

BAB II

KAJIAN LITERATUR

2.1 Suara

Suara manusia merupakan bunyi yang dihasilkan oleh getaran pita suara di laring ketika udara dari paru-paru melewatinya [20]. Getaran tersebut menghasilkan gelombang tekanan yang merambat melalui udara dan ditangkap oleh sistem pendengaran manusia [20], [21]. Setiap individu memiliki karakteristik suara yang berbeda karena dipengaruhi oleh perbedaan panjang, massa, dan ketegangan pita suara [20]. Suara dipahami sebagai sinyal kontinu yang membawa informasi identitas yang khas, dipengaruhi oleh fisik pita suara serta rongga pada tenggorokan [21], [22].

Karakteristik suara setiap individu dibentuk oleh tiga acuan akustik utama, yaitu *pitch*, *timbre*, dan *amplitude*. *Pitch* adalah tinggi rendahnya nada suara yang ditentukan oleh frekuensi getaran pita suara. Pita suara yang lebih panjang dan tebal menghasilkan frekuensi yang lebih rendah, sehingga penutur laki-laki dewasa umumnya memiliki *pitch* lebih rendah dibandingkan perempuan [23], [24]. *Timbre* adalah warna atau kualitas suara yang membedakan satu individu dari individu lain meskipun *pitch*-nya sama. *Timbre* dibentuk oleh *formant*, yaitu puncak energi pada frekuensi tertentu yang dihasilkan oleh bentuk dan ukuran rongga saluran tenggorokan vokal setiap individu [23], [25], [26]. *Amplitude* menentukan keras lemahnya suara yang terdengar, yang dipengaruhi oleh besarnya tekanan udara dari paru-paru [20].

Selain pita suara, organ artikulasi, yaitu lidah, bibir, dan rahang, bergerak aktif untuk membentuk bunyi vokal dan konsonan. Pergerakan organ-organ ini berlangsung terus-menerus selama berbicara dan membuat karakteristik sinyal suara berubah dari waktu ke waktu [27], [28]. Analisis akustik yang hanya bersandar pada domain waktu (*waveform*) tidak mampu menangkap seluruh informasi tersebut, sehingga diperlukan representasi domain waktu-frekuensi dalam bentuk spektrogram yang dapat mengamati interaksi antara waktu, frekuensi, dan amplitudo secara bersamaan [29], [30]. Pada representasi waktu-frekuensi seperti spektrogram, pita energi yang tersusun konsisten mencerminkan letak *formant* atau karakteristik resonansi vokal yang menjadi ciri pembeda antarindividu [31], [32].

2.2 Sinyal Audio Digital

Suara manusia berupa gelombang yang bersifat kontinu, baik dalam amplitudo maupun waktu. Agar sinyal ini dapat diolah dan dianalisis oleh komputer, sinyal analog tersebut harus melalui proses digitalisasi. Digitalisasi merupakan tahapan yang mengubah getaran suara menjadi representasi matematis diskret yang dapat disimpan dalam memori komputer dan diproses oleh model AI [31], [33]. Sinyal digital yang telah terbentuk dapat divisualisasikan melalui dua bentuk utama. Bentuk pertama adalah *waveform* yang menampilkan perubahan amplitudo terhadap waktu. Visualisasi ini memungkinkan pengamatan ritme, durasi, dan energi suara melalui *signal envelope* yang terbentuk sepanjang waktu. Bentuk kedua adalah spektrogram yang menampilkan distribusi energi pada sumbu frekuensi dan waktu secara simultan. Sumbu vertikal menunjukkan frekuensi, sedangkan sumbu horizontal menunjukkan waktu, dan kekuatan sinyal direpresentasikan melalui gradasi warna. Visualisasi ini membantu mengidentifikasi artefak manipulasi digital yang bersifat halus [31], [32].

2.2.1 Sampling dan Teorema Nyquist

Proses konversi sinyal analog ke digital dimulai dengan operasi *sampling* pada domain waktu. *Sampling* merupakan proses pengambilan nilai sinyal analog pada interval waktu yang seragam. Pengambilan ini ditentukan oleh parameter *sampling rate* dengan satuan Hertz (Hz). Nilai *sampling rate* tidak boleh ditentukan sembarangan karena sangat memengaruhi kualitas rekonstruksi sinyal digital [32], [34]. Berdasarkan Teorema Nyquist-Shannon, frekuensi *sampling* (f_s) harus minimal dua kali frekuensi tertinggi (f_{max}) yang terkandung dalam sinyal asli [27], [35]. Ketentuan ini bertujuan menghindari fenomena *aliasing*, yaitu tumpang tindihnya informasi frekuensi yang menyebabkan distorsi pada sinyal hasil rekonstruksi. Syarat *sampling* ini dirumuskan pada persamaan (1).

$$f_s \geq 2f_{max} \dots\dots\dots(1)$$

Keterangan:

f_s = Frekuensi *sampling* (Hz).
 f_{max} = Frekuensi maksimum (Hz).

Pada penelitian ini, seluruh dataset menggunakan *sampling rate* 44.100 Hz. Nilai tersebut telah memenuhi kriteria Nyquist karena melebihi dua kali frekuensi maksimum pendengaran manusia, yaitu sekitar 20.000 Hz, sehingga informasi frekuensi tinggi dapat dipertahankan tanpa menimbulkan *aliasing* [27], [28].

2.2.2 Bit Depth

Setelah proses sampling menghasilkan titik-titik sampel pada domain waktu, tahap berikutnya adalah kuantisasi amplitudo. Kuantisasi merupakan proses memetakan nilai amplitudo kontinu dari setiap sampel ke level amplitudo diskret terdekat yang direpresentasikan dalam bentuk biner. Ketelitian proses kuantisasi ditentukan oleh parameter *bit depth*, yaitu jumlah *bit* yang digunakan untuk merepresentasikan nilai amplitudo setiap sampel. Peningkatan *bit depth* sejalan dengan bertambahnya level amplitudo, yang memungkinkan rentang dinamis sinyal direpresentasikan secara lebih presisi [27], [36]. Hubungan antara jumlah tingkatan kuantisasi (L) dengan *bit depth* (n) dinyatakan dalam persamaan (2).

$$L = 2^n \dots\dots\dots(2)$$

Keterangan:

- L = Jumlah level kuantisasi
- n = *Bit depth*
- 2 = Bilangan basis sistem biner.

2.3 Deepfake

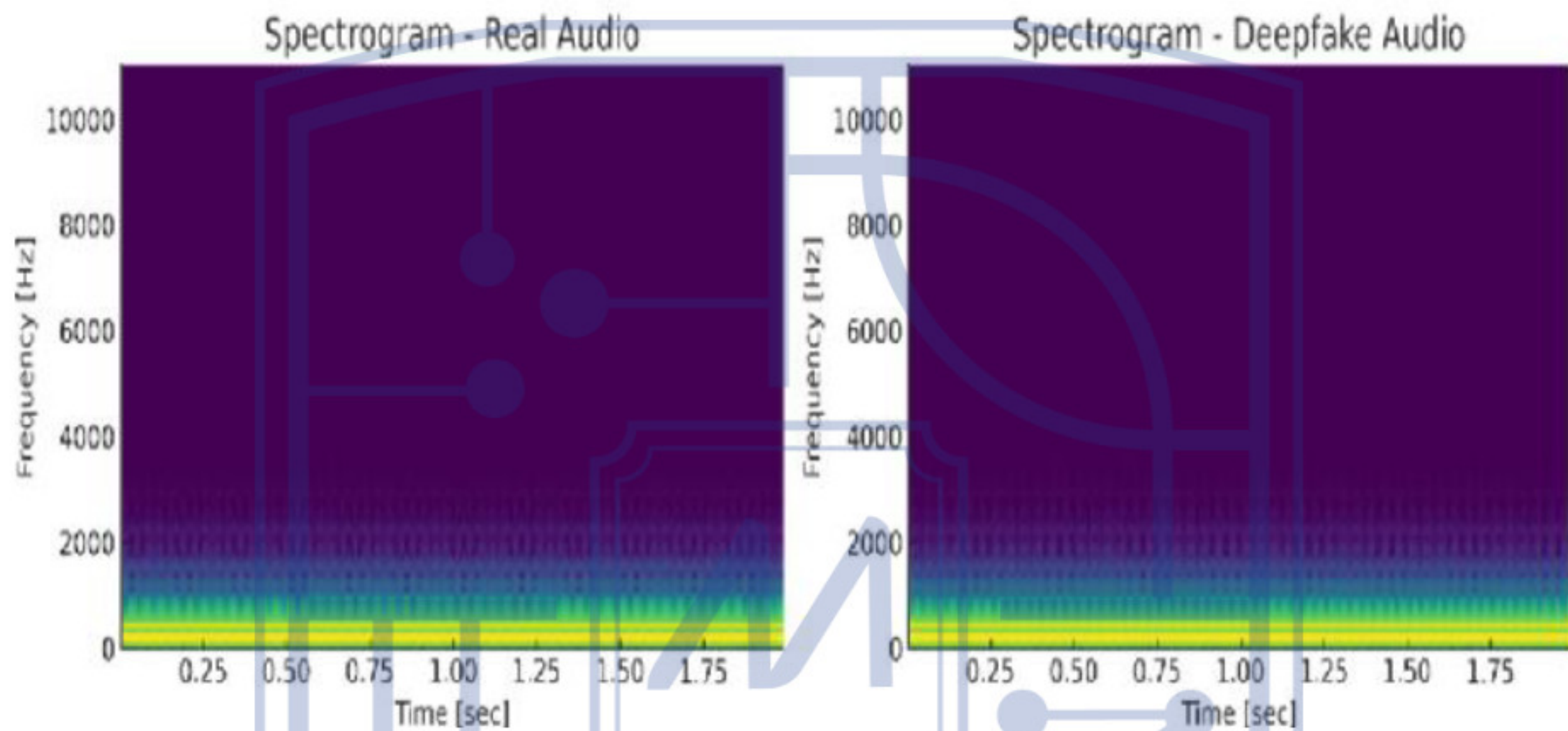
Istilah *deepfake* merupakan penggabungan dari kata *deep learning* dan *fake*. *Deepfake* merujuk pada media sintetis berupa video, gambar, maupun audio yang dimanipulasi atau dihasilkan menggunakan model AI sehingga menyerupai media asli. Perkembangan *deepfake* didukung oleh arsitektur *deep learning*, khususnya *generative modeling*, yang mampu mereplikasi karakteristik biometrik suara manusia dengan presisi tinggi [1], [37].

2.3.1 Definisi dan Kategori Deepfake

Berdasarkan cara kerjanya, teknologi *deepfake* dapat dibedakan menjadi dua pendekatan utama. Pendekatan pertama adalah modifikasi media yang sudah ada, seperti mengubah ekspresi wajah atau kata-kata yang diucapkan oleh penutur asli (*Voice Conversion*). Pendekatan kedua adalah sintesis generatif penuh, yaitu menciptakan media buatan dari awal tanpa mengandalkan rekaman asli, melainkan menggunakan teks sebagai masukan melalui teknologi *Text-to-Speech* (TTS) untuk menghasilkan suara sintetis[2], [38].

Pada awalnya, manipulasi *deepfake* didominasi oleh domain visual, menggunakan arsitektur seperti *Generative Adversarial Networks* (GAN) untuk menukar wajah seseorang dalam sebuah video. Namun, seiring kemajuan algoritma pemrosesan sinyal digital, fokus

manipulasi mulai beralih ke domain audio. *Deepfake* audio kini mampu mereplikasi frekuensi dasar (*pitch*), warna suara (*timbre*), hingga ritme bicara seseorang hanya dengan sampel suara berdurasi beberapa detik [6], [38], [39]. Meskipun suara *deepfake* sulit dibedakan dari suara asli oleh pendengaran manusia, proses sintesisnya masih meninggalkan artefak halus pada spektrogram dan *Mel-Frequency Cepstral Coefficients* (MFCC). Perbedaan artefak spektral antara audio asli dan audio *deepfake* dapat diamati melalui perbandingan spektrogram, sebagaimana ditunjukkan pada Gambar 2.1 [40].



Gambar 2.1 Perbandingan Spektrogram Audio Asli dan Audio *Deepfake* [40]

Berdasarkan Gambar 2.1, spektrogram audio asli pada bagian kiri menampilkan distribusi energi yang terkonsentrasi secara alami pada rentang frekuensi rendah (0–2.000 Hz), dengan transisi antarfrekuensi yang halus yang mencerminkan karakteristik prosodi manusia. Sebaliknya, spektrogram audio *deepfake* pada bagian kanan memperlihatkan pola distribusi energi yang lebih seragam dan kurang alami [40].

Suara asli umumnya memiliki pola harmonik dan distribusi energi yang lebih alami, sedangkan audio *deepfake* cenderung menunjukkan pola spektral yang tidak konsisten akibat keterbatasan algoritma generatif dalam mereplikasi resonansi vokal manusia. Identifikasi anomali pada spektrum frekuensi tersebut menjadi salah satu dasar dalam sistem keaslian audio [38], [39].

2.3.2 Voice Cloning

Voice cloning merupakan salah satu teknik utama dalam menghasilkan audio *deepfake*, yang melatih model *deep learning* untuk mempelajari serta mereplikasi karakteristik vokal seseorang. Berbeda dengan sistem *Text-to-Speech* (TTS) konvensional

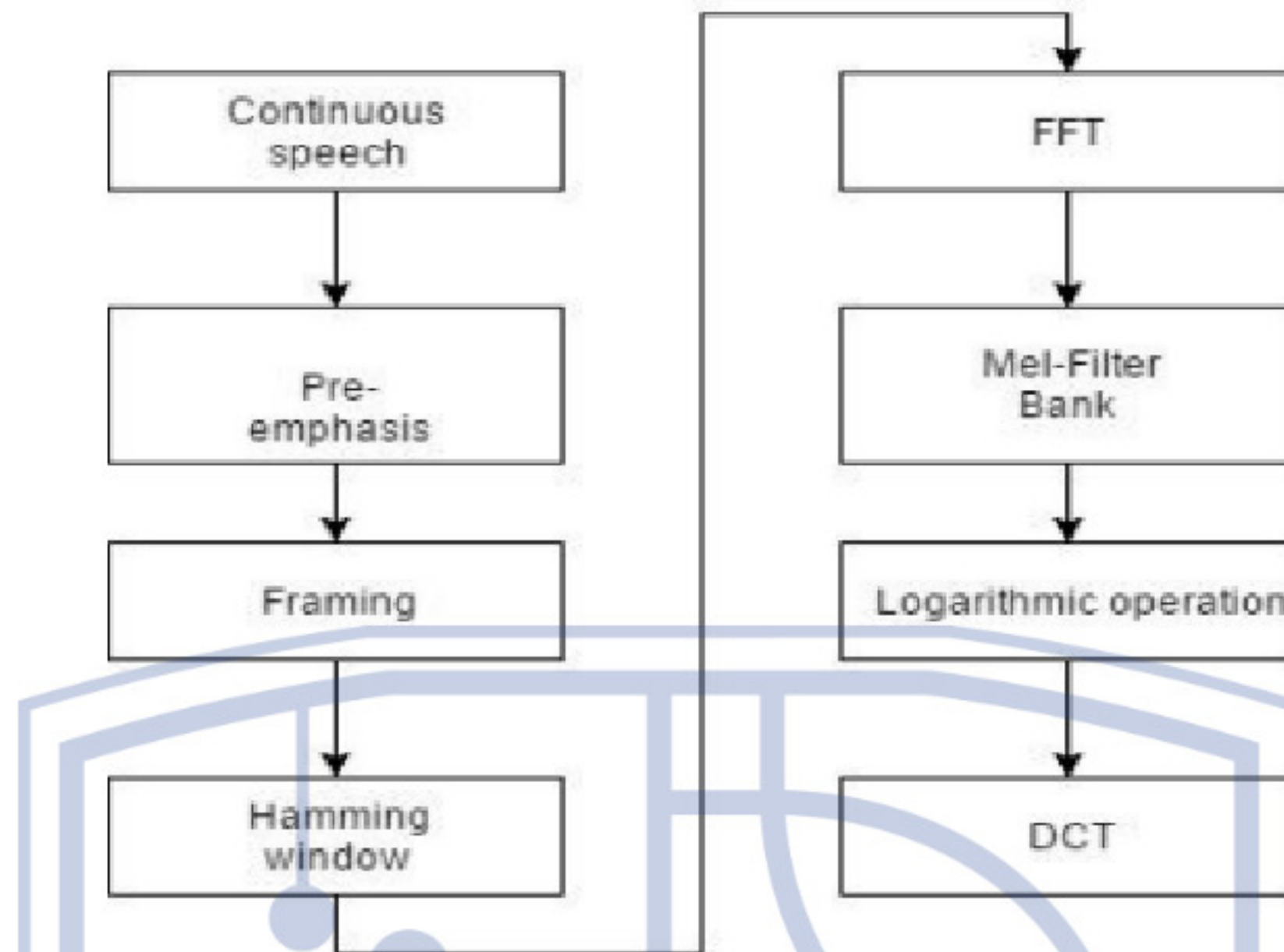
yang menghasilkan suara dengan karakteristik lebih terbatas, model *voice cloning* modern mampu mereplikasi karakteristik vokal penutur dari seorang penutur, termasuk dialek, aksen, intonasi emosional, dan ritme bicara [2], [3]. Pada awalnya, proses *voice cloning* membutuhkan rekaman audio target berdurasi berjam-jam di studio profesional. Namun, perkembangan arsitektur jaringan saraf dan penerapan *few-shot learning*, yaitu metode yang memungkinkan model mempelajari karakteristik suara penutur hanya dari sampel audio berdurasi kurang dari lima detik. Sampel tersebut dapat diperoleh dari berbagai sumber seperti media sosial, wawancara publik, atau panggilan telepon singkat [38].

2.4 Segmentasi dan Pemotongan

Pada proses klasifikasi audio, bagian keheningan (*non-speech*) pada rekaman perlu diproses terlebih dahulu. Rekaman suara manusia asli umumnya diawali dengan jeda keheningan sesaat sebelum penutur mulai berbicara, sedangkan audio hasil sintesis *Text-to-Speech* (TTS) sering kali tidak memiliki jeda keheningan awal. Perbedaan karakteristik pada bagian awal rekaman dapat memengaruhi proses pembelajaran model. Oleh karena itu, bagian keheningan yang tidak mengandung informasi tuturan dihilangkan melalui proses *trimming*. Dengan menghilangkan bagian ini, model diarahkan untuk belajar dari segmen wicara yang informatif [41], [39]. Setelah bagian keheningan dihilangkan, tahapan berikutnya adalah segmentasi, yaitu pembagian sinyal audio menjadi potongan-potongan berdurasi seragam. Segmentasi bertujuan menghasilkan ukuran *input* yang konsisten serta meningkatkan jumlah sampel pelatihan melalui pembagian setiap rekaman menjadi beberapa segmen. Panjang segmen dipilih agar mampu mempertahankan informasi temporal yang diperlukan untuk menangkap karakteristik suara [15], [19].

2.5 Mel-Frequency Cepstral Coefficients (MFCC)

Mel-Frequency Cepstral Coefficients (MFCC) merupakan salah satu metode ekstraksi fitur yang paling mendasar dan menjadi standar utama dalam bidang pengolahan sinyal wicara (*speech processing*). Fungsi utamanya adalah mengonversi gelombang suara mentah berdimensi tinggi menjadi representasi fitur berdimensi lebih rendah dengan tetap mempertahankan informasi akustik yang relevan [43], [44]. Alur keseluruhan dari proses ekstraksi fitur ini dapat dilihat melalui diagram blok yang disajikan pada Gambar 2.2 [45].



Gambar 2.2 Diagram Alur Tahapan Ekstraksi Fitur MFCC [45]

Berdasarkan Gambar 2.2, sistem ekstraksi MFCC menerima *input* berupa sinyal audio mentah yang kemudian diproses secara sekuensial melalui dua tahapan utama, yaitu pra-pemrosesan sinyal dan transformasi inti. [43].

2.5.1 Prinsip Kerja dan Skala Mel

Prinsip kerja MFCC mengacu pada mekanisme sistem pendengaran manusia. Koklea pada telinga manusia tidak merespons frekuensi suara secara linear. Sistem pendengaran lebih peka terhadap perubahan nada pada frekuensi rendah (di bawah 1.000 Hz) dibandingkan pada frekuensi tinggi [43]. Untuk memodelkan fenomena tersebut, MFCC memetakan spektrum frekuensi linear ke dalam Skala Mel agar sesuai dengan persepsi pendengaran manusia [43], [44]. Hubungan matematis antara frekuensi linear f (Hz) dan Skala Mel m dinyatakan pada persamaan (3).

$$m = 2595 \log_{10} \left(1 + \frac{f}{700} \right) \dots \dots \dots (3)$$

Keterangan:

- m = Nilai frekuensi dalam Skala Mel.
 f = Nilai frekuensi aktual sinyal dalam satuan Hertz (Hz).

2.5.2 Tahapan Pra-pemrosesan Sinyal

Proses ekstraksi MFCC diawali dengan tiga tahap pra-pemrosesan sinyal yang bertujuan mempersiapkan sinyal audio sebelum memasuki proses transformasi fitur.

1. *Pre-emphasis*

Tahap ini menerapkan *high-pass* filter pada sinyal audio untuk menguatkan amplitudo pada frekuensi tinggi. Langkah ini menyeimbangkan spektrum sinyal suara manusia yang secara alami mengalami redaman di pita frekuensi tinggi [43], [44]. Operasi *pre-emphasis* dirumuskan secara matematis sebagai filter orde pertama pada persamaan (4).

$$y[n] = x[n] - \alpha x[n - 1] \dots\dots\dots(4)$$

Keterangan:

$y[n]$ = Sinyal *output* setelah *pre-emphasis*.
 $x[n]$ = Sinyal *input* pada waktu ke- n
 $x[n - 1]$ = Sampel *input* sebelumnya.
 α = Koefisien filter (0,95–0,97)

2. *Framing*

Framing merupakan proses membagi sinyal audio menjadi frame-frame berdurasi pendek, umumnya 20-30 ms [27]. Pemotongan ini dilakukan karena suara manusia bersifat non-stasioner. Namun, dalam rentang waktu yang sangat sempit tersebut, organ vokal belum sempat mengubah bentuk artikulasinya sehingga sinyal dapat diasumsikan bersifat stasioner [40], [43]. Jumlah sampel yang tercakup dalam satu *frame* dan jarak antar-*frame* (*hop length*) ditentukan berdasarkan durasi *frame* dan *hop* yang ditetapkan, dikalikan dengan persamaan (5) dan persamaan (6).

$$N_{frame} = T_{frame} \times f_s \dots\dots\dots(5)$$

$$N_{hop} = T_{hop} \times f_s \dots\dots\dots(6)$$

Keterangan:

N_{frame} = jumlah sampel dalam satu *frame*.
 N_{hop} = jarak antar-*frame* dalam satuan sampel.
 T_{frame} = durasi *frame*.
 T_{hop} = durasi *hop*.
 f_s = frekuensi *sampling* (Hz).

3. *Windowing*

Pemotongan pada tahap *framing* sering memicu fenomena *spectral leakage* pada ujung-ujung *frame* akibat diskontinuitas sinyal. Tahap *windowing* mengalikan setiap frame

dengan *window function* sehingga amplitudo pada kedua ujung frame mendekati nol [43]. Fungsi yang paling umum digunakan adalah *Hamming Window*, dengan persamaan (6).

$$w[n] = 0,54 - 0,46 \cos\left(\frac{2\pi n}{N-1}\right) \dots\dots\dots (6)$$

Keterangan:

$w[n]$ = Nilai window pada indeks ke- n .

N = jumlah sample dalam satu window

2.5.3 Tahapan Transformasi Inti

Setelah melewati blok pra-pemrosesan, sinyal memasuki fase transformasi inti yang mengalir melalui empat operasi perhitungan untuk membentuk matriks fitur.

1. *Fast Fourier Transform* (FFT)

Algoritma FFT diterapkan untuk mengonversi setiap *frame* sinyal dari domain waktu menjadi domain frekuensi. FFT merupakan algoritma yang menghitung *Discrete Fourier Transform* (DFT) secara efisien sehingga diperoleh spektrum frekuensi dari setiap frame sinyal, yang dirumuskan pada persamaan (7) [43].

$$X[k] = \sum_{n=0}^{N-1} y[n] e^{-j \frac{2\pi kn}{N}} \dots\dots\dots (7)$$

Keterangan:

$X[k]$ = Spektrum frekuensi pada indeks ke- k .

$y[n]$ = Sinyal *input* hasil windowing.

j = Bilangan imajiner

2. Penerapan Mel-Filterbank

Spektrum magnitudo dari tahap FFT dilewatkan ke dalam sekumpulan filter segitiga (*filterbank*) yang disusun berdasarkan Skala Mel. Pemetaan ini bertujuan untuk merepresentasikan energi spektral sesuai dengan karakteristik persepsi pendengaran manusia [43]. Hubungan matematis untuk mengonversi frekuensi linear f (dalam Hertz) menjadi frekuensi Skala Mel m dinyatakan dalam persamaan (3).

3. *Transformasi Logaritmik*

Setelah melewati *filterbank*, nilai logaritma diterapkan pada energi keluaran setiap filter. Proses kompresi ini meniru persepsi pendengaran manusia dalam merespons

kenyaringan suara (*loudness*) secara non-linear. Energi logaritmik pada filter ke- m dihitung dengan rumus persamaan (8).

$$S[m] = \log_{10}(\sum_{k=0}^{N-1} |X[k]|^2 H_m[k]) \dots\dots\dots(8)$$

Keterangan:

$S[m]$ = Energi spektral logaritmik pada filter Skala Mel ke- m .

$|X[k]|^2$ = Spektrum daya atau magnitudo kuadrat dari *output* FFT.

$H_m[k]$ = Fungsi respons dari filter segitiga ke- m .

4. Discrete Cosine Transform (DCT)

Tahap akhir melibatkan operasi DCT untuk mengonversi data dari domain log-Mel ke dalam domain spektral baru yang disebut domain *cepstral*. DCT melakukan dekorelasi koefisien hasil Mel-Filterbank. Perhitungan koefisien akhir MFCC dirumuskan melalui persamaan (9) [43], [44].

$$C_n = \sum_{m=1}^M S[m] \cos\left(\frac{\pi n(m-0,5)}{M}\right) \dots\dots\dots(9)$$

Keterangan:

C_n = Koefisien MFCC ke- m

M = Jumlah filter Mel-Filterbank

$S[m]$ = Energi logaritmik filter ke- m

Hasil akhir dari rangkaian proses transformasi ini adalah sekumpulan koefisien numerik. Vektor-vektor koefisien tersebut kemudian disusun menjadi matriks fitur spektral temporal berdimensi $F \times T$ (dengan F mewakili jumlah koefisien fitur frekuensi dan T mewakili jumlah *frame* waktu). Matriks ini secara menyeluruh merangkum identitas akustik dari sinyal suara, dan selanjutnya digunakan sebagai *input* bagi arsitektur *deep learning* [46], [40].

2.5.4 Normalisasi Fitur

Normalisasi fitur adalah teknik pra-pemrosesan data yang bertujuan menyamakan skala antar fitur, sehingga atribut dengan rentang nilai besar tidak mendominasi atribut dengan rentang nilai kecil pada saat proses pembelajaran model [47]. Terdapat beberapa metode yang umum digunakan untuk melakukan normalisasi fitur, salah satunya adalah *StandardScaler* yang dikenal juga dengan istilah *Z-score normalization* [47]. Metode ini bekerja dengan cara menstandarkan data sehingga memiliki nilai rata-rata (*mean*) sebesar 0

dan standar deviasi (*standard deviation*) sebesar 1, dengan menyesuaikan setiap nilai data berdasarkan seberapa jauh data tersebut menyimpang dari nilai rata-ratanya tanpa mengubah bentuk distribusi aslinya [47]. Pertimbangan inilah yang membuat StandardScaler banyak diterapkan sebagai tahap standar pra-pemrosesan sebelum data dilatih pada arsitektur *deep learning*, karena fitur dengan magnitudo lebih besar berpotensi mendominasi proses pembaruan bobot dan memperlambat proses apabila tidak dinormalisasi [48]. Secara matematis, perhitungan StandardScaler dinyatakan pada Persamaan (10).

$$z = \frac{x - \mu}{\sigma} \dots\dots\dots(10)$$

Keterangan:

z = Nilai fitur setelah dinormalisasi (*Z-score*).

x = Nilai fitur asli sebelum dinormalisasi.

μ = Nilai rata-rata (*mean*) dari fitur pada seluruh sampel data.

σ = Standar deviasi (*standard deviation*) dari fitur tersebut.

2.6 Deep Learning

Deep learning merupakan subbidang dari *Machine Learning* yang menggunakan sistem berlapis seperti jaringan saraf manusia untuk mengekstraksi pola tingkat tinggi dari data mentah secara otomatis. Berbeda dengan algoritma *Machine Learning* tradisional yang memerlukan rekayasa fitur manual, *deep learning* memiliki kemampuan hierarkis untuk mempelajari pola data dari tingkat yang paling dasar hingga abstrak. Kemampuan ini menjadikannya unggul dalam memproses data kompleks berdimensi tinggi, seperti analisis citra spektrogram dan matriks fitur suara [49], [50].

2.6.1 Artificial Neural Network (ANN)

Dasar *deep learning* adalah Artificial Neural Network yang memproses aliran data melalui penyesuaian bobot (*weight*) dan bias secara iteratif [51]. Setiap unit neuron dalam jaringan menghitung jumlah terbobot dari *input* yang diterima sebelum dilewatkan ke *Activation Function*. Operasi dasar pada satu unit neuron dirumuskan pada persamaan (12).

$$y = f(\sum_{i=1}^n w_i x_i + b) \dots\dots\dots(12)$$

Keterangan:

y = *Output* neuron.

f = Fungsi aktivasi.

w_i = Bobot *input* ke- i .

x_i = Nilai *input* ke-i.
 b = Bias

2.6.2 Fungsi Aktivasi

Fungsi aktivasi merupakan komponen yang memperkenalkan sifat non-linearitas ke dalam jaringan saraf tiruan. Tanpa fungsi aktivasi, jaringan saraf hanya akan berfungsi sebagai model transformasi linear yang sangat terbatas dalam mempelajari hubungan data yang kompleks [52]. Dua fungsi aktivasi yang digunakan dalam arsitektur penelitian ini adalah *Rectified Linear Unit* (ReLU) dan Sigmoid, yang masing-masing memiliki peran spesifik pada *layer* yang berbeda [53].

1. Rectified Linear Unit (ReLU)

ReLU merupakan fungsi aktivasi non-linear yang paling umum digunakan pada *hidden layer*. Fungsi ini menjaga efisiensi perhitungan dan mencegah masalah *vanishing gradient* selama proses pembaruan bobot. Mekanisme kerjanya sederhana namun efektif, yaitu melewati nilai *input* positif secara langsung dan mengubah seluruh nilai *input* negatif menjadi nol. Operasi ini dinyatakan pada persamaan (13) [53].

$$f(x) = \max(0, x) \dots \dots \dots (13)$$

Keterangan:

$f(x)$ = Nilai *output* dari fungsi aktivasi ReLU.
 x = *input*.
 $\max(0, x)$ = nilai maksimal antara 0 dan x

2. Sigmoid

Fungsi *Sigmoid* adalah fungsi aktivasi dengan kurva menyerupai huruf 'S' yang dirancang untuk *layer* klasifikasi akhir (*output layer*). Fungsi ini memetakan nilai perhitungan bilangan riil ke dalam rentang probabilitas 0 hingga 1. Transformasi ini mengubah *output* jaringan saraf menjadi nilai probabilitas, sehingga sangat sesuai untuk tugas klasifikasi biner. Fungsi Sigmoid dinyatakan pada persamaan (14) [53].

$$f(x) = \frac{1}{1+e^{-x}} \dots \dots \dots (14)$$

Keterangan:

$f(x)$ = Probabilitas *output* 0-1.
 e = Bilangan Euler atau konstanta

x = Nilai *input* dari *layer* neuron sebelumnya.

2.6.3 Optimization Algorithm

Optimizer berfungsi meminimalkan nilai *loss* dengan memperbarui bobot model berdasarkan gradien. Adam (*Adaptive Moment Estimation*) merupakan *optimization algorithm* yang menggabungkan keunggulan metode *Root Mean Square Propagation* (RMSProp) dan Momentum untuk melakukan pembaruan bobot secara adaptif [54]. Algoritma ini menghitung *learning rate* yang adaptif dengan mempertimbangkan estimasi momen pertama (m_t) dan momen kedua (v_t) dari gradien. Prinsip pembaruan parameter θ pada waktu ke- t dinyatakan dalam persamaan (15).

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t \dots \dots \dots (15)$$

Keterangan:

- θ_t = Bobot model setelah pembaruan.
- η = *Learning Rate*.
- $\hat{m}_t$ = Estimasi momen pertama terkoreksi.
- $\hat{v}_t$ = Estimasi momen kedua terkoreksi.
- ϵ = Konstanta stabilitas numerik.

2.6.4 Binary Cross-Entropy

Binary Cross-Entropy (BCE), yang juga dikenal sebagai *Log Loss*, *loss function* yang paling umum digunakan dalam permasalahan klasifikasi biner, yaitu ketika target *output* hanya memiliki dua kemungkinan kelas, seperti audio asli (*real*) atau audio palsu (*deepfake*). Fungsi ini mengukur perbedaan antara dua distribusi probabilitas, yaitu label aktual (nilai target) dan probabilitas prediksi yang dihasilkan oleh model melalui *sigmoid activation function* [55]. Secara matematis, *Binary Cross-Entropy* didefinisikan melalui persamaan (16).

$$L = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \dots \dots \dots (16)$$

Keterangan:

- L = Nilai loss total.
- N = Jumlah sampel perbatch.
- y_i = Label aktual (0 = Asli, 1 = Palsu).
- $\hat{y}_i$ = Nilai probabilitas prediksi model.

Nilai *loss* meningkat ketika probabilitas prediksi semakin jauh dari label aktual. Jika probabilitas prediksi mendekati nilai label asli, maka nilai *loss* akan mendekati nol. Sebaliknya, semakin besar perbedaan antara probabilitas prediksi dan label aktual, semakin besar nilai *loss* yang dihasilkan. Mekanisme ini berfungsi untuk memberikan sinyal gradien kepada *optimization algorithm* untuk memperbarui bobot jaringan secara iteratif hingga mencapai konvergensi yang optimal [55].

2.6.5 Dropout

Teknik regularisasi merupakan strategi dalam pengembangan yang bertujuan meningkatkan kemampuan generalisasi model dan mengurangi risiko *overfitting*. Salah satu metode yang paling luas digunakan adalah *Dropout*, yang menonaktifkan unit neuron tertentu secara acak dengan probabilitas p yang telah ditentukan selama fase pelatihan [49]. Mekanisme ini mendorong jaringan untuk mempelajari representasi fitur yang lebih beragam, karena jaringan tidak dapat bergantung pada keberadaan neuron tertentu, sehingga meningkatkan ketangguhan model saat diuji pada data yang belum pernah dilihat sebelumnya. Secara matematis, operasi ini dinyatakan pada persamaan (17).

$$y = r \cdot f(x) \dots\dots\dots(17)$$

Keterangan:

- y = Vektor *output*.
- r = Vektor masker biner acak pada y .
- $f(x)$ = Fungsi aktivasi pada x .

2.6.6 Hyperparameter

hyperparameter adalah variabel konfigurasi yang nilainya harus ditetapkan secara manual oleh peneliti sebelum proses pelatihan dimulai [56]. Sebaliknya, parameter model seperti bobot (*weight*) dan *bias* dipelajari serta diperbarui oleh algoritma berdasarkan pola pada data latih. Pemilihan nilai *hyperparameter* yang tepat memengaruhi cara model belajar, sehingga ikut menentukan performa sistem yang dibangun [32], [56].

Beberapa *hyperparameter* fundamental yang umum digunakan dalam pelatihan adalah berikut:

- a. *Epoch* menyatakan satu siklus pelatihan penuh, yaitu saat seluruh data latih telah melewati proses maju (*forward pass*) dan mundur (*backward pass*) pada jaringan saraf. Semakin banyak epoch, semakin sering model melakukan proses pembaruan parameter berdasarkan data latih. Jumlah *epoch* yang terlalu sedikit dapat menyebabkan

underfitting karena model belum cukup belajar, sedangkan jumlah yang terlalu banyak berisiko menyebabkan *overfitting* [56].

- b. *Batch size* menentukan jumlah sampel data yang diproses bersamaan dalam satu iterasi sebelum bobot model diperbarui. Ukuran *batch* yang besar menghasilkan estimasi gradien yang lebih stabil, tetapi membutuhkan memori komputasi lebih tinggi. Sebaliknya, ukuran *batch* yang kecil cenderung memberi efek regularisasi yang membantu model menghindari *local minima* [57].
- c. *Learning rate* merupakan *hyperparameter* yang paling berpengaruh karena mengontrol besar langkah pembaruan bobot model pada setiap iterasi. Nilai *learning rate* yang terlalu besar dapat membuat proses optimasi tidak stabil karena model melompati titik optimalnya, sedangkan nilai yang terlalu kecil membuat proses belajar berjalan lambat dan memakan waktu komputasi lebih lama [32].

2.6.7 Early Stopping

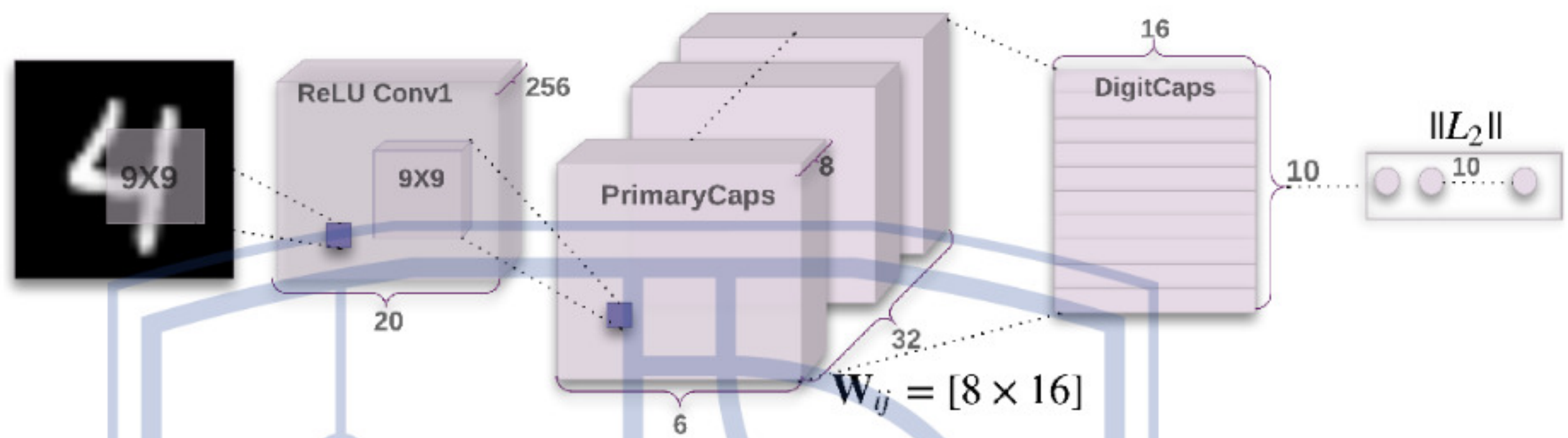
Early stopping merupakan salah satu teknik *regularization* yang digunakan secara luas untuk mencegah terjadinya *overfitting* selama proses pelatihan [58]. *Overfitting* terjadi ketika model terlalu menghafal detail dan *noise* dari data latih, sehingga kehilangan kemampuannya untuk menggeneralisasi atau mengenali data baru yang belum pernah dilihat sebelumnya [59].

Teknik ini bekerja dengan cara memantau metrik performa model, seperti nilai *validation loss* pada himpunan data validasi di setiap akhir iterasi atau *epoch*. Proses pelatihan dihentikan apabila nilai *validation loss* tidak menunjukkan penurunan selama sejumlah *epoch* yang ditentukan oleh parameter *patience* [58], [59]. Ketika pelatihan dihentikan secara otomatis, *model weights* akan dikembalikan ke kondisi terbaik yang tercatat sebelum performa validasinya mulai memburuk [59]. Oleh karena itu, bobot model yang digunakan merupakan bobot dengan performa validasi terbaik sebelum terjadi *overfitting*.

2.7 Capsule Network

Capsule Network (CapsNet) merupakan arsitektur *deep learning* yang dikembangkan untuk mengatasi keterbatasan pada arsitektur konvensional, khususnya *Convolutional Neural Network* (CNN). Masalah utama pada CNN terletak pada operasi *max-pooling* yang mereduksi dimensi data demi efisiensi perhitungan, namun secara bersamaan menghilangkan informasi hierarkis dan hubungan spektral antarfitur di dalam sebuah

matriks (seperti citra spektrogram). CapsNet menggunakan representasi berbasis kapsul sebagai pengganti neuron berbasis skalar menjadi struktur kapsul berbasis vektor [60], [61]. Arsitektur CapsNet yang diperkenalkan oleh Sabour et al. ditunjukkan pada Gambar 2.3 [62].



Gambar 2.3 Arsitektur *Capsule Network* [62]

Berdasarkan Gambar 2.3, arsitektur CapsNet terdiri atas tiga komponen utama yang tersusun secara hierarkis. Komponen pertama Adalah *Conv1* yang mengekstrak fitur tingkat rendah dari data *input* melalui operasi konvolusi dengan fungsi aktivasi ReLU. Komponen kedua adalah *PrimaryCaps* yang mengubah *feature map* hasil konvolusi menjadi kumpulan vektor kapsul, dengan setiap kapsul menggambarkan keberadaan sekaligus sifat spektral suatu fitur. Komponen ketiga adalah *DigitCaps* yang menerima vektor kapsul dari *PrimaryCaps* melalui mekanisme *dynamic routing* dan menghasilkan vektor kapsul untuk setiap kelas. Panjang vektor pada *layer DigitCaps* dihitung menggunakan norma L_2 yang menunjukkan probabilitas keberadaan fitur dari kelas yang bersangkutan [52]. Secara matematis, panjang atau norma Euclidean dari sebuah vektor kapsul v berdimensi n dihitung melalui akar kuadrat dari jumlah kuadrat pada persamaan (18).

$$\|v\| = \sqrt{\sum_{k=1}^n v_k^2} \dots \dots \dots (18)$$

Keterangan:

$\|v\|$ = Panjang vektor

v_k = Elemen vektor.

n = Jumlah dimensi vektor.

2.7.1 Convolutional Layer

Convolutional layer dalam arsitektur *Capsule Network* berfungsi sebagai tahap pra-pemrosesan fitur yang bertugas mengekstraksi fitur tingkat rendah (*low-level features*) dari

data *input* [13]. *Layer* ini melakukan pemindaian terhadap matriks MFCC untuk mendeteksi pola dasar seperti tepi, tekstur frekuensi, dan perubahan intensitas spektral [39].

Proses pada *layer* ini menggunakan sejumlah filter atau *kernel* yang bergerak di atas data *input* melalui operasi konvolusi. Secara matematis, operasi konvolusi dua dimensi didefinisikan pada persamaan (19).

$$y(i, j) = (x * w)(i, j) = \sum_m \sum_n x(i + m, j + n) \cdot w(m, n) \dots \dots \dots (19)$$

Keterangan:

$y(i, j)$ = Nilai pada peta fitur (*feature map*).

x = Citra atau matriks *input*.

w = Filter atau kernel konvolusi.

m, n = Dimensi dari filter konvolusi.

Hasil operasi ini adalah kumpulan peta fitur yang kemudian diberi fungsi aktivasi non-linear, umumnya *Rectified Linear Unit* (ReLU). Berbeda dengan CNN tradisional yang menggunakan *layer pooling* untuk mereduksi dimensi, pada CapsNet *output convolutional layer* ini langsung diteruskan ke unit *Primary Capsules* untuk diubah menjadi vektor yang mempertahankan informasi spasial [50], [51].

2.7.2 Primary Capsule

Primary Capsule merupakan *layer* pertama yang menghasilkan vektor kapsul dari *feature map* hasil *Convolutional Layer*. Berbeda dengan neuron biasa yang menghasilkan nilai skalar tunggal, setiap kapsul menghasilkan vektor yang secara bersamaan menyimpan probabilitas keberadaan fitur sekaligus properti spasialnya, seperti posisi dan orientasi [13], [61]. *Primary Capsule* menerapkan operasi konvolusi terhadap *feature map input* untuk membentuk kumpulan vektor kapsul berdimensi 8, kemudian dilakukan proses *reshape* sehingga membentuk tensor yang siap diteruskan ke *layer DigitCaps* melalui mekanisme *dynamic routing* [62].

Setiap vektor kapsul merepresentasikan satu entitas fitur pada posisi tertentu dalam *feature map*. Panjang vektor mencerminkan kemungkinan kehadiran fitur tersebut, sedangkan arah vektor menyimpan informasi tentang tampilannya, termasuk skala, rotasi, dan hubungan spasialnya terhadap fitur lain [13], [61], [62]. Karakteristik tersebut memungkinkan *Primary Capsule* mempertahankan informasi spasial melalui representasi berbasis vektor ketika diteruskan ke *layer* berikutnya.

2.7.3 Dynamic Routing

Sebagai pengganti operasi *pooling* statis, CapsNet menggunakan mekanisme *dynamic routing* yang didasarkan pada algoritma *Routing-by-Agreement*. Secara konseptual, mekanisme ini beroperasi berdasarkan prinsip probabilitas hierarkis, yang mengasumsikan bahwa entitas bagian (fitur lokal) harus memiliki kesepakatan orientasi dengan entitas keseluruhan (fitur global) [13], [62]. Pada mekanisme *dynamic routing*, sebuah kapsul pada tingkat hierarki yang lebih rendah menghasilkan vektor prediksi untuk kapsul pada tingkat yang lebih tinggi melalui tahapan perhitungan pada persamaan (20), (21), (22) dan (23) [13], [62].

1. Prediction Vector

Setiap kapsul di *layer* bawah memprediksi *output* kapsul di *layer* atas dengan mengalikan *output*-nya (u_i) dengan matriks bobot transformasi (W_{ij}):

$$\hat{u}_{j|i} = W_{ij}u_i \dots\dots\dots(20)$$

Keterangan:

- $\hat{u}_{j|i}$ = Vektor prediksi.
- W_{ij} = Matriks bobot transformasi.
- u_i = Vektor *output*.

2. Routing Coefficient

Distribusi informasi dari kapsul tingkat rendah ke kapsul tingkat tinggi ditentukan oleh *coupling coefficients* yang dihitung menggunakan fungsi *softmax*:

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})} \dots\dots\dots(21)$$

Keterangan:

- c_{ij} = Koefisien perutean.
- b_{ij} = Nilai yang menyatakan probabilitas.
- K = Indeks yang mewakili seluruh kapsul.

3. Weighted Sum

Vektor *input* total bagi kapsul tingkat tinggi dihitung sebagai penjumlahan berbobot dari seluruh vektor prediksi, dengan koefisien perutean sebagai bobotnya:

$$s_j = \sum_i c_{ij}\hat{u}_{j|i} \dots\dots\dots(22)$$

Keterangan:

s_j = Vektor *input* total bagi kapsul j.

c_{ij} = Koefisien perutean yang menghubungkan kapsul i dan kapsul j.

$\widehat{u}_{j|i}$ = Vektor prediksi dari kapsul i untuk kapsul j.

4. Agreement Update

Tingkat keselarasan diukur melalui operasi *dot product* antara vektor prediksi dan vektor *output* aktual kapsul j (v_j). Jika keduanya selaras, nilai kesepakatan akan diperbarui untuk memperkuat hubungan antarkapsul:

$$b_{ij} \leftarrow b_{ij} + \widehat{u}_{j|i} \cdot v_j \dots\dots\dots(23)$$

Keterangan:

b_{ij} = Nilai *log-prior* yang diperbarui.

$\widehat{u}_{j|i}$ = Vektor prediksi yang berasal dari kapsul i.

v_j = Vektor *output* aktual dari kapsul j

$\widehat{u}_{j|i} \cdot v_j$ = Operasi *dot product* (skalar).

2.7.4 Squashing Function

Karena arsitektur ini bekerja pada data berbentuk vektor, fungsi aktivasi skalar konvensional tidak dapat digunakan. Sebagai gantinya, CapsNet menggunakan fungsi aktivasi non-linear khusus yang bekerja pada domain vektor, yaitu *Squashing Function* [62]. *Squashing Function* memetakan panjang vektor ke rentang 0 hingga 1 sehingga nilainya dapat diinterpretasikan sebagai probabilitas keberadaan suatu fitur. Keunggulan fungsi ini adalah kemampuannya memampatkan magnitudo vektor secara proporsional tanpa mengubah arah orientasi spasial vektor aslinya [62], [61]. Persamaan fungsi tersebut dirumuskan pada Persamaan (24)

$$v_j = \left(\frac{|s_j|^2}{|s_j|(1+|s_j|^2)} \right) s_j \dots\dots\dots(24)$$

Keterangan:

v_j = Vektor *output*.

s_j = Vektor *input* total.

$||s_j||$ = Panjang (norma) dari vektor *input* s_j .

$$\frac{\|s_j\|^2}{1+\|s_j\|^2} = \text{Komponen skalar.}$$

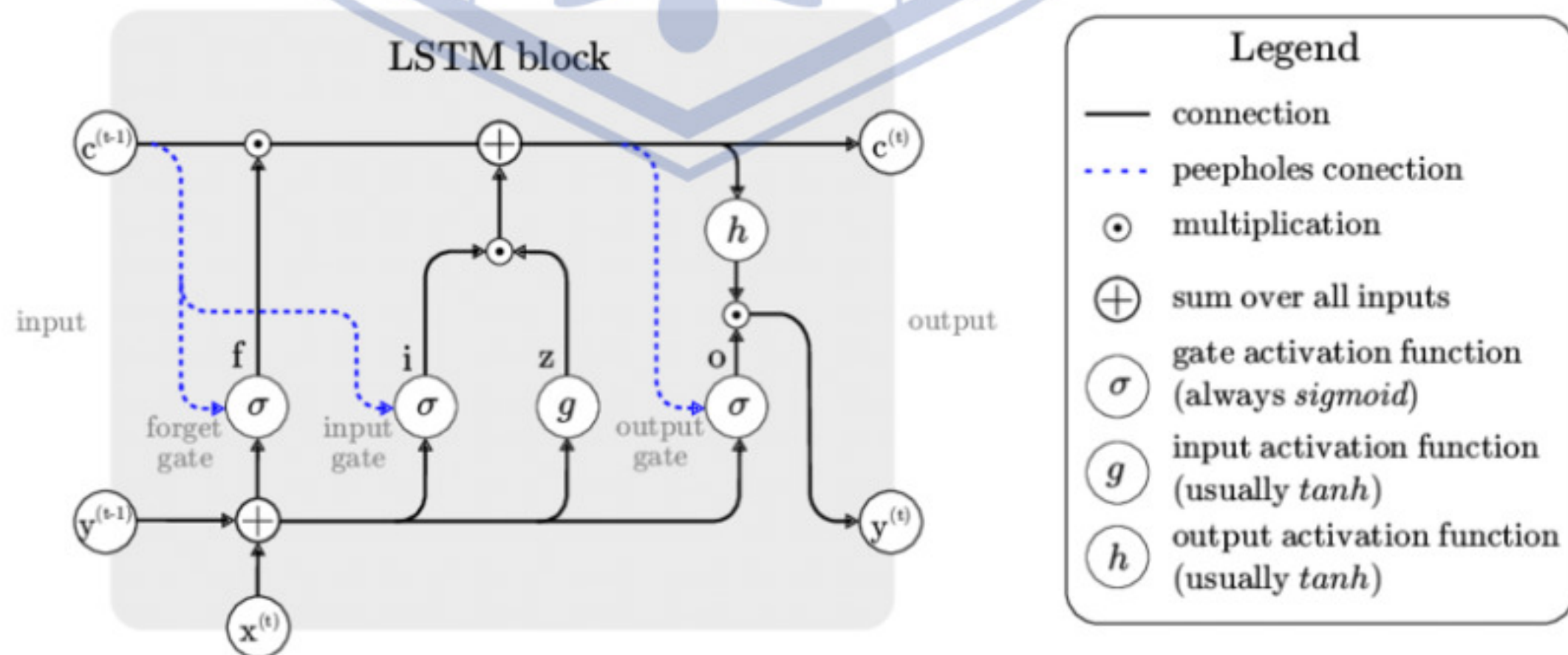
$$\frac{s_j}{\|s_j\|} = \text{Vektor satuan.}$$

Dengan mempertahankan arah vektor, model dapat menjaga informasi properti fisik suara sementara panjang vektor menunjukkan apakah pola tersebut merupakan artefak manipulasi atau karakteristik suara asli manusia [51], [52].

2.8 Long Short-Term Memory

Long Short-Term Memory (LSTM) adalah salah satu arsitektur tingkat lanjut dari *Recurrent Neural Network* (RNN) yang dirancang khusus untuk memodelkan data berurutan atau sekuensial. Berbeda dengan *feedforward neural network* konvensional yang memproses titik data secara independen, LSTM memiliki koneksi umpan balik (*feedback connections*) yang memungkinkannya memproses urutan data kompleks, seperti analisis deret waktu (*time series*), pemrosesan bahasa alami, hingga klasifikasi sinyal audio [63], [64].

Inti dari arsitektur LSTM terletak pada komponen *cell state* dan unit memorinya. *Cell state* berfungsi sebagai jalur memori utama yang melintasi seluruh rangkaian waktu. Jalur ini memungkinkan informasi dari masa lalu diteruskan ke langkah waktu berikutnya dengan sedikit perubahan linear. Konsep "memori" pada LSTM tidak dipertahankan secara pasif, melainkan diatur oleh struktur interaktif yang disebut *gate*. *Gate* ini terdiri atas layer dengan fungsi aktivasi sigmoid dan operasi perkalian elemen (*element-wise multiplication*) yang mengendalikan informasi yang dipertahankan, diperbarui, maupun dihapus dari *cell state* seiring berjalannya waktu [64], [65]. Visualisasi aliran data dan interaksi antar-*gate* di dalam sel LSTM dapat dilihat pada Gambar 2.4 [66].



Gambar 2.4 Arsitektur LSTM [66]

Berdasarkan Gambar 2.4 [66], garis horizontal utama di bagian atas menggambarkan cell state (C_t) yang mengalir lurus melintasi seluruh *timestep*. Kotak-kotak di bagian bawah menunjukkan bagaimana informasi *input* (x_t) dan memori sebelumnya (h_{t-1}) diproses melalui *sigmoid activation function* (σ) dan Tanh sebelum akhirnya memodifikasi *cell state* tersebut [67]. Implementasi pada penelitian ini menggunakan LSTM standar tanpa *peephole connection*, sesuai konfigurasi *default* TensorFlow/Keras.

1. Forget Gate

Tahap pertama di dalam LSTM adalah memutuskan informasi mana dari *cell state* sebelumnya yang tidak lagi relevan dan harus dibuang. Keputusan ini dikendalikan oleh *forget gate*. Gate ini menerima *input* dari *hidden state* waktu sebelumnya (h_{t-1}) dan data *input* pada waktu saat ini (x_t), lalu melewatinya pada *sigmoid activation function* untuk menghasilkan nilai antara 0 (buang sepenuhnya) dan 1 (simpan sepenuhnya). Secara matematis, operasi *forget gate* tersebut dirumuskan pada Persamaan (25) [63], [65].

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \dots \dots \dots (25)$$

Keterangan:

- f_t = Vektor *output* gate pada waktu t.
- σ = Sigmoid activation function.
- W_f = Matriks bobot gate.
- h_{t-1} = Hidden state sebelumnya.
- x_t = Vektor *input* pada waktu t.
- b_f = Bias gate.

2. Input Gate

Setelah membuang informasi yang tidak relevan LSTM harus memutuskan informasi baru mana yang akan disimpan untuk memperbarui unit memori. Proses ini dikelola oleh *input gate* yang menentukan nilai mana yang akan diperbarui, dikombinasikan dengan *layer tanh* yang menghasilkan vektor kandidat nilai memori baru ($\tilde{C}_t$). Informasi lama kemudian dikalikan dengan *output forget gate*, lalu ditambahkan dengan informasi baru yang telah disaring untuk menghasilkan *cell state* yang diperbarui (C_t). Persamaan ini dinyatakan dalam tiga persamaan berikut, yaitu (26), (27) dan (28) [65], [67].

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \dots \dots \dots (26)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \dots\dots\dots(27)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \dots\dots\dots(28)$$

Keterangan:

- i_t = Vektor *output* gate pada waktu t
 $\tilde{C}_t$ = Vektor kandidat nilai memori baru
 $\tanh$ = Fungsi aktivasi tangen hiperbolik
 C_t = Cell state yang diperbarui pada waktu t
 C_{t-1} = Cell state pada langkah waktu sebelumnya
 $*$ = Operasi perkalian elemen (element-wise multiplication)

3. Output Gate

Tahap terakhir adalah menentukan bagian mana dari *cell state* yang akan dikeluarkan sebagai *hidden state* untuk langkah waktu selanjutnya. Proses ini diatur oleh *output gate*. *Cell state* yang telah diperbarui (C_t) akan dilewatkan melalui fungsi *tanh* untuk menekan nilainya agar berada dalam rentang -1 hingga 1, kemudian dikalikan dengan hasil evaluasi dari *output gate*, sehingga hanya memori yang paling relevan yang akan diteruskan ke *layer* berikutnya [64], [68]. Rumus *output gate* dan penentuan *hidden state* dinyatakan pada Persamaan (26) dan (27).

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \dots\dots\dots(29)$$

$$h_t = o_t * \tanh(C_t) \dots\dots\dots(30)$$

Keterangan:

- o_t = Vektor *output* gate pada waktu t
 h_t = Hidden state *output* akhir pada waktu t
 W = Matriks bobot pada masing-masing *gate*.
 b = Nilai bias pada masing-masing *gate*.
 x_t = nilai input pada waktu t.
 h_{t-1} = *Hidden state* dari langkah waktu sebelumnya.

2.9 Arsitektur Hibrida

Arsitektur hibrida adalah penggabungan dua atau lebih model ke dalam satu sistem pemrosesan. Pendekatan ini banyak dipakai untuk mengatasi keterbatasan model tunggal. Setiap arsitektur memiliki karakteristik dan keunggulan yang berbeda dalam mengekstraksi

informasi, sehingga menggabungkannya membuat sistem dapat memakai keunggulan masing-masing secara saling melengkapi sekaligus menutupi kelemahannya [69], [70].

Pada penelitian ini, CapsNet-LSTM digabungkan melalui skema pemrosesan paralel (*parallel processing*). Berbeda dengan arsitektur berurutan, skema paralel memberikan matriks MFCC yang sama kepada kedua cabang model secara bersamaan. Pada cabang CapsNet, matriks MFCC diperlakukan sebagai representasi dua dimensi sehingga pola lokal pada sumbu frekuensi dapat dipelajari sebagai fitur spasial. Cabang kedua adalah LSTM, yang menganalisis dinamika temporal, yaitu ketergantungan urutan waktu dan pola prosodi pada deret fitur yang sama. *Output* kedua cabang tersebut kemudian digabungkan (*concatenate*) menjadi satu vektor fitur tunggal yang memuat informasi spektral dan temporal secara bersamaan. Vektor gabungan ini selanjutnya diteruskan ke *Dense Layer* (*fully connected layer*), yaitu *layer* neuron yang setiap unitnya terhubung penuh ke seluruh elemen vektor *input*, sehingga model dapat mempelajari kombinasi non-linear antara kedua jenis fitur tersebut. *Dense layer terakhir menggunakan sigmoid activation function, sedangkan Binary Cross-Entropy digunakan sebagai loss function pada proses pelatihan* [13], [70].

Dengan skema ini, arsitektur hibrida paralel melakukan dua analisis sekaligus, yaitu mengenali anomali atau artefak pada struktur spektral, dan melacak ketidakwajaran ritme atau transisi prosodi sepanjang rekaman. Secara teoretis, pendekatan paralel berpotensi meningkatkan representasi fitur dibandingkan model tunggal maupun arsitektur berurutan karena kedua cabang menerima matriks fitur yang sama tanpa melalui proses reduksi pada cabang lainnya [71], [13], [72].

2.10 Evaluasi Model

Setelah arsitektur selesai dilatih, kinerja model tersebut harus diukur menggunakan metrik evaluasi. Evaluasi digunakan untuk mengukur kemampuan model dalam mengenali anomali pada data audio baru [1], [73].

1. Confusion Matrix

Evaluasi pada tugas klasifikasi biner, seperti pemisahan antara suara manusia asli dan suara *deepfake*, umumnya didasarkan pada *Confusion Matrix*. Matriks ini menampilkan tabulasi silang antara hasil prediksi model dan label *ground truth* dari dataset. Terdapat empat komponen dasar yang menyusun matriks ini [1], [73]:

- a. *True Positive* (TP): sampel yang secara aktual merupakan kelas positif dan diprediksi dengan benar sebagai kelas positif oleh model.
- b. *True Negative* (TN): sampel yang secara aktual merupakan kelas negatif dan diprediksi dengan benar sebagai kelas negatif oleh model.
- c. *False Positive* (FP): sampel yang secara aktual merupakan kelas negatif, namun salah diprediksi sebagai kelas positif (dikenal sebagai Kesalahan Tipe I / *Type I Error*).
- d. *False Negative* (FN): sampel yang secara aktual merupakan kelas positif, namun salah diprediksi sebagai kelas negatif (dikenal sebagai Kesalahan Tipe II / *Type II Error*).

2. Accuracy

Accuracy merupakan metrik evaluasi paling dasar yang mengukur rasio total prediksi yang benar (baik positif maupun negatif) terhadap keseluruhan data yang dievaluasi. Secara matematis, nilai akurasi dirumuskan sebagai Persamaan (31) [1], [73].

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \dots\dots\dots(31)$$

Keterangan:

- Accuracy* = Nilai tingkat akurasi prediksi model secara keseluruhan.
 TP = Jumlah sampel kelas positif yang diprediksi dengan benar.
 TN = Jumlah sampel kelas negatif yang diprediksi dengan benar.
 FP = Jumlah sampel kelas negatif yang salah diprediksi sebagai kelas positif.
 FN = Jumlah sampel kelas positif yang salah diprediksi sebagai kelas negatif.

Meskipun akurasi memberikan gambaran umum mengenai performa sistem, metrik ini dapat menjadi bias dan menyesatkan apabila distribusi sampel antar-kelas pada dataset sangat tidak seimbang (*imbalanced dataset*) [1], [73].

3. Precision

Precision mengukur tingkat ketepatan model dalam memberikan prediksi positif [1], [73]. Metrik ini mengukur persentase data yang secara aktual benar-benar merupakan kelas positif, dibandingkan dengan keseluruhan data yang diklasifikasikan sebagai kelas positif oleh model. *Precision* dirumuskan pada persamaan (32).

$$Precision = \frac{TP}{TP+FP} \dots\dots\dots(32)$$

Keterangan:

- Precision = Nilai ketepatan prediksi model untuk kelas positif.
 TP = Jumlah sampel kelas positif yang diprediksi dengan benar.
 FP = Jumlah sampel kelas negatif yang salah diprediksi sebagai kelas positif.

4. Recall

Recall mengukur kemampuan model dalam menemukan seluruh sampel positif yang ada [1], [73]. Metrik ini mengukur persentase sampel kelas positif aktual yang berhasil diklasifikasikan dengan benar oleh model tanpa terlewat. *Recall* dirumuskan pada persamaan (33).

$$Recall = \frac{TP}{TP+FN} \dots\dots\dots(33)$$

Keterangan:

- Recall = Nilai model dalam menemukan keseluruhan data kelas positif.
 TP = Jumlah sampel kelas positif yang diprediksi dengan benar.
 FN = Jumlah sampel kelas positif yang salah diprediksi sebagai kelas negatif.

5. F1-Score

Deep learning sering kali menunjukkan fenomena *trade-off*, di mana upaya meningkatkan nilai *precision* menyebabkan penurunan nilai *recall*. Untuk menyeimbangkan kedua metrik tersebut, digunakan *F1-Score*. Metrik ini dihitung berdasarkan rata-rata harmonik dari presisi dan *recall*, yang meredam pengaruh nilai ekstrem dari salah satu variabel [1], [73]. Rumus perhitungannya dinyatakan pada Persamaan (34).

$$F1-Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \dots\dots\dots(34)$$

Keterangan:

- F1-Score = Nilai rata-rata yang menyeimbangkan metrik presisi dan recall.
Precision = Nilai tingkat ketepatan prediksi positif yang dihasilkan model.
Recall = Nilai sensitivitas pendeteksian positif yang dihasilkan model.

Nilai *F1-Score* yang tinggi (mendekati 1) mengindikasikan bahwa model klasifikasi memiliki kemampuan diskriminatif yang baik, ditandai oleh nilai FP dan FN yang relatif rendah [1], [73].

2.11 Shortcut Learning

Shortcut Learning adalah kondisi di mana model tidak mempelajari karakteristik mendasar dari objek yang ingin diklasifikasikan, melainkan mengandalkan *spurious correlations* atau petunjuk dangkal yang kebetulan berulang di dalam data latih [74]. Dalam klasifikasi audio *deepfake*, model yang mengalami *shortcut learning* dapat mengabaikan artefak spektral atau kejanggalan temporal yang dihasilkan algoritma sintesis, melainkan memanfaatkan pola sekunder seperti perbedaan panjang keheningan (*silence*) di awal rekaman atau karakteristik derau latar belakang (*background noise*), atau jenis format kompresi tiap kelas [75]. Smeu dkk. (2025) menunjukkan efektivitas pendekatan tersebut secara empiris pada dua dataset *deepfake* yang umum digunakan. Sebuah pengklasifikasi sederhana yang hanya mengandalkan keheningan di awal rekaman mampu mencapai performa hampir sempurna, yaitu di atas 98%, dan ketika keheningan tersebut dihilangkan performanya turun drastis hingga mendekati tebakan acak, yaitu sekitar 50% [41]. Ketergantungan pada petunjuk semacam ini membuat model tampak memiliki akurasi yang sangat tinggi saat diuji pada data serupa, tetapi performanya akan menurun tajam ketika diterapkan pada data dengan kondisi yang berbeda [41], [75].

2.12 K-Fold Cross-Validation

K-Fold Cross-Validation adalah teknik validasi yang membagi dataset menjadi K bagian (*fold*) berukuran relatif sama, kemudian melatih dan menguji model sebanyak K kali dengan kombinasi *fold* yang berbeda pada setiap iterasi [76]. Pada iterasi ke-k, satu *fold* digunakan sebagai data uji sementara K-1 *fold* sisanya digunakan sebagai data latih, sehingga setiap *fold* mendapat giliran menjadi data uji tepat satu kali sepanjang keseluruhan proses [76]. Performa akhir model dilaporkan sebagai rata-rata dari metrik evaluasi pada seluruh *fold*, dinyatakan pada Persamaan (35).

$$\bar{M} = \frac{1}{K} \sum_{k=1}^K M_k \dots\dots\dots(35)$$

Keterangan:

$\bar{M}$ = Nilai rata-rata metrik evaluasi pada seluruh *fold*.

M_k = Nilai metrik evaluasi pada *fold* ke-k.

K = Jumlah *fold*.

Selain rata-rata, standar deviasi antar-*fold* turut dilaporkan untuk menggambarkan konsistensi performa model pada berbagai pembagian data, dinyatakan pada Persamaan (36).

$$\sigma_M = \sqrt{\frac{1}{K} \sum_{k=1}^K (M_k - \bar{M})^2} \dots\dots\dots(36)$$

Keterangan:

σ_M = Standar deviasi metrik evaluasi antar-*fold*.

Dibandingkan dengan satu kali pembagian data latih dan data uji yang tetap, *K-Fold Cross-Validation* mengurangi ketergantungan hasil evaluasi pada satu kombinasi pembagian data tertentu, karena setiap bagian dataset pernah berperan sebagai data uji [77].

