, sedangkan sumbu X merepresentasikan perubahan fitur tersebut sepanjang 300 *time frames*.

4.2 Hasil Kinerja Model Rubin

Sebagai langkah awal eksperimen dan titik acuan (*baseline*) untuk mengukur efektivitas metode optimasi *hyperparameter*, dilakukan pelatihan model menggunakan arsitektur CNN 2D yang mengadaptasi konfigurasi dari penelitian Rubin et al. (2016). Konfigurasi ini menggunakan nilai *hyperparameter* yang ditetapkan secara manual atau *default* tanpa melalui proses pencarian otomatis.

4.2.1 Konfigurasi Hyperparameter Rubin

Model dilatih menggunakan konfigurasi *hyperparameter* statis yang dirancang untuk mereplikasi struktur dasar arsitektur terdahulu. Rincian konfigurasi yang digunakan adalah sebagai berikut:

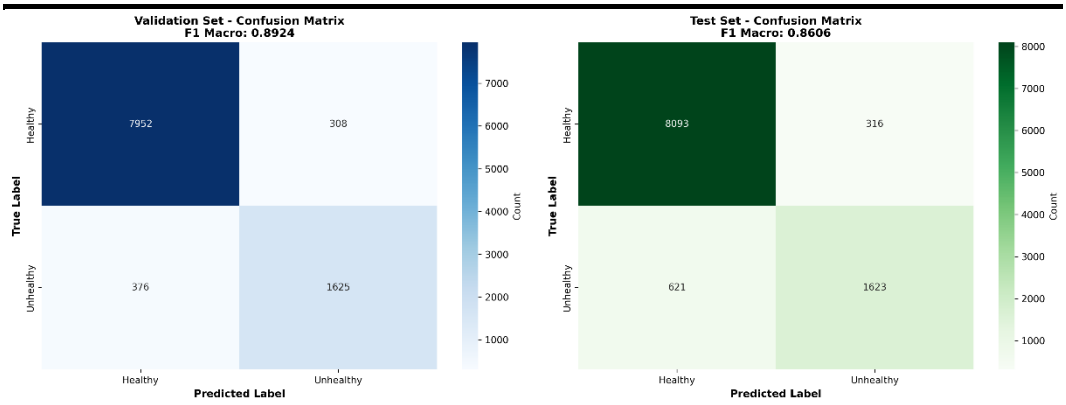
Tabel 4.2 Tabel Konfigurasi Hyperparameter Rubin

Hyperparameter	***Range/Option***
Kernel Conv Layer 1	[64]
Kernel Conv Layer 2	[64]
Kernel Size Conv Layer 1	[(2,20)]
Kernel Size Conv Layer 2	[(2,10)]
Dense Unit Layer 1	[1024]
Dense Unit Layer 2	[512]
Drop Out Rate Layer 1	[0.85565561]
Drop Out Rate Layer 2	[0.85565561]
Learning Rate	[0.000158]
Batch Size	[256]
Optimizer	[Adam]

4.2.2 Kinerja Model Rubin

Table 4.3 Table Rekapitulasi Kinerja Model Rubin

Metrik Evaluasi	*Train*	*Validation*	*Test*
Accuracy	0,974	0,933	0,912
F1-Score (Macro)	0,962	0,892	0,860
F1-Score (Weighted)	0,974	0,932	0,909



Gambar 4.4 *Confusion Matrix* Rubin

4.2.3 Performa Perkelas

Untuk memahami kemampuan deteksi model secara lebih spesifik, dilakukan analisis performa terpisah untuk kelas *Healthy* dan *Unhealthy* pada *Train*, *Validation* dan *Test set*.

Table 4.3 Tabel performa Kelas pada *Training Set* Rubin

Kelas	*Precision*	*Recall*	*F1-Score*	Jumlah Sample
Healthy	0,982	0,984	0,983	36.983
Unhealthy	0,944	0,937	0,941	10.340
Rata-rata (Macro)	0,963	0,961	0,962	47.323
Rata-rata (Weighted)	0,974	0,974	0,974	47.323

Table 4.4 Tabel performa Kelas pada *Validation Set* Rubin

Kelas	*Precision*	*Recall*	*F1-Score*	Jumlah Sample
Healthy	0,954	0,962	0,958	8.260
Unhealthy	0,840	0,812	0,826	2.001
Rata-rata (Macro)	0,897	0,887	0,892	10.261
Rata-rata (Weighted)	0,932	0,933	0,932	10.261

Table 4.5 Tabel performa Kelas pada *Test Set* Rubin

Kelas	*Precision*	*Recall*	*F1-Score*	Jumlah Sample
Healthy	0,928	0,962	0,945	8.409
Unhealthy	0,837	0,723	0,776	2.244
Rata-rata (Macro)	0,882	0,842	0,860	10.653
Rata-rata (Weighted)	0,909	0,912	0,909	10.653

Berdasarkan perbandingan ketiga tabel di atas, dapat disimpulkan bahwa model masih menunjukkan permasalahan performa yang signifikan akibat ketidakseimbangan data. *Overfitting* terjadi lebih parah pada kelas minoritas (*Unhealthy*), yang terlihat dari penurunan *F1-Score* yang jauh lebih drastis dibandingkan kelas mayoritas. Hal ini juga mencerminkan adanya bias terhadap kelas *Healthy* akibat dominasi jumlah data, sehingga model lebih cenderung memprediksi kondisi normal, yang ditunjukkan oleh tingginya *recall* pada kelas *Healthy* namun rendah pada kelas *Unhealthy*. Temuan ini menegaskan bahwa konfigurasi baseline belum optimal untuk menangani imbalance data dan meningkatkan generalisasi model, sehingga diperlukan optimasi *hyperparameter* lebih lanjut untuk meningkatkan kemampuan model dalam mendeteksi kondisi abnormal.

4.3 Hasil Hyperparameter Optimization

Hyperparameter Optimization dilakukan menggunakan empat metode berbeda untuk mencari konfigurasi terbaik yang dapat meningkatkan performa deteksi, khususnya pada kelas *Unhealthy*. Berikut adalah paparan hasil dari masing-masing metode.

4.3.1 Grid Search

Metode *Grid Search* dijalankan menggunakan strategi *Reduced Grid* sebanyak 192 iterasi untuk mengatasi kendala komputasi dengan durasi 24 jam. Proses ini mengevaluasi kombinasi parameter secara sistematis pada titik-titik yang telah ditentukan. Untuk *hyperparameter search space* yang digunakan dalam *reduce grid* adalah sebagai berikut:

Table 4.6 Table Search Space Reduce Grid

Kategori	***Hyperparameter***	***Range/Option***
Architecture Parameters	*Kernel Conv Layer 1*	[64]
	Kernel Conv Layer 2	[64]
	Kernel Size Conv Layer 1	[(2,20]
	Kernel Size Conv Layer 2	[(2,10)]
	Dense Unit Layer 1	[1024, 1536]
	Dense Unit Layer 2	[512, 768]
Regularization Parameters	*Drop Out Rate Layer 1*	[0.2, 0.4]
	Drop Out Rate Layer 2	[0.2, 0.4]
Training Parameters	*Learning Rate*	[0.0001, 0.001, 0.01]
	Batch Size	[128, 256]
	Optimizer	[Adam, Nadam]
Total Search Space	1x1x1x1x2x2x2x2x3x2x2	192

Berdasarkan hasil eksperimen, konfigurasi terbaik ditemukan pada Iterasi 76 Rincian konfigurasi parameter terbaik disajikan pada Tabel 4.7

Table 4.7 Table Konfigurasi *Hyperparameter* Terbaik *Grid Search*

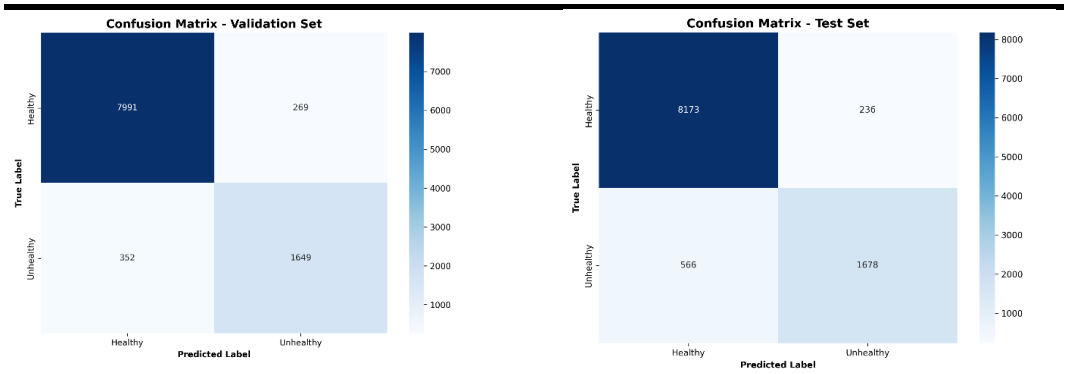
Hyperparameter	***Range/Option***
Kernel Conv Layer 1	[64]
Kernel Conv Layer 2	[64]
Kernel Size Conv Layer 1	[(2,20)]
Kernel Size Conv Layer 2	[(2,10)]
Dense Unit Layer 1	[1024]
Dense Unit Layer 2	[768]
Drop Out Rate Layer 1	[0.4]
Drop Out Rate Layer 2	[0.2]
Learning Rate	[0.0001]
Batch Size	[256]
Optimizer	[Nadam]

4.3.1.1 Hasil Kinerja Grid Search

Analisa dilakukan secara komprehensif pada tiga himpunan data: Training, Validation, dan Testing:

Table 4.8 Table Rekapitulasi Kinerja Hasil *Grid Search*

Metrik Evaluasi	*Train*	*Validation*	*Test*
Accuracy	0,999	0,939	0,924
F1-Score (Macro)	0,999	0,902	0,880
F1-Score (Weighted)	0,999	0,938	0,922

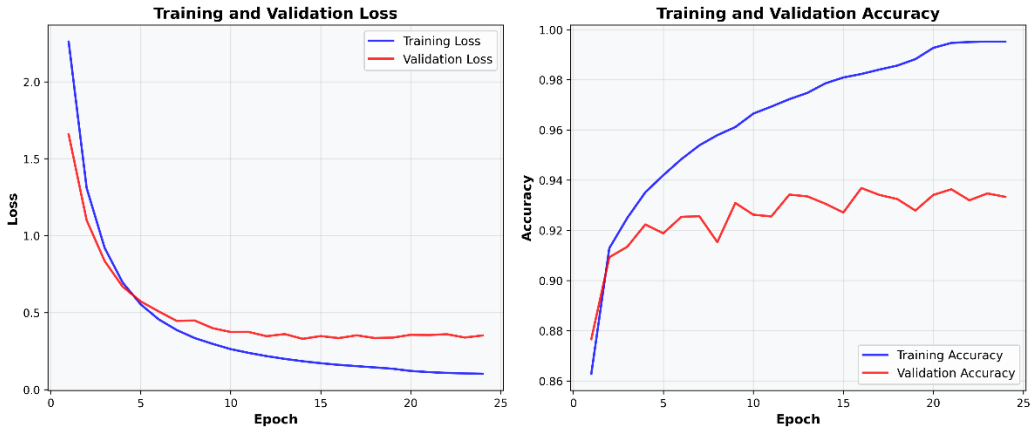


Gambar 4.5 *Confusion Matrix* Hasil *Grid Search*

Berdasarkan hasil eksperimen optimasi *hyperparameter* menggunakan metode *Grid Search*, ditemukan konfigurasi terbaik pada konfigurasi 182 yang menghasilkan performa klasifikasi yang tinggi, meskipun terdapat indikasi *overfitting* antara data latih dan data uji.

Pada Tabel 4.8 Rekapitulasi Kinerja, model mencatatkan akurasi pelatihan (*Train Accuracy*) yang sempurna sebesar 0,999. Namun, terjadi penurunan performa pada tahap validasi menjadi 0,938 dan pada tahap pengujian (*Test*) menjadi 0,924. Kesenjangan (*gap*) antara akurasi *training* dan *test* ini mengindikasikan bahwa model cenderung menghafal pola data latih dengan sangat baik namun sedikit mengalami penurunan kemampuan generalisasi saat dihadapkan pada data baru.

Dari sisi keseimbangan performa antar kelas, model menghasilkan nilai F1-Score Macro pada *Test set* sebesar 0,880. Nilai ini menunjukkan bahwa model memiliki kemampuan yang cukup baik dalam menangani kedua kelas, meskipun tidak setinggi nilai akurasi globalnya.



Gambar 4.6 Grafik *Loss* dan *Accuracy Training* dan *Validation Grid Search*

Hasil pelatihan konfigurasi terbaik dari *Grid Search* menunjukkan performa yang sangat solid dengan konvergensi cepat. Model mampu mencapai *Validation Accuracy* yang stabil di angka 93–94%, sementara *Training Accuracy* terus meningkat hingga mendekati 100%. Meskipun demikian, terdapat indikasi *overfitting* ringan mulai epoch ke-10 di mana *Validation Loss* mengalami stagnasi sementara *loss* data latih terus turun. Secara keseluruhan, model ini terbukti *robust*,

namun penghentian pelatihan di sekitar epoch ke-15 sangat disarankan untuk menjaga efisiensi dan generalisasi terbaik.

4.3.1.2 Performa per kelas

Untuk memahami kemampuan deteksi model secara lebih spesifik, dilakukan analisis performa terpisah untuk kelas *Healthy* dan *Unhealthy* pada *Train*, *Validation* dan *Test set*.

Table 4.9 Tabel performa Kelas pada *Training Set Grid Search*

Kelas	*Precision*	*Recall*	*F1-Score*	Jumlah Sample
Healthy	0,999	0,999	0,999	36.983
Unhealthy	0,999	0,999	0,999	10.340
Rata-rata (Macro)	0,999	0,999	0,999	47.323
Rata-rata (Weighted)	0,999	0,999	0,999	47.323

Table 4.10 Tabel performa Kelas pada *Validation Set Grid Search*

Kelas	*Precision*	*Recall*	*F1-Score*	Jumlah Sample
Healthy	0,958	0,967	0,963	8.260
Unhealthy	0,860	0,824	0,842	2.001
Rata-rata (Macro)	0,909	0,896	0,902	10.261
Rata-rata (Weighted)	0,939	0,939	0,939	10.261

Table 4.11 Tabel performa Kelas pada *Test Set Grid Search*

Kelas	*Precision*	*Recall*	*F1-Score*	Jumlah Sample
Healthy	0,935	0,972	0,953	8.409
Unhealthy	0,877	0,748	0,807	2.244
Rata-rata (Macro)	0,906	0,860	0,880	10.653
Rata-rata (Weighted)	0,923	0,925	0,922	10.653

4.3.2 Random Search

Metode *Random Search* melakukan eksplorasi ruang parameter secara acak sebanyak 200 iterasi. Proses optimasi ini menghabiskan total durasi waktu komputasi selama 14 jam. Model terbaik ditemukan cukup awal, yaitu pada iterasi ke-178, dengan konfigurasi parameter sebagai berikut:

Table 4.12 Table Konfigurasi *Hyperparameter* Terbaik *Random Search*

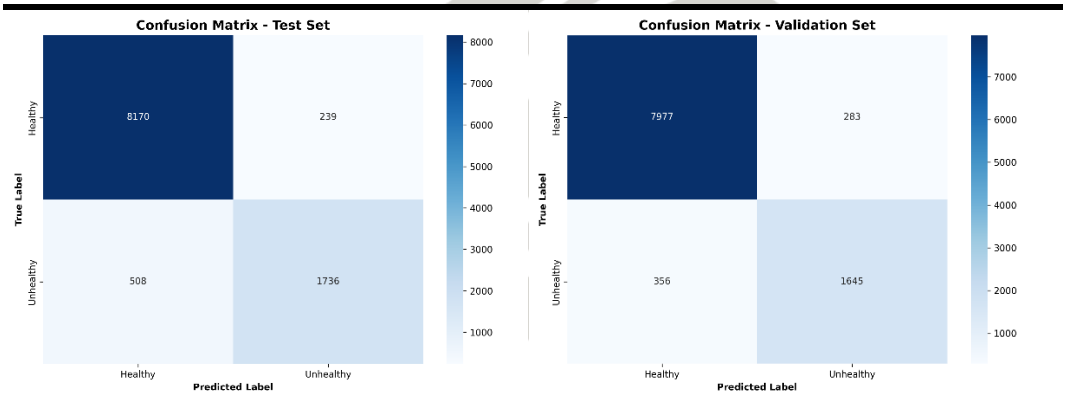
Hyperparameter	***Range/Option***
Kernel Conv Layer 1	[80]
Kernel Conv Layer 2	[80]
Kernel Size Conv Layer 1	[(2,25)]
Kernel Size Conv Layer 2	[(3,10)]
Dense Unit Layer 1	[1024]
Dense Unit Layer 2	[384]
Drop Out Rate Layer 1	[0.2]
Drop Out Rate Layer 2	[0.2]
Learning Rate	[0.0001]
Batch Size	[128]
Optimizer	[Nadam]

4.3.2.1 Hasil Kinerja Random Search

Analisa dilakukan secara komprehensif pada tiga himpunan data: Training, Validation, dan Testing:

Table 4.13 Table Rekapitulasi Kinerja Hasil *Random Search*

Metrik Evaluasi	*Train*	*Validation*	*Test*
Accuracy	0,998	0,937	0,929
F1-Score (Macro)	0,998	0,899	0,889
F1-Score (Weighted)	0,998	0,937	0,928

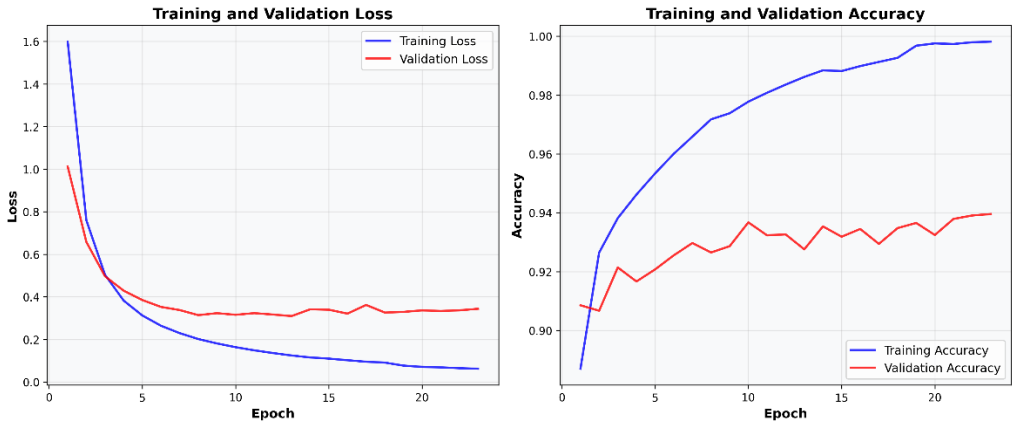


Gambar 4.7 *Confusion Matrix* Hasil *Random Search*

Berdasarkan hasil eksperimen optimasi *hyperparameter* menggunakan metode *Random Search*, ditemukan konfigurasi terbaik pada konfigurasi 57 yang menghasilkan performa klasifikasi yang tinggi, meskipun terdapat indikasi *overfitting* antara data latih dan data uji.

Merujuk pada Tabel 4.13 Rekapitulasi Kinerja Hasil Random Search, model menunjukkan kemampuan pembelajaran yang sangat kuat dengan Akurasi Pelatihan (*Train Accuracy*) mencapai 0,998. Kinerja ini sedikit menurun pada tahap validasi menjadi 0,937 dan pada tahap pengujian (*Test*) menjadi 0,929. Penurunan performa dari *training* ke *test* sebesar kurang lebih 7,6% ini mengindikasikan adanya gejala *overfitting*, di mana model sangat presisi dalam mengenali data latih namun mengalami sedikit penurunan performa saat memproses data baru yang

belum pernah dilihat sebelumnya.Dalam hal keseimbangan deteksi antar kelas, metode ini menghasilkan F1-Score Macro pada *Test set* sebesar 0,889.



Gambar 4.8 Grafik *Loss* dan *Accuracy Training* dan *Validation Random Search*

Hasil pelatihan konfigurasi terbaik dari *Random Search* menunjukkan efektivitas pembelajaran yang tinggi, di mana *Validation Accuracy* berhasil bertahan di kisaran stabil 93–94% dengan *Training Accuracy* yang nyaris sempurna. Namun, grafik *loss* memperlihatkan gejala *overfitting* yang muncul lebih awal; *Validation Loss* mulai stagnan di sekitar angka 0.32 setelah epoch ke-7, sementara *Training Loss* terus menurun drastis.

4.3.2.2 Performa per kelas

Untuk memahami kemampuan deteksi model secara lebih spesifik, dilakukan analisis performa terpisah untuk kelas *Healthy* dan *Unhealthy* pada *Train*, *Validation* dan *Test set*.

Table 4.14 Tabel performa Kelas pada *Training Set Random Search*

Kelas	*Precision*	*Recall*	*F1-Score*	Jumlah Sample
Healthy	0,998	0,998	0,998	36.983
Unhealthy	0,997	0,997	0,997	10.340
Rata-rata (Macro)	0,998	0,998	0,998	47.323
Rata-rata (Weighted)	0,998	0,998	0,998	47.323

Table 4.15 Tabel performa Kelas pada *Validation Set Random Search*

Kelas	*Precision*	*Recall*	*F1-Score*	Jumlah Sample
Healthy	0,957	0,966	0,961	8.260
Unhealthy	0,853	0,822	0,837	2.001
Rata-rata (Macro)	0,905	0,894	0,899	10.261
Rata-rata (Weighted)	0,937	0,938	0,937	10.261

Table 4.16 Tabel performa Kelas pada *Test Set Random Search*

Kelas	*Precision*	*Recall*	*F1-Score*	Jumlah Sample
Healthy	0,941	0,972	0,956	8.409
Unhealthy	0,879	0,774	0,823	2.244
Rata-rata (Macro)	0,910	0,873	0,889	10.653
Rata-rata (Weighted)	0,928	0,930	0,928	10.653

4.3.3 Bayesian Optimization

Metode *Bayesian Optimization* pendekatan probabilistik untuk memandu pencarian parameter secara cerdas berdasarkan evaluasi sebelumnya sebanyak 200 iterasi. Proses optimasi ini menghabiskan durasi 17 jam, karena kompleksitas proses pemodelan *surrogate*. Namun, investasi waktu ini terbayar dengan ditemukannya model terbaik pada tahap akhir proses, yaitu iterasi ke-148. Konfigurasi terbaiknya adalah:

Table 4.17 Table Konfigurasi *Hyperparameter* Terbaik *Bayesian Optimization*

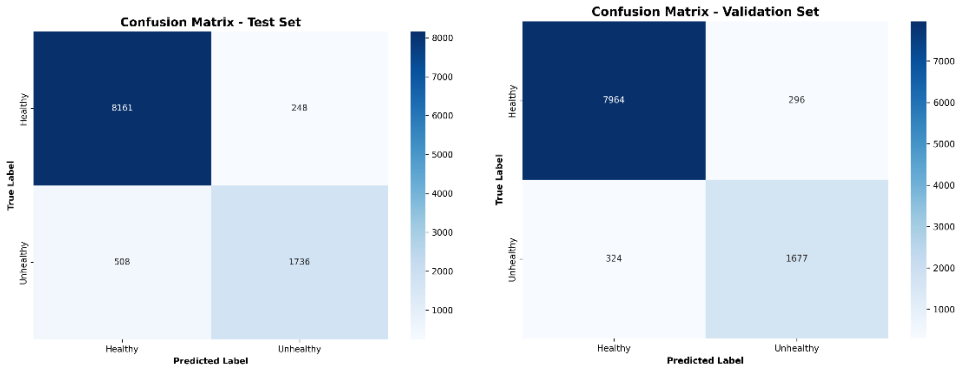
Hyperparameter	***Range/Option***
Kernel Conv Layer 1	[64]
Kernel Conv Layer 2	[48]
Kernel Size Conv Layer 1	[(3,20)]
Kernel Size Conv Layer 2	[(2,10)]
Dense Unit Layer 1	[1536]
Dense Unit Layer 2	[768]
Drop Out Rate Layer 1	[0.2]
Drop Out Rate Layer 2	[0.4]
Learning Rate	[0.0001]
Batch Size	[128]
Optimizer	[Adam]

4.3.3.1 Hasil Kinerja Bayesian Optimization

Analisa dilakukan secara komprehensif pada tiga himpunan data: Training, Validation, dan Testing:

Table 4.18 Table Rekapitulasi Kinerja Hasil *Bayesian Optimization*

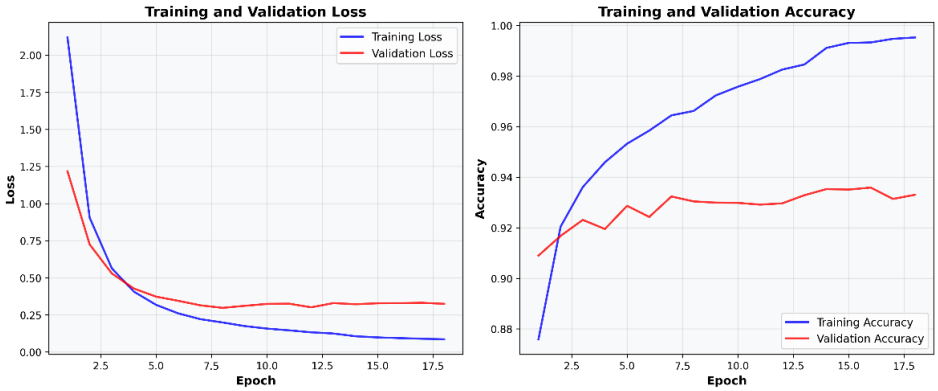
Metrik Evaluasi	*Train*	*Validation*	*Test*
Accuracy	0,998	0,939	0,929
F1-Score (Macro)	0,998	0,903	0,888
F1-Score (Weighted)	0,998	0,939	0,927



Gambar 4.9 *Confusion Matrix* Hasil *Bayesian Optimization*

Hasil eksperimen optimasi hyperparameter menggunakan metode *Bayesian Optimization* menunjukkan kinerja model yang sangat solid dan seimbang, dengan karakteristik yang sedikit lebih unggul dalam hal akurasi pelatihan dibandingkan metode lainnya.

Merujuk pada Tabel 4.18 Rekapitulasi Kinerja Hasil Bayesian Optimization, model mencatatkan Akurasi Pelatihan (Train Accuracy) yang sangat tinggi, mencapai 0,998. Performa ini sedikit menurun pada tahap validasi menjadi 0,939 dan pada tahap pengujian (Test) menjadi 0,929. Pola penurunan dari training ke test ini konsisten dengan metode optimasi lainnya, mengindikasikan adanya gap generalisasi yang wajar dalam model Deep Learning yang dilatih pada dataset dengan karakteristik kompleks seperti suara jantung. Dalam hal kemampuan mendeteksi kedua kelas secara seimbang, metode ini menghasilkan F1-Score Macro pada Test set sebesar 0,888



Gambar 4.10 Grafik *Loss* dan *Accuracy Training* dan *Validation Bayesian Optimization*

Hasil pelatihan konfigurasi terbaik dari *Bayesian Optimization* menunjukkan efisiensi konvergensi yang sangat cepat. Model berhasil mempertahankan *Validation Accuracy* yang stabil di kisaran 93–94%, sementara *Training Accuracy* meningkat konsisten hingga mendekati 100%. Namun, grafik *loss* memperlihatkan bahwa titik optimal generalisasi tercapai lebih awal; *Validation Loss* mulai stagnan (*plateau*) di kisaran 0.30 setelah epoch ke-7, sedangkan *Training Loss* terus menurun. Meskipun model terbukti *robust*, penerapan *Early Stopping* di sekitar epoch ke-8 hingga ke-10 sangat direkomendasikan untuk mencegah *overfitting* yang tidak perlu serta menghemat sumber daya komputasi.

4.3.3.2 Performa per kelas

Untuk memahami kemampuan deteksi model secara lebih spesifik, dilakukan analisis performa terpisah untuk kelas *Healthy* dan *Unhealthy* pada *Train*, *Validation* dan *Test set*.

Table 4.19 Tabel performa Kelas pada *Training Set Bayesian Optimization*

Kelas	*Precision*	*Recall*	*F1-Score*	Jumlah Sample
Healthy	1,000	0,999	0,999	36.983
Unhealthy	0,995	0,999	0,997	10.340
Rata-rata (Macro)	0,998	0,999	0,998	47.323
Rata-rata (Weighted)	0,999	0,999	0,998	47.323

Table 4.20 Tabel performa Kelas pada *Validation Set Bayesian Optimization*

Kelas	*Precision*	*Recall*	*F1-Score*	Jumlah Sample
Healthy	0,961	0,964	0,963	8.260
Unhealthy	0,850	0,838	0,844	2.001
Rata-rata (Macro)	0,905	0,901	0,903	10.261
Rata-rata (Weighted)	0,939	0,940	0,939	10.261

Table 4.21 Tabel performa Kelas pada *Test Set Bayesian Optimization*

Kelas	*Precision*	*Recall*	*F1-Score*	Jumlah Sample
Healthy	0,941	0,971	0,956	8.409
Unhealthy	0,875	0,774	0,821	2.244
Rata-rata (Macro)	0,908	0,872	0,888	10.653
Rata-rata (Weighted)	0,927	0,929	0,927	10.653

4.3.4 Genetic Algorithm

Metode *Genetic Algorithm* (GA) menerapkan prinsip evolusi biologis melalui seleksi, *crossover* pada populasi parameter selama 10 generasi. Proses evolusi menghabiskan waktu total selama 14 jam dan konfigurasi terbaik ditemukan pada iterasi ke 30. Berikut adalah *hyperparameter* terbaik yang dihasilkan:

Table 4.22 Table Konfigurasi *Hyperparameter* Terbaik *Genetic Algorithm*

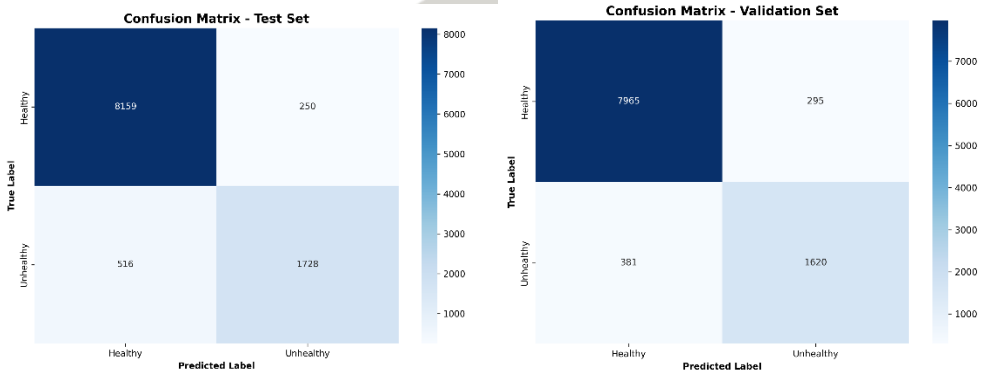
Hyperparameter	***Range/Option***
Kernel Conv Layer 1	[64]
Kernel Conv Layer 2	[48]
Kernel Size Conv Layer 1	[(3,25)]
Kernel Size Conv Layer 2	[(3,12)]
Dense Unit Layer 1	[1204]
Dense Unit Layer 2	[384]
Drop Out Rate Layer 1	[0.4]
Drop Out Rate Layer 2	[0.8]
Learning Rate	[0.0001]
Batch Size	[64]
Optimizer	[Nadam]

4.3.4.1 Hasil Kinerja Genetic Algorithm

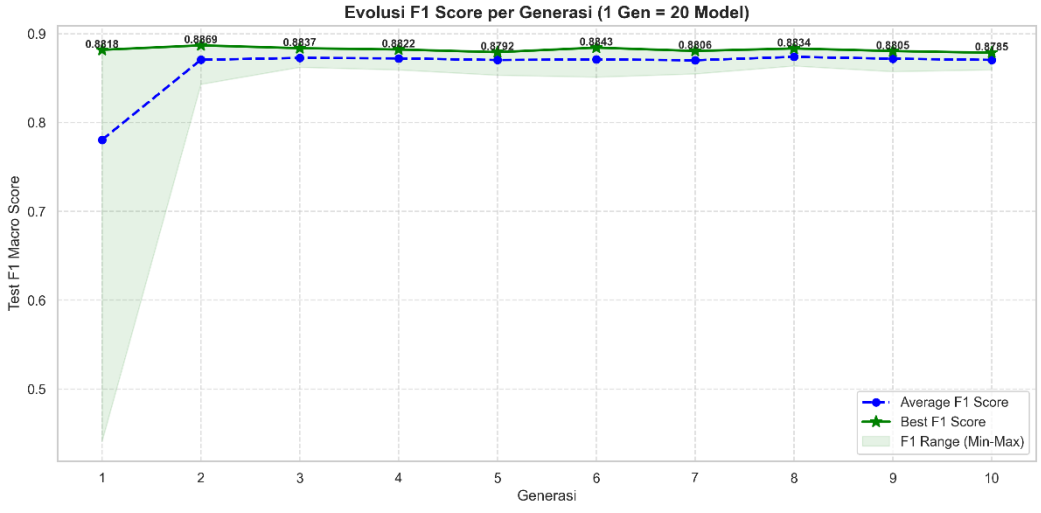
Analisa dilakukan secara komprehensif pada tiga himpunan data: Training, Validation, dan Testing:

Table 4.23 Table Rekapitulasi Kinerja Hasil *Genetic Algorithm*

Metrik Evaluasi	*Train*	*Validation*	*Test*
Accuracy	0,996	0,934	0,928
F1-Score (Macro)	0,994	0,893	0,886
F1-Score (Weighted)	0,996	0,933	0,926



Gambar 4.11 Confusion Matrix Hasil Genetic Algorithm



Gambar 4.12 *Chart F1-Score Range per Generation*

Hasil eksperimen optimasi *hyperparameter* menggunakan metode Genetic Algorithm menunjukkan kinerja yang sangat kompetitif, dengan karakteristik yang unik yaitu efisiensi komputasi yang tinggi (waktu konvergensi tercepat) namun tetap mampu mempertahankan akurasi yang setara dengan metode lainnya.

Merujuk pada Tabel 4.23 Rekapitulasi Kinerja Hasil Genetic Algorithm, model menunjukkan kemampuan pembelajaran yang sangat baik dengan Akurasi Pelatihan (*Train Accuracy*) mencapai 0,996. Serupa dengan metode lainnya, terjadi penurunan pada tahap validasi menjadi 0,934 dan pada tahap pengujian (*Test*) menjadi 0,928. Selisih antara akurasi *training* dan *test* ini menunjukkan pola *overfitting* yang umum terjadi pada *Deep Learning*, namun model masih mampu mempertahankan kemampuan generalisasi yang baik dengan akurasi di atas 91%.

Dalam hal kemampuan menangani ketidakseimbangan kelas, metode ini menghasilkan F1-Score Macro pada *Test set* sebesar 0,886. Hal ini mengindikasikan bahwa meskipun proses pencariannya berbasis evolusi dan stokastik, *Genetic Algorithm* mampu menemukan konfigurasi yang memberikan performa klasifikasi yang stabil.

Gambar 4.13 Grafik *Loss* dan *Accuracy Training* dan *Validation Genetic Algorithm*

Hasil pelatihan konfigurasi terbaik dari Genetic Algorithm menunjukkan pola pembelajaran yang sangat efisien. Model mampu mencapai stabilitas Validation Accuracy di kisaran 93–94%, sementara Training Accuracy terus menanjak hingga hampir menyentuh 99%. Namun, grafik loss mengindikasikan

bahwa kemampuan generalisasi model mencapai puncaknya cukup dini; Validation Loss mulai mendatar di kisaran 0.28–0.30 setelah epoch ke-7, sedangkan Training Loss masih terus menurun. Meskipun model ini sangat robust, penerapan Early Stopping di sekitar epoch ke-8 hingga ke-10 sangat disarankan untuk menjaga efisiensi dan mencegah model terlalu "menghafal" data latih.

4.3.4.2 Performa per kelas

Untuk memahami kemampuan deteksi model secara lebih spesifik, dilakukan analisis performa terpisah untuk kelas *Healthy* dan *Unhealthy* pada *Train*, *Validation* dan *Test set*.

Table 4.24 Tabel performa Kelas pada *Training Set Genetic Algorithm*

Kelas	*Precision*	*Recall*	*F1-Score*	Jumlah Sample
Healthy	0,997	0,998	0,998	36.983
Unhealthy	0,994	0,991	0,992	10.340
Rata-rata (Macro)	0,995	0,995	0,994	47.323
Rata-rata (Weighted)	0,997	0,997	0,996	47.323

Table 4.26 Tabel performa Kelas pada *Validation Set Genetic Algorithm*

Kelas	*Precision*	*Recall*	*F1-Score*	Jumlah Sample
Healthy	0,954	0,964	0,959	8.260
Unhealthy	0,846	0,810	0,827	2.001
Rata-rata (Macro)	0,900	0,887	0,893	10.261
Rata-rata (Weighted)	0,933	0,934	0,933	10.261

Table 4.27 Tabel performa Kelas pada *Test Set Genetic Algorithm*

Kelas	*Precision*	*Recall*	*F1-Score*	Jumlah Sample
Healthy	0,941	0,970	0,955	8.409
Unhealthy	0,874	0,770	0,819	2.244
Rata-rata (Macro)	0,907	0,870	0,886	10.653
Rata-rata (Weighted)	0,926	0,928	0,926	10.653

4.4 Evaluasi Hyperparameter Optimization

4.4.1 Analisis Komparatif Antar Metode

Setelah dilakukan eksperimen pada keempat metode optimasi (*Grid Search, Random Search, Bayesian Optimization, Genetic Algorithm*), tahap selanjutnya adalah melakukan penilaian silang untuk mengidentifikasi metode yang paling unggul. Evaluasi dilakukan secara komprehensif berdasarkan empat dimensi utama: Efektivitas, Efisiensi, Stabilitas, dan Skalabilitas.

4.4.1.1 Efektivitas Optimization

Efektivitas diukur berdasarkan kemampuan metode dalam menemukan konfigurasi *hyperparameter* yang menghasilkan *F1-Score* tertinggi.

Table 4.28 Table Perbandingan Efektivitas Metode HPO

Model	*Test F1-Score Macro*	Peningkatan performa
Grid Search	0,880	0,02
Random Search	0,889	0,029
Bayesian Optimization	0,888	0,028
Genetic Algorithm	0,886	0,026

Berdasarkan hasil uji efektivitas metode *Random Search* terbukti menjadi metode yang paling unggul dalam eksperimen ini. Metode ini mencatatkan Test F1-

Score Macro tertinggi sebesar 0,889, yang memberikan peningkatan performa terbesar yaitu 0,026 (2,9%) dari model Rubin

4.4.1.2 Efisiensi Komputasi

Efisiensi dievaluasi berdasarkan sumber daya waktu yang dibutuhkan untuk mencapai konvergensi.

Table 4.29 Table Perbandingan Efisiensi Metode HPO

Model	*Time to convergence*	*Number of Trials*	*Computational Cost*	*Time per Trial*
Grid Search	3,2 Jam	59	23,8 Jam	7,5 Menit
Random Search	13 Jam	160	14,2 Jam	4,2 Menit
Bayesian Optimization	15,3 Jam	191	17,4 Jam	5,1 Menit
Genetic Algorithm	2,5 Jam	29	14,1 Jam	4,2 Menit

Analisis efisiensi komputasi menunjukkan bahwa *Genetic Algorithm* merupakan metode yang paling unggul dalam hal kecepatan konvergensi, mampu mencapai target performa hanya dalam waktu 2,5 jam. Dari sisi efisiensi eksekusi per iterasi, *Random Search* dan *Genetic Algorithm* terbukti paling ringan dengan rata-rata waktu 4,2 menit per *trial*, menjadikannya opsi hemat sumber daya dengan total biaya komputasi sekitar 14 jam. Sebaliknya, *Grid Search* menjadi metode dengan biaya komputasi tertinggi (23,8 jam) dan waktu eksekusi per *trial* terlama (7,5 menit), sementara *Bayesian Optimization* membutuhkan waktu konvergensi terlama (13,5 jam) karena proses eksplorasinya yang ekstensif sebelum mengeksploitasi solusi optimal.

4.4.1.3 Stabilitas Performance

Evaluasi stabilitas bertujuan untuk memastikan bahwa performa tinggi yang dicapai oleh model terbaik bukanlah kebetulan semata, melainkan hasil dari konfigurasi *hyperparameter*. Pengujian dilakukan dengan melatih ulang

(*retraining*) konfigurasi terbaik dari masing-masing metode optimasi sebanyak 5 kali pengulangan (*runs*) menggunakan inisialisasi *random seed* yang berbeda (42, 123, 456, 789, 1024).Metode yang stabil ditandai dengan rendahnya nilai *Standard Deviation* dan *Coefficient of Variation* (CV) pada deretan solusi terbaiknya. Berikut adalah hasil 5 kali pengulangan dengan random seed dan juga hasil analisis nya.

Table 4.30 Table Hasil 5 kali pengulangan dengan *random seed*

Model	*42*	*123*	*456*	*789*	*1024*
Grid Search	0,883	0,880	0,866	0,878	0,868
Random Search	0,874	0,873	0,874	0,864	0,869
Bayesian Optimization	0,869	0,853	0,876	0,855	0,878
Genetic Algorithm	0,878	0,862	0,882	0,873	0,865

Table 4.31 Table Analisis Stabilitas

Model	*Mean F1-Score*	*Standar Deviasi*	*Coefficient of Variation*	*Range*
Grid Search	0,875	0,007	0,89%	0,017
Random Search	0,871	0,004	0,49%	0,01
Bayesian Optimization	0,866	0,011	1,32%	0,025
Genetic Algorithm	0,872	0,008	0,96%	0,02

Analisis stabilitas menunjukkan bahwa *Genetic Algorithm* merupakan metode yang paling konsisten dan *robust* dalam penelitian ini, ditandai dengan pencapaian nilai Standar Deviasi (0,008) dan *Coefficient of Variation* (0,96%)

terendah, serta rentang performa yang sangat sempit (0,02). Di sisi lain, *Grid Search* menunjukkan keseimbangan performa yang sangat baik dengan mencatatkan rata-rata kinerja (*Mean F1-Score*) tertinggi sebesar 0,875 sambil mempertahankan stabilitas yang tinggi (CV 0,89%). Sebaliknya, *Bayesian Optimization* teridentifikasi sebagai metode yang paling tidak stabil dengan variasi kinerja terbesar (CV 1,32%), mengindikasikan sensitivitas tinggi terhadap pemilihan ruang parameter.

4.4.1.4 Skalabilitas

Analisis skalabilitas dilakukan untuk mengevaluasi ketahanan metode optimasi terhadap variasi ukuran dataset dan kompleksitas komputasi pada pada dataset *sizes* yang berbeda 25%, 50%, 100%. Dalam analisanya akan dilakukan analisan efektivitas performance pada *F1-Score Macro*. Berikut adalah hasil analisanya:

Table 4.32 Table Analisis Skalabilitas

Model	25 %	50 %	100 %
Grid Search	0,840	0,851	0,880
Random Search	0,843	0,859	0,889
Bayesian Optimization	0,839	0,863	0,888
Genetic Algorithm	0,852	0,864	0,886

Analisis skalabilitas menunjukkan tren positif di mana kinerja seluruh metode optimasi meningkat seiring dengan bertambahnya volume data latih dari 25% hingga 100%, mengonfirmasi bahwa ketersediaan data yang lebih besar memperkuat kemampuan generalisasi model

4.4.2 Statistical Significance Testing

Untuk memvalidasi bahwa perbedaan kinerja antar metode *Hyperparameter Optimization* (HPO) adalah nyata secara statistik dan bukan sekadar kebetulan akibat variasi acak (*random variation*), dilakukan analisis statistik inferensial

terhadap data F1-Score dari 5 kali pengulangan eksperimen (*repeated runs*) dengan random seed 42, 123, 456, 789, dan 1024 untuk menjamin *reproducibility* sekaligus mengeksplorasi variasi yang disebabkan oleh *random initialization*.

4.4.2.1 Uji Normalitas

Langkah pertama adalah memastikan asumsi distribusi data menggunakan uji Normalitas. Pengujian dilakukan menggunakan metode Shapiro-Wilk, yang dikenal sensitif dan akurat untuk ukuran sampel kecil (n < 50). Uji ini menentukan apakah data hasil akurasi dari masing-masing metode berdistribusi normal. Apabila data berdistribusi normal, analisis dilanjutkan dengan uji *One-Way ANOVA*. Sebaliknya, jika data tidak berdistribusi normal, digunakan uji statistik *Kruskal Wallis Test.*

Table 4.33 Hasil Uji Normalitas

	Method	Kolmogrov-Smirnov			Shapiro-Wilk		
		Statistic	df	Sig	Statistic	df	Sig
Score	Grid Search	0,254	5	0,200	0,884	5	0,327
	Random Search	0,295	5	0,180	0,825	5	0,127
	Bayesian Optimization	0,232	5	0,200	0,866	5	0,252
	Genetic Algorithm	0,196	5	0,200	0,944	5	0,694

Sesuai dengan kriteria pengambilan keputusan:

- Jika *Sig.* > 0.05, maka H0 diterima (Data berdistribusi normal).
- Jika *Sig.* < 0.05, maka H0 ditolak (Data tidak berdistribusi normal).

Karena seluruh nilai signifikansi yang diperoleh lebih besar dari 0.05, maka terima H0. Dengan terpenuhinya asumsi normalitas ini, analisis perbedaan kinerja antar metode dapat dilanjutkan menggunakan uji statistik parametrik, yaitu One-Way ANOVA.

4.4.2.2 One-Way ANOVA

Setelah asumsi normalitas terpenuhi, dilakukan uji One-Way ANOVA untuk mengetahui apakah terdapat perbedaan rata-rata performa (*F1-Score*) yang signifikan antara keempat metode optimasi yang diuji (*Grid Search, Random Search, Bayesian Optimization, Genetic Algorithm*). Uji ini bertujuan untuk membuktikan hipotesis bahwa pemilihan metode optimasi memberikan dampak nyata terhadap kinerja model, bukan sekadar kebetulan.

Table 4.34 Hasil Tes ANOVA

	Sum of Squares	df	Mean Square	F	Sig
Between Groups	0,000	3	0,000	0,945	0,443
Within Groups	0,002	16	0,000		
Total	0,002	16			

Sesuai dengan kriteria pengambilan keputusan hipotesis:

- Jika *Sig.* > 0.05, maka H0 diterima (Tidak ada perbedaan signifikan).
- Jika *Sig.* < 0.05, maka H0 ditolak (Terdapat perbedaan signifikan).

Karena nilai signifikansi yang diperoleh adalah 0.442 > 0.05, maka terima H0. Hal ini membuktikan bahwa secara statistik tidak terdapat perbedaan rata-rata *F1-Score* yang signifikan antara keempat metode optimasi *hyperparameter* yang diuji. Temuan ini mengindikasikan bahwa baik metode konvensional (*Grid Search, Random Search*) maupun metode heuristik lanjut (*Bayesian Optimization, Genetic Algorithm*) mampu menghasilkan konfigurasi *hyperparameter* dengan tingkat efektivitas yang setara (komparabel) dalam kasus klasifikasi ini. Oleh karena itu, penentuan metode terbaik tidak lagi didasarkan semata-mata pada perbedaan skor akurasi yang tipis, melainkan dapat difokuskan pada aspek efisiensi komputasi (waktu dan sumber daya).

4.4.3 Analisis Efisiensi Komputasi (CPU vs GPU)

Mengingat kompleksitas arsitektur dan besarnya ruang pencarian hyperparameter, efisiensi waktu komputasi menjadi faktor krusial dalam keberhasilan eksperimen. Untuk mengukur dampak akselerasi perangkat keras, dilakukan analisis perbandingan antara estimasi waktu pelatihan menggunakan CPU dan waktu aktual menggunakan GPU.

Estimasi waktu CPU dihitung berdasarkan rata-rata durasi pelatihan satu model (trial) yang kemudian diproyeksikan ke total iterasi eksperimen (192 iterasi untuk Grid Search dan 200 iterasi untuk metode lainnya).

Table 4.35 Perbandingan estimasi waktu komputasi CPU vs GPU

Metode Optimasi	Durasi Training 1 Iterasi CPU	Proyeksi Durasi Total CPU	Total Waktu Training GPU
Grid Search	28,56 Menit	95,2 Jam	23,8 Jam
Random Search	22,89 Menit	73,2 Jam	14,2 Jam
Bayesian Optimization	16,94 Menit	56,5 Jam	17,4 Jam
Genetic Algorithm	17,95 Menit	59,8 Jam	14,1 Jam

Hasil proyeksi menunjukkan inefisiensi signifikan pada penggunaan CPU, dengan durasi pelatihan terlama mencapai 95,2 jam pada Grid Search. Penerapan GPU terbukti memangkas waktu secara drastis, di mana peningkatan efisiensi tertinggi terjadi pada Random Search dengan percepatan 5,15 kali lipat (menjadi 14,2 jam), diikuti oleh Grid Search dengan percepatan 4 kali lipat. Data ini menegaskan bahwa akselerasi perangkat keras mutlak diperlukan untuk menyelesaikan eksperimen dalam durasi yang wajar.

4.5 Analisis Dampak Ketidakseimbangan Data

Mengingat dataset yang digunakan dalam penelitian ini memiliki distribusi kelas yang tidak seimbang (*imbalanced*), penggunaan metrik akurasi global

(akurasi keseluruhan) berpotensi memberikan gambaran kinerja yang bias. Model cenderung memprediksi kelas mayoritas dengan sangat baik, namun sering kali gagal mengenali kelas minoritas yang justru menjadi fokus utama deteksi.

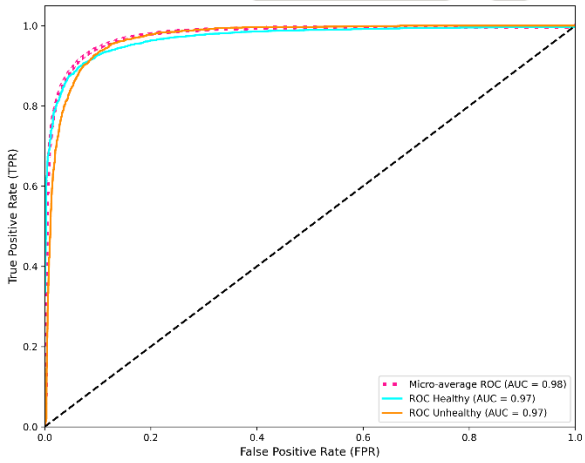
Oleh karena itu, evaluasi mendalam dilakukan menggunakan kurva *Receiver Operating Characteristic* (ROC) dan *Area Under Curve* (AUC). Analisis ini bertujuan untuk mengukur seberapa baik setiap metode optimasi dalam memisahkan kelas *Healthy* dan *Unhealthy* secara adil, tanpa memihak pada kelas yang memiliki jumlah data lebih banyak.

Dalam penelitian ini, ROC dan AUC yang digunakan merupakan hasil dari model terbaik yang diperoleh dari setiap metode optimasi hiperparameter.

4.5.1 Evaluasi ROC Curve per Metode Optimasi

4.5.1.1 Evaluasi ROC Curve Grid Search

Berikut adalah grafik *ROC Curve Grid Search:*



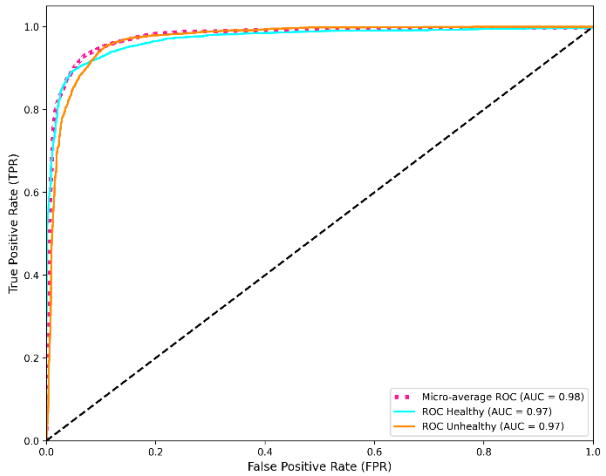
Gambar 4.14 Grafik ROC Curve Grid Search

Berdasarkan gambar 4.14 model yang dioptimasi menggunakan *Grid Search* menunjukkan performa yang sangat stabil dan seimbang (*robust*). Secara keseluruhan, model mencapai nilai *Micro-average AUC* sebesar 0.98, yang mengindikasikan kemampuan klasifikasi yang nyaris sempurna. Kesetaraan nilai AUC ini (0.97 vs 0.97) membuktikan bahwa *Grid Search* berhasil mengatasi masalah *imbalance*. Algoritma ini mampu menemukan konfigurasi hiperparameter

yang membuat model mengenali kelas penyakit (*Unhealthy*) sama baiknya dengan kelas sehat (*Healthy*), tanpa adanya bias terhadap salah satu kelas.

4.5.1.2 Evaluasi ROC Curve Random Seacrh

Berikut adalah grafik *ROC Curve Random Search*:

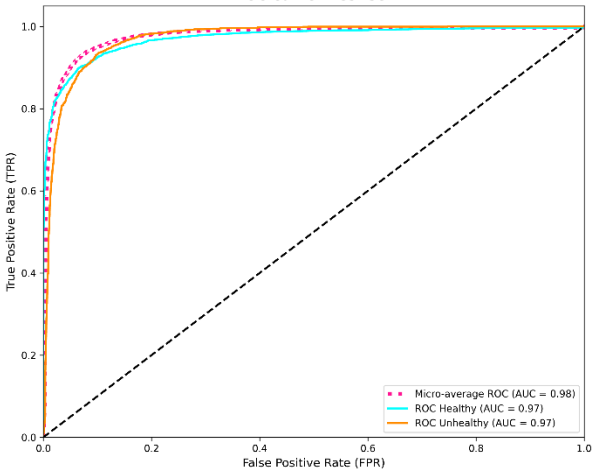


Gambar 4.15 Grafik *ROC Curve Random Search*

Berdasarkan gambar 4.15 model yang dioptimasi menggunakan *Random Search* menunjukkan performa yang sangat stabil dan seimbang (*robust*). Secara keseluruhan, model mencapai nilai *Micro-average AUC* sebesar 0.98, yang mengindikasikan kemampuan klasifikasi yang nyaris sempurna. Kesetaraan nilai AUC ini (0.97 vs 0.97) membuktikan bahwa *Random Search* berhasil mengatasi masalah *imbalance*. Algoritma ini mampu menemukan konfigurasi hiperparameter yang membuat model mengenali kelas penyakit (*Unhealthy*) sama baiknya dengan kelas sehat (*Healthy*), tanpa adanya bias terhadap salah satu kelas.

4.5.1.3 Evaluasi ROC Bayesian Optimization

Berikut adalah grafik *ROC Curve Bayesian Optimization*:

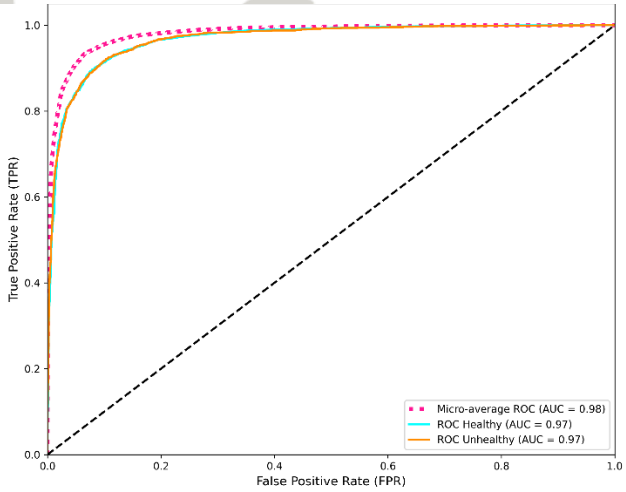


Gambar 4.16 Grafik ROC *Curve Bayesian Optimization*

Berdasarkan gambar 4.16 model yang dioptimasi menggunakan *Bayesian Optimization* menunjukkan performa yang sangat stabil dan seimbang (*robust*). Secara keseluruhan, model mencapai nilai *Micro-average AUC* sebesar 0.98, yang mengindikasikan kemampuan klasifikasi yang nyaris sempurna. Algoritma ini mampu menemukan konfigurasi hiperparameter yang membuat model mengenali kelas penyakit (*Unhealthy*) sama baiknya dengan kelas sehat (*Healthy*), tanpa adanya bias terhadap salah satu kelas.

4.5.1.4 Evaluasi ROC Genetic Algorithm

Berikut adalah grafik *ROC Curve Genetic Algorithm*:



Gambar 4.17 Grafik ROC *Curve Genetic Algorithm*

Berdasarkan gambar 4.17 model yang dioptimasi menggunakan *Bayesian Optimization* menunjukkan performa yang sangat stabil dan seimbang (*robust*). Secara keseluruhan, model mencapai nilai *Micro-average AUC* sebesar 0.98, yang mengindikasikan kemampuan klasifikasi yang nyaris sempurna. Kesetaraan nilai AUC ini (0.97 vs 0.97) membuktikan bahwa *Bayesian Optimization* berhasil mengatasi masalah *imbalance*. Algoritma ini mampu menemukan konfigurasi hiperparameter yang membuat model mengenali kelas penyakit (*Unhealthy*) sama baiknya dengan kelas sehat (*Healthy*), tanpa adanya bias terhadap salah satu kelas.

4.5.2 Kesimpulan Analisis Imbalance

Dari hasil evaluasi ROC keempat metode dapat disimpulkan bahwa model yang dihasilkan memiliki tingkat ketahanan (*robustness*) yang tinggi. Meskipun dilatih menggunakan dataset yang tidak seimbang, model mampu mempertahankan generalisasi yang baik. Kemampuan model untuk mendeteksi kelas *Unhealthy* dengan AUC 0.97 membuktikan bahwa sistem ini sangat layak untuk diimplementasikan sebagai alat bantu diagnosis, karena risiko kesalahan dalam mendeteksi (*False Negative*) telah diminimalisir secara signifikan.

4.6 Analisis Komparatif Model Rubin dengan Metode Genetic Algorithm

4.6.1 Perbandingan Konfigurasi Hyperparameter

Berikut adalah table perbandingan Konfigurasi *Hyperparameter* Model Rubin dengan Konfigurasi Terbaik *Genetic Algorithm*:

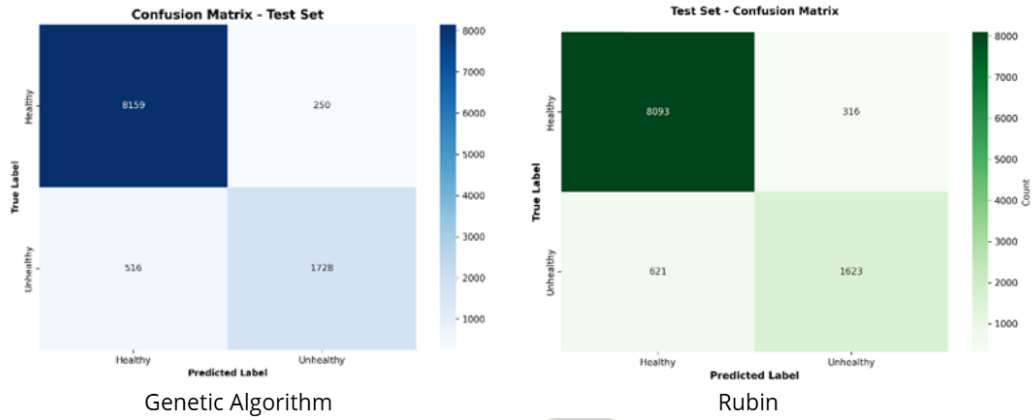
Table 4.36 Table Perbandingan Konfigurasi *Hyperparameter* Rubin dengan *Genetic Algorithm*

Hyperparameter	***Rubin***	***Genetic Algorithm***
Kernel Conv Layer 1	[64]	[64]
Kernel Conv Layer 2	[64]	[48]
Kernel Size Conv Layer 1	[(2,20)]	[(3,25)]
Kernel Size Conv Layer 2	[(2,10)]	[(3,12)]
Dense Unit Layer 1	[1024]	[1204]
Dense Unit Layer 2	[512]	[384]
Drop Out Rate Layer 1	[0.85565561]	[0.4]

Hyperparameter	***Rubin***	***Genetic Algorithm***
Drop Out Rate Layer 2	[0.85565561]	[0.8]
Learning Rate	[0.000158]	[0.0001]
Batch Size	[256]	[64]
Optimizer	[Adam]	[Nadam]

Analisis perbandingan *hyperparameter* pada table 4.35 menunjukkan perbedaan strategi yang signifikan antara model referensi (Rubin) dan hasil optimasi *Genetic Algorithm* (GA). Pada arsitektur CNN, meskipun blok pertama sama-sama menggunakan 64 kernel, GA mengurangi kernel blok kedua menjadi 48 dibandingkan Rubin yang tetap menggunakan 64. Namun, GA memperbesar ukuran kernel menjadi (3,25) dan (3,12) dibandingkan Rubin yang hanya (2,20) dan (2,10), yang mengindikasikan bahwa model GA lebih memprioritaskan cakupan fitur temporal yang luas daripada sekadar jumlah kernel. Perbedaan berlanjut ke lapisan *fully connected*, di mana GA meningkatkan kapasitas *layer* pertama menjadi 1204 unit namun merampingkan *layer* kedua menjadi 384 unit, berbeda dengan Rubin yang menggunakan 1024 dan 512 unit. Selain itu, GA menyeimbangkan regularisasi dengan menurunkan *Dropout Rate* lapisan pertama secara drastis menjadi 0.4 dari angka agresif 0.85 milik Rubin, sehingga aliran informasi menjadi lebih optimal. Terakhir, efisiensi pelatihan ditingkatkan oleh GA melalui penggantian *optimizer* dari Adam ke Nadam, penurunan *Batch Size* signifikan dari 256 ke 64, serta penyesuaian *Learning Rate* menjadi 0.0001, yang secara kolektif bertujuan untuk menghindari *local minima* dan meningkatkan presisi konvergensi.

4.6.2 Analisis Perbandingan Confusion Matrix

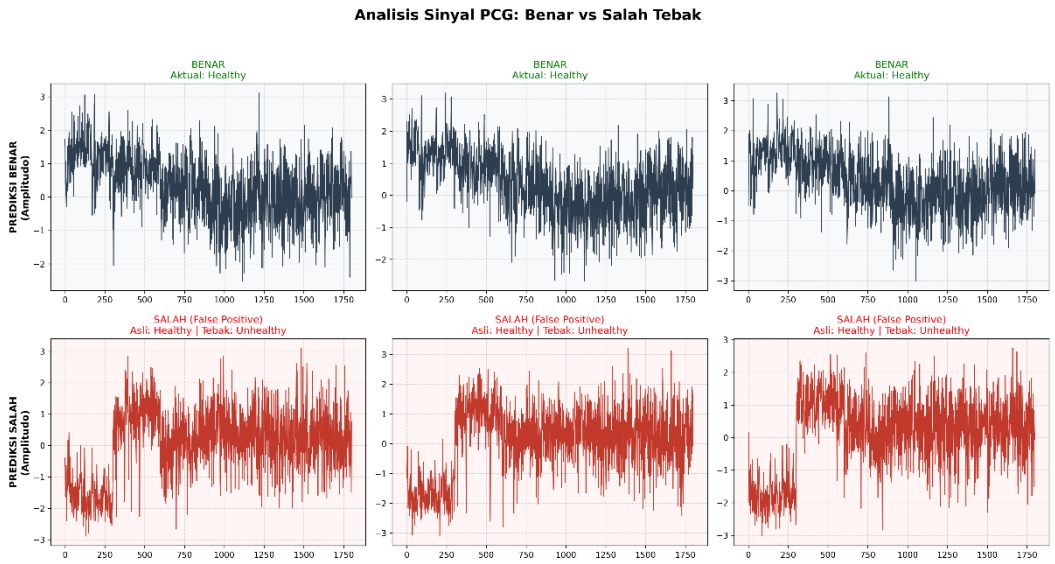


Gambar 4.18 Gambar perbandingan *Confusion Matrix*

Evaluasi komparatif pada Test Set menunjukkan keunggulan konsisten model hasil optimasi *Genetic Algorithm* (GA) dibandingkan model referensi Rubin di seluruh kuadran *Confusion Matrix*. Pada deteksi kelas *healthy*, model GA terbukti lebih presisi dengan meningkatkan *True Negative* menjadi 8.159 dan menekan angka "alarm palsu" (*False Positive*) menjadi 250 kasus. Peningkatan yang lebih krusial secara medis terlihat pada deteksi kelas *unhealthy*, di mana model GA berhasil menaikkan angka *True Positive* menjadi 1.728 sekaligus mereduksi *False Negative* menjadi 516 kasus, yang mengindikasikan sensitivitas yang lebih baik dalam mencegah terlewatnya diagnosis pasien sakit. Secara keseluruhan, total prediksi benar meningkat menjadi 9.887, membuktikan bahwa strategi pencarian hyperparameter menggunakan *Genetic Algorithm* efektif menghasilkan konfigurasi model yang lebih *robust* dan akurat dibandingkan konfigurasi statis pada penelitian referensi.

4.7 Analisis Kesalahan Prediksi (*Error Analysis*) Genetic Algorithm

Sebagai tahap evaluasi mendalam, dilakukan analisis kesalahan prediksi (*error analysis*) terhadap model terbaik hasil optimasi *Genetic Algorithm*. Analisis ini mengambil sampel acak yang terdiri dari 3 contoh klasifikasi benar (*True Positive*) dan tiga contoh klasifikasi salah (*False Positive*) untuk membandingkan karakteristik visual sinyalnya secara langsung.



Gambar 4.19 Gambar Perbandingan Segmentasi Audio *True Positive* dan *False Positive*

Berdasarkan pengamatan visual, terungkap perbedaan karakteristik sinyal yang signifikan antara kedua kelompok tersebut. Pada data *True Positive*, sinyal tampak stabil dengan osilasi yang konsisten di sekitar garis dasar atau titik nol. Sebaliknya, pada kasus *False Positive* (data sehat yang terprediksi sakit), terlihat adanya anomali teknis yang mencolok berupa pergeseran garis dasar (*baseline shift*) yang ekstrem, di mana terjadi lonjakan amplitudo mendadak (*step-change*) yang diikuti oleh perubahan rata-rata sinyal secara permanen. Model CNN diduga menginterpretasikan lonjakan energi mendadak dan ketidakaturan struktur gelombang ini sebagai fitur patologis (seperti *murmur*), sehingga memicu kesalahan klasifikasi.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil Analisa komparatif antar metode dan analisis statistik yang dilakukan terhadap empat metode *Hyperparameter Optimization* (HPO), dapat disimpulkan bahwa setiap metode memiliki karakteristik keunggulan yang berbeda, namun efisiensi menjadi faktor pembeda utama. Secara nominal pada pengujian tunggal, metode *Random Search* mampu mencatatkan *F1-Score* tertinggi sebesar 0,889. Secara statistik melalui uji statistik *One-Way ANOVA* menunjukkan nilai signifikansi sebesar 0,442 (> 0,05). Hal ini membuktikan bahwa tidak terdapat perbedaan rata-rata kinerja yang signifikan antara *Grid Search, Random Search, Bayesian Optimization,* dan *Genetic Algorithm*. Artinya, keempat metode tersebut memiliki kemampuan yang setara dalam menemukan konfigurasi *hyperparameter* yang efektif.

Mengingat tidak adanya perbedaan signifikan dari sisi statistik, penentuan metode terbaik didasarkan pada aspek efisiensi komputasi. *Genetic Algorithm* terbukti menjadi metode yang paling unggul secara menyeluruh karena mampu mencapai konvergensi optimal hanya dalam waktu 2,5 jam yang dimana ini jauh lebih cepat dibandingkan *Grid Search* yang memakan waktu hingga 23,8 jam sambil tetap mempertahankan performa *F1-Score* yang setara dengan metode lainnya. Selain itu, seluruh metode menunjukkan tren skalabilitas yang positif, di mana performa model meningkat seiring dengan penambahan volume data latih. Oleh karena itu, dengan mempertimbangkan keseimbangan antara efisiensi waktu yang superior dan akurasi yang kompetitif secara statistik, *Genetic Algorithm* ditetapkan sebagai metode terbaik dalam penelitian ini.

Selain hasil evaluasi berbasis *F1-Score*, analisis tambahan menggunakan ROC *Curve* dan AUC turut memperkuat kesimpulan penelitian ini. Dari hasil evaluasi ROC terhadap model terbaik dari keempat metode, dapat disimpulkan bahwa model yang dihasilkan memiliki tingkat ketahanan (*robustness*) yang tinggi, bahkan ketika dilatih pada dataset yang tidak seimbang.

5.2 Saran

Dari hasil uji efisiensi komputasi dan analisis statistik, disarankan dalam melakukan *hyperparameter optimization* menggunakan *Genetic Algorithm*. Hal ini dikarenakan metode tersebut terbukti memiliki kecepatan konvergensi yang paling superior, mampu menyelesaikan proses optimasi hanya dalam waktu 2,5 jam, jauh lebih efisien dibandingkan metode *Grid Search* maupun *Bayesian Optimization* yang membutuhkan waktu berjam-jam. Meskipun memiliki waktu eksekusi yang sangat singkat, hasil uji statistik menunjukkan bahwa performa akurasi yang dihasilkan oleh *Genetic Algorithm* tidak berbeda secara signifikan dengan metode lainnya, sehingga menjadikannya solusi yang paling praktis dan hemat sumber daya tanpa mengorbankan kualitas model secara drastis. Namun demikian, hasil analisis grafik *training* dan *validation loss* serta *accuracy* menunjukkan adanya indikasi *overfitting*, di mana performa model pada data pelatihan meningkat secara signifikan tetapi tidak diikuti oleh peningkatan yang sepadan pada data validasi. Oleh karena itu, meskipun Genetic Algorithm direkomendasikan sebagai metode optimasi terbaik dari sisi efisiensi, penggunaannya tetap perlu disertai kehati-hatian, misalnya dengan penerapan teknik regularisasi, *early stopping*, atau validasi silang, guna memastikan kemampuan generalisasi model tetap terjaga.

DAFTAR PUSTAKA

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16) (pp. 265-283). USENIX Association.

Agarap, A. F. (2018). Deep learning using rectified linear units (ReLU). arXiv preprint arXiv:1803.08375. https://doi.org/10.48550/arXiv.1803.08375

Baghel, N., Dutta, M. K., & Burget, R. (2020). Automatic diagnosis of multiple cardiac diseases from PCG signals using convolutional neural network. Computer Methods and Programs in Biomedicine, 197, 105750. https://doi.org/10.1016/j.cmpb.2020.105750

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), Neural Networks: Tricks of the Trade (pp. 437-478). Springer. https://doi.org/10.1007/978-3-642-35289-8_26

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13(1), 281-305.

Bonow, R. O., O'Gara, P. T., Adams, D. H., Badhwar, V., Bavaria, J. E., Elmariah, S., ... & Woo, Y. J. (2021). 2020 Focused update of the 2017 ACC expert consensus decision pathway on the management of mitral regurgitation. Journal of the American College of Cardiology, 77(23), 2925-2946. https://doi.org/10.1016/j.jacc.2021.02.005

Brochu, E., Cora, V. M., & De Freitas, N. (2010). A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv preprint arXiv:1012.2599. https://doi.org/10.48550/arXiv.1012.2599

Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J., & Dahl, G. E. (2020). On empirical comparisons of optimizers for deep learning. arXiv preprint arXiv:1910.05446. https://doi.org/10.48550/arXiv.1910.05446

Chollet, F. (2021). Deep Learning with Python (2nd ed.). Manning Publications.

Claesen, M., & De Moor, B. (2015). Hyperparameter search in machine learning. arXiv preprint arXiv:1502.02127. https://doi.org/10.48550/arXiv.1502.02127

Davis, J., & Goadrich, M. (2006). The relationship between Precision-Recall and ROC curves. In Proceedings of the 23rd International Conference on Machine Learning (pp. 233-240). ACM. https://doi.org/10.1145/1143844.1143874

Davis, S., & Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. IEEE Transactions on Acoustics, Speech, and Signal Processing, 28(4), 357-366. https://doi.org/10.1109/TASSP.1980.1163420

Deng, Y., & Bentley, P. J. (2021). Algorithmic approaches for automated heart sound analysis. Physiological Measurement, 42(1), 01TR01. https://doi.org/10.1088/1361-6579/abc6bb

DeVries, T., & Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552. https://doi.org/10.48550/arXiv.1708.04552

Dubey, S. R., Singh, S. K., & Chaudhuri, B. B. (2022). Activation functions in deep learning: A comprehensive survey and benchmark. Neurocomputing, 503, 92-108. https://doi.org/10.1016/j.neucom.2022.06.111

Eiben, A. E., & Smith, J. E. (2015). Introduction to Evolutionary Computing (2nd ed.). Springer. https://doi.org/10.1007/978-3-662-44874-8

Fawcett, T. (2006). An introduction to ROC analysis. Pattern Recognition Letters, 27(8), 861-874. https://doi.org/10.1016/j.patrec.2005.10.010

Frank, M. J., Gelfand, E. V., & Levine, R. A. (2022). The innocent murmur: A clinical update. American Journal of Medicine, 135(8), 913-919. https://doi.org/10.1016/j.amjmed.2022.04.015

Frazier, P. I. (2018). A tutorial on Bayesian optimization. arXiv preprint arXiv:1807.02811. https://doi.org/10.48550/arXiv.1807.02811

Gal, Y., & Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In International Conference on Machine Learning (pp. 1050-1059).

Géron, A. (2022). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (3rd ed.). O'Reilly Media.

Gharehbaghi, A., Babic, A., Ask, P., & Sörnmo, L. (2021). A decision support system for distinguishing between innocent and pathological murmurs. Biomedical Signal Processing and Control, 65, 102363. https://doi.org/10.1016/j.bspc.2020.102363

Ghiasi, G., Lin, T. Y., & Le, Q. V. (2018). DropBlock: A regularization method for convolutional networks. In Advances in Neural Information Processing Systems 31 (pp. 10727-10737).

Goldberg, D. E., & Holland, J. H. (1988). Genetic algorithms and machine learning. Machine Learning, 3(2), 95-99. https://doi.org/10.1023/A:1022602019183

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

GoodStats. (2023). Prevalensi penyakit jantung di Indonesia tahun 2023. Retrieved from https://goodstats.id

Hanin, B., & Seluk, D. (2018). How to start training: The effect of initialization and architecture. In Advances in Neural Information Processing Systems 31 (pp. 571-581).

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357-362. https://doi.org/10.1038/s41586-020-2649-2

Holland, J. H. (1992). Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. MIT Press.

Hossin, M., & Sulaiman, M. N. (2015). A review on evaluation metrics for data classification evaluations. International Journal of Data Mining & Knowledge Management Process, 5(2), 1-11. https://doi.org/10.5121/ijdkp.2015.5201

Hubel, D. H., & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. The Journal of Physiology, 160(1), 106-154. https://doi.org/10.1113/jphysiol.1962.sp006837

Humayun, A. I., Ghaffarzadegan, S., Feng, Z., & Hasan, T. (2020). Learning front-end kernel-bank parameters using convolutional neural networks for abnormal heart sound detection. In 2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC) (pp. 1408-1411). IEEE. https://doi.org/10.1109/EMBC44109.2020.9176240

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90-95. https://doi.org/10.1109/MCSE.2007.55

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the 32nd International Conference on Machine Learning (pp. 448-456). PMLR.

Japkowicz, N., & Shah, M. (2011). Evaluating Learning Algorithms: A Classification Perspective. Cambridge University Press. https://doi.org/10.1017/CBO9780511921803

Kaggle. (2024). KaggleHub Documentation. Retrieved from https://github.com/Kaggle/kagglehub

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. In 5th International Conference on Learning Representations (ICLR). https://openreview.net/forum?id=H1oyRlYgg

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In 3rd International Conference on Learning Representations (ICLR). https://arxiv.org/abs/1412.6980

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25 (pp. 1097-1105). Curran Associates, Inc.

Labach, A., Salehinejad, H., & Valaee, S. (2019). Survey of dropout methods for deep neural networks. arXiv preprint arXiv:1904.13310. https://doi.org/10.48550/arXiv.1904.13310

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444. https://doi.org/10.1038/nature14539

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. https://doi.org/10.1109/5.726791

Liashchynskyi, P., & Liashchynskyi, P. (2019). Grid search, random search, genetic algorithm: A big comparison for NAS. arXiv preprint arXiv:1912.06059. https://doi.org/10.48550/arXiv.1912.06059

Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., & Han, J. (2020). On the variance of the adaptive learning rate and beyond. In 8th International Conference on Learning Representations (ICLR). https://openreview.net/forum?id=rkgz2aEKDr

Liu, Z., Mao, H., Wu, C. Y., Feichtenhofer, C., Darrell, T., & Xie, S. (2022). A ConvNet for the 2020s. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 11976-11986). https://doi.org/10.1109/CVPR52688.2022.01167

Lloyd-Jones, D. M., Allen, N. B., Anderson, C. A., Black, T., Brewer, L. C., Foraker, R. E., ... & Wilkins, J. T. (2022). Life's Essential 8: Updating and enhancing the American Heart Association's construct of cardiovascular health - A presidential advisory from the American Heart Association. Circulation, 146(5), e18-e43. https://doi.org/10.1161/CIR.0000000000001078

Lorenzo, P. R., Nalepa, J., Kawulok, M., Ramos, L. S., & Pastor, J. R. (2017). Particle swarm optimization for hyper-parameter selection in deep neural networks. In Proceedings of the Genetic and Evolutionary Computation Conference (pp. 481-488). https://doi.org/10.1145/3071178.3071208

Loshchilov, I., & Hutter, F. (2017). SGDR: Stochastic gradient descent with warm restarts. In 5th International Conference on Learning Representations (ICLR). https://openreview.net/forum?id=Skq89Scxx

Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. In 7th International Conference on Learning Representations (ICLR). https://openreview.net/forum?id=Bkg6RiCqY7

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In Proceedings of the 30th International Conference on Machine Learning (Vol. 30, No. 1, p. 3).

Maknickas, V., & Maknickas, A. (2017). Recognition of normal-abnormal phonocardiographic signals using deep convolutional neural networks and mel-frequency spectral coefficients. Physiological Measurement, 38(8), 1671-1684. https://doi.org/10.1088/1361-6579/aa7841

Mantovani, R. G., Rossi, A. L., Vanschoren, J., Bischl, B., & De Carvalho, A. C. (2015). Effectiveness of random search in SVM hyper-parameter tuning. In 2015 International Joint Conference on Neural Networks (IJCNN) (pp. 1-8). IEEE. https://doi.org/10.1109/IJCNN.2015.7280664

Masters, D., & Luschi, C. (2018). Revisiting small batch training for deep neural networks. arXiv preprint arXiv:1804.07612. https://doi.org/10.48550/arXiv.1804.07612

McCarthy, J., Minsky, M. L., Rochester, N., & Shannon, C. E. (1956). A proposal for the Dartmouth summer research project on artificial intelligence. AI Magazine, 27(4), 12-14.

McFee, B., Raffel, C., Liang, D., Ellis, D. P., McVicar, M., Battenberg, E., & Nieto, O. (2015). librosa: Audio and music signal analysis in Python. In Proceedings of the 14th Python in Science Conference (Vol. 8, pp. 18-25). https://doi.org/10.25080/Majora-7b98e3ed-003

McKinney, W. (2010). Data structures for statistical computing in Python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51-56). https://doi.org/10.25080/Majora-92bf1922-00a

Mensah, G. A., Fuster, V., Murray, C. J. L., & Roth, G. A. (2023). Global burden of cardiovascular diseases and risks, 1990-2022. Journal of the American College of Cardiology, 82(25), 2350-2473. https://doi.org/10.1016/j.jacc.2023.11.007

Messner, T., Dillier, R., & Badertscher, P. (2023). Phonocardiography in the era of artificial intelligence: Current applications and future perspectives. European Heart Journal - Digital Health, 4(2), 91-101. https://doi.org/10.1093/ehjdh/ztac078

Mockus, J. (1975). On Bayesian methods for seeking the extremum. In Optimization Techniques IFIP Technical Conference (pp. 400-404). Springer.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (ICML-10) (pp. 807-814).

Nishimura, R. A., Otto, C. M., Bonow, R. O., Carabello, B. A., Erwin, J. P., Gentile, F., ... & Toly, C. (2021). 2020 ACC/AHA guideline for the management of patients with valvular heart disease. Journal of the American College of Cardiology, 77(4), e25-e197. https://doi.org/10.1016/j.jacc.2020.11.018

Nogueira, D. M., Ferreira, C. A., Gomes, E. F., & Jorge, A. M. (2019). Classifying heart sounds using images of motifs, MFCC and temporal features. Journal of Medical Systems, 43(6), 168. https://doi.org/10.1007/s10916-019-1286-5

O'Malley, T., Bursztein, E., Long, J., Chollet, F., Jin, H., Invernizzi, L., et al. (2019). KerasTuner. Retrieved from https://github.com/keras-team/keras-tuner

Otto, C. M., Nishimura, R. A., Bonow, R. O., Carabello, B. A., Erwin III, J. P., Gentile, F., ... & O'Gara, P. T. (2020). 2020 ACC/AHA guideline for the management of patients with valvular heart disease: Executive summary. Circulation, 143(5), e35-e71. https://doi.org/10.1161/CIR.0000000000000932

Paleyes, A., Pullin, M., Mahsereci, M., McCollum, C., Lawrence, N. D., & González, J. (2021). Emulation of physical processes with Emukit. In Second Workshop on Machine Learning and the Physical Sciences (NeurIPS 2019). arXiv preprint arXiv:1910.13321.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830.

Potes, C., Parvaneh, S., Rahman, A., & Conroy, B. (2016). Ensemble of feature-based and deep learning-based classifiers for detection of abnormal heart sounds. In 2016 Computing in Cardiology Conference (CinC) (pp. 621-624). IEEE. https://doi.org/10.22489/CinC.2016.182-399

Powers, D. M. (2011). Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. Journal of Machine Learning Technologies, 2(1), 37-63.

Probst, P., Boulesteix, A. L., & Bischl, B. (2020). Tunability: Importance of hyperparameters of machine learning algorithms. Journal of Machine Learning Research, 20(53), 1-32.

Roth, G. A., Mensah, G. A., Johnson, C. O., Addolorato, G., Ammirati, E., Baddour, L. M., ... & GBD-NHLBI-JACC Global Burden of Cardiovascular Diseases Writing Group. (2020). Global burden of cardiovascular diseases and risk factors, 1990–2019: Update from the GBD 2019 study. Journal of the American College of Cardiology, 76(25), 2982-3021. https://doi.org/10.1016/j.jacc.2020.11.010

Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747. https://doi.org/10.48550/arXiv.1609.04747

Russell, S. J., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach (4th ed.). Pearson.

Scherer, D., Müller, A., & Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. In Artificial Neural Networks–ICANN 2010 (pp. 92-101). Springer. https://doi.org/10.1007/978-3-642-15825-4_10

Sejnowski, T. J. (2020). The unreasonable effectiveness of deep learning in artificial intelligence. Proceedings of the National Academy of Sciences, 117(48), 30033-30038. https://doi.org/10.1073/pnas.1907373117

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & De Freitas, N. (2016). Taking the human out of the loop: A review of Bayesian optimization. Proceedings of the IEEE, 104(1), 148-175. https://doi.org/10.1109/JPROC.2015.2494218

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In International Conference on Learning Representations.

Smith, S. L., Dherin, B., Barrett, D. G., & De, S. (2021). On the origin of implicit regularization in stochastic gradient descent. In 9th International Conference on Learning Representations (ICLR). https://openreview.net/forum?id=rq_Qr0c1Hyo

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In Advances in Neural Information Processing Systems 25 (pp. 2951-2959). Curran Associates, Inc.

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. Information Processing & Management, 45(4), 427-437. https://doi.org/10.1016/j.ipm.2009.03.002

Springer, D. B., Tarassenko, L., & Clifford, G. D. (2016). Logistic regression-HSMM-based heart sound segmentation. IEEE Transactions on Biomedical Engineering, 63(4), 822-832.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.

Stevens, S. S., Volkmann, J., & Newman, E. B. (1937). A scale for the measurement of the psychological magnitude pitch. The Journal of the Acoustical Society of America, 8(3), 185-190. https://doi.org/10.1121/1.1915893

Tan, M., & Le, Q. (2021). EfficientNetV2: Smaller models and faster training. In International Conference on Machine Learning (pp. 10096-10106). PMLR.

Virani, S. S., Alonso, A., Benjamin, E. J., Bittencourt, M. S., Callaway, C. W., Carson, A. P., ... & American Heart Association Council on Epidemiology and Prevention Statistics Committee and Stroke Statistics Subcommittee. (2020). Heart disease and stroke statistics—2020 update: A report from the American Heart Association. Circulation, 141(9), e139-e596. https://doi.org/10.1161/CIR.0000000000000757

Waring, J., Lindvall, C., & Umeton, R. (2020). Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. Artificial Intelligence in Medicine, 104, 101822. https://doi.org/10.1016/j.artmed.2020.101822

Waskom, M. L. (2021). seaborn: statistical data visualization. Journal of Open Source Software, 6(60), 3021. https://doi.org/10.21105/joss.03021

World Health Organization. (2021). Cardiovascular diseases (CVDs) fact sheet. Retrieved from https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds

Yang, L., & Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. Neurocomputing, 415, 295-316. https://doi.org/10.1016/j.neucom.2020.07.061

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., ... & Hsieh, C. J. (2020). Large batch optimization for deep learning: Training BERT in 76

minutes. In 8th International Conference on Learning Representations (ICLR). https://openreview.net/forum?id=Syx4wnEtvH

Zhang, W., Han, J., & Deng, S. (2022). Heart sound classification based on scaled spectrogram and partial transfer learning. IEEE Sensors Journal, 22(16), 16285-16294. https://doi.org/10.1109/JSEN.2022.3163753

Royston, J. P. (1982). *Some Techniques for Assessing Multivariate Normality Based on the Shapiro–Wilk W. Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 32(2), 121–133.

Laerd Statistics. (2024). *Repeated Measures ANOVA and Tests of Normality using SPSS*. Retrieved from https://statistics.laerd.com

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks

Hershey, S., Chaudhuri, S., Ellis, D. P., Gemmeke, J. F., Jansen, A., Moore, R. C., ... & Wilson, K. (2017). CNN architectures for large-scale audio classification. In *IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 131-135).

Pons, J., Lidy, T., & Serra, X. (2017). Experimenting with musically motivated convolutional neural networks. In *IEEE International Workshop on Content-Based Multimedia Indexing* (pp. 1-6).