STUDI MACHINE LEARNING MENGGUNAKAN ARDUINO NANO 33 SENSE

Ferdinand Vincent Halim
NIM: 312110019

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI DAN DESAIN
UNIVERSITAS MA CHUNG
MALANG
2025

KATA PENGANTAR

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan laporan penelitian yang berjudul "Studi Penerapan Machine Learning pada Arduino Nano 33 BLE Sense." Penelitian ini disusun sebagai salah satu syarat untuk menyelesaikan program studi di Fakultas Teknologi dan Desain, Universitas Ma Chung. Penelitian ini bertujuan untuk mengembangkan sistem berbasis machine learning pada perangkat Arduino Nano 33 BLE Sense, yang diharapkan dapat berkontribusi dalam pengembangan teknologi IoT dan memberikan solusi praktis dalam berbagai bidang aplikasi sensorik.

Penulis menyadari bahwa penelitian ini tidak akan dapat terselesaikan tanpa bantuan, bimbingan, dan dukungan dari berbagai pihak. Oleh karena itu, pada kesempatan ini penulis ingin mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Bapak Prof. Dr.Eng. Romy Budhi Widodo, selaku dosen pembimbing, yang telah memberikan bimbingan, arahan, serta motivasi selama proses penelitian hingga penyusunan laporan ini.
2. Rekan-rekan mahasiswa, yang selalu memberikan dukungan dan kerjasama selama masa studi.
3. Keluarga tercinta, yang selalu memberikan doa, dukungan, dan motivasi tanpa henti kepada penulis.

Penulis menyadari bahwa laporan ini masih jauh dari sempurna, oleh karena itu kritik dan saran yang membangun sangat penulis harapkan untuk perbaikan di masa mendatang. Semoga laporan penelitian ini dapat memberikan manfaat bagi kita semua, khususnya dalam pengembangan teknologi dan penerapan machine learning pada perangkat IoT di berbagai bidang.

Malang, Mei 2025

Ferdinand Vincent Halim

DAFTAR ISI

DAFTAR GAMBAR

Bab I
Pendahuluan

1.1. Latar Belakang

Di era digital yang semakin berkembang, penggunaan mikrokontroler seperti Raspberry Pi dan Arduino Uno telah menjadi komponen vital dalam berbagai aplikasi Teknologi. Raspberry Pi, dengan kemampuan komputasi yang setara komputer mini, telah diimplementasikan dalam proyek-proyek mulai dari sistem otomasi rumah hingga stasiun pemantau cuaca. Sementara itu, Arduino Uno dengan arsitektur sederhananya telah menjadi pilihan utama para pengembang untuk proyek elektronika dan robotika skala kecil hingga menengah. Kedua platform ini menawarkan fleksibilitas tinggi dengan harga terjangkau, memungkinkan innovator dan peneliti mengembangkan Solusi Teknologi yang dapat diterapkan dalam kehidupan sehari-hari.

Seiring dengan revolusi industry 4.0, kebutuhan akan sistem cerdas yang mampu belajar dan beradaptasi semakin meningkat. Machine learning hadir sebagai Solusi untuk menganalisis data dalam jumlah besar dan menghasilkan prediksi atau keputusan yang akurat. Dalam konteks industry, kemampuan machine learning untuk mendeteksi anomali, mengoptimalkan proses produksi, dan melakukan prediksi perawatan telah terbukti meningkatkan efisiensi dan mengurangi biaya operasional. Integrasi machine learning dengan perangkat mikrokontroler membuka peluang baru dalam pengembangan sistem cerdas yang dapat beroperasi secara mandiri di edge device, mengurangi ketergantungan pada koneksi cloud dan meningkatkan respons real-time.

Perkembangan teknologi *machine learning* telah mendorong kemajuan di berbagai bidang, terutama dalam otomatisasi dan analisis data berbasis sensor. Dengan kemampuan machine learning untuk mengenali pola dan membuat keputusan berdasarkan data, penerapannya kini menjangkau perangkat kecil dan berdaya rendah melalui konsep TinyML (Tiny Machine Learning). TinyML memungkinkan model machine learning dapat dijalankan di perangkat mikro,

seperti mikrokontroler, sehingga membuka peluang bagi pengembangan solusi IoT yang hemat energi.

Salah satu perangkat mikrokontroler yang mendukung penerapan TinyML adalah Arduino Nano 33 BLE Sense. Perangkat ini dilengkapi dengan beragam sensor—seperti akselerometer, mikrofon, dan sensor suhu—serta konektivitas Bluetooth Low Energy. Fitur-fitur ini membuatnya ideal untuk aplikasi berbasis sensor yang membutuhkan pemrosesan data real-time di lokasi, seperti pemantauan lingkungan, pengenalan suara, dan analisis gerakan. Kebutuhan akan sistem yang dapat bekerja secara mandiri dengan daya rendah menjadikan Arduino Nano 33 BLE Sense sebagai salah satu platform yang potensial.

Di sisi lain, penerapan machine learning pada perangkat berdaya rendah menghadapi beberapa tantangan. Mikrokontroler seperti Arduino Nano 33 BLE Sense memiliki keterbatasan dalam hal memori dan kapasitas komputasi. Oleh karena itu, pemilihan model machine learning yang efisien menjadi hal yang sangat penting agar sistem dapat berjalan dengan lancar tanpa mengorbankan akurasi prediksi. Selain itu, pengolahan data dari sensor yang dapat digunakan sebagai input bagi model machine learning harus dioptimalkan untuk meminimalkan penggunaan daya.

Implementasi TinyML juga mengharuskan adanya metode kompresi dan optimasi model yang cermat. Model yang digunakan harus cukup ringan untuk dijalankan pada perangkat dengan kapasitas terbatas tanpa mengorbankan kinerja. Selain itu, diperlukan pemahaman mendalam tentang cara kerja model machine learning pada lingkungan berdaya rendah agar aplikasi dapat bertahan dalam jangka waktu yang lama tanpa intervensi pengguna.

Penelitian ini bertujuan untuk mengeksplorasi penerapan machine learning pada Arduino Nano 33 BLE Sense, yang dapat menjadi referensi untuk mengembangkan aplikasi IoT berbasis machine learning dengan konsumsi daya yang rendah. Diharapkan, penelitian ini dapat memberikan solusi yang efisien dan aplikatif bagi berbagai kebutuhan teknologi masa kini yang memerlukan sistem prediktif berbasis sensor pada perangkat dengan sumber daya terbatas.

1.2. **Batasan Masalah**

1. Penelitian ini berfokus pada implementasi model machine learning berbasis TensorFlow Lite pada Arduino Nano 33 BLE Sense.
2. Data yang diolah berasal dari sensor bawaan Arduino Nano 33 BLE Sense, yaitu akselerometer, mikrofon, dan sensor suhu, tanpa menggunakan sensor tambahan dari perangkat eksternal.
3. Model yang dikembangkan difokuskan pada klasifikasi berbasis data dari sensor bawaan.

1.3. **Tujuan**

Tujuan dari penelitian ini adalah:

1. Mengembangkan model machine learning yang efisien dan sesuai untuk diterapkan pada perangkat Arduino Nano 33 BLE Sense dengan memanfaatkan data dari sensor bawaan.
2. Mengimplementasikan proses pengolahan data dari sensor bawaan (akselerometer, mikrofon, dan sensor suhu) secara optimal agar dapat dimanfaatkan sebagai input bagi model machine learning.
3. Mengevaluasi performa model dalam kondisi nyata menggunakan data dari sensor bawaan pada Arduino Nano 33 BLE Sense.

1.4. **Manfaat**

Penelitian ini diharapkan dapat memberikan manfaat sebagai berikut:

1. Menyediakan referensi bagi pengembangan aplikasi IoT berbasis machine learning pada perangkat berdaya rendah.
2. Memberikan solusi efisien untuk aplikasi yang membutuhkan pemrosesan prediktif secara real-time di perangkat mikro.
3. Mendorong pengembangan teknologi TinyML yang hemat energi untuk berbagai bidang seperti kesehatan, lingkungan, dan otomasi

Bab II

Gambaran Umum Perusahaan

2.1. **Ma Chung Human-Machine Interaction Research Center**

Teknik Informatika merupakan disiplin keilmuan yang memfokuskan dirinya pada penyediaan kebutuhan penggunaan dan organisasi terhadap teknologi komputer. Sebagai perkiraan untuk saat ini dan masa yang akan datang, teknologi informasi menjadi tulang punggung pertumbuhan ekonomi bangsa. Saat ini pun inovasi teknologi informasi sudah terasa di berbagai bidang kehidupan manusia. Dalam hal ini, pemerintah telah mencanangkan bahwa pengembangan pendidikan tinggi dalam bidang komputer dan informatika merupakan salah satu program prioritas, bersama-sama dengan disiplin ilmu lainnya seperti rekayasa, perilaku, manajemen, akuntansi, dan kesenian.

Pendidikan tinggi diarahkan untuk mempersiapkan bangsa Indonesia dalam menghadapi era pembangunan industri dan informasi. Untuk itu pemerintah melalui Direktorat Jendral Pendidikan Tinggi pada tanggal 07 Juli 2007 menginstruksikan untuk membuka Program Studi S1 Teknik Informatika berdasarkan Surat Keputusan Penyelenggaraan 15274/D/T/K-VII/2013, pada tanggal 22 Maret 2013. Pada saat ini, Program Studi Teknik Informatika memperoleh nilai akreditasi B berdasarkan Keputusan BAN-PT No. 2546/SK/BAN-PT/Ak-PPJ/S/IV/2021, tanggal 28 April 2021.

Pesatnya kemajuan juga tidak lepas dari aspek interaksi antara manusia dan dalam pengembangannya. Untuk menunjang pengembangan dan terapannya kepada human welfare dan aspek interaksinya maka dibentuk Ma Chung Human-Machine Interaction Research Center sesuai Surat Keputusan 031/MACHUNG/FST/SK-DEK/IX/2019, pada tanggal 11 September 2019.

Ma Chung Human-Machine Interaction Research Center bergerak dalam bidang kajian meliputi namun tidak terbatas kepada: machine vision, human-computer interaction, untuk manusia berkebutuhan khusus, dan aplikasi mobile yang mendukung penggunaan aplikasi yang lebih meluas.

2.2. **Visi dan Misi Ma Chung Human-Machine Interaction Research Center**

Ma Chung Human-Machine Interaction Research Center merupakan pusat studi yang berada di bawah naungan Teknik Informatika Universitas Ma Chung yang memiliki visi dan misi yang sama dalam setiap aspeknya. Berikut adalah visi dari Teknik Informatika Universitas Ma Chung:

"Pada tahun 2025 Menjadi Program Studi Teknik Informatika aras utama di Indonesia Timur yang mendukung eksplorasi sumber daya alam beserta pengelolaan bisnisnya sebagai perwujudan memuliakan Tuhan dan berkontribusi nyata bagi kesejahteraan masyarakat."

Berikut adalah misi dari Teknik Informatika Universitas Ma Chung:

a. Menyelenggarakan pengajaran, penelitian dan pengabdian kepada masyarakat yang berfokus pada pengembangan ilmu-ilmu 5elola informatika untuk pengelolaan sumberdaya alam dan bisnis.

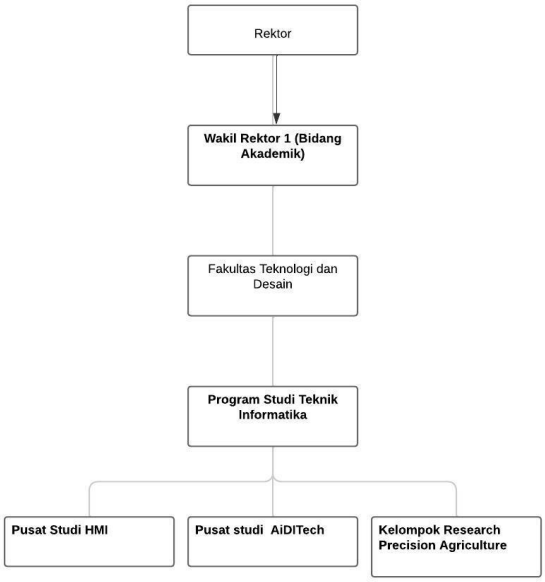
b. Membentuk dan mengembangkan generasi motivator yang mempunyai jiwa pemimpin dan wirausahawan dengan bertitik berat pada perkembangan akhlak, bersikap rendah hati, dan berwawasan.

c. Membentuk lulusan siap pakai yang berkualitas tinggi dan mampu bersaing pada pasar informasi global.

d. Menyelenggarakan Program Studi dengan tata kelola yang baik dan profesional.

2.3. **Struktur Organisasi Ma Chung Human-Machine Interaction Research Center**

Pada struktur organisasi Pusat Studi Interaksi Manusia dan Mesin Universitas Ma Chung yaitu bertujuan untuk pengembangan ilmu dan penelitian. Gambar 2.1 menunjukkan struktur organisasi Ma Chung Human-Machine Interaction Research



Gambar 2. 1 Struktur organisasi dari Rektor hingga pusat studi

2.4. **Publikasi Ilmiah Ma Chung HMI Research Center**

Ma Chung Human-Machine Interaction Research Center pada saat ini memiliki topik-topik riset sebagai berikut:

a. Rancang Bangun Pengontrol Gerakan Robot Openmanipulator dengan Matlab

Penelitian ini bertujuan mengembangkan aplikasi untuk mengontrol Gerakan robot arm menggunakan MATLAB. Robot arm yang digunakan terdiri dari perangkat U2D2 board,adaptor daya, dan servo Dynamixel. Pengujian dilakukan dengan menggerakkan robot arm ke posisi yang telah ditentukan, dan hasilnya menunjukkan keberhasilan 100% dalam menjalankan pergerakan. Meskipun demikian, penyempurnaan lebih lanjut diperlukan agar robot arm dapat berfungsi lebih optimal di masa mendatang.

b. Analisis Perbandingan Waktu Reaksi pada individu Usia Dewasa Muda dan Usia Lanjut dalam Tugas Kognitif

Penelitian ini menguji perbedaan waktu reaksi kognitif antara kelompok usia 13-25 tahun(masa remaja) dan 50 tahun ke atas (akhir kedewasaan), yang masing-masing ditandai oleh perkembangan optimal dan penurunan kognitif bertahap. Delapan subjek dengan variasi usia dianalisis menggunakan software GoStats, dengan uji

Liliefors untuk normalitas data dan uji Mann-Whitney U untuk mengukur perbedaan waktu reaksi. Hasil menunjukkan data tidak berdistribusi normal ($p < 0.05$) dan tidak ada perbedaan signifikan antara kedua kelompok pada empat tugas kognitif yang diuji ($p > 0.05$), yaitu Simple Reaction, Physical Matching, dan Class Matching. Temuan ini mengindikasikan bahwa perbedaan usia tidak selalu mempengaruhi semua aspek fungsi kognitif. Penelitian lanjutan diperlukan untuk mengeksplorasi factor-faktor lain seperti jenis tugas, Tingkat kesulitan, dan pengalaman individu.

c. Hyperparameter Tuning for Malaria Detection using Convulation Neural network

Malaria, yang disebabkan oleh parasit Plasmodium, dapat dideteksi melalui pemeriksaan blood smear menggunakan mikroskop setelah pewarnaan fluoresensi. Selain itu, metode computer vision dan deep learning telah digunakan untuk mendeteksi malaria dari citra sel darah merah, dengan model Convolutional Neural Network (CNN) terbukti memberikan hasil akurasi tinggi. Penelitian ini mengevaluasi pengaruh hyperparameter tuning terhadap akurasi, presisi, recall, f1-score, dan Matthew Correlation Coefficient (MCC). Dua arsitektur CNN, yaitu Rajaraman dan BaselineNet, digunakan dalam eksperimen untuk membandingkan performa model.

Bab III
Tinjauan Pustaka

3.1. Rujukan Pustaka

Tiny Machine Learning (TinyML) adalah implementasi machine learning pada perangkat berdaya rendah, seperti mikrokontroler, untuk menjalankan model yang mampu melakukan inferensi secara lokal tanpa memerlukan koneksi cloud. Salah satu pustaka yang digunakan adalah TensorFlow Lite for Microcontrollers, yang memungkinkan penggunaan model machine learning dengan konsumsi daya di bawah 1 mW, sesuai dengan karakteristik perangkat seperti Arduino Nano 33 BLE Sense.

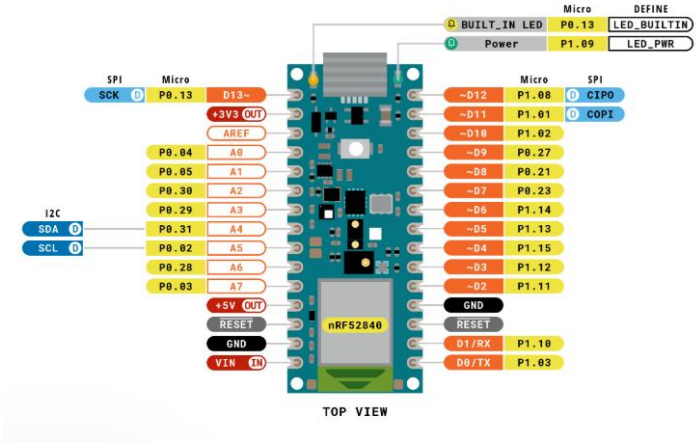
Dalam buku TinyML: *Machine Learning* with TensorFlow Lite on Arduino and Ultra-low-power Microcontrollers, Pete Warden dan Daniel Situnayake menjelaskan proses pengembangan model machine learning, mulai dari pengumpulan data, desain arsitektur model, pelatihan, hingga deployment ke perangkat embedded. Fokus utamanya adalah pada optimasi sumber daya untuk mengakomodasi keterbatasan perangkat mikrokontroler.

3.2. Penjelasan Istilah Asing

1. **TinyML**: Istilah untuk teknologi machine learning yang diterapkan pada perangkat mikro dengan daya dan kapasitas terbatas.
2. **TensorFlow Lite for Microcontrollers**: Versi TensorFlow yang dioptimalkan untuk perangkat berdaya rendah dan kapasitas memori kecil.
3. **Inferensi**: Proses menjalankan model machine learning untuk menghasilkan prediksi atau pengenalan berdasarkan input data.
4. **Deployment**: Tahap penerapan model ke perangkat target, seperti mikrokontroler, agar model dapat digunakan secara langsung.
5. **Arduino Nano 33 BLE Sense**: Mikrokontroler yang dilengkapi dengan berbagai sensor dan mendukung konektivitas Bluetooth Low Energy (BLE).

3.3. **Arduino Nano 33 Sense BLE**

Arduino Nano 33 Sense adalah salah satu papan mikrokontroler dari keluarga Arduino yang dirancang khusus untuk aplikasi berbasis sensor dan pengolahan data IoT (Internet of Things). Papan ini dilengkapi dengan berbagai sensor bawaan dan menggunakan mikrokontroler berdaya rendah untuk memungkinkan penggunaan di perangkat kecil dan portabel.



Gambar 3. 1 Arduino Nano 33 Sense BLE

Gambar 3.1 adalah pinout diagram dari Arduino Nano 33 BLE yang menggunakan nRF52840 sebagai mikrokontrolernya. Diagram ini menunjukkan semua pin yang tersedia, termasuk fungsinya dalam berbagai mode operasi seperti GPIO, SPI, I2C, dan UART. Berikut adalah penjelasan bagian-bagiannya:

1. Mikrokontroler

nRF52840 adalah mikrokontroler berbasis ARM Cortex-M4F yang mendukung komunikasi nirkabel seperti Bluetooth Low Energy (BLE).

2. Power Pins

+3V3 OUT → Output tegangan 3.3V.

+5V OUT → Output tegangan 5V (tergantung sumber daya).

VIN IN → Tegangan input eksternal.

GND → Ground (0V).

RESET → Pin untuk mereset mikrokontroler.

3. Analog Pins (A0 - A7)

Digunakan untuk membaca sinyal analog (0-3.3V).

Mapped ke port P0.04 - P0.03 dari nRF52840.

4. Digital Pins (D0 - D13)

Digunakan untuk input/output digital.

Beberapa pin mendukung PWM (ditandai dengan ~).

Contohnya:

D13 (P0.13) → Terhubung ke LED_BUILTIN.

D12 - D2 → Dapat digunakan untuk GPIO atau fungsi lainnya.

5. SPI Pins

SCK (P0.13) → Serial Clock.

CIPO (P1.08) → Controller In Peripheral Out (MISO).

COPI (P1.02) → Controller Out Peripheral In (MOSI).

6. I2C Pins

SDA (P0.20) → Data line.

SCL (P0.19) → Clock line.

7. UART Pins (Serial Communication)

D1/RX (P1.10) → Menerima data serial.

D0/TX (P1.03) → Mengirim data serial.

8. LED Indicators

BUILT_IN LED (P0.13) → LED internal yang dapat dikendalikan dengan kode.

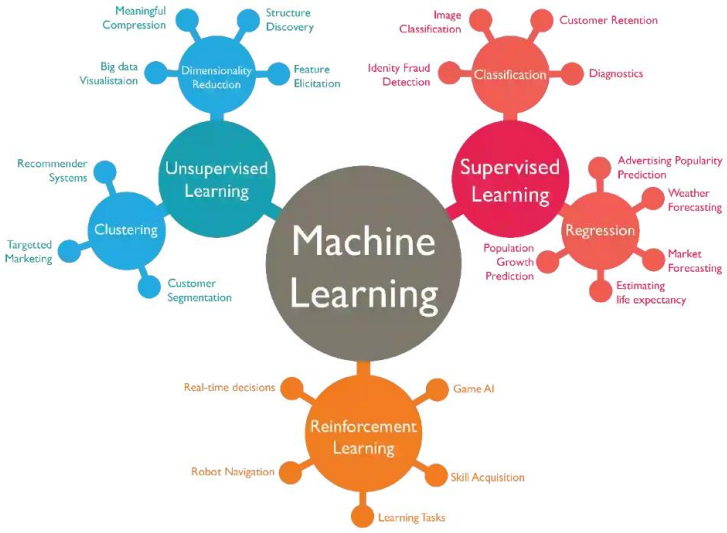
LED_PWR (P1.09) → Menunjukkan bahwa board mendapatkan daya.

Spesifikasi utama Arduino Nano 33 BLE:

1. Mikrokontroler: Arduino Nano 33 Sense menggunakan Nordic nRF52840, sebuah mikrokontroler ARM Cortex-M4 32-bit yang memiliki fitur Bluetooth Low Energy (BLE).
2. Sensor Bawaan: Papan ini dilengkapi dengan berbagai sensor bawaan, termasuk:
    a. IMU (Inertial Measurement Unit): Sensor 9-axis untuk mendeteksi akselerasi, rotasi, dan medan magnet.
    b. Sensor Suhu dan Kelembaban: Untuk memantau kondisi lingkungan.
    c. Sensor Cahaya: Mengukur intensitas cahaya.
    d. Microphone: Menggunakan sensor suara untuk aplikasi pemrosesan audio.
    e. Gesture Sensor: Untuk mendeteksi gerakan seperti sapuan tangan.
3. Konektivitas: Bluetooth Low Energy (BLE) untuk komunikasi data secara nirkabel.
4. Konsumsi Daya Rendah: Dirancang untuk proyek yang memerlukan efisiensi energi, sehingga cocok untuk perangkat bertenaga baterai.
5. Ukuran: Kompak, dengan form factor yang serupa dengan Arduino Nano, mempermudah integrasi pada proyek yang sudah menggunakan papan Nano.

3.4. Machine Learning

Machine Learning (ML) adalah cabang dari kecerdasan buatan (Artificial Intelligence, AI) yang berfokus pada pengembangan sistem komputer yang dapat belajar dari data dan meningkatkan kinerjanya secara otomatis tanpa harus diprogram secara eksplisit. ML menggunakan algoritma untuk menganalisis data, mengenali pola, dan membuat prediksi atau keputusan berdasarkan data tersebut. Gambar 3.2 menunjukkan jenis-jenis *Machine Learning*.



Gambar 3. 2 Machine Learning

Cara Kerja *Machine Learning*:

1. Data Input: Data dikumpulkan dan disiapkan, mencakup data historis atau data yang relevan dengan masalah yang ingin diselesaikan.
2. Pelatihan Model: Algoritma ML dilatih menggunakan data. Model belajar mengenali pola dalam data dan membuat hubungan antara input dan output.
3. Prediksi/Inferensi: Setelah dilatih, model digunakan untuk memprediksi hasil berdasarkan data baru.
4. Evaluasi dan Penyesuaian: Model dievaluasi untuk mengukur kinerjanya menggunakan metrik tertentu. Jika model kurang akurat, proses pelatihan dan penyempurnaan dilakukan kembali.

Jenis *Machine Learning:*

1. Supervised Learning:Model dilatih menggunakan data berlabel, di mana setiap input memiliki output yang diketahui.

   Contoh:

   a. Prediksi harga rumah berdasarkan luas dan lokasi.
   b. Klasifikasi email sebagai spam atau bukan.
2. Unsupervised Learning: Model belajar dari data yang tidak memiliki label, dengan tujuan menemukan pola tersembunyi.

   Contoh:

a. Klasterisasi pelanggan berdasarkan kebiasaan belanja.
b. Deteksi anomali dalam data jaringan.

3. Reinforcement Learning: Model belajar melalui interaksi dengan lingkungan, mendapatkan umpan balik dalam bentuk reward atau punishment.

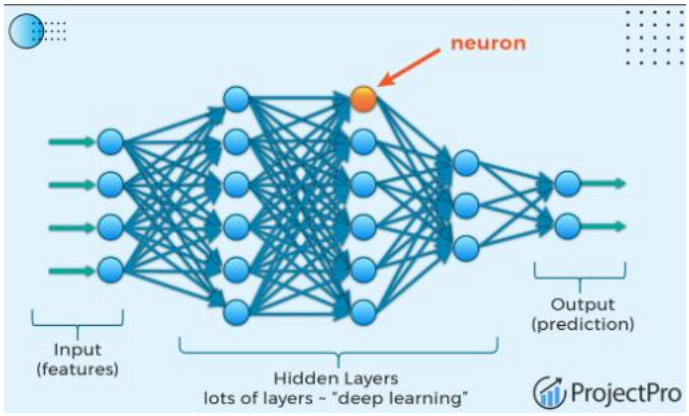
   Contoh:

   a. Robot yang belajar berjalan.
   b. Sistem bermain game (misalnya, AlphaGo).

4. Semi-Supervised Learning: Kombinasi dari supervised dan unsupervised learning, menggunakan sebagian data berlabel dan sebagian tidak berlabel.

3.5. Deep Learning

Deep Learning adalah salah satu cabang dari Machine Learning (Pembelajaran Mesin) yang menggunakan jaringan saraf tiruan (neural networks) dengan banyak lapisan (deep layers) untuk memodelkan dan memecahkan masalah kompleks. Deep Learning meniru cara kerja otak manusia dalam memproses informasi, khususnya dalam hal pengenalan pola dan pengambilan keputusan. Gambar 3.3 menunjukkan arsitektur Deep Learning.



Gambar 3. 3 Contoh Arsitektur Deep Learning

Berikut penjelasan tentang Deep Learning:

1. Konsep Dasar Deep Learning

Deep Learning menggunakan struktur jaringan saraf tiruan yang terdiri dari beberapa lapisan (layers). Setiap lapisan terdiri dari neuron (unit pemroses) yang menerima input, melakukan komputasi, dan menghasilkan output. Lapisan-lapisan ini dibagi menjadi:

- Input Layer: Lapisan pertama yang menerima data mentah.
- Hidden Layers: Lapisan tersembunyi yang melakukan transformasi matematis pada data.
- Output Layer: Lapisan terakhir yang menghasilkan hasil akhir (prediksi atau klasifikasi).

Semakin banyak lapisan tersembunyi, semakin "dalam" (deep) jaringan tersebut, sehingga disebut Deep Learning.

2. Cara Kerja Deep Learning

- Alur Kerja Deep Learning

  Pada bagian sebelumnya, kita telah menjelaskan sebuah skenario penggunaan deep learning untuk memprediksi kapan sebuah mesin pabrik kemungkinan akan mengalami kerusakan. Deep learning dapat menjadi solusi yang sangat efektif dalam mendeteksi anomali atau kegagalan mesin dengan menganalisis pola dari data historis yang tersedia. Namun, untuk membangun model yang akurat dan dapat diandalkan, diperlukan serangkaian langkah yang sistematis.

  Dalam bagian ini, kita akan membahas langkah-langkah yang diperlukan untuk mengembangkan model deep learning dari awal hingga dapat digunakan untuk inferensi dan evaluasi. Proses ini mencakup berbagai tahapan yang penting untuk memastikan bahwa model yang dihasilkan dapat bekerja dengan baik dalam kondisi nyata.

  Proses ini melibatkan langkah-langkah berikut:

  a. Menentukan tujuan
  b. Mengumpulkan dataset
  c. Merancang arsitektur model

d. Melatih model
e. Mengonversi model
f. Menjalankan inferensi
g. Mengevaluasi dan memperbaiki model

Langkah a sampai d adalah bagian penting dari Deep Learning yang menghasilkan suatu model. Langkah e sampai g adalah bagian yang membawa model untuk di upload hardware, hal ini merupakan inti dari Tiny *Machine Learning*. Setiap langkah ini memiliki peran krusial dalam membangun sistem prediksi berbasis deep learning yang andal. Mari kita bahas satu per satu:

a. Menentukan Tujuan

Ketika merancang algoritma, penting untuk menetapkan tujuan yang jelas sejak awal. Tanpa tujuan yang terdefinisi dengan baik, sulit untuk menentukan data yang dibutuhkan, metode yang akan digunakan, dan bagaimana cara mengevaluasi keberhasilan model.

Dalam *Machine Learning*, tujuan umumnya berkaitan dengan prediksi atau klasifikasi. Kita perlu menentukan apa yang ingin diprediksi agar dapat memilih dataset yang relevan serta arsitektur model yang tepat. Jika tujuan tidak didefinisikan dengan jelas, hasil yang diperoleh mungkin tidak sesuai dengan harapan.

Dalam contoh ini, kita ingin memprediksi apakah mesin pabrik akan mengalami kerusakan. Ini dapat dianggap sebagai masalah klasifikasi, yaitu tugas *Machine Learning* yang mengkategorikan data ke dalam beberapa kelompok yang telah ditentukan sebelumnya.

Sebagai contoh, kita bisa membagi kondisi mesin menjadi dua kelas utama:

- "Normal": Mesin beroperasi tanpa gangguan dan tidak menunjukkan tanda-tanda akan rusak.
- "Abnormal": Mesin menunjukkan indikasi adanya masalah dan kemungkinan akan mengalami kegagalan dalam waktu dekat.

Dengan demikian, tujuan kita adalah menciptakan model yang mampu mengklasifikasikan data sensor mesin ke dalam kategori "normal" atau "abnormal" dengan tingkat akurasi yang tinggi.

- Mengumpulkan Dataset

  Setelah menentukan tujuan model, langkah berikutnya adalah mengumpulkan dataset yang akan digunakan untuk melatih model deep learning. Data yang digunakan dalam pelatihan harus mencerminkan berbagai kondisi operasional mesin agar model dapat mengenali pola dengan baik.

- Memilih Data yang Relevan

  Model deep learning memiliki kemampuan untuk menyaring informasi yang tidak relevan, tetapi lebih baik jika sejak awal kita hanya menggunakan data yang benar-benar berkontribusi terhadap prediksi yang akan dilakukan.

  Sebagai contoh, dalam skenario prediksi kegagalan mesin pabrik, ada banyak jenis data yang tersedia. Ini bisa mencakup suhu operasi, tingkat getaran, tekanan, kecepatan rotasi, atau bahkan faktor eksternal seperti kondisi cuaca atau jadwal pemeliharaan rutin.

  Namun, tidak semua data tersebut relevan untuk tujuan kita. Misalnya, menu makanan di kantin pabrik kemungkinan besar tidak memiliki pengaruh terhadap kondisi mesin. Jika kita memasukkan data yang tidak relevan, model bisa saja belajar pola yang salah, misalnya menghubungkan kerusakan mesin dengan hari-hari tertentu ketika menu tertentu disajikan. Oleh karena itu, pemilihan fitur atau variabel yang akan digunakan dalam model harus dilakukan dengan hati-hati. Kita dapat mengandalkan kombinasi antara pemahaman domain (pengetahuan industri) dan teknik statistik untuk menentukan variabel yang benar-benar berkontribusi terhadap prediksi. Jika masih ragu, pendekatan eksperimental dapat dilakukan, misalnya dengan membandingkan dua model—satu dengan dataset lengkap dan satu lagi dengan dataset yang sudah difilter—untuk melihat mana yang memberikan hasil terbaik.

  Dalam skenario ini, kita memilih laju produksi, suhu, dan getaran sebagai fitur utama yang akan digunakan dalam pelatihan model. Setelah fitur ditentukan, langkah selanjutnya adalah mengumpulkan data dalam jumlah yang cukup agar model dapat belajar dengan baik.

- Mengumpulkan Data

  Salah satu tantangan utama dalam deep learning adalah menentukan jumlah data yang cukup untuk pelatihan. Jumlah data yang diperlukan tergantung pada berbagai faktor seperti:

  - Kompleksitas hubungan antara variabel dalam dataset
  - Tingkat kebisingan (noise) dalam data
  - Seberapa mudah model dapat membedakan antara kelas "normal" dan "abnormal"

  Secara umum, semakin banyak data yang dikumpulkan, semakin baik model dalam mengenali pola. Namun, data harus mencakup berbagai kondisi dan variasi agar model dapat bekerja dengan baik di berbagai situasi.

  Sebagai contoh, jika suhu mesin bervariasi antara musim panas dan musim dingin, maka dataset harus mencakup data dari kedua musim tersebut. Jika mesin dapat mengalami berbagai jenis kegagalan, kita harus memastikan bahwa semua jenis kegagalan tersebut terwakili dalam dataset. Data dalam pabrik biasanya direkam dalam bentuk time series, yaitu serangkaian pengukuran yang dilakukan secara berkala. Contoh data yang bisa dikumpulkan adalah:

  - Suhu mesin dicatat setiap 1 menit
  - Laju produksi dicatat setiap 2 menit
  - Getaran dicatat setiap 10 detik

  Setelah data dikumpulkan, langkah selanjutnya adalah mengubahnya ke dalam format yang sesuai untuk model deep learning.

- Pelatihan Model

  Pelatihan model adalah proses di mana model belajar menghasilkan output yang benar berdasarkan sekumpulan data masukan. Proses ini melibatkan pemberian data pelatihan ke dalam model dan menyesuaikan parameter model (berupa bobot dan bias) secara bertahap agar prediksinya semakin akurat.

  Model deep learning terdiri dari jaringan neuron buatan yang direpresentasikan dalam bentuk array angka yang tersusun dalam

lapisan-lapisan. Bobot awal dalam model biasanya diatur secara acak, sedangkan bias dimulai dari nol. Ketika data dimasukkan ke dalam model, ia mengalami berbagai transformasi matematis berdasarkan bobot dan bias di setiap lapisan untuk menghasilkan output. Proses pelatihan menggunakan algoritma backpropagation, yang secara bertahap memperbaiki bobot dan bias berdasarkan perbedaan antara output model dan output yang diharapkan. Pelatihan berlangsung selama beberapa epoch hingga model mencapai performa optimal atau tidak mengalami peningkatan lebih lanjut.

Kinerja model dinilai menggunakan loss dan accuracy:

- Loss mengukur seberapa jauh prediksi model dari nilai yang benar.
- Accuracy mengukur persentase prediksi yang benar.

Jika model terlalu sederhana dan tidak dapat mengenali pola dalam data, ini disebut underfitting. Sebaliknya, jika model terlalu menghafal data pelatihan tanpa dapat mengenali pola baru, ini disebut overfitting.

Untuk mencegah overfitting, beberapa teknik digunakan:

- Regularisasi (L1/L2, dropout): Membatasi kompleksitas model agar tidak menghafal data pelatihan secara berlebihan.
- Augmentasi Data: Membuat variasi data baru dari data asli untuk meningkatkan keberagaman input.
- Meningkatkan Ukuran Dataset: Semakin banyak data yang digunakan, semakin baik model dalam memahami pola.

Dataset biasanya dibagi menjadi tiga bagian:

- Training set (60%): Digunakan untuk melatih model.
- Validation set (20%): Digunakan untuk mengevaluasi kinerja model selama pelatihan.
- Test set (20%): Digunakan setelah pelatihan selesai untuk memastikan model tidak overfit terhadap data pelatihan dan validasi.

3. Arsitektur Deep Learning

Beberapa arsitektur Deep Learning yang populer meliputi:

- Convolutional Neural Networks (CNNs): Digunakan untuk pemrosesan gambar dan video. CNNs menggunakan lapisan konvolusi untuk mengekstrak fitur dari data spasial.
- Recurrent Neural Networks (RNNs): Dirancang untuk data berurutan seperti teks atau time series. RNNs memiliki memori internal untuk mengingat informasi dari langkah sebelumnya.
- Transformers: Arsitektur yang digunakan dalam pemrosesan bahasa alami (NLP) seperti model BERT dan GPT. Transformers menggunakan mekanisme attention untuk memproses data secara paralel.
- Autoencoders: Digunakan untuk kompresi data dan reduksi dimensi.
- Generative Adversarial Networks (GANs): Dua jaringan (generator dan discriminator) bekerja bersama untuk menghasilkan data baru yang mirip dengan data asli.

4. Keunggulan Deep Learning

Berikut adalah beberapa keunggulan Deep Learning:

- Kemampuan Menangani Data Kompleks: Deep Learning dapat memproses data yang tidak terstruktur seperti gambar, teks, dan suara.
- Otomatisasi Fitur: Tidak memerlukan ekstraksi fitur manual karena jaringan dapat mempelajari fitur secara otomatis.
- Akurasi Tinggi: Dalam banyak kasus, Deep Learning mencapai akurasi yang lebih tinggi dibandingkan metode tradisional.

5. Tantangan Deep Learning

Berikut adalah beberapa tantangan Deep Learning:

- Kebutuhan Data Besar: Deep Learning memerlukan dataset yang besar untuk melatih model secara efektif.
- Sumber Daya Komputasi: Proses pelatihan membutuhkan GPU atau TPU yang mahal dan waktu yang lama.

- Overfitting: Model mungkin terlalu spesifik pada data pelatihan dan gagal generalisasi ke data baru.
- Interpretabilitas: Model Deep Learning sering dianggap sebagai "black box" karena sulit untuk memahami bagaimana keputusan diambil.

6. Aplikasi Deep Learning

Deep Learning digunakan dalam berbagai bidang, seperti:

- Computer Vision: Pengenalan wajah, deteksi objek, dan mobil otonom.
- Natural Language Processing (NLP): Terjemahan mesin, chatbot, dan analisis sentimen.
- Speech Recognition: Asisten virtual seperti Siri dan Alexa.
- Kesehatan: Diagnosis medis dan analisis gambar medis.
- Permainan: AI dalam game seperti AlphaGo.

7. Perkembangan Terkini

Deep Learning terus berkembang dengan inovasi seperti:

- Reinforcement Learning: Kombinasi Deep Learning dengan pembelajaran penguatan untuk aplikasi seperti robotika.
- Transfer Learning: Menggunakan model yang sudah dilatih untuk tugas baru dengan sedikit data.
- Explainable AI (XAI): Upaya untuk membuat model Deep Learning lebih transparan dan dapat diinterpretasikan.

Dengan kemampuannya yang luar biasa dalam memproses data kompleks, Deep Learning telah menjadi fondasi bagi banyak kemajuan dalam kecerdasan buatan (AI) dan terus membuka peluang baru di berbagai industri.

3.6. **TensorFlow Lite**

TensorFlow Lite adalah versi ringan (lightweight) dari TensorFlow, sebuah framework populer untuk *Machine Learning* dan deep learning yang dikembangkan oleh Google. TensorFlow Lite dirancang khusus untuk perangkat mobile, embedded systems, dan perangkat IoT (Internet of Things) yang memiliki sumber daya terbatas, seperti memori, daya komputasi, dan baterai. Tujuannya adalah

memungkinkan penerapan model *Machine Learning* secara efisien di perangkat edge (perangkat lokal) tanpa bergantung pada cloud.

Berikut penjelasan tentang TensorFlow Lite:

1. Tujuan TensorFlow Lite

- Optimasi untuk Perangkat Edge: TensorFlow Lite memungkinkan model *Machine Learning* berjalan di perangkat dengan sumber daya terbatas, seperti smartphone, mikrokontroler, dan perangkat IoT.
- Efisiensi: Mengurangi ukuran model dan kebutuhan komputasi sehingga cocok untuk perangkat dengan daya dan memori terbatas.
- Latensi Rendah: Memproses data secara lokal di perangkat, mengurangi ketergantungan pada koneksi internet dan meningkatkan kecepatan respons.
- Privasi: Data tetap berada di perangkat lokal, meningkatkan keamanan dan privasi pengguna.

2. Arsitektur TensorFlow Lite

TensorFlow Lite terdiri dari beberapa komponen utama:

- TensorFlow Lite Converter: Alat untuk mengonversi model TensorFlow standar (dalam format SavedModel atau Keras) ke format TensorFlow Lite (.tflite). Proses ini melibatkan optimasi seperti kuantisasi (quantization) dan pemangkasan (pruning) untuk mengurangi ukuran model.
- TensorFlow Lite Interpreter: Mesin yang menjalankan model .tflite di perangkat target. Interpreter ini dirancang untuk efisiensi dan kompatibilitas dengan berbagai platform.
- TensorFlow Lite Model: Model yang telah dioptimalkan dan dikonversi ke format .tflite.
- Dukungan Hardware Accelerator: TensorFlow Lite mendukung akselerasi hardware seperti GPU, DSP (Digital Signal Processor), dan Neural Processing Units (NPU) untuk meningkatkan performa.

3. Fitur Utama TensorFlow Lite

   Berikut adalah beberapa Fitur utama TensorFlow Lite:

   - Kuantisasi (Quantization): Teknik untuk mengurangi ukuran model dan mempercepat inferensi dengan mengubah bobot dan aktivasi dari floating-point (32-bit) ke integer (8-bit). Ini mengurangi kebutuhan memori dan daya komputasi.
   - Pemangkasan (Pruning): Menghilangkan bobot yang tidak penting dari model untuk mengurangi ukuran dan kompleksitas.
   - Selective Operator Kernels: TensorFlow Lite hanya menyertakan operator yang diperlukan untuk model tertentu, mengurangi ukuran biner (binary size).
   - Dukungan Multi-Platform: TensorFlow Lite mendukung Android, iOS, Linux, dan mikrokontroler (melalui TensorFlow Lite for Microcontrollers).
   - Pre-Trained Models: Menyediakan model pra-latih (pre-trained) yang siap digunakan untuk tugas seperti klasifikasi gambar, deteksi objek, dan NLP.

4. Workflow Penggunaan TensorFlow Lite

   Berikut adalah Workflow penggunaan TensorFlow Lite:

   - Pelatihan Model: Model dilatih menggunakan TensorFlow atau Keras di lingkungan yang mendukung (seperti PC atau cloud).
   - Konversi Model: Model diubah ke format TensorFlow Lite menggunakan TensorFlow Lite Converter. Proses ini melibatkan optimasi seperti kuantisasi.
   - Deploy ke Perangkat: Model .tflite di-deploy ke perangkat target (smartphone, mikrokontroler, dan lain lain).
   - Inferensi: Model dijalankan di perangkat menggunakan TensorFlow Lite Interpreter.

   5. Aplikasi TensorFlow Lite

   - Mobile Applications: Aplikasi seperti pengenalan gambar, terjemahan teks, dan asisten virtual di smartphone.

- IoT Devices: Perangkat pintar seperti kamera keamanan, sensor lingkungan, dan perangkat wearables.
- Mikrokontroler:TensorFlowLiteforMicrocontrollers memungkinkan model ML berjalan di perangkat dengan sumber daya sangat terbatas, seperti Arduino dan ESP32.

6. Keunggulan TensorFlow Lite

- Ringan dan Cepat: Dirancang untuk performa tinggi dengan sumber daya minimal.
- Fleksibel: Mendukung berbagai jenis model dan tugas *Machine Learning*.
- Kompatibilitas Luas: Dapat diintegrasikan dengan Android, iOS, dan sistem embedded.
- Dukungan Komunitas: TensorFlow Lite didukung oleh komunitas besar dan dokumentasi yang lengkap.

7. Contoh Penggunaan TensorFlow Lite

- Pengenalan Gambar: Aplikasi yang mengidentifikasi objek dalam gambar menggunakan model CNN.
- Pemrosesan Bahasa Alami (NLP): Aplikasi chat atau terjemahan teks di perangkat mobile.
- Deteksi Suara: Sistem perintah suara di perangkat IoT.
- Kesehatan: Aplikasi yang menganalisis data sensor untuk memantau kesehatan pengguna.

8. Cara Penerapan TensorFlow Lite

Saat ini, library Arduino_TensorFlowLite sudah tidak tersedia melalui Library Manager di Arduino IDE. Hal ini dikarenakan permintaan dari pengelola TensorFlow Lite Micro untuk menghapusnya dari daftar Library Manager. Oleh karena itu, pemasangan library ini harus dilakukan secara manual.

Berikut adalah langkah-langkah untuk menginstal library TensorFlow Lite secara manual di Arduino IDE:

Langkah 1: Unduh Library dari GitHub

- Buka halaman resmi repository TensorFlow Lite Micro untuk Arduino:
- https://github.com/tensorflow/tflite-micro-arduino-examples
- Klik tombol "Code" dan pilih "Download ZIP" untuk mengunduh seluruh repository sebagai file ZIP.

Langkah 2: Instal Library di Arduino IDE

Buka Arduino IDE.

- Navigasikan ke menu Sketch > Include Library > Add .ZIP Library....
- Pilih file ZIP yang telah diunduh sebelumnya dan klik "Open".
- Tunggu hingga proses instalasi selesai.

Langkah 3: Verifikasi Instalasi

Setelah instalasi, Anda dapat memverifikasi bahwa library telah terpasang dengan membuka menu Sketch > Include Library dan mencari Arduino_TensorFlowLite dalam daftar library yang tersedia.

Alternatif: Menggunakan Git untuk Clone Repository

Jika Anda lebih nyaman menggunakan Git, Anda dapat meng-clone repository langsung ke folder library Arduino Anda:

- Buka terminal atau command prompt.
- Navigasikan ke folder library Arduino Anda. Lokasi default biasanya:

Windows: C:\Users\<NamaPengguna>\Documents\Arduino\libraries

macOS/Linux: ~/Documents/Arduino/libraries

- Jalankan perintah berikut:

git clone https://github.com/tensorflow/tflite-micro-arduino-examples Arduino_TensorFlowLite

- Setelah proses clone selesai, library akan tersedia di Arduino IDE.

9. Perkembangan Terkini

- TensorFlow Lite for Microcontrollers: Versi yang dirancang untuk perangkat dengan sumber daya sangat terbatas, seperti mikrokontroler.
- Model Garden: Koleksi model pra-latih yang siap digunakan untuk berbagai tugas.
- Dukungan Hardware Terbaru: Integrasi dengan hardware accelerator seperti Coral Edge TPU dan ARM NPU.

Dengan TensorFlow Lite, pengembang dapat membawa kekuatan *Machine Learning* ke perangkat edge, memungkinkan aplikasi yang lebih cepat, lebih aman, dan lebih efisien.

3.7. **Tiny Machine Learning (TinyML)**

Buku ini bertujuan untuk membantu pengembang dengan pengalaman dasar dalam menggunakan terminal dan editor kode agar dapat mulai membangun proyek yang menjalankan *Machine Learning* (ML) pada perangkat embedded. Dengan semakin berkembangnya teknologi, kini ML tidak hanya dapat dijalankan di komputer atau server berkekuatan tinggi, tetapi juga pada perangkat kecil dengan daya rendah.

Ketika penulis bergabung dengan Google pada tahun 2014, ia menemukan banyak proyek internal yang menarik. Salah satu yang paling menonjol adalah pekerjaan yang dilakukan oleh tim "OK Google". Mereka berhasil menjalankan jaringan neural hanya sebesar 14KB pada prosesor sinyal digital (DSP) yang terdapat di sebagian besar ponsel Android. DSP ini digunakan untuk terus mendengarkan kata pemicu "OK Google" tanpa harus mengaktifkan CPU utama, sehingga dapat menghemat daya baterai.

Keberhasilan ini menunjukkan bahwa jaringan neural dapat berjalan pada perangkat dengan sumber daya terbatas. Biasanya, menjalankan model ML membutuhkan daya yang besar, tetapi dalam kasus ini, konsumsi daya dapat ditekan hingga hanya beberapa miliwatt (mW). Dari sinilah muncul gagasan tentang TinyML, yaitu *Machine Learning* yang berjalan pada perangkat dengan konsumsi

daya di bawah 1mW. Dengan konsumsi daya sekecil ini, perangkat dapat beroperasi selama bertahun-tahun hanya dengan baterai koin, tanpa memerlukan intervensi manusia.

Seiring berkembangnya teknologi, muncul berbagai platform seperti Raspberry Pi dan NVIDIA Jetson. Meskipun keduanya merupakan perangkat luar biasa dalam menjalankan ML, namun konsumsi daya mereka jauh lebih tinggi dibandingkan TinyML. Bahkan, Raspberry Pi yang paling kecil sekalipun masih membutuhkan daya ratusan miliwatt, sementara Jetson bisa mengonsumsi hingga 12 watt saat beroperasi penuh. Hal ini membuat perangkat-perangkat tersebut sulit digunakan untuk aplikasi yang membutuhkan konsumsi daya ultra-rendah dan tidak memiliki sumber daya listrik yang stabil.

Sebaliknya, mikrokontroler 32-bit menjadi pilihan yang lebih cocok untuk TinyML. Mikrokontroler ini jauh lebih murah, bahkan bisa didapatkan dengan harga di bawah $1 per unit, sehingga memungkinkan produsen untuk menggantikan sistem kontrol analog atau elektromekanis dengan alternatif berbasis perangkat lunak. Dengan harga yang terjangkau, teknologi ini dapat diterapkan dalam berbagai bidang, seperti sensor cerdas di lingkungan bangunan, konservasi satwa liar, atau sistem pemantauan industri.

a. Perangkat Embedded

TinyML bergantung pada dunia perangkat embedded, yang dahulu dianggap sulit untuk diakses oleh pengembang umum. Sebelumnya, perangkat embedded menggunakan arsitektur 8-bit dengan alat pengembangan yang tertutup dan sulit dipahami. Namun, dengan kehadiran Arduino, pengembangan perangkat embedded menjadi lebih mudah karena adanya antarmuka yang ramah pengguna dan ekosistem yang lebih terbuka.

Saat ini, sebagian besar mikrokontroler menggunakan CPU Arm Cortex-M, yang telah menjadi standar dalam dunia embedded. Meskipun perangkat ini memiliki keterbatasan, seperti kapasitas RAM dan penyimpanan yang kecil serta kecepatan pemrosesan yang rendah, namun keunggulannya adalah konsumsi daya yang sangat efisien. Tidak seperti komputer atau ponsel, perangkat embedded tidak

memiliki sistem operasi Linux yang lengkap karena keterbatasan sumber daya, sehingga pengembang harus bekerja dengan sistem yang lebih ringan.

Selain itu, banyak sistem embedded menghindari penggunaan alokasi memori dinamis seperti malloc() atau new karena sistem ini dirancang agar dapat berjalan dalam waktu yang lama dengan tingkat keandalan tinggi. Fragmentasi memori dapat menjadi masalah besar dalam sistem yang harus berjalan tanpa gangguan selama bertahun-tahun. Oleh karena itu, pengembangan perangkat embedded memerlukan pendekatan yang berbeda dibandingkan dengan pemrograman pada komputer atau ponsel.

Meskipun terdapat tantangan dalam pengembangan perangkat embedded, ada juga banyak keuntungan. Salah satu keunggulannya adalah sederhananya model pemrosesan, karena perangkat ini tidak memiliki proses lain yang mengganggu jalannya program utama. Selain itu, dengan arsitektur prosesor yang lebih sederhana dan tanpa prediksi cabang atau pipelining instruksi yang kompleks, optimasi manual dalam bahasa assembly menjadi lebih mudah dilakukan dibandingkan pada CPU yang lebih canggih.

b. Perkembangan Cepat

TinyML merupakan bidang yang masih sangat baru dan berkembang dengan cepat. Baik dari sisi perangkat keras, perangkat lunak, maupun penelitian, semuanya berubah dengan sangat dinamis. Buku ini ditulis berdasarkan kondisi teknologi pada tahun 2019, namun dalam dunia TinyML, perubahan bisa terjadi dalam hitungan bulan. Oleh karena itu, beberapa bagian dari buku ini mungkin akan terasa usang dalam waktu singkat.

Namun, penulis berusaha untuk memberikan panduan dengan menggunakan platform yang diperkirakan akan tetap tersedia dalam jangka panjang. Selain itu, framework TensorFlow Lite yang digunakan dalam buku ini memiliki API yang cukup stabil dan terus mendapatkan dukungan dari komunitas pengembang. Untuk mengikuti perkembangan terbaru, disediakan pula tautan ke sumber daya online yang selalu diperbarui, termasuk kode contoh dan dokumentasi terbaru.

Lebih dari sekadar membahas teknologi spesifik, buku ini juga berfokus pada pemahaman konsep fundamental, seperti teknik debugging, pembuatan model, dan dasar-dasar deep learning. Dengan memahami konsep-konsep ini, pembaca dapat tetap relevan dalam industri, meskipun perangkat keras dan perangkat lunak yang digunakan terus berkembang. Tiny *Machine Learning* (TinyML) adalah cabang dari pembelajaran mesin (*Machine Learning*) yang fokus pada implementasi model pembelajaran mesin di perangkat kecil dengan daya sangat rendah, seperti mikrocontroller. TinyML memungkinkan perangkat tersebut untuk melakukan inferensi atau prediksi secara lokal tanpa memerlukan konektivitas internet atau cloud. Salah satu karakteristik utama dari TinyML adalah kemampuannya untuk beroperasi dengan konsumsi daya di bawah 1 mW, menjadikannya sangat efisien untuk aplikasi yang membutuhkan masa pakai baterai yang lama atau bahkan menggunakan sumber daya energi alternatif seperti pengumpulan energi (energy harvesting).

TinyML memanfaatkan berbagai teknik optimasi seperti quantization, yang mengurangi ukuran model dengan menurunkan presisi numerik, dan pruning, yang menghilangkan neuron-neuron yang tidak penting dari jaringan saraf. Kombinasi teknik ini memastikan bahwa model deep learning yang kompleks dapat berjalan pada perangkat keras dengan memori yang sangat terbatas, sering kali hanya memiliki puluhan hingga ratusan kilobyte RAM. Contoh aplikasi TinyML meliputi deteksi suara untuk wake-word seperti “OK Google,” pengenalan gerakan menggunakan akselerometer, dan deteksi objek menggunakan kamera kecil. Dengan mengolah data secara lokal, TinyML tidak hanya mengurangi konsumsi daya tetapi juga meningkatkan privasi data, menjadikannya ideal untuk aplikasi IoT dan perangkat yang tersebar luas dalam skala besar.

- Konversi Model

Dalam buku ini, kita menggunakan TensorFlow untuk membangun dan melatih model. Model TensorFlow pada dasarnya adalah seperangkat instruksi yang memberi tahu interpreter bagaimana cara mengubah data untuk menghasilkan output tertentu. Saat ingin menggunakan model, kita cukup memuatnya ke dalam memori dan menjalankannya dengan interpreter TensorFlow.

```
include <algorithm>
include "Arduino.h"
include "constants.h"
include "output_handler.h"
include "tensorflow/lite/micro/micro_log.h"

        int led = LED_BUILTIN;
        bool initialized = false;

        void HandleOutput(float x_value, float y_value) {
        if (!initialized) {
        pinMode(led, OUTPUT);
        initialized = true;}

        int brightness = (int)(127.5f * (y_value + 1));
        int brightness_clamped = std::min(255, std::max(0, brightness));
        analogWrite(led,brightness_clamped);
        Serial.write(brightness);
        MicroPrintf("%d\n", brightness);
        delay(33);
        }
```

Gambar 3.3 Kode untuk proses konversi

Namun, interpreter TensorFlow dirancang untuk berjalan di komputer desktop dan server yang memiliki daya pemrosesan tinggi. Karena kita ingin menjalankan model di mikrocontroller kecil dengan daya terbatas, kita memerlukan interpreter yang lebih ringan dan efisien.

TensorFlow menyediakan alat bernama TensorFlow Lite, yang memungkinkan model dapat berjalan pada perangkat kecil dan berdaya rendah. Dengan menggunakan TensorFlow Lite, kita dapat menjalankan model pembelajaran mesin tanpa harus mengandalkan perangkat keras yang kuat seperti komputer atau server.

Sebelum TensorFlow Lite dapat menjalankan model, model tersebut harus dikonversi ke dalam format khusus yang lebih ringan. Proses ini dilakukan menggunakan alat bernama TensorFlow Lite Converter. Konversi ini penting karena mengoptimalkan ukuran dan kecepatan model agar lebih sesuai untuk perangkat dengan sumber daya terbatas.

Selain mengonversi format model, TensorFlow Lite Converter juga dapat menerapkan berbagai teknik optimasi. Optimasi ini bertujuan untuk mengurangi ukuran model dan meningkatkan kecepatan eksekusi tanpa mengorbankan performa prediksi yang dihasilkan oleh model.

- Menjalankan Inferensi

Setelah model dikonversi ke dalam format TensorFlow Lite, model siap untuk diterapkan ke perangkat target. Untuk menjalankannya, kita akan menggunakan pustaka TensorFlow Lite for Microcontrollers yang ditulis dalam bahasa pemrograman C++.

Saat model diterapkan ke dalam sistem, kita perlu memastikan bahwa data masukan yang diberikan ke model memiliki format yang sama dengan data yang digunakan saat pelatihan. Ini penting agar model dapat memberikan hasil prediksi yang akurat.

Hasil yang dihasilkan oleh model berupa skor probabilitas untuk setiap kelas yang telah ditentukan sebelumnya. Sebagai contoh, dalam kasus model klasifikasi, skor ini menunjukkan kemungkinan apakah suatu kondisi termasuk "normal" atau "abnormal."

Misalnya, jika model menghasilkan skor berikut:

Skor Normal	Skor Abnormal	
0.1	0.9	Model yakin terjadi keadaan abnormal
0.7	0.3	Model yakin keadaan normal
0.49	0.51	Hasil tidak meyakinkan

Dari tabel di atas, kita dapat melihat bahwa semakin besar selisih antara skor kelas yang dihasilkan, semakin tinggi tingkat kepastian model terhadap prediksinya.

- Menangani Noise dan Glitch dalam Data

Dalam banyak kasus, model pembelajaran mesin hanya mempertimbangkan snapshot data dalam periode waktu tertentu. Sebagai contoh, dalam sistem pemantauan mesin industri, model dapat mengevaluasi kondisi mesin berdasarkan data sensor dalam 10 detik terakhir.

Namun, data dunia nyata sering kali tidak bersih dan bisa mengandung gangguan atau glitch. Hal ini dapat menyebabkan model membuat prediksi yang keliru. Sebagai contoh, lonjakan suhu akibat kesalahan sensor dapat menyebabkan model mengklasifikasikan kondisi mesin sebagai abnormal, padahal kondisi sebenarnya masih normal.

Untuk menghindari masalah ini, kita bisa menerapkan teknik penyaringan data. Salah satu metode yang efektif adalah dengan menghitung rata-rata dari beberapa hasil inferensi sebelumnya. Dengan cara ini, keputusan tidak akan terlalu dipengaruhi oleh gangguan sementara dan lebih mencerminkan kondisi yang sebenarnya.

Sebagai contoh, kita dapat menjalankan model setiap 10 detik dan mengambil rata-rata hasil inferensi selama satu menit terakhir. Jika kondisi abnormal terdeteksi secara konsisten selama periode tersebut, maka sistem dapat mengambil tindakan, seperti mematikan mesin atau memberi peringatan kepada tim pemeliharaan.

- Evaluasi dan Troubleshooting Model

Setelah model diterapkan ke dalam perangkat target, kita perlu mengevaluasi apakah model benar-benar bekerja dengan baik dalam kondisi nyata. Meskipun model dapat menunjukkan performa yang baik pada data uji, hasilnya di dunia nyata bisa berbeda.

Ada beberapa alasan mengapa performa model di lapangan bisa menurun. Salah satunya adalah karena data pelatihan tidak sepenuhnya mencerminkan kondisi operasional sebenarnya. Sebagai contoh, jika model dilatih menggunakan data suhu dari lingkungan yang lebih hangat, sedangkan perangkat digunakan di lokasi dengan suhu lebih dingin, maka model mungkin tidak berfungsi seakurat yang diharapkan.

Selain itu, masalah overfitting juga bisa menjadi penyebab performa yang buruk. Overfitting terjadi ketika model terlalu menghafal pola dari data pelatihan dan kurang mampu beradaptasi dengan data baru. Jika model terlalu spesifik terhadap dataset pelatihan, ia mungkin gagal memberikan prediksi yang akurat ketika digunakan dalam skenario yang berbeda.

- Langkah-Langkah Troubleshooting

Jika model tidak bekerja sesuai harapan, ada beberapa langkah yang bisa kita lakukan untuk mencari penyebab masalah dan memperbaikinya:

i.Periksa perangkat keras

- Pastikan sensor berfungsi dengan baik dan tidak mengalami gangguan.

- Periksa apakah ada noise atau interferensi yang dapat memengaruhi kualitas data.

2. Bandingkan data dunia nyata dengan data pelatihan

- Ambil sampel data dari perangkat di lapangan dan bandingkan dengan dataset pelatihan.
- Pastikan tidak ada perbedaan signifikan dalam karakteristik data yang digunakan.

3. Mengatasi Overfitting

- Jika overfitting terdeteksi, latih ulang model menggunakan lebih banyak data yang lebih bervariasi.
- Terapkan teknik regularisasi untuk mengurangi kompleksitas model agar lebih mampu menangani data baru.
- Gunakan augmentasi data untuk meningkatkan keragaman dalam dataset pelatihan.

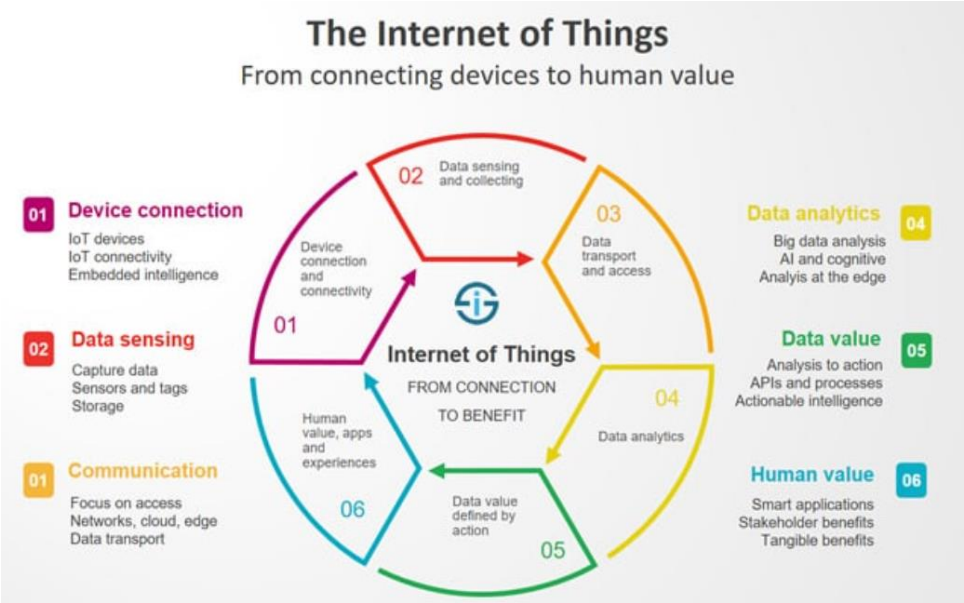
3.8. **Internet of Things (IoT)**

Internet of Things (IoT) adalah konsep di mana berbagai perangkat fisik, seperti sensor, alat rumah tangga, kendaraan, dan mesin industri, dapat terhubung ke internet dan berkomunikasi satu sama lain. Dengan adanya IoT, perangkat dapat mengumpulkan, bertukar, dan menganalisis data secara otomatis. Hal ini memungkinkan sistem untuk bekerja lebih efisien, meningkatkan produktivitas, serta memberikan kenyamanan dan keamanan bagi penggunanya.

IoT berkembang pesat di berbagai sektor, termasuk rumah pintar (smart home), kota pintar (smart city), otomasi industri, kesehatan digital, dan masih banyak lagi. Keunggulan utama IoT adalah kemampuannya dalam mengoptimalkan proses, mengurangi keterlibatan manusia dalam tugas-tugas rutin, serta meningkatkan pengambilan keputusan berbasis data yang akurat dan real-time.

Dengan adanya teknologi IoT, berbagai perangkat dapat saling terhubung dan bekerja secara otomatis berdasarkan data yang dikumpulkan. Sebagai contoh, di sektor kesehatan, IoT memungkinkan pemantauan pasien secara real-time melalui sensor medis yang dapat mendeteksi tanda-tanda vital dan mengirimkan laporan langsung ke dokter. Dalam sektor industri, IoT dapat digunakan untuk

mendeteksi kerusakan mesin sebelum terjadi kegagalan total, sehingga dapat mengurangi downtime produksi.



Gambar 3. 4 Internet of Things

Gambar 3.4 menjelaskan bagaimana IoT bekerja, mulai dari tahap awal menghubungkan perangkat hingga menghasilkan manfaat bagi manusia. Proses ini terdiri dari enam tahap utama yang membentuk siklus IoT:

1. Device Connection (Koneksi Perangkat)

Tahap pertama dalam sistem IoT adalah menghubungkan perangkat ke jaringan internet. Perangkat yang digunakan dalam IoT biasanya memiliki kemampuan untuk terhubung melalui jaringan seperti Wi-Fi, Bluetooth, atau protokol komunikasi khusus IoT seperti LoRa dan Zigbee.

Selain itu, perangkat ini sering kali dilengkapi dengan kecerdasan tertanam (embedded intelligence) yang memungkinkan mereka untuk memproses data sebelum mengirimkannya ke sistem pusat. Contoh perangkat IoT meliputi sensor suhu, kamera pintar, smartwatches, dan mesin industri yang terkoneksi dengan cloud.

2. Data Sensing (Penginderaan Data)

Setelah perangkat terhubung, tahap selanjutnya adalah pengumpulan data. IoT menggunakan berbagai sensor untuk menangkap informasi dari lingkungan sekitar. Sensor ini bisa berupa sensor suhu, kelembaban, tekanan, gerakan, dan sebagainya.

Setelah data dikumpulkan, informasi tersebut disimpan dalam sistem penyimpanan seperti database atau cloud. Data yang dikumpulkan ini akan menjadi dasar bagi proses analisis yang dilakukan di tahap selanjutnya.

3. Communication (Komunikasi Data)

Data yang telah dikumpulkan oleh sensor harus dikirim ke pusat pemrosesan untuk dianalisis. Proses ini melibatkan komunikasi data melalui jaringan yang telah dikonfigurasi sebelumnya.

Terdapat berbagai metode komunikasi yang digunakan dalam IoT, mulai dari jaringan kabel hingga teknologi nirkabel seperti 4G, 5G, dan satelit. Cloud computing dan edge computing juga memainkan peran penting dalam tahap ini, karena memungkinkan data untuk diproses lebih dekat ke sumbernya sehingga mengurangi latensi.

4. Data Analytics (Analisis Data)

Setelah data diterima oleh sistem pusat, langkah berikutnya adalah menganalisis data tersebut. Analisis dilakukan menggunakan teknologi seperti big data analytics, kecerdasan buatan (AI), dan komputasi kognitif.

Analisis ini bertujuan untuk menemukan pola, mendeteksi anomali, serta membuat prediksi yang dapat membantu dalam pengambilan keputusan. Proses analisis ini bisa dilakukan baik di cloud maupun di edge (lebih dekat dengan perangkat sumber data) untuk meningkatkan kecepatan dan efisiensi.

5. Data Value (Nilai Data)

Data yang telah dianalisis akan dikonversi menjadi informasi yang dapat dimanfaatkan. Tahap ini melibatkan penggunaan Application Programming Interface (API) dan proses otomatisasi untuk menghasilkan actionable intelligence atau wawasan yang dapat langsung digunakan.

Sebagai contoh, dalam sistem pemantauan lingkungan, jika sensor mendeteksi polusi udara yang tinggi, sistem dapat secara otomatis mengaktifkan alarm peringatan atau memberi tahu pihak yang berwenang untuk mengambil tindakan.

6. Human Value (Manfaat bagi Manusia)

Tahap terakhir dalam siklus IoT adalah memberikan manfaat nyata kepada manusia. Wawasan yang telah dihasilkan melalui analisis data dapat diterapkan dalam berbagai aplikasi pintar untuk meningkatkan efisiensi dan kualitas hidup.

Sebagai contoh, dalam industri manufaktur, jika sistem IoT mendeteksi potensi kerusakan pada suatu mesin, informasi ini dapat digunakan untuk menjadwalkan pemeliharaan sebelum terjadi kegagalan. Dalam sistem rumah pintar, IoT dapat digunakan untuk mengatur pencahayaan dan suhu ruangan secara otomatis berdasarkan kebiasaan penghuni. Manfaat ini tidak hanya dirasakan oleh individu, tetapi juga oleh bisnis dan pemerintah dalam meningkatkan efisiensi operasional serta mengurangi biaya.

7. Kesimpulan

Gambar ini menggambarkan siklus kerja IoT dari awal hingga memberikan manfaat bagi manusia. Proses ini dimulai dengan menghubungkan perangkat, menangkap data, mengirimkannya untuk dianalisis, hingga menghasilkan wawasan yang berharga dan dapat digunakan dalam berbagai aplikasi cerdas.

Dengan memahami alur kerja ini, kita bisa lebih efektif dalam menerapkan IoT untuk meningkatkan efisiensi, produktivitas, dan kenyamanan di berbagai bidang. IoT tidak hanya sebatas teknologi masa depan, tetapi sudah menjadi bagian dari kehidupan sehari-hari yang terus berkembang dan memberikan dampak positif bagi berbagai sektor.

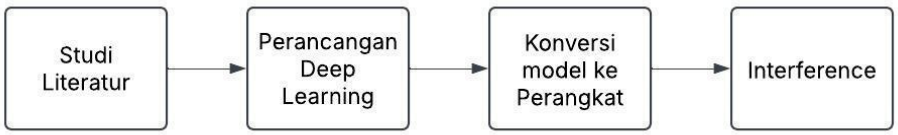
Dalam konteks IoT, TinyML berperan penting sebagai teknologi pendukung untuk memproses data langsung di perangkat ujung (edge devices). Biasanya, perangkat IoT bergantung pada cloud untuk memproses data sensor, tetapi pendekatan ini membutuhkan konektivitas yang konstan dan memakan daya. Dengan menggunakan TinyML, perangkat IoT dapat memproses data sensor secara langsung, mengurangi ketergantungan pada cloud, meningkatkan efisiensi energi,

dan mempercepat waktu respons. Sebagai contoh, sensor gerakan yang dilengkapi dengan model TinyML dapat mengenali aktivitas tertentu dan hanya mengirimkan data penting ke cloud, sehingga menghemat bandwidth. Kombinasi IoT dan TinyML menciptakan solusi yang lebih cerdas, hemat daya, dan cocok untuk aplikasi berskala besar seperti pertanian pintar, pemantauan kesehatan jarak jauh, dan pelacakan logistik.

Bab IV
Deskripsi Data Dan Hasil Praktik Kerja Lapangan

4.1. Metode atau Prosedur yang Digunakan

Gambar 4.1 menunjukkan prosedur yang digunakan untuk praktik kerja lapangan.



Gambar 4. 1 Prosedur yang digunakan dalam Praktik Kerja Lapangan

1. Studi Literatur

Tahap pertama dalam proses ini adalah melakukan studi literatur, yang bertujuan untuk memahami konsep dasar Deep Learning, teknik yang digunakan, serta bagaimana model dapat diterapkan pada perangkat target. Studi ini mencakup eksplorasi berbagai sumber, seperti jurnal ilmiah, buku, artikel, dan penelitian terdahulu yang relevan.

Selain memahami teori Deep Learning, dalam tahap ini juga dilakukan analisis terhadap perangkat keras yang akan digunakan, seperti Arduino, Raspberry Pi, atau mikrokontroler lainnya. Hal ini penting karena keterbatasan daya komputasi dan memori pada perangkat kecil mempengaruhi bagaimana model harus dirancang dan dioptimalkan.

Studi literatur juga mencakup pemilihan algoritma dan metode yang paling sesuai untuk diterapkan dalam skenario tertentu. Misalnya, jika proyek berfokus pada pengolahan citra, maka arsitektur Convolutional Neural Network (CNN) mungkin menjadi pilihan terbaik. Sementara itu, jika proyek berkaitan dengan pemrosesan teks, maka model berbasis Recurrent Neural Network (RNN) atau Transformer dapat lebih cocok digunakan.

2. Perancangan Deep Learning

Setelah memahami teori dan memilih pendekatan yang tepat, tahap selanjutnya adalah merancang model Deep Learning. Pada tahap ini, peneliti atau pengembang menentukan arsitektur model, jumlah lapisan neuron, fungsi aktivasi, algoritma optimasi, serta parameter lain yang akan digunakan selama pelatihan.

Proses pelatihan model biasanya dilakukan menggunakan dataset yang sesuai dengan permasalahan yang ingin diselesaikan. Dataset ini bisa berupa kumpulan gambar, teks, atau data sensor yang dikumpulkan dari perangkat. Model kemudian diuji dengan data pelatihan dan validasi untuk memastikan bahwa ia dapat mengenali pola dengan baik tanpa mengalami overfitting.

Selama tahap perancangan, berbagai teknik dapat diterapkan untuk meningkatkan performa model. Misalnya, penggunaan augmentasi data untuk meningkatkan keberagaman data pelatihan atau penerapan regularisasi untuk mengurangi risiko overfitting.

3. Konversi Model ke Perangkat

Setelah model Deep Learning berhasil dilatih di lingkungan komputasi yang lebih kuat (misalnya di komputer dengan GPU), model tersebut harus dikompresi dan dikonversi agar dapat berjalan di perangkat kecil dengan keterbatasan daya dan memori.

Proses konversi ini dilakukan menggunakan alat seperti TensorFlow Lite yang memungkinkan model menjadi lebih ringan dan tetap mempertahankan performa yang optimal. Dalam beberapa kasus, dilakukan juga kuantisasi model, yaitu teknik untuk mengurangi ukuran model dengan mengganti tipe data floating point menjadi integer tanpa mengorbankan akurasi secara signifikan.

Konversi ini sangat penting karena model Deep Learning yang tidak dioptimalkan dapat membebani perangkat dan menyebabkan kinerja yang lambat atau bahkan tidak dapat berjalan sama sekali. Oleh karena itu, diperlukan teknik khusus untuk memastikan model tetap akurat tetapi juga efisien dalam penggunaan sumber daya.

4. Inference (Inferensi)

Tahap terakhir dalam proses ini adalah inferensi, yaitu ketika model yang telah dikonversi mulai dijalankan di perangkat target. Pada tahap ini, model menerima data input secara real-time, seperti gambar dari kamera atau data dari sensor, kemudian memprosesnya untuk menghasilkan prediksi.

Sebagai contoh, dalam sistem deteksi anomali mesin industri, sensor yang dipasang pada mesin akan mengirimkan data suhu dan getaran ke model yang telah dioptimalkan. Model kemudian menentukan apakah mesin dalam keadaan normal atau mengalami anomali berdasarkan pola data yang telah dipelajari sebelumnya. Jika terdeteksi anomali secara konsisten, sistem dapat mengirimkan peringatan untuk segera dilakukan pemeriksaan atau pemeliharaan.

Selain itu, dalam sistem pengenalan suara seperti wake-word detection, model yang telah dikonversi ke TensorFlow Lite akan mendeteksi perintah suara pengguna dan mengaktifkan perangkat sesuai dengan instruksi yang diberikan. Proses inferensi ini harus berlangsung cepat dan efisien, sehingga optimalisasi model pada tahap sebelumnya menjadi sangat penting.

Dalam banyak kasus, model inferensi juga menggunakan teknik tambahan seperti pengambilan keputusan berbasis rata-rata untuk menghindari kesalahan prediksi akibat gangguan sesaat dalam data. Sebagai contoh, jika model mendeteksi anomali selama satu detik tetapi kembali normal di detik berikutnya, sistem dapat mengabaikannya agar tidak terjadi false alarm.

5. Kesimpulan

Flowchart ini memberikan gambaran lengkap mengenai bagaimana model Deep Learning dikembangkan, dioptimalkan, dan diterapkan dalam perangkat kecil seperti mikrokontroler atau sistem IoT.

Dengan melalui tahapan studi literatur, pengembang memastikan bahwa pendekatan yang digunakan sesuai dengan kebutuhan proyek. Selanjutnya, pada tahap perancangan, model dikembangkan dan diuji untuk mencapai tingkat akurasi yang diharapkan. Kemudian, dalam tahap konversi, model dioptimalkan agar dapat berjalan dengan baik dalam keterbatasan sumber daya perangkat. Akhirnya, pada tahap inferensi, model diterapkan untuk melakukan prediksi secara real-time dan memberikan output yang dapat digunakan untuk pengambilan keputusan.

Proses ini menunjukkan pentingnya kombinasi antara pengembangan model yang akurat dan optimasi model yang efisien agar sistem berbasis Deep Learning dapat berjalan dengan baik di perangkat yang memiliki keterbatasan daya komputasi.

5.1. Perancangan Deep Learning

Gambar 4.2 menunjukkan cuplikan kode Python yang berisi perintah instalasi dan impor pustaka yang umum digunakan dalam *Machine Learning* dan deep learning.

```
1  !pip install protobuf==3.20.3
2  !pip install tensorflow==2.8.0rc0
3  import tensorflow as tf
4  import tensorflow.keras.layers as mylayers
5  import numpy as np
6  import matplotlib.pyplot as plt
7  import math
```

Gambar 4. 2 Library

Berikut penjelasan gambar 4.2:

1. Instalasi Pustaka

!pip install protobuf==3.20.3

!pip install tensorflow==2.8.0rc0

- protobuf==3.20.3: Menginstal versi tertentu dari pustaka Protocol Buffers yang digunakan oleh TensorFlow untuk komunikasi data yang efisien.
- tensorflow==2.8.0rc0: Menginstal versi kandidat rilis (release candidate) dari TensorFlow 2.8.0.

2. Import Pustaka

Line 3 sampai 7 dijelaskan berikut:

- import tensorflow as tf: Mengimpor TensorFlow dengan alias tf, yang sering digunakan dalam deep learning.
- import tensorflow.keras.layers as mylayers: Mengimpor modul keras.layers dari TensorFlow dan memberinya alias mylayers untuk mempermudah akses ke berbagai jenis layer dalam neural network.

- import numpy as np: Mengimpor NumPy dengan alias np, pustaka yang digunakan untuk komputasi numerik dan operasi matriks.
- import matplotlib.pyplot as plt: Mengimpor pyplot dari matplotlib dengan alias plt, yang digunakan untuk visualisasi data.
- import math: Mengimpor pustaka matematika bawaan Python untuk perhitungan matematis dasar.

Gambar 4.3 menunjukkan cuplikan kode Python yang digunakan untuk menghasilkan dan memvisualisasikan data berbasis fungsi sinus dengan sedikit noise.

```
SAMPLES = 1000
SEED = 1337
np.random.seed(SEED)
tf.random.set_seed(SEED)
x_values = np.random.uniform(low=0, high=2*math.pi, size=SAMPLES)
np.random.shuffle(x_values)
y_values = np.sin(x_values)
y_values += 0.1 * np.random.randn(*y_values.shape)
plt.plot(x_values, y_values, 'b.')
plt.show()
```

Gambar 4. 3 Kode untuk membuat 1000 data sinus secara random

1. Inisialisasi Variabel

SAMPLES = 1000

SEED = 1337

- SAMPLES = 1000: Menentukan jumlah sampel data yang akan dihasilkan.
- SEED = 1337: Menentukan nilai seed agar hasil randomisasi tetap konsisten setiap kali kode dijalankan.

2. Pengaturan Seed untuk Reproduksibilitas

np.random.seed(SEED)

tf.random.set_seed(SEED)

- np.random.seed(SEED): Mengatur seed untuk generator angka acak dari NumPy.
- tf.random.set_seed(SEED): Mengatur seed untuk generator angka acak dari TensorFlow.

3. Pembuatan Data

x_values = np.random.uniform(low=0, high=2*math.pi, size=SAMPLES)

- np.random.uniform(low=0, high=2*math.pi, size=SAMPLES): Menghasilkan 1000 nilai acak dari distribusi uniform dalam rentang [0, 2π].

np.random.shuffle(x_values)

- np.random.shuffle(x_values): Mengacak urutan nilai x_values.

y_values = np.sin(x_values)

- Menghitung nilai sinus dari setiap elemen dalam x_values.

4. Penambahan Noise (Gangguan Acak) pada Data

y_values += 0.1 * np.random.randn(*y_values.shape)

- np.random.randn(*y_values.shape): Menghasilkan noise dari distribusi normal (mean = 0, standar deviasi = 1).
- * np.random.randn(*y_values.shape): Memperkecil noise dengan mengalikannya dengan 0.1 agar gangguan tidak terlalu besar.
- y_values += ...: Menambahkan noise ke nilai y_values agar data lebih realistis dan tidak terlalu sempurna.

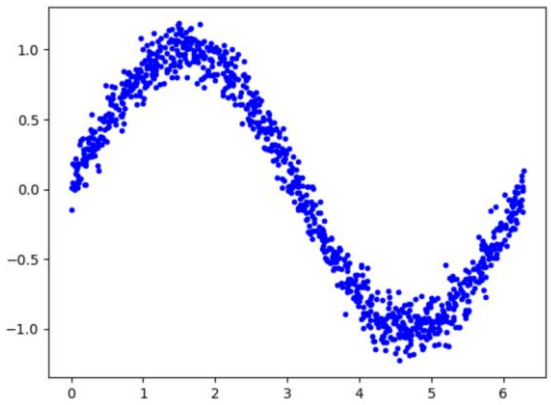
5. Visualisasi Data

plt.plot(x_values, y_values, 'b.')

plt.show()

- plt.plot(x_values, y_values, 'b.'): Membuat scatter plot dengan titik berwarna biru ('b.').
- plt.show(): Menampilkan plot.

Gambar 4.3 adalah hasil plot dari kode sebelumnya yang menggunakan fungsi sinus dengan noise acak



Gambar 4. 4 Hasil Plot dari kode sebelumnya

Gambar 4.4 menunjukkan kode untuk membagi dataset menjadi tiga bagian: train (pelatihan), validate (validasi), dan test (pengujian), serta memvisualisasikannya dengan warna berbeda.

```
TRAIN_SPLIT = int(0.6 * SAMPLES)
TEST_SPLIT = int(0.2 * SAMPLES + TRAIN_SPLIT)
x_train, x_validate, x_test = np.split(x_values, [TRAIN_SPLIT, TEST_SPLIT])
y_train, y_validate, y_test = np.split(y_values, [TRAIN_SPLIT, TEST_SPLIT])
plt.plot(x_train, y_train, 'b.', label="Train")
plt.plot(x_validate, y_validate, 'y.', label="Validate")
plt.plot(x_test, y_test, 'r.', label="Test")
plt.legend()
plt.show()
```

Gambar 4. 5 Pembagian dataset dan proses Training

1. Pembagian Dataset

TRAIN_SPLIT = int(0.6 * SAMPLES)

TEST_SPLIT = int(0.2 * SAMPLES + TRAIN_SPLIT)

- TRAIN_SPLIT = int(0.6 * SAMPLES): Menentukan indeks batas untuk data training (60% dari total SAMPLES).
- TEST_SPLIT = int(0.2 * SAMPLES + TRAIN_SPLIT): Menentukan indeks batas untuk data testing (20% setelah data training), sehingga sisanya (20%) otomatis menjadi validation set.

2. Memisahkan Data

x_train, x_validate, x_test = np.split(x_values, [TRAIN_SPLIT, TEST_SPLIT])

y_train, y_validate, y_test = np.split(y_values, [TRAIN_SPLIT, TEST_SPLIT])

- np.split(x_values, [TRAIN_SPLIT, TEST_SPLIT]): Memisahkan x_values menjadi tiga bagian:

x_train (60%)

x_validate (20%)

x_test (20%)

- np.split(y_values, [TRAIN_SPLIT, TEST_SPLIT]): Memisahkan y_values dengan cara yang sama.

3. Visualisasi Data

plt.plot(x_train, y_train, 'b.', label="Train")

plt.plot(x_validate, y_validate, 'y.', label="Validate")

plt.plot(x_test, y_test, 'r.', label="Test")

- plt.plot(x_train, y_train, 'b.', label="Train"): Menampilkan data training dengan titik biru ('b.').
- plt.plot(x_validate, y_validate, 'y.', label="Validate"): Menampilkan data validasi dengan titik kuning ('y.').
- plt.plot(x_test, y_test, 'r.', label="Test"): Menampilkan data uji dengan titik merah ('r.').

4. Menampilkan Plot

plt.legend()

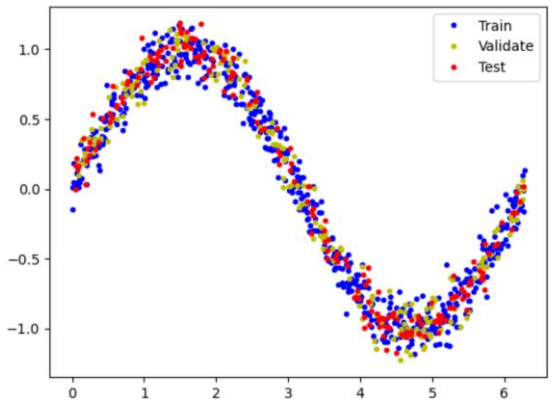
plt.show()

- plt.legend(): Menampilkan legenda untuk membedakan kategori data.
- plt.show(): Menampilkan plot.

Gambar 4.6 menunjukkan pembagian dataset sinus dengan noise menjadi tiga bagian:

1. Training Set (Train) → Ditandai dengan titik biru (60% data).

2. Validation Set (Validate) → Ditandai dengan titik kuning (20% data).
3. Test Set (Test) → Ditandai dengan titik merah (20% data).



Gambar 4. 6 Hasil Dataset Sinus

Gambar 4.7 menunjukkan arsitektur model neural network (MLP) sederhana yang dibuat menggunakan TensorFlow dan Keras.

```
model_1 = tf.keras.Sequential()
model_1.add(mylayers.Dense(16, activation='relu',input_shape=(1,)))
model_1.add(mylayers.Dense(1))
model_1.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
model_1.summary()
```

Gambar 4. 7 Membuat model sequential

1. Membuat Model Sequential

model_1 = tf.keras.Sequential()

- Membuat model sequential, yang berarti lapisan (layers) akan ditambahkan secara berurutan dari input ke output.

2. Menambahkan Hidden Layer

model_1.add(mylayers.Dense(16, activation='relu', input_shape=(1,)))

- mylayers.Dense(16, activation='relu', input_shape=(1,))
- 16 neuron di layer tersembunyi.
- Fungsi aktivasi: ReLU (Rectified Linear Unit), yang umum digunakan dalam deep learning untuk mengatasi masalah vanishing gradient.

- input_shape=(1,): Model menerima satu fitur input (karena hanya ada satu variabel input).

3. Menambahkan Output Layer

model_1.add(mylayers.Dense(1))

- mylayers.Dense(1)
- Layer keluaran (output layer) memiliki 1 neuron, yang cocok untuk tugas regresi (memprediksi nilai kontinu).

4. Mengompilasi Model

model_1.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])

- optimizer='rmsprop'
- RMSprop (Root Mean Square Propagation) adalah algoritma optimasi yang cocok untuk pelatihan model dengan gradien yang bervariasi.
- loss='mse'
- Mean Squared Error (MSE) digunakan sebagai fungsi kerugian (loss function), cocok untuk regresi.
- metrics=['mae']
- Mean Absolute Error (MAE) digunakan sebagai metrik evaluasi untuk menilai performa model.

5. Menampilkan Ringkasan Model

model_1.summary() Menampilkan struktur model, jumlah parameter trainable, serta jumlah layer.

Gambar 4.8 menunjukkan ringkasan arsitektur model neural network yang dibuat menggunakan TensorFlow/Keras.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 16)                32

 dense_4 (Dense)             (None, 1)                 17

=================================================================
Total params: 49
Trainable params: 49
Non-trainable params: 0
_________________________________________________________________
```

Gambar 4. 8 Arsitektur Model Neural Network

Gambar 4.9 menunjukkan proses pelatihan model neural network menggunakan metode .fit() pada TensorFlow/Keras.

```
history_1 = model_1.fit(
    x_train, y_train,
    epochs=1000,
    batch_size=16,
    validation_data=(x_validate, y_validate)
)
```

Gambar 4. 9 Kode Proses pelatihan model Neural Network

Rincian Parameter

1. x_train, y_train

Ini adalah dataset pelatihan yang digunakan model untuk belajar.

x_train berisi fitur (input), sementara y_train adalah label atau target yang ingin diprediksi.

2. epochs=1000

Model akan dilatih selama 1000 iterasi penuh pada dataset.

Semakin tinggi jumlah epoch, semakin lama pelatihan berlangsung.

Bisa menyebabkan overfitting jika terlalu besar.

3. batch_size=16

Data akan diproses dalam kelompok kecil berisi 16 sampel per iterasi.

Penggunaan batch membantu dalam optimalisasi dan penggunaan memori.

4. validation_data=(x_validate, y_validate)

Model akan dievaluasi menggunakan data validasi setelah setiap epoch.

Berguna untuk melihat apakah model mengalami overfitting atau underfitting.

Gambar 4.10 menunjukkan output hasil training model neural network menggunakan TensorFlow/Keras.

```
Epoch 1/1000
38/38 [==============================] - 1s 7ms/step - loss: 0.7994 - mae: 0.7884 - val_loss: 0.5914 - val_mae: 0.6926
Epoch 2/1000
38/38 [==============================] - 0s 2ms/step - loss: 0.4854 - mae: 0.6188 - val_loss: 0.4723 - val_mae: 0.6022
Epoch 3/1000
38/38 [==============================] - 0s 2ms/step - loss: 0.4214 - mae: 0.5637 - val_loss: 0.4276 - val_mae: 0.5725
Epoch 4/1000
38/38 [==============================] - 0s 2ms/step - loss: 0.3848 - mae: 0.5383 - val_loss: 0.3897 - val_mae: 0.5418
Epoch 5/1000
38/38 [==============================] - 0s 3ms/step - loss: 0.3516 - mae: 0.5157 - val_loss: 0.3540 - val_mae: 0.5120
```

Gambar 4. 10 Output Hasil Training Model Meural Network

```
Epoch 996/1000
38/38 [==============================] - 0s 3ms/step - loss: 0.1545 - mae: 0.3077 - val_loss: 0.1591 - val_mae: 0.3065
Epoch 997/1000
38/38 [==============================] - 0s 3ms/step - loss: 0.1538 - mae: 0.3065 - val_loss: 0.1594 - val_mae: 0.3131
Epoch 998/1000
38/38 [==============================] - 0s 3ms/step - loss: 0.1537 - mae: 0.3068 - val_loss: 0.1575 - val_mae: 0.3099
Epoch 999/1000
38/38 [==============================] - 0s 3ms/step - loss: 0.1538 - mae: 0.3069 - val_loss: 0.1648 - val_mae: 0.3181
Epoch 1000/1000
38/38 [==============================] - 0s 4ms/step - loss: 0.1549 - mae: 0.3082 - val_loss: 0.1585 - val_mae: 0.3119
```

Gambar 4.11 menunjukkan kode Python untuk memvisualisasikan loss selama proses pelatihan model.

```python
loss = history_1.history['loss']
val_loss = history_1.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'g.', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Gambar 4. 11 Kode Untuk Visualisasi Loss Selama Proses Pelatihan

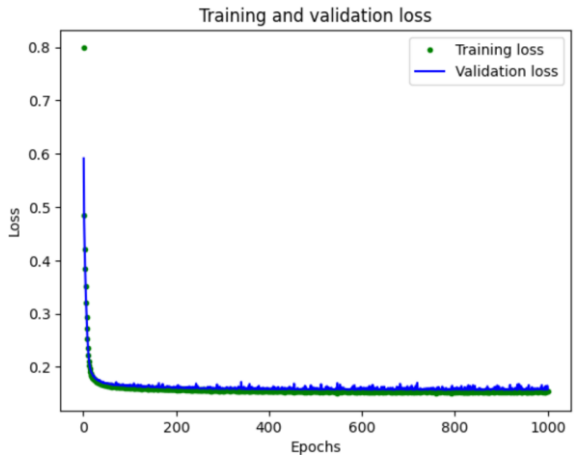
Rincian Fungsi

1. Mengambil History Loss

- history_1.history['loss'] → berisi nilai loss selama training.
- history_1.history['val_loss'] → berisi nilai loss selama validasi.

2. Membuat Rentang Epochs
    - epochs = range(1, len(loss) + 1) → Membuat list dari 1 sampai jumlah epoch untuk sumbu x.
3. Plot Loss Training & Validation
    - plt.plot(epochs, loss, 'g.', label='Training loss')
    - Menampilkan loss training dengan warna hijau (g.) dan titik kecil.
    - plt.plot(epochs, val_loss, 'b', label='Validation loss')
    - Menampilkan loss validasi dengan warna biru (b) dan garis kontinu.
4. Memberikan Label pada Grafik
    - plt.title('Training and validation loss') → Menambahkan judul.
    - plt.xlabel('Epochs'), plt.ylabel('Loss') → Memberikan label sumbu x dan y.
    - plt.legend() → Menampilkan legenda untuk membedakan antara training loss dan validation loss.
5. Menampilkan Grafik
    - plt.show() → Menampilkan grafik di layar.

Gambar 4.12 menunjukkan grafik loss selama proses training dan validasi model neural network menggunakan TensorFlow/Keras.



Gambar 4. 12 Hasil Grafik Loss selama proses training dan validasi Model Neral Network

Gambar 4.13 menunjukkan kode Python untuk memvisualisasikan training loss dan validation loss dengan mengabaikan 100 epoch pertama dalam grafik.

```
SKIP = 100
 plt.plot(epochs[SKIP:], loss[SKIP:], 'g.', label='Training loss')
 plt.plot(epochs[SKIP:], val_loss[SKIP:], 'b.', label='Validation loss')
 plt.title('Training and validation loss')
 plt.xlabel('Epochs')
 plt.ylabel('Loss')
 plt.legend()
plt.show()
```

Gambar 4. 13 Kode Untuk Visualisasi Training Loss dan Validasi

Rincian Fungsi

1. Melewati 100 Epoch Pertama

SKIP = 100 → Hanya menampilkan loss setelah epoch ke-100.

epochs[SKIP:], loss[SKIP:], dan val_loss[SKIP:] → Memotong data sehingga hanya menampilkan nilai setelah epoch ke-100.

2. Plot Loss Training & Validation

plt.plot(epochs[SKIP:], loss[SKIP:], 'g.', label='Training loss')

Menampilkan loss training dengan warna hijau (g.) dan titik kecil.

plt.plot(epochs[SKIP:], val_loss[SKIP:], 'b.', label='Validation loss')

Menampilkan loss validasi dengan warna biru (b.) dan titik kecil.

3. Memberikan Label pada Grafik

plt.title('Training and validation loss') → Judul grafik.

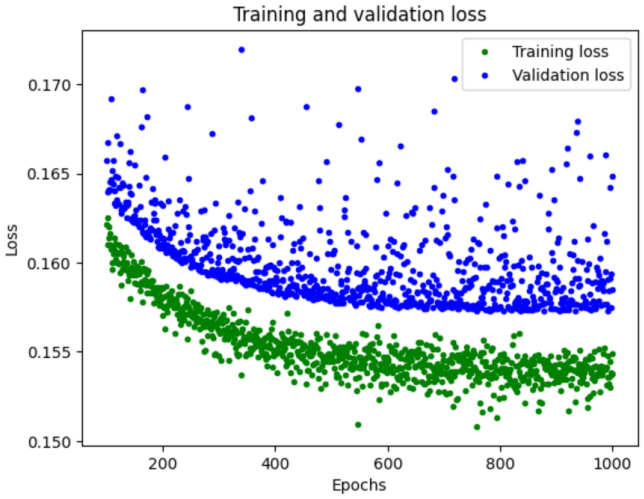
plt.xlabel('Epochs'), plt.ylabel('Loss') → Memberikan label sumbu x dan y.

plt.legend() → Menampilkan legenda untuk membedakan antara training loss dan validation loss.

4. Menampilkan Grafik

plt.show() → Menampilkan grafik di layar.

Gambar 4.14 menunjukkan grafik Training Loss dan Validation Loss setelah mengabaikan 100 epoch pertama selama pelatihan model



Gambar 4. 14 Grafik Training Loss dan Validation

Gambar 4.15 menunjukkan kode Python untuk memvisualisasikan Mean Absolute Error (MAE) pada training dan validation data selama proses pelatihan model.

```
mae = history_1.history['mae']
val_mae = history_1.history['val_mae']
plt.plot(epochs[SKIP:], mae[SKIP:], 'g.', label='Training MAE')
plt.plot(epochs[SKIP:], val_mae[SKIP:], 'b.', label='Validation MAE')
plt.title('Training and validation mean absolute error')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.legend()
plt.show()
```

Gambar 4. 15 Kode untuk visualisasi Mean Absolute Error

Rincian Fungsi

1. Mengambil Data MAE dari Training & Validation

history_1.history['mae'] → Menyimpan nilai Mean Absolute Error (MAE) dari training.

history_1.history['val_mae'] → Menyimpan nilai MAE dari validasi.

2. Melewati Epoch Awal yang Tidak Stabil

epochs[SKIP:] → Memotong epoch awal agar tidak ditampilkan.

mae[SKIP:], val_mae[SKIP:] → Menampilkan MAE hanya setelah sejumlah epoch tertentu.

3. Plot MAE Training dan Validation

plt.plot(epochs[SKIP:], mae[SKIP:], 'g.', label='Training MAE')

Menampilkan MAE training dengan warna hijau (g.).

plt.plot(epochs[SKIP:], val_mae[SKIP:], 'b.', label='Validation MAE')

Menampilkan MAE validasi dengan warna biru (b.).

4. Memberikan Label pada Grafik

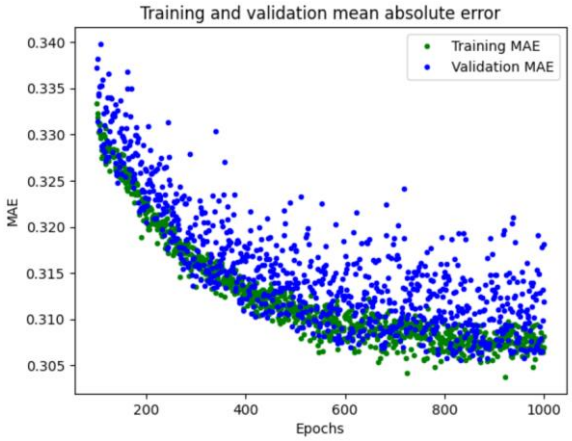
plt.title('Training and validation mean absolute error') → Judul grafik.

plt.xlabel('Epochs'), plt.ylabel('MAE') → Memberikan label sumbu x dan y.

plt.legend() → Menampilkan legenda agar grafik lebih mudah dibaca.

5. Menampilkan Grafik

plt.show() → Menampilkan grafik di layar.



Gambar 4. 16 Pelatihan Dan validasi Mean Absolute Error

```
predictions = model_1.predict(x_train)
plt.clf()
plt.title('Training data predicted vs actual values')
plt.plot(x_test, y_test, 'b.', label='Actual')
plt.plot(x_train, predictions, 'r.', label='Predicted')
plt.legend()
plt.show()
```

Gambar 4. 17 Kode untuk membuat plot prediksi

Rincian Fungsi Gambar 4.17 :

1. Melakukan Prediksi pada Data Latih

predictions = model_1.predict(x_train)

model_1.predict(x_train) → Model menghasilkan prediksi untuk data latih (x_train).

predictions akan berisi nilai-nilai prediksi yang dihasilkan oleh model.

2. Menghapus Grafik Sebelumnya

plt.clf()

plt.clf() → Membersihkan grafik sebelum membuat plot baru agar tidak tercampur dengan grafik sebelumnya.

Menambahkan Judul Grafik

plt.title('Training data predicted vs actual values')

Memberikan judul pada grafik: "Training data predicted vs actual values"

3. Menampilkan Data Aktual

plt.plot(x_test, y_test, 'b.', label='Actual')

plt.plot(x_test, y_test, 'b.', label='Actual')

Menampilkan data aktual dengan titik berwarna biru (b.).

x_test, y_test → Data uji digunakan sebagai representasi data aktual.

4. Menampilkan Data Prediksi

plt.plot(x_train, predictions, 'r.', label='Predicted')

plt.plot(x_train, predictions, 'r.', label='Predicted')

Menampilkan data prediksi model dengan titik merah (r.).

x_train, predictions → Data latih digunakan untuk mengevaluasi prediksi model.

5. Menampilkan Legenda dan Grafik

plt.legend()

plt.show()

plt.legend() → Menampilkan legenda agar lebih mudah membedakan antara data aktual (biru) dan prediksi (merah).

plt.show() → Menampilkan grafik hasil visualisasi.

```
model_2 = tf.keras.Sequential()
model_2.add(mylayers.Dense(16, activation='relu', input_shape=(1,)))
model_2.add(mylayers.Dense(16, activation='relu'))
model_2.add(mylayers.Dense(1))
model_2.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
model_2.summary()
```

Gambar 4. 18 Kode untuk membuat Model 2

Penjelasan Gambar 4.18

1. model_2 = tf.keras.Sequential()

tf.keras.Sequential() → Membuat model Sequential, yaitu model dengan susunan layer bertingkat dari input hingga output.

1. model_2.add(mylayers.Dense(16, activation='relu', input_shape=(1,)))

Layer pertama (Hidden Layer 1)

mylayers.Dense(16, activation='relu', input_shape=(1,))

Memiliki 16 neuron dengan fungsi aktivasi ReLU.

input_shape=(1,) menunjukkan bahwa input model memiliki 1 fitur.

2. model_2.add(mylayers.Dense(16, activation='relu'))

Layer kedua (Hidden Layer 2)

Memiliki 16 neuron.

Menggunakan ReLU sebagai fungsi aktivasi.

3. model_2.add(mylayers.Dense(1))

Layer Output

Memiliki 1 neuron, sesuai untuk regresi (karena ingin memprediksi nilai numerik).

Tidak menggunakan fungsi aktivasi, sehingga hasilnya berupa nilai kontinu.

4. model_2.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])

Mengompilasi model dengan:

optimizer='rmsprop' → Menggunakan RMSprop sebagai optimasi.

loss='mse' → Menggunakan Mean Squared Error (MSE) sebagai fungsi loss.

metrics=['mae'] → Memantau Mean Absolute Error (MAE) selama pelatihan.

5. model_2.summary()

Menampilkan ringkasan model termasuk jumlah parameter, jumlah layer, dan ukuran setiap layer.

```
history_2 = model_2.fit(
    x_train, y_train,
    epochs=600,
    batch_size=16,
    validation_data=(x_validate, y_validate)
)
```

Gambar 4. 19 Kode untuk pelatihan Model 2

Penjelasan Gambar 4.19 :

1. model_2.fit(...)

Metode fit() digunakan untuk melatih model neural network dengan data latih (x_train, y_train).

2. epochs=600

Model akan dilatih sebanyak 600 epoch (iterasi) untuk meningkatkan akurasi.

3. batch_size=16

Data akan diproses dalam batch berisi 16 sampel dalam satu iterasi.

4. validation_data=(x_validate, y_validate)

Model akan dievaluasi pada data validasi (x_validate, y_validate) di setiap epoch untuk memantau performa selama pelatihan.

```
loss = history_2.history['loss']
val_loss = history_2.history['val_loss']
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'g.', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Gambar 4. 20 Kode Untuk menampilkan hasil loss pada model 2

Penjelasan Kode Gambar 4.20

1. Mengambil Data Loss dari History Training

loss = history_2.history['loss']

val_loss = history_2.history['val_loss']

history_2.history['loss'] → Menyimpan nilai loss (kerugian) pada data training selama proses pelatihan model.

history_2.history['val_loss'] → Menyimpan nilai validation loss (kerugian pada data validasi).

2. Membuat Rentang Epochs

epochs = range(1, len(loss) + 1)

range(1, len(loss) + 1) → Membuat rentang epoch dari 1 hingga jumlah epoch terakhir.

Digunakan sebagai sumbu X dalam plot.

3. Membuat Grafik Loss

plt.plot(epochs, loss, 'g.', label='Training loss')

plt.plot(epochs, val_loss, 'b', label='Validation loss')

plt.plot(epochs, loss, 'g.', label='Training loss') → Memplot training loss dengan titik warna hijau ('g.').

plt.plot(epochs, val_loss, 'b', label='Validation loss') → Memplot validation loss dengan garis warna biru ('b').

4. Menambahkan Judul dan Label Grafik

plt.title('Training and validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

plt.title('Training and validation loss') → Menambahkan judul grafik.

plt.xlabel('Epochs') → Memberi label sumbu X sebagai Epochs.

plt.ylabel('Loss') → Memberi label sumbu Y sebagai Loss.

plt.legend() → Menampilkan keterangan untuk training loss dan validation loss.

plt.show() → Menampilkan grafik.

```
SKIP = 100
plt.clf()
plt.plot(epochs[SKIP:], loss[SKIP:], 'g.', label='Training loss')
plt.plot(epochs[SKIP:], val_loss[SKIP:], 'b.', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Gambar 4. 21 Kode untuk menggambarkan proses training dan validasi diatas 100 epoch

Penjelasan Kode Gambar 4.21:

1. SKIP = 100

Variabel SKIP digunakan untuk melewatkan 100 epoch pertama agar grafik lebih jelas dan tidak terlalu padat.

2. plt.clf()

plt.clf() digunakan untuk membersihkan plot sebelum membuat plot baru, mencegah tumpang tindih gambar.

plt.plot(epochs[SKIP:], loss[SKIP:], 'g.', label='Training loss')

plt.plot(epochs[SKIP:], val_loss[SKIP:], 'b.', label='Validation loss')

plt.plot(...) digunakan untuk menggambar grafik:

Training loss ditampilkan dalam warna hijau ('g.').

Validation loss ditampilkan dalam warna biru ('b.').

[SKIP:] berarti hanya menampilkan epoch setelah 100 pertama.

plt.title('Training and validation loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

3. Judul & Label:

plt.title(...) → Menetapkan judul grafik.

plt.xlabel('Epochs') → Label sumbu X sebagai "Epochs".

plt.ylabel('Loss') → Label sumbu Y sebagai "Loss".

plt.legend() → Menampilkan keterangan warna pada grafik.

plt.show() → Menampilkan grafik.

```
plt.clf()
mae = history_2.history['mae']
val_mae = history_2.history['val_mae' ]
plt.plot( epochs[SKIP : ], mae [SKIP : ], 'g .', label= 'Training MAE')
plt.plot( epochs[SKIP: ], val_mae[SKIP : ], 'b .', label='Validation MAE')
plt.title ('Training and validation mean absolute error ')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.legend ()
plt.show ()
```

Gambar 4. 22 Kode untuk menggambarkan Mean Absolute Error

Penjelasan Kode:

1. plt.clf()

plt.clf() digunakan untuk membersihkan plot sebelum membuat grafik baru, mencegah tumpang tindih gambar.

2. mae = history_2.history['mae']

val_mae = history_2.history['val_mae']

mae → Menyimpan nilai Mean Absolute Error (MAE) saat training dari history_2.history['mae'].

val_mae → Menyimpan nilai MAE saat validasi dari history_2.history['val_mae'].

3. plt.plot(epochs[SKIP:], mae[SKIP:], 'g.', label='Training MAE')

plt.plot(epochs[SKIP:], val_mae[SKIP:], 'b.', label='Validation MAE')

Training MAE ditampilkan dalam warna hijau ('g.').

Validation MAE ditampilkan dalam warna biru ('b.').

[SKIP:] digunakan untuk melewatkan sejumlah epoch pertama (biasanya agar grafik lebih jelas).

4. plt.title('Training and validation mean absolute error')

plt.xlabel('Epochs')

plt.ylabel('MAE')

plt.legend()

plt.show()

5. Judul & Label:

plt.title(...) → Menetapkan judul grafik.

plt.xlabel('Epochs') → Label sumbu X sebagai "Epochs".

plt.ylabel('MAE') → Label sumbu Y sebagai "Mean Absolute Error".

plt.legend() → Menampilkan keterangan warna pada grafik.

plt.show() → Menampilkan grafik.

```
loss = model_2.evaluate(x_test, y_test)
predictions = model_2.predict(x_test)
plt.clf()
plt.title('Comparison of predictions and actual values')
plt.plot(x_test, y_test, 'b.', label='Actual')
plt.plot(x_test, predictions, 'r.', label='Predicted')
plt.legend()
plt.show()
```

Gambar 4. 23 Kode untuk menampilkan perbedaan antara prediksi dan Actualnya

Penjelasan Gambar 4.23:

1. loss = model_2.evaluate(x_test, y_test)

model_2.evaluate(x_test, y_test) → Mengevaluasi performa model pada data uji (test set) dan menyimpan nilai error atau loss.

2. predictions = model_2.predict(x_test)

model_2.predict(x_test) → Menghasilkan prediksi model berdasarkan data uji (x_test).

3. plt.clf()

plt.clf() → Membersihkan plot sebelum menggambar yang baru.

4. plt.title('Comparison of predictions and actual values')

Judul grafik: "Comparison of predictions and actual values" (Perbandingan antara prediksi dan nilai aktual).

5. plt.plot(x_test, y_test, 'b.', label='Actual')

plt.plot(x_test, predictions, 'r.', label='Predicted')

Data aktual (ground truth) divisualisasikan dengan warna biru ('b.').

Prediksi model divisualisasikan dengan warna merah ('r.').

6. plt.legend()

plt.show()

plt.legend() → Menampilkan label (keterangan) untuk warna pada grafik.

plt.show() → Menampilkan grafik.

```
converter = tf.lite.TFLiteConverter.from_keras_model(model_2)
tflite_model = converter.convert()
open("sine_model.tflite", "wb").write(tflite_model)
converter = tf.lite.TFLiteConverter.from_keras_model(model_2)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
def representative_dataset_generator():
    for value in x_test:
        yield [np.array(value, dtype=np.float32, ndmin=2)]
converter.representative_dataset = representative_dataset_generator
tflite_model = converter.convert()
open("sine_model_quantized.tflite", "wb").write(tflite_model)
```

Gambar 4. 24 Kode untuk menkonversi Model Keras ke TFLite

Penjelasan Gambar 4.24:

1. Konversi Model Keras ke TFLite(Line 1)

converter = tf.lite.TFLiteConverter.from_keras_model(model_2)

tflite_model = converter.convert()

open("sine_model.tflite", "wb").write(tflite_model)

tf.lite.TFLiteConverter.from_keras_model(model_2) → Membuat converter untuk mengubah model Keras (model_2) menjadi model TensorFlow Lite (TFLite).

converter.convert() → Mengkonversi model ke format TFLite.

open("sine_model.tflite", "wb").write(tflite_model) → Menyimpan model yang sudah dikonversi ke dalam file "sine_model.tflite".

2. Konversi dengan Kuantisasi

converter = tf.lite.TFLiteConverter.from_keras_model(model_2)

converter.optimizations = [tf.lite.Optimize.DEFAULT]

converter.optimizations = [tf.lite.Optimize.DEFAULT] → Mengaktifkan optimasi kuantisasi untuk membuat model lebih kecil dan lebih cepat tanpa kehilangan akurasi secara signifikan.

3. Membuat Representative Dataset untuk Kuantisasi**

def representative_dataset_generator():

```
    for value in x_test:
        yield [np.array(value, dtype=np.float32, ndmin=2)]
```

Representative dataset digunakan dalam kuantisasi model untuk memastikan performa tetap optimal pada perangkat dengan sumber daya terbatas.

Fungsi representative_dataset_generator():

Mengambil nilai dari x_test sebagai contoh data.

Mengubah setiap nilai menjadi array float32 dengan dimensi minimal 2 (ndmin=2).

4. Menetapkan Dataset dan Mengkonversi Model

converter.representative_dataset = representative_dataset_generator

tflite_model = converter.convert()

open("sine_model_quantized.tflite", "wb").write(tflite_model)

converter.representative_dataset = representative_dataset_generator → Menetapkan dataset perwakilan untuk kuantisasi.

converter.convert() → Mengkonversi model dengan kuantisasi.

Menyimpan model yang telah dikonversi dengan kuantisasi ke dalam file "sine_model_quantized.tflite".

5.2. Konversi Model ke Perangkat

Konversi model *Machine Learning* ke perangkat merupakan langkah penting dalam implementasi model di lingkungan dengan keterbatasan sumber daya, seperti mikrokontroler, edge devices, atau sistem IoT. Proses ini bertujuan untuk mengoptimalkan ukuran, kecepatan, dan efisiensi model agar dapat berjalan secara real-time dengan konsumsi daya yang minimal.

1. Persiapan lingkungan kerja.

Sebelum melakukan konversi model machine learning ke perangkat embedded seperti Arduino Nano 33 BLE Sense, penting untuk menyiapkan lingkungan

pengembangan yang mendukung proses konversi dan pengujian. Adapun langkah-langkah persiapan lingkungan kerja meliputi:

Instalasi Python dan Pustaka Pendukung:

- Pastikan Python 3.x telah terinstal pada sistem.

Instal pustaka TensorFlow dengan perintah berikut:

- pip install tensorflow

Untuk mikrokontroler, Anda juga perlu:

- pip install numpy

Instalasi Arduino IDE:

Unduh dan pasang Arduino IDE versi terbaru dari situs resmi:

https://www.arduino.cc/en/software

Tambahkan board Arduino Nano 33 BLE Sense melalui Board Manager.

Instal pustaka tambahan, seperti:

- Arduino_TensorFlowLite (diinstal manual melalui GitHub karena tidak tersedia di Library Manager)

Arduino_LSM9DS1, PDM, dan pustaka sensor lain jika diperlukan

Unduh TensorFlow Lite Micro Examples:

- Clone repository TensorFlow Lite for Microcontrollers untuk mengakses contoh dan template proyek: git clone https://github.com/tensorflow/tflite-micro

Pemasangan xxd (untuk konversi ke format C array):

Pada sistem berbasis Unix/Linux/MacOS, xxd biasanya sudah tersedia. Untuk Windows, xxd bisa didapat dari paket vim atau gunakan versi portabel CLI dari internet.

- xxd -i model_quantized.tflite > model_data.cc

Verifikasi Versi dan Dependensi:

Pastikan semua pustaka dan tools yang dibutuhkan kompatibel dengan versi TensorFlow dan Arduino yang digunakan. Perbedaan versi dapat menyebabkan kegagalan saat konversi atau deploy model ke perangkat.

Dengan lingkungan kerja yang telah disiapkan secara lengkap, proses konversi, kuantisasi, dan deploy model ke perangkat mikrokontroler dapat dilakukan dengan lancar dan efisien.

2. Ekspor Model yang Telah Dilatih

Langkah pertama dalam proses konversi adalah mengekspor model yang telah dilatih dalam format yang lebih ringan. Model deep learning biasanya dikembangkan dan dilatih dalam framework seperti TensorFlow, PyTorch, atau Scikit-learn. Setelah model mencapai akurasi yang diinginkan, model perlu diekspor ke format yang lebih kompatibel dengan perangkat tujuan.Untuk model yang dikembangkan menggunakan TensorFlow, model harus dikonversi ke TensorFlow Lite (TFLite) menggunakan perintah berikut:

import tensorflow as tf

- Memuat model yang telah dilatih

model = tf.keras.models.load_model("model.h5")

- Mengonversi model ke format TensorFlow Lite

converter = tf.lite.TFLiteConverter.from_keras_model(model)

tflite_model = converter.convert()

- Menyimpan model hasil konversi

Pada tahap ini, model yang awalnya dalam format H5 atau SavedModel telah dikonversi ke format TFLite yang lebih ringan.

3. Optimasi dan Kuantisasi Model

Model deep learning yang dikembangkan di komputer umumnya menggunakan floating-point precision (float32) yang memerlukan sumber daya besar untuk komputasi. Untuk mengurangi ukuran model dan mempercepat inferensi, dilakukan kuantisasi yang mengubah bobot model dari format float32 ke int8 atau float16.

Kuantisasi dapat dilakukan dengan beberapa metode, antara lain:

- Post-Training Quantization (PTQ): Kuantisasi dilakukan setelah model selesai dilatih.
- Full Integer Quantization: Seluruh bobot dan aktivasi model dikonversi menjadi bilangan bulat (int8) agar bisa dijalankan di perangkat dengan prosesor sederhana.
- Hybrid Quantization: Menggabungkan floating-point dan integer untuk keseimbangan antara akurasi dan efisiensi.
- Mengubah model menjadi TensorFlow Lite dengan Kuantisasi

Gambar 4.10 Line 4 adalah contoh kode untuk melakukan kuantisasi model:

- Mengaktifkan optimasi kuantisasi

Gambar 4.10 Line 5 melakukan optimasi

```
converter.optimizations = [tf.lite.Optimize.DEFAULT]
```

- Menyediakan dataset perwakilan untuk kuantisasi

```
def representative_dataset():
    for data in x_test:
        yield [np.array(data, dtype=np.float32, ndmin=2)]
converter.representative_dataset = representative_dataset
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
```

- Konversi model dengan kuantisasi penuh

```
tflite_quantized_model = converter.convert()
```

- Menyimpan model yang telah dikonversi

```
with open("model_quantized.tflite", "wb") as f:
    f.write(tflite_quantized_model)
```

Dengan kuantisasi ini, ukuran model bisa berkurang hingga 75% dan meningkatkan efisiensi komputasi hingga 4 kali lebih cepat dibandingkan model float32.

4. Deploy Model ke Perangkat Tujuan

Setelah model dikonversi ke format TFLite, langkah selanjutnya adalah memasukkannya ke perangkat yang akan digunakan, seperti Arduino, Raspberry Pi, atau ESP32. Untuk mikrokontroler seperti Arduino Nano 33 BLE Sense, model dapat dijalankan menggunakan pustaka TensorFlow Lite for Microcontrollers. Berikut adalah langkah-langkahnya:

1. Mengompilasi model ke dalam kode sumber C++

TensorFlow Lite untuk mikrokontroler tidak mendukung file .tflite secara langsung, sehingga model harus dikonversi ke dalam array biner menggunakan skrip berikut:

xxd -i model_quantized.tflite > model_data.cc

Hasilnya adalah array yang dapat disisipkan langsung dalam kode C++.

2. Memuat model di Arduino

Dalam kode Arduino (file .ino), model yang telah dikompilasi ditambahkan menggunakan pustaka TensorFlow Lite:

Gambar 4.11 menunjukkan potongan kode dalam bahasa pemrograman C++ yang digunakan untuk menginisialisasi model TensorFlow Lite di perangkat dengan sumber daya terbatas, seperti mikrokontroler atau edge devices.

```
#include <TensorFlowLite.h>
#include "model_data.h"

// Inisialisasi model TensorFlow Lite
const tflite::Model* model = tflite::GetModel(model_data);
```

Gambar 4. 25 Potongan Kode untuk Menginisialisasi Model TensorFlow Lite

Kode Gambar 4.25 memastikan bahwa model dapat dimuat dan digunakan di perangkat dengan memori terbatas.

3. Menjalankan Inferensi di Perangkat

Setelah model di-deploy, kita bisa menggunakannya untuk menjalankan inferensi (prediksi) secara real-time. Untuk model regresi sederhana, kita bisa menggunakan kode berikut:

Gambar 4.26 menunjukkan potongan kode dalam bahasa C++ yang digunakan untuk menjalankan inferensi pada model TensorFlow Lite yang telah dimuat sebelumnya.

```
float input_value = 1.5;

input->data.f[0] = input_value;
if (interpreter->Invoke() != kTfLiteOk) {
Serial.println("Error running inference!");
return;
}

float output_value = output->data.f[0];
Serial.println(output_value);
```

Gambar 4. 26 Kode Untuk menjalankan inferensi pada Model TensorFlow Lite

Dengan cara di Gambar 4.26, model yang telah dikonversi dapat berjalan di Arduino atau ESP32 tanpa memerlukan komputer atau cloud.

4. Pengujian dan Validasi Model

Setelah model berhasil dijalankan di perangkat, tahap terakhir adalah pengujian dan validasi untuk memastikan model bekerja dengan baik. Pengujian ini dapat dilakukan dengan:

1. Membandingkan hasil inferensi di perangkat dengan hasil di komputer untuk memastikan tidak ada perbedaan signifikan.
2. Menggunakan dataset pengujian untuk melihat apakah model tetap akurat setelah dikonversi dan dikompresi.
3. Mengukur waktu inferensi untuk memastikan model berjalan dengan kecepatan yang sesuai dengan kebutuhan sistem real-time.

5. Kesimpulan

Proses konversi model ke perangkat adalah langkah penting dalam penerapan machine learning di perangkat embedded. Dengan menggunakan teknik seperti TensorFlow Lite, kuantisasi, dan optimasi model, kita dapat menjalankan model deep learning di perangkat dengan daya rendah seperti Arduino, ESP32, atau Raspberry Pi.

Proses utama yang dilakukan dalam konversi model meliputi:

- Ekspor model yang telah dilatih ke format yang kompatibel.
- Optimasi dan kuantisasi untuk mengurangi ukuran model dan meningkatkan efisiensi.
- Deploy ke perangkat menggunakan pustaka TensorFlow Lite.
- Pengujian dan validasi untuk memastikan akurasi dan performa tetap terjaga.

5.3. Inference

Proses *Inference* dalam TinyML merupakan tahap akhir dari implementasi model machine learning, di mana model yang telah dilatih digunakan untuk membuat prediksi atau keputusan secara langsung di perangkat dengan sumber daya terbatas, seperti mikrokontroler dan edge devices. Tidak seperti pelatihan model yang memerlukan komputasi tinggi dan sering kali dilakukan di server atau cloud, inference dilakukan secara lokal di perangkat kecil dengan efisiensi tinggi.

1. Persiapan Data Input

Langkah pertama dalam inference adalah menyiapkan data yang akan digunakan sebagai input ke model. Data ini dapat berasal dari berbagai sensor yang terhubung ke perangkat, seperti kamera untuk pengenalan gambar, mikrofon untuk pengenalan suara, atau akselerometer untuk mendeteksi gerakan. Setiap model memiliki format input yang berbeda, sehingga data yang masuk harus diproses agar sesuai dengan format yang diharapkan oleh model.

Sebagai contoh, jika model memerlukan gambar dengan resolusi 28x28 piksel dalam skala abu-abu, maka gambar dari sensor kamera perlu diubah ukurannya terlebih dahulu. Jika model membutuhkan nilai numerik dari sensor suhu, maka data harus dikonversi ke format yang kompatibel, misalnya dalam bentuk bilangan desimal dengan skala yang sesuai. Setelah data siap, nilai tersebut kemudian dimasukkan ke dalam tensor input yang dialokasikan dalam memori perangkat

2. Memuat Model dan Interpreter

Setelah input siap, langkah berikutnya adalah memuat model yang telah dikonversi ke format TensorFlow Lite (TFLite) ke dalam perangkat. Model ini biasanya telah dikompresi dan dikonversi dari model yang lebih besar agar dapat berjalan dengan efisien di sistem dengan RAM dan penyimpanan terbatas.

Interpreter TensorFlow Lite bertanggung jawab untuk menjalankan model dan mengelola proses inferensi di perangkat. Interpreter ini memastikan bahwa model dapat berjalan dengan baik meskipun terdapat keterbatasan komputasi. Untuk menginisialisasi model, perangkat akan mengalokasikan memori yang diperlukan untuk input, output, dan variabel perantara lainnya.

Pada Gambar 4.27 adalah contoh cara memuat model ke dalam interpreter (Pada file hello_world.ino) :

```
1  const tflite::Model* model = nullptr;
2  tflite::MicroInterpreter* interpreter = nullptr;
```

Gambar 4. 27 Kode untuk memuat model ke dalam interpreter

Dalam contoh ini, model yang telah dikonversi ke format TFLite disimpan dalam variabel model_data dan kemudian diproses oleh interpreter agar bisa digunakan untuk inference.

3. Menjalankan Model dengan Interpreter

Setelah model dimuat ke dalam interpreter, langkah berikutnya adalah menjalankan proses inferensi dengan data input yang telah dipersiapkan(Pada file hello_world.ino). Ini dilakukan dengan memanggil fungsi Invoke() pada interpreter, yang akan mengeksekusi model menggunakan input yang telah diberikan.

Gambar 4.28 adalah contoh kode untuk menjalankan inference:

```
if (interpreter->Invoke() != kTfLiteOk) {
MicroPrintf("Error: Invoke() failed!\n");
return;
}
```

Gambar 4. 28 Kode untuk menjalankan inference

Jika Invoke() berhasil, model akan memproses data dan menghasilkan output berdasarkan hasil prediksi. Namun, jika terjadi kesalahan dalam eksekusi model, sistem akan menampilkan pesan error agar dapat dilakukan perbaikan.

4. Mengambil Hasil Inferensi dari Model

Setelah model selesai dijalankan, hasil dari inferensi disimpan dalam tensor output. Output ini dapat berupa nilai numerik, probabilitas dari berbagai kategori, atau hasil perhitungan regresi, tergantung pada jenis model yang digunakan.

Gambar 4.29 adalah contoh cara mengambil nilai output dari tensor hasil prediksi:

```
float y_value = output->data.f[0];
```

Gambar 4. 29 Kode untuk mengambil nilai output

Dalam contoh ini, hasil prediksi model diambil dari elemen pertama dalam array output bertipe float. Nilai ini kemudian dapat digunakan untuk berbagai tujuan, seperti mengendalikan perangkat keras, menampilkan hasil di layar, atau mengirim data ke sistem lain.

5. Menggunakan Hasil Inferensi untuk Pengambilan Keputusan

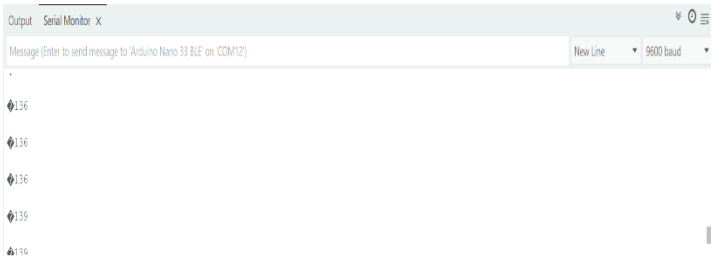
Setelah mendapatkan hasil prediksi dari model, langkah terakhir adalah mengambil tindakan berdasarkan hasil tersebut. Misalnya, dalam sistem smart home, jika model mendeteksi bahwa suhu ruangan terlalu tinggi, maka kipas atau pendingin ruangan dapat diaktifkan secara otomatis. Jika model digunakan untuk mengenali suara perintah tertentu, perangkat bisa merespons dengan menjalankan instruksi yang sesuai.

Gambar 4.30 adalah contoh kode untuk menangani hasil inferensi berdasarkan kategori output (Pada file arduino_output_handler.cpp):

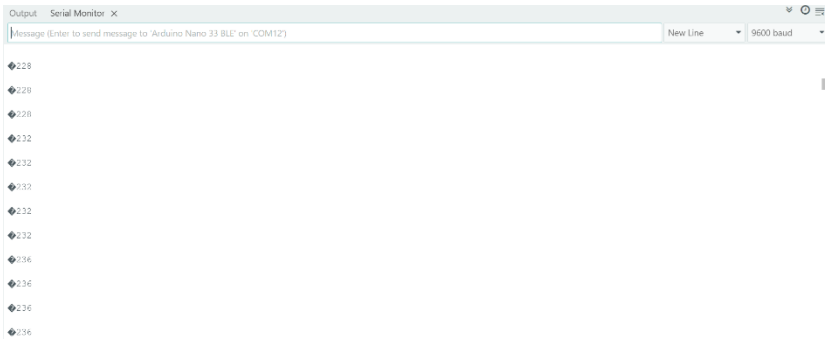
```
void HandleOutput(float x_value, float y_value) {
if (!initialized) {
    // Set the LED pin to output
    pinMode(led, OUTPUT);
    initialized = true;
}
```

Gambar 4. 30 Kode untuk menangani hasil inferensi

Dalam contoh Gambar 4.30, jika hasil inferensi menunjukkan nilai lebih dari 0.5, sistem akan menganggap bahwa suara telah dikenali dan akan memberikan respons dengan menyalakan LED. Jika nilai di bawah ambang batas, LED akan dimatikan.



Gambar 4. 31 Hasil Serial Monitor



Gambar 4. 32 Hasil Serial Monitor

Gambar 4.31 dan 4.32 menampilkan deretan karakter seperti 136, 139, dan angka-angka yang berubah secara berkala. Karakter tersebut merupakan hasil dari data yang dikirim oleh program Arduino ke komputer melalui komunikasi serial. Setiap satu detik, program mengirimkan nilai acak yang berada dalam rentang tertentu. Nilai ini kemudian diterima dan ditampilkan oleh Serial Monitor dalam bentuk karakter atau angka sesuai dengan data yang dikirimkan dari mikrokontroler. Tampilan yang berubah secara periodik menunjukkan bahwa program berjalan secara berulang dan terus menghasilkan output baru setiap detiknya.

6. Hasil

Gambar 4.33 adalah tampilan fisik dari papan mikrokontroler Arduino Nano 33 BLE yang digunakan dalam proses pengembangan sistem (terhubung melalui kabel USB seperti tampak pada gambar).

Gambar 4. 33 Tampilan Papan Arduino Nano 33 BLE

Dalam Gambar 4.33, terlihat bahwa papan Arduino Nano 33 BLE telah terhubung ke sumber daya melalui kabel USB, yang ditandai dengan menyala-nya LED indikator berwarna hijau di bagian atas papan. Papan ini memiliki bentuk kecil dan kompak, serta dilengkapi dengan modul komunikasi nirkabel berbasis Bluetooth Low Energy (BLE) dan sensor IMU bawaan. Fitur-fitur ini memungkinkan papan untuk digunakan dalam berbagai proyek berbasis Internet of Things (IoT), seperti pemantauan lingkungan, deteksi gerakan, dan aplikasi wearable. Keaktifan LED juga menunjukkan bahwa papan telah berhasil diinisialisasi dan siap digunakan untuk proses selanjutnya dalam sistem.

Selama kegiatan Praktik Kerja Lapangan (PKL), berhasil dilakukan implementasi dan pengujian perangkat mikrokontroler menggunakan papan pengembangan Arduino Nano 33 BLE yang dilengkapi dengan konektivitas Bluetooth dan sensor bawaan. Perangkat dihubungkan ke komputer menggunakan kabel USB dan berhasil terdeteksi oleh perangkat lunak Arduino IDE. Indikator LED menyala dengan stabil, menandakan bahwa papan berfungsi dengan baik. Program sederhana berhasil diunggah untuk membaca data dari sensor serta mengirimkan informasi secara nirkabel. Hasil ini menunjukkan bahwa perangkat bekerja secara optimal dan dapat dimanfaatkan sebagai prototipe awal untuk berbagai aplikasi berbasis Internet of Things (IoT), seperti pelacakan lokasi, pemantauan lingkungan, atau pengendalian perangkat jarak jauh.

7. Kesimpulan

Proses inference dalam TinyML mencakup beberapa tahapan utama, mulai dari persiapan input data, memuat model ke interpreter, menjalankan model, membaca hasil output, dan mengambil tindakan berdasarkan hasil prediksi. Dengan menggunakan TensorFlow Lite for Microcontrollers, model machine learning dapat berjalan secara efisien di perangkat dengan keterbatasan sumber daya.

Inference yang dioptimalkan memungkinkan berbagai aplikasi machine learning berbasis edge computing, seperti pengenalan suara, deteksi gerakan, atau analisis sensor, untuk berjalan tanpa memerlukan koneksi ke server atau cloud. Hal ini memberikan keunggulan dalam efisiensi daya, kecepatan pemrosesan, dan keamanan data, sehingga memungkinkan implementasi AI di perangkat kecil dengan daya rendah.

Untuk informasi lebih lanjut serta materi tambahan yang mendukung pembelajaran dalam buku ini, Anda dapat mengunduhnya melalui tautan berikut: https://tinymlbook.com/supplemental. Situs tersebut menyediakan berbagai sumber daya tambahan seperti kode program, dataset, dan petunjuk teknis yang relevan dengan topik TinyML.

Bab V

Penutup

5.1. Kesimpulan

Pada Praktik Kerja Lapangan (PKL) ini, telah dilakukan pengembangan model Machine Learning mulai dari tahap pelatihan hingga proses inference pada perangkat Arduino Sense. Model yang digunakan dalam penelitian ini berbasis persamaan sinus, yang memungkinkan perangkat untuk mengenali dan memprediksi pola berbasis fungsi sinusoidal. Proses pengembangan mencakup pembuatan dataset, pelatihan model, konversi ke format yang kompatibel dengan TensorFlow Lite, serta implementasi pada perangkat keras dengan sumber daya terbatas. Dengan pendekatan ini, model dapat berjalan secara efisien di mikrokontroler, memungkinkan aplikasi seperti analisis sinyal, pengenalan pola, dan pemrosesan data real-time secara langsung di perangkat.

Implementasi proses pengolahan data belum sempat dikerjakan pada PKL ini disebabkan oleh beberapa kendala yang dihadapi selama pelaksanaan. Salah satu faktor utama adalah terbatasnya waktu yang tersedia untuk menyelesaikan seluruh tahapan pengembangan, mulai dari pelatihan model hingga deployment pada perangkat. Selain itu, kendala teknis seperti kompatibilitas perangkat keras, keterbatasan sumber daya komputasi pada mikrokontroler, serta tantangan dalam konversi model ke format yang lebih ringan juga menjadi faktor penghambat. Di samping itu, proses pengolahan data memerlukan pengujian dan validasi lebih lanjut, yang membutuhkan waktu serta dukungan infrastruktur yang memadai. Oleh karena itu, tahap ini masih menjadi bagian yang dapat dikembangkan lebih lanjut dalam penelitian atau implementasi berikutnya.

Evaluasi performa model dalam kondisi nyata telah dilakukan dengan menampilkan hasil inference di Serial Monitor, sehingga output model dapat diamati secara langsung. Namun, dalam implementasi ini, penggunaan sensor bawaan Arduino Sense belum dapat dilakukan sepenuhnya. Hal ini disebabkan oleh keterbatasan dalam integrasi model dengan sensor, baik dari segi kompatibilitas perangkat lunak maupun konfigurasi perangkat keras. Sebagai alternatif, nilai input diberikan secara manual atau melalui simulasi, sehingga prediksi model tetap dapat dianalisis meskipun tanpa data sensor langsung. Untuk pengembangan selanjutnya,

diperlukan optimalisasi lebih lanjut agar model dapat berinteraksi langsung dengan sensor, memungkinkan evaluasi yang lebih akurat dalam kondisi nyata.

5.2. Saran

Untuk pengembangan lebih lanjut, terdapat beberapa saran yang dapat diterapkan agar implementasi Machine Learning pada perangkat Arduino Sense menjadi lebih optimal dan aplikatif.

1. Integrasi Model dengan Sensor Bawaan

Pada penelitian ini, model masih dievaluasi menggunakan input simulasi tanpa melibatkan sensor bawaan. Ke depan, perlu dilakukan optimalisasi agar model dapat langsung menerima data dari sensor akselerometer, mikrofon, atau sensor lainnya. Dengan demikian, hasil inferensi akan lebih akurat dan mencerminkan kondisi nyata.

2. Optimasi dan Kuantisasi Model

Untuk meningkatkan efisiensi komputasi pada perangkat dengan keterbatasan daya dan memori, model dapat dioptimalkan lebih lanjut menggunakan teknik post-training quantization atau pruning. Teknik ini dapat mempercepat waktu inferensi dan mengurangi penggunaan memori tanpa mengorbankan akurasi secara signifikan.

3. Pengolahan Data Secara Langsung di Perangkat

Pada PKL ini, proses pengolahan data belum dapat diimplementasikan sepenuhnya. Untuk penelitian selanjutnya, disarankan agar perangkat dapat melakukan preprocessing data secara langsung, seperti filtering, normalisasi, atau ekstraksi fitur. Dengan cara ini, model tidak hanya menerima input mentah, tetapi juga data yang telah diproses untuk meningkatkan akurasi prediksi.

4. Pengujian Model dalam Berbagai Kondisi

Evaluasi model sebaiknya dilakukan dalam berbagai kondisi lingkungan nyata untuk memastikan ketahanannya terhadap perubahan variabel eksternal. Misalnya, jika model diterapkan untuk analisis gerakan, pengujian dapat mencakup berbagai tingkat pencahayaan, variasi suhu, atau latensi dalam pemrosesan sinyal sensor.

5. Implementasi Model pada Perangkat Lain

Selain Arduino Sense, model juga dapat diuji pada platform IoT lainnya, seperti ESP32, Raspberry Pi, atau perangkat edge AI lainnya. Dengan melakukan perbandingan performa pada berbagai platform, dapat ditentukan perangkat yang paling efisien dan sesuai dengan kebutuhan aplikasi tertentu.

6. Penggunaan Model yang Lebih Kompleks

Ke depan, model yang digunakan dapat diperluas dengan arsitektur yang lebih kompleks, seperti Convolutional Neural Networks (CNN) untuk pengolahan citra atau Recurrent Neural Networks (RNN) untuk analisis data sekuensial. Hal ini memungkinkan implementasi aplikasi yang lebih luas, seperti pengenalan suara, deteksi objek, atau analisis pola waktu nyata.

Dengan menerapkan saran-saran ini, diharapkan pengembangan Machine Learning pada Arduino Sense dapat berjalan lebih optimal, efisien, dan memiliki dampak yang lebih luas dalam aplikasi dunia nyata.

Daftar Pustaka

Warden, P. and Situnayake, D., 2019, *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-low-power Microcontrollers*, O'Reilly Media, Inc., Sebastopol, USA.

TensorFlow, 2023, *TensorFlow Lite Guide*, Available at: https://www.tensorflow.org/lite [Accessed 11 June 2025].

Arduino, 2023, *Arduino Nano 33 BLE Sense Overview*, Available at: https://docs.arduino.cc/hardware/nano-33-ble-sense [Accessed 11 June 2025].

Goodfellow, I., Bengio, Y. and Courville, A., 2016, *Deep Learning*, MIT Press, Cambridge, USA.

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

Sze, V., Chen, Y.H., Yang, T.J. and Emer, J.S., 2017, Efficient Processing of Deep Neural Networks: A Tutorial and Survey, *Proceedings of the IEEE*, 105(12):2295–2329. https://doi.org/10.1109/JPROC.2017.2761740

Banzi, M. and Shiloh, M., 2014, *Getting Started with Arduino* (3rd ed.), Maker Media, Inc., Sebastopol, USA.

Duda, R.O., Hart, P.E. and Stork, D.G., 2001, *Pattern Classification* (2nd ed.), Wiley-Interscience, New York, USA.

Zhang, C., Bengio, S., Hardt, M., Recht, B. and Vinyals, O., 2017, Understanding Deep Learning Requires Rethinking Generalization, *arXiv preprint*, arXiv:1611.03530.

Lane, N.D., Bhattacharya, S., Georgiev, P., Forlivesi, C., Jiao, P., Qendro, L. and Kawsar, F., 2015, DeepX: A Software Accelerator for Low-power Deep Learning Inference on Mobile Devices, *Proceedings of the 14th International Conference on Information Processing in Sensor Networks (IPSN '15)*, 23–34.