BAB 2
LANDASAN TEORI

2.1 *Deepfake*

Deepfake adalah teknologi manipulasi konten visual dan audio yang didukung oleh kecerdasan buatan, khususnya teknik *deep learning*, untuk menghasilkan media sintetis—baik gambar, suara, maupun video—yang tampak sangat realistis namun sebenarnya palsu. Istilah "*deepfake*" sendiri merupakan gabungan dari kata "*deep learning*" dan "*fake*", yang mencerminkan metode dan tujuan dari teknologi ini, yaitu menciptakan pemalsuan tingkat tinggi menggunakan pembelajaran mendalam. Teknologi ini umumnya memanfaatkan arsitektur seperti *Generative Adversarial Networks* (GANs), *Autoencoders*, serta jaringan saraf lainnya seperti ResNet dan Long Short-Term Memory (LSTM) untuk meniru ekspresi wajah, gerakan bibir, intonasi suara, dan pola perilaku visual atau audio manusia dengan presisi tinggi [10, 11]. Proses pembuatannya melibatkan pelatihan model terhadap kumpulan data wajah atau suara target, yang kemudian digunakan untuk menyintesis ekspresi dan gerakan pada konten baru seolah-olah dilakukan oleh orang yang ditiru. Kemampuan *deepfake* untuk merekayasa kenyataan secara sangat meyakinkan menjadikannya ancaman serius dalam konteks keamanan informasi, disinformasi politik, penipuan identitas, serta pelanggaran privasi individu [12, 13]. Dalam beberapa kasus, teknologi ini telah digunakan untuk mencemarkan nama baik, menyebarkan berita palsu, dan bahkan memanipulasi opini publik melalui media sosial, sehingga memunculkan tantangan besar terhadap etika, hukum, dan kepercayaan masyarakat terhadap konten digital.

2.1.1 Deteksi Deepfake Menggunakan LSTM dan ResNet

Untuk mengatasi ancaman dari teknologi *deepfake*, banyak studi telah mengembangkan sistem deteksi berbasis arsitektur hibrida. Salah satu pendekatan yang menjanjikan adalah kombinasi antara ResNet dan Long Short-Term Memory (LSTM). ResNet digunakan untuk mengekstraksi fitur spasial dari wajah dan elemen visual, sementara LSTM mempelajari dinamika temporal dari video, seperti perubahan ekspresi wajah dan sinkronisasi bibir, untuk mengidentifikasi manipulasi video secara lebih akurat [10, 11, 14, 15].

Hasil eksperimen pada berbagai dataset seperti FaceForensics++, Celeb-DF,

dan DFDC menunjukkan bahwa model kombinasi ResNet-LSTM mampu mencapai tingkat akurasi yang tinggi dan ketahanan terhadap teknik *deepfake* yang semakin canggih.

2.1.2 Masalah dan Ancaman dari Deepfake

Meskipun dapat dimanfaatkan untuk hiburan dan pelestarian budaya, *deepfake* membawa sejumlah masalah serius:

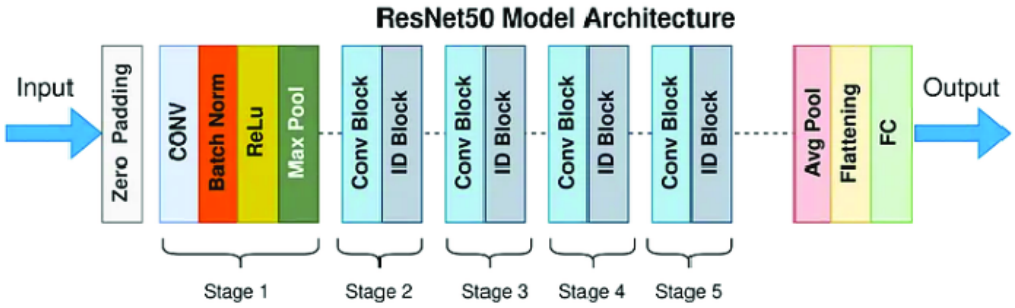
- Penyebaran disinformasi politik — Video palsu dapat digunakan untuk menyebarkan propaganda dan memanipulasi opini publik.
- Penipuan dan kejahatan *cyber* — *Deepfake* suara dan wajah sering dimanfaatkan dalam penipuan identitas, *phishing*, dan rekayasa sosial.
- Pelanggaran privasi dan martabat individu — Konten palsu yang menyerupai seseorang dapat merusak reputasi secara permanen.
- Mengaburkan batas antara kenyataan dan kebohongan — Ini mengancam integritas media digital dan kepercayaan publik terhadap informasi visual [12, 13].

2.2 ResNet-50

ResNet-50 merupakan arsitektur jaringan *convolutional* dalam yang terdiri dari 50 lapisan dan dikenal dengan penggunaan *residual connections* untuk mengatasi degradasi performa pada jaringan yang sangat dalam. Dengan struktur ini, ResNet-50 mampu mengekstraksi fitur kompleks dari citra tanpa mengalami permasalahan *vanishing gradient*. Dalam penelitian ini, ResNet-50 dimanfaatkan sebagai *feature extractor* untuk memperoleh representasi spasial yang kaya dari setiap frame video sebelum diproses oleh model *sequential* seperti LSTM untuk deteksi *deepfake* [16].

2.2.1 Arsitektur ResNet-50

Arsitektur ResNet-50 adalah jaringan konvolusional dalam yang terdiri dari 50 lapisan dan mengadopsi pendekatan pembelajaran residual untuk mempermudah pelatihan jaringan yang sangat dalam. Gambar 2.5 berikut ini menunjukkan alur lengkap dari input hingga output dalam arsitektur ResNet-50.



Gambar 2.1. Arsitektur ResNet-50 secara keseluruhan dari lapisan input hingga klasifikasi output.
Sumber: [17]

Gambar 2.1 mengilustrasikan alur data dalam jaringan ResNet-50, dimulai dari input citra yang melalui beberapa tahap pemrosesan sebagai berikut:

- *Zero Padding*: Menambahkan piksel nol di sekitar tepi citra untuk mempertahankan dimensi spasial setelah konvolusi awal.
- *Convolution Layer*: Konvolusi awal dilakukan dengan filter besar untuk menangkap fitur dasar.
- *Batch Normalization*: Menormalkan hasil konvolusi agar distribusi data tetap stabil selama pelatihan.
- ReLU: Fungsi aktivasi non-linear yang membantu model menangani kompleksitas data.
- *Max Pooling*: Mengurangi dimensi spasial sambil mempertahankan fitur penting melalui pemilihan nilai maksimum dalam jendela lokal.

Setelah tahap awal tersebut, data melewati lima tahap utama (*stage*) yang terdiri dari kombinasi blok *convolutional* dan *identity* (ID) blocks:

- Stage 2–5: Terdiri dari blok residual yang terdiri dari *Conv Block* dan beberapa *Identity Block*. Conv Block digunakan untuk menyesuaikan dimensi input, sementara Identity Block mempertahankan dimensi dan memungkinkan koneksi shortcut langsung.

Tahap akhir dari arsitektur ini terdiri dari:

- *Average Pooling*: Mengambil rata-rata dari fitur map sebagai representasi global dari citra.
- *Flattening*: Mengubah data multidimensi menjadi vektor satu dimensi.
- *Fully Connected Layer* (FCL): Lapisan klasifikasi akhir yang biasanya menggunakan fungsi aktivasi *Softmax* untuk menghasilkan probabilitas *output* dari setiap kelas.

Struktur blok residual dengan koneksi *shortcut* dalam arsitektur ini memungkinkan pelatihan jaringan yang sangat dalam tanpa mengalami degradasi performa, karena gradien dapat mengalir lebih efektif ke lapisan awal selama proses pelatihan.

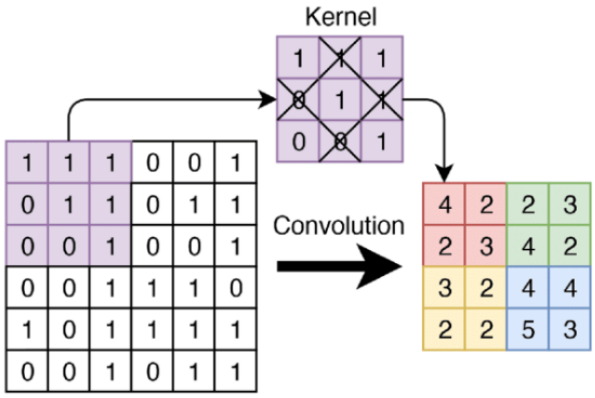
2.2.2 *Convolution Layer*

ResNet-50 menggunakan *Convolutional Layer* sebagai dasar dalam ekstraksi fitur dari data masukan berupa gambar. Konvolusi dalam ResNet-50 mempertahankan hubungan spasial antar piksel dan mempelajari fitur gambar melalui filter yang dapat dilatih. Arsitektur ResNet-50 memperkenalkan konsep *cardinality*, yaitu jumlah jalur (*paths*) paralel yang memproses informasi secara bersamaan dalam satu blok residual. Struktur ini memungkinkan model untuk menangkap lebih banyak fitur tanpa menambah jumlah parameter secara signifikan [18].

Rumus 2.1 menunjukkan cara penjumlahan residual pada jaringan neural.

$$y = x + F(x) \tag{2.1}$$

UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 2.2. Simulasi cara kerja *convolution layer*
Sumber: [19]

Gambar 2.2 mengilustrasikan proses dasar dari operasi konvolusi pada sebuah *convolution layer*. Matriks di sebelah kiri merepresentasikan citra masukan, sedangkan matriks kecil di atasnya adalah *kernel* atau filter. Kernel ini digeser ke seluruh area citra, dan pada setiap langkah, dilakukan operasi perkalian elemen per elemen antara kernel dan bagian citra yang ditutupi, kemudian hasilnya dijumlahkan. Nilai hasil penjumlahan ini membentuk elemen baru dalam matriks hasil konvolusi di sebelah kanan. Proses ini memungkinkan jaringan saraf untuk mengekstraksi fitur seperti tepi, tekstur, dan pola dari citra, yang kemudian digunakan untuk proses klasifikasi atau tugas pengenalan lainnya [19].

2.2.3 *Bottleneck Block*

Blok utama dalam arsitektur ResNet-50 adalah *Bottleneck Block*, yang terdiri dari beberapa jalur paralel dengan konvolusi skala kecil yang lebih efisien dibandingkan dengan pendekatan tradisional. Setiap jalur melakukan transformasi konvolusi identik dan hasilnya dijumlahkan untuk membentuk keluaran akhir. Blok ini mempertahankan keunggulan *deep residual learning* yang diperkenalkan dalam ResNet-50, tetapi dengan parameter yang lebih sedikit dan efisiensi yang lebih tinggi [18].

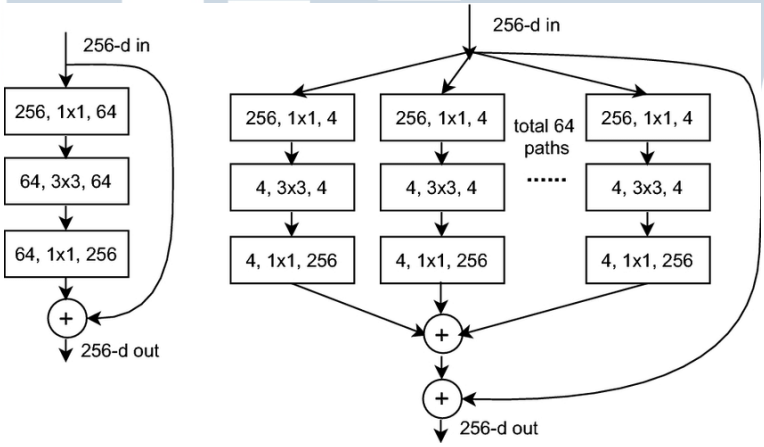
2.2.4 *Aggregation Layer*

Lapisan agregasi (*Aggregation Layer*) dalam ResNet-50 berfungsi untuk menggabungkan hasil dari beberapa jalur transformasi paralel dalam blok residual. Setiap jalur melakukan transformasi konvolusi terhadap masukan yang sama,

dan hasilnya dijumlahkan untuk membentuk keluaran akhir. Penjumlahan ini memungkinkan model untuk mempelajari berbagai representasi fitur secara paralel tanpa meningkatkan kompleksitas jaringan secara signifikan, karena struktur ini memungkinkan informasi untuk diproses lebih efisien melalui beberapa jalur yang memiliki struktur identik [17].

Rumus 2.2 menunjukkan perhitungan untuk lapisan *Aggregation Layer*

$$y = x + \sum_{i=1}^{C} T_i(x) \tag{2.2}$$



Gambar 2.3. Struktur *Aggregation Layer* dalam ResNet-50
Sumber: [20]

Gambar 2.3 memperlihatkan dua tipe struktur blok residual dalam ResNet-50, yaitu *bottleneck* tradisional (kiri) dan versi dengan *cardinality* tinggi (kanan). Pada struktur *bottleneck* standar, data masukan 256-dimensi diproses secara berurutan melalui tiga lapisan konvolusi: konvolusi 1×1 untuk reduksi dimensi, diikuti oleh konvolusi 3×3, dan akhirnya konvolusi 1×1 untuk mengembalikan dimensi ke 256. Hasilnya kemudian dijumlahkan dengan input awal melalui koneksi *shortcut*. Sebaliknya, struktur di sisi kanan menerapkan konsep *cardinality*, di mana data diproses secara paralel melalui 64 jalur transformasi identik. Masing-masing jalur terdiri dari tiga lapisan konvolusi serupa, namun dengan dimensi lebih kecil (misalnya dari 256 ke 4), yang memungkinkan pemrosesan lebih efisien. Hasil dari semua jalur dijumlahkan dan digabungkan kembali dengan input awal untuk menghasilkan keluaran akhir 256-dimensi. Pendekatan ini meningkatkan kemampuan representasi fitur dari jaringan tanpa menambah kompleksitas parameter secara signifikan, mendukung pembelajaran fitur yang

lebih beragam dan kaya [20].

2.2.5 *Pooling Layer*

Pooling layer bertujuan untuk mengurangi dimensi dari setiap peta aktivasi, namun tetap mempertahankan informasi yang paling penting. Gambar masukan dibagi menjadi sejumlah persegi panjang yang tidak tumpang tindih. Setiap wilayah ini di-*downsample* melalui operasi *non-linear* seperti rata-rata atau maksimum. Lapisan ini mencapai generalisasi yang lebih baik, konvergensi yang lebih cepat, ketahanan terhadap translasi dan distorsi, dan biasanya ditempatkan di antara lapisan-lapisan konvolusi [21].

Rumus 2.3 menunjukkan cara kerja *Pooling Layer* dalam jaringan neural

$$p_{j,m} = \max(h_j, (m-1)N + r) \tag{2.3}$$

Gambar 2.4. Simulasi cara kerja *max pooling* pada *pooling layer*
Sumber: [22]

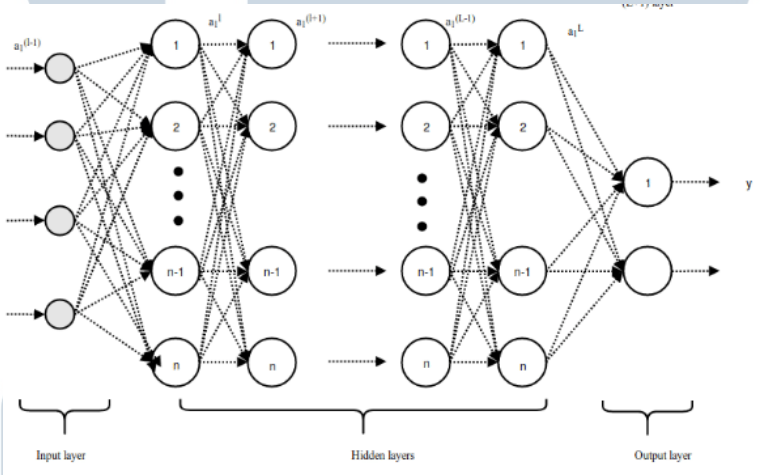
Gambar 2.4 memperlihatkan proses *max pooling* dengan menggunakan filter berukuran 2×2 dan langkah (*stride*) sebesar 2. Citra masukan dibagi menjadi beberapa blok berukuran 2×2 tanpa tumpang tindih. Dalam setiap blok, hanya nilai maksimum yang diambil sebagai representasi dari blok tersebut. Sebagai contoh, pada blok kiri atas yang terdiri dari nilai $\{3, 1, 5, 1\}$, nilai maksimum adalah 5. Proses ini dilakukan untuk seluruh blok pada citra masukan. Hasil akhirnya adalah citra baru dengan dimensi yang lebih kecil, namun tetap mempertahankan fitur paling menonjol dari citra aslinya. Metode ini sangat membantu dalam mengurangi kompleksitas komputasi, serta meningkatkan ketahanan model terhadap variasi posisi objek dalam gambar [22].

2.2.6 *Fully Connected Layer*

Fully Connected Layer (FCL) dalam ResNet-50 digunakan untuk mengklasifikasikan fitur yang telah diekstraksi oleh blok residual sebelumnya. Setiap neuron dalam FCL terhubung dengan semua neuron di lapisan sebelumnya, memungkinkan model untuk menggabungkan informasi dari berbagai fitur yang telah dipelajari. FCL sering menggunakan fungsi aktivasi *Softmax* untuk menghasilkan probabilitas klasifikasi [23].

Rumus 2.4 menunjukkan cara kerja Fully Connected Layer

$$a = \sigma(WX + b) \tag{2.4}$$



Gambar 2.5. Struktur *fully connected layer*
Sumber: [22]

Gambar 2.5 menunjukkan arsitektur dasar dari *Fully Connected Layer* (FCL), yang terdiri dari tiga bagian utama: *input layer*, *hidden layers*, dan *output layer*. Setiap neuron pada lapisan tertentu terhubung secara penuh dengan seluruh neuron pada lapisan berikutnya, yang direpresentasikan oleh garis-garis penghubung antar neuron. Informasi dari input diteruskan ke setiap neuron di lapisan tersembunyi, di mana bobot (*weights*) dan bias diterapkan, diikuti oleh fungsi aktivasi seperti ReLU atau sigmoid. Proses ini berlanjut hingga ke lapisan keluaran, yang biasanya menggunakan fungsi *Softmax* untuk mengubah nilai keluaran menjadi probabilitas untuk klasifikasi. Struktur ini memungkinkan jaringan untuk mempelajari hubungan kompleks antar fitur yang telah diekstraksi

dari lapisan sebelumnya [22].

2.2.7 *Activation Function*

Fungsi aktivasi berperan penting dalam arsitektur ResNet-50 untuk menangani non-linearitas dalam data yang memungkinkan sebuah jaringan untuk dapat menyelesaikan permasalahan permasalahan *non trivial*. Setiap fungsi aktivasi mengambil sebuah nilai dan melakukan operasi matematika. Pada arsitektur ResNet-50, fungsi aktivasi terletak pada perhitungan akhir keluaran fitur map atau sesudah proses perhitungan konvolusi atau *pooling* untuk menghasilkan suatu pola fitur. Fungsi aktivasi yang digunakan dalam ResNet-50 adalah *Rectified Liniear Unit* (ReLU) [24].

Rumus 2.5 menunjukkan fungsi ReLU

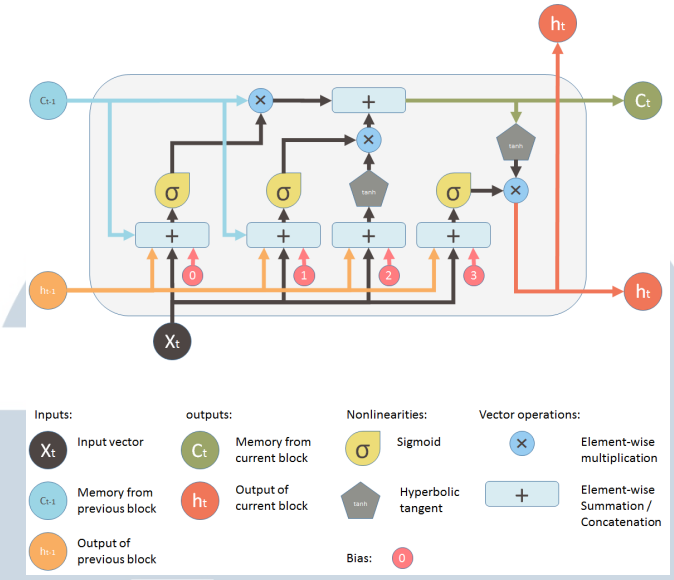
$$f(x) = \max(0, x) \tag{2.5}$$

2.3 Long Short Term Memory (LSTM)

Long Short-Term Memory (LSTM) digunakan dalam penelitian ini sebagai model *sequential* untuk menangkap informasi temporal antar *frame* dalam video. Kemampuannya dalam mempertahankan informasi dari urutan data yang panjang menjadikannya cocok untuk mendeteksi pola perubahan wajah atau ekspresi dalam rangkaian gambar. Dengan memproses representasi fitur dari setiap *frame* yang dihasilkan oleh ResNet-50, LSTM membantu meningkatkan akurasi dalam membedakan antara video asli dan video *deepfake* [10].

2.3.1 Arsitektur LSTM

Long Short-Term Memory (LSTM) adalah arsitektur jaringan saraf berulang (Recurrent Neural Network/RNN) yang dirancang untuk menangani data sekuensial dengan ketergantungan jangka panjang. LSTM mengatasi masalah *vanishing gradient* yang umum pada RNN standar dengan memperkenalkan *memory cell* dan tiga gerbang pengontrol utama: *forget gate*, *input gate*, dan *output gate*.



Gambar 2.6. Arsitektur internal dari unit Long Short-Term Memory (LSTM). Sumber: [25]

Gambar 2.6 menunjukkan arsitektur internal dari unit LSTM. Pada setiap langkah waktu t, LSTM menerima input x_t, hidden state sebelumnya h_{t-1}, dan cell state sebelumnya C_{t-1}. Proses di dalam sel LSTM dibagi menjadi beberapa tahap utama sebagai berikut: [26]

- *Forget Gate* (f_t): Gerbang ini menentukan informasi apa yang akan dibuang dari *cell state* sebelumnya C_{t-1}. Ini dilakukan dengan menerapkan fungsi sigmoid terhadap gabungan $[h_{t-1}, x_t]$.
- *Input Gate* (i_t): Gerbang ini menentukan informasi baru apa yang akan ditambahkan ke *cell state*. *Output* sigmoid digunakan untuk memfilter informasi, kemudian dikalikan dengan kandidat *cell state* $\tilde{C}_t$.
- *Candidate Cell State* ($\tilde{C}_t$): Merupakan representasi informasi baru yang mungkin ditambahkan ke *cell state*, dihitung menggunakan fungsi aktivasi tanh.
- *Cell State* (C_t): Nilai *cell state* diperbarui dengan mengombinasikan informasi yang diteruskan oleh *forget gate* dan *input gate*: $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$.
- *Output Gate* (o_t): Menentukan bagian dari *cell state* yang akan dikeluarkan sebagai *hidden state* h_t.

- *Hidden State* (h_t): Nilai akhir yang diteruskan ke waktu berikutnya dan digunakan dalam proses prediksi. Dihitung sebagai: $h_t = o_t * \tanh(C_t)$.

Rumus dasar pada LSTM didefinisikan sebagai berikut [25]:
Rumus 2.6 menunjukkan *Forget Gate* pada LSTM

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.6}$$

Rumus 2.7 menunjukkan *Input Gate* pada LSTM

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.7}$$

Rumus 2.8 menunjukkan *Candidate Cell State* pada LSTM

$$\tilde{C}t = \tanh(W_C \cdot [ht-1, x_t] + b_C) \tag{2.8}$$

Rumus 2.9 menunjukkan *Cell State* pada LSTM

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}t \tag{2.9}$$

Rumus 2.10 menunjukkan *Output Gate* pada LSTM

$$o_t = \sigma(W_o \cdot [ht-1, x_t] + b_o) \tag{2.10}$$

Rumus 2.11 menunjukkan *Hidden State* pada LSTM

$$h_t = o_t * \tanh(C_t) \tag{2.11}$$

2.3.2 Fungsi Aktivasi pada LSTM

Fungsi aktivasi memiliki peran penting dalam mengatur aliran informasi di dalam sel LSTM. Pada umumnya, fungsi aktivasi yang digunakan adalah

fungsi Sigmoid dan fungsi *hyperbolic tangent* (tanh). Fungsi Sigmoid digunakan pada gerbang *input*, *forget*, dan *output*. Fungsi ini membatasi keluaran dalam rentang 0 hingga 1, yang merepresentasikan seberapa besar informasi yang harus dilewatkan oleh masing-masing gerbang. Sementara itu, fungsi tanh digunakan untuk menormalisasi nilai di dalam *memory cell*. Fungsi ini menghasilkan nilai dalam rentang -1 hingga 1, sehingga membantu menjaga kestabilan data yang disimpan dalam sel memori [25].

Rumus 2.12 menunjukkan Fungsi Sigmoid pada LSTM

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.12}$$

Rumus 2.13 menunjukkan Fungsi Tanh pada LSTM

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{2.13}$$

2.3.3 Optimasi LSTM

Salah satu tantangan utama dalam implementasi Long Short-Term Memory (LSTM) adalah kompleksitas komputasinya yang tinggi, terutama ketika diterapkan pada data sekuensial yang panjang dan kompleks. Untuk mengatasi hal ini, berbagai metode optimasi telah dikembangkan. Salah satu pendekatan yang umum digunakan adalah teknik *batch learning* dengan algoritma *Backpropagation Through Time* (BPTT), yang memungkinkan pembaruan bobot jaringan secara efisien berdasarkan akumulasi error dalam rentang waktu tertentu [27]. Selain itu, upaya pengurangan jumlah parameter juga dilakukan melalui modifikasi arsitektur, seperti penggunaan *Cell-Expanded LSTM*. Varian ini menambahkan lebih banyak *memory cell* untuk meningkatkan kapasitas model dalam mengenali pola-pola kompleks dalam data sekuensial, tanpa harus secara drastis meningkatkan jumlah parameter pelatihan [28]. Pendekatan-pendekatan ini berkontribusi pada peningkatan efisiensi dan akurasi model LSTM, menjadikannya lebih layak untuk diterapkan dalam sistem yang memerlukan prediksi real-time atau sumber daya terbatas.

2.3.4 Aplikasi LSTM dalam Deteksi *Deepfake*

LSTM telah membuktikan efektivitasnya dalam berbagai aplikasi pemrosesan data temporal, termasuk dalam deteksi video *deepfake*. Dalam konteks ini, LSTM digunakan untuk menganalisis rangkaian ekspresi wajah, gerakan bibir, dan pola mikro-ekspresi yang terjadi secara berurutan dari satu *frame* ke *frame* lainnya. Model ini sangat andal dalam menangkap dinamika temporal yang sering kali tidak konsisten dalam video hasil manipulasi. Sejumlah studi telah menunjukkan bahwa kombinasi antara Convolutional Neural Network (CNN), seperti ResNet, dengan LSTM dapat meningkatkan performa klasifikasi antara video asli dan video palsu secara signifikan. CNN bertugas mengekstraksi fitur spasial dari setiap *frame*, sementara LSTM memproses hubungan temporal antar *frame* tersebut. Model hibrida ini terbukti mampu mendeteksi anomali temporal yang halus, yang biasanya tidak terlihat pada analisis spasial semata [29, 30]. Dengan demikian, integrasi CNN dan LSTM menjadi strategi yang sangat efektif dalam menghadapi tantangan deteksi *deepfake* yang semakin kompleks.

2.3.5 Varian LSTM dan Perbandingan

Seiring berkembangnya kebutuhan akan model yang lebih efisien dan akurat, beberapa varian dari LSTM telah dikembangkan untuk meningkatkan performa jaringan tanpa meningkatkan kompleksitas secara signifikan. Salah satu varian tersebut adalah Bidirectional LSTM (BiLSTM), yang memungkinkan informasi diproses dalam dua arah, yaitu maju (*forward*) dan mundur (*backward*) secara simultan. Pendekatan ini sangat berguna dalam situasi di mana konteks masa depan sama pentingnya dengan konteks masa lalu dalam menentukan makna dari data sekuensial [31]. Di sisi lain, Gated Recurrent Unit (GRU) merupakan penyederhanaan dari LSTM dengan menggabungkan gerbang input dan *forget* menjadi satu gerbang tunggal. Dengan arsitektur yang lebih ringkas, GRU menawarkan efisiensi komputasi yang lebih baik, dan dalam banyak kasus, performanya setara dengan atau bahkan melebihi LSTM [32]. Pemilihan antara LSTM, BiLSTM, atau GRU sangat bergantung pada kebutuhan spesifik dari aplikasi yang dikembangkan, seperti kompleksitas data, kapasitas komputasi, dan target performa yang diinginkan.

2.3.6 Implementasi Perangkat Keras

Dalam rangka memperluas aplikasi LSTM ke dalam sistem dengan keterbatasan sumber daya, implementasi perangkat keras menjadi bidang riset yang penting. Salah satu pendekatan yang menonjol adalah implementasi LSTM pada perangkat keras seperti *Field Programmable Gate Array* (FPGA), yang mampu meningkatkan efisiensi konsumsi daya sekaligus mempercepat proses komputasi. Arsitektur berbasis memristor telah diperkenalkan sebagai alternatif yang menjanjikan untuk meniru perilaku sinapsis biologis, dengan keunggulan dalam efisiensi energi dan kecepatan akses memori [25]. Selain itu, berbagai metode optimasi perangkat keras telah dikembangkan untuk meminimalkan konsumsi daya dan jumlah sumber daya logika yang digunakan, seperti optimasi *layout* sirkuit, kompresi bobot jaringan, serta penggunaan metode kuantisasi untuk representasi data [33]. Pendekatan ini sangat penting terutama dalam pengembangan sistem deteksi *real-time* pada perangkat *edge*, seperti kamera pengawas cerdas atau perangkat IoT yang membutuhkan pengenalan pola dengan latensi rendah dan konsumsi daya yang minimal.

2.4 *ForgeryNet Dataset*

ForgeryNet adalah *dataset* besar yang dirancang untuk mendukung penelitian dalam deteksi manipulasi gambar dan video. *Dataset* ini mencakup berbagai jenis manipulasi, seperti *splicing*, *copy-move*, dan *deepfake*, yang mencerminkan tantangan dunia nyata dalam bidang *media forensics*. *ForgeryNet* berisi lebih dari 2.9 juta gambar dan 220 ribu video dengan anotasi detail yang mencakup *mask* manipulasi serta metadata tambahan[34].

2.4.1 *Karakteristik Dataset*

Dataset ini memiliki beberapa karakteristik utama yang membuatnya unik dan berharga bagi komunitas penelitian:

- Skala Besar: Dengan jutaan sampel gambar dan ratusan ribu video, ForgeryNet menyediakan data yang cukup untuk melatih model deteksi forensik dengan performa tinggi.
- Beragam Teknik Manipulasi: *Dataset* ini mencakup berbagai jenis teknik pemalsuan, mulai dari manipulasi tradisional hingga metode berbasis

kecerdasan buatan.

- Anotasi Detail: Setiap sampel dilengkapi dengan mask manipulasi yang memungkinkan model untuk dilatih dengan pendekatan segmentasi.
- Variasi Domain: Data dalam ForgeryNet mencakup berbagai domain, termasuk wajah manusia, objek, serta pemandangan, sehingga meningkatkan kemampuan generalisasi model yang dilatih.

2.4.2 *Kegunaan Dataset*

ForgeryNet digunakan dalam berbagai skenario deteksi pemalsuan, termasuk:

1. Deteksi Manipulasi Gambar: Model dapat dilatih untuk mengidentifikasi apakah suatu gambar telah dimanipulasi serta menandai area yang telah dimodifikasi.
2. Deteksi *Deepfake* Video: *Dataset* ini mendukung penelitian dalam mendeteksi video *deepfake* dengan fitur temporal yang lebih kompleks.
3. Segmentasi Area Manipulasi: Dengan anotasi *mask* yang tersedia, model dapat dilatih untuk menentukan area spesifik yang telah dimanipulasi dalam suatu gambar atau video.

2.5 Eksplorasi Hyperparameter

Dalam penelitian ini, dilakukan eksplorasi sistematis terhadap sejumlah kombinasi *hyperparameter* utama untuk mengidentifikasi konfigurasi optimal yang dapat memaksimalkan performa model dalam mendeteksi video *deepfake*. Pemilihan rentang nilai untuk masing-masing *hyperparameter* tidak hanya berdasarkan praktik umum dalam literatur *deep learning*, tetapi juga disesuaikan dengan karakteristik khusus dari *dataset* ForgeryNet dan kompleksitas temporal-spasial dari video manipulasi yang dianalisis. Tujuan dari eksplorasi ini adalah untuk memperoleh keseimbangan optimal antara kapasitas representasi model, stabilitas proses pelatihan, dan generalisasi terhadap data tak terlihat.

2.5.1 LSTM Units (512, 1024, 2048)

Jumlah unit pada lapisan LSTM merupakan faktor krusial yang menentukan kapasitas model dalam menyimpan dan mengolah informasi urutan jangka panjang. Dalam konteks deteksi video *deepfake*, dinamika temporal seperti pergerakan bibir, pola kedipan, atau perubahan ekspresi wajah membutuhkan representasi memori yang kaya dan berkelanjutan. Dengan menggunakan jumlah unit sebesar 512, 1024, dan 2048, eksplorasi ini bertujuan untuk memahami seberapa besar kapasitas memori dibutuhkan untuk menangkap sinyal temporal yang bermakna tanpa kehilangan efisiensi. Unit yang lebih besar seperti 2048 berpotensi memberikan performa klasifikasi yang lebih tinggi karena mampu mempelajari relasi temporal yang kompleks antar *frame*. Namun, peningkatan jumlah unit juga membawa risiko tingginya kompleksitas model, meningkatnya kebutuhan memori GPU, serta kemungkinan *overfitting* terutama jika data pelatihan tidak mencukupi. Oleh karena itu, *trade-off* antara akurasi dan efisiensi dihitung secara cermat dalam konfigurasi ini [35].

2.5.2 Sequence Length (8, 16, 24)

Panjang sekuens input atau *sequence length* menjadi parameter penting dalam model-model sekuensial seperti LSTM karena menentukan berapa banyak *frame* yang digunakan sebagai konteks saat membuat prediksi. Dalam eksperimen ini, panjang sekuens diuji pada 8, 16, dan 24 *frame* untuk melihat bagaimana cakupan temporal memengaruhi kualitas deteksi. Panjang sekuens pendek (misalnya 8 *frame*) cenderung memiliki manfaat dari sisi efisiensi pelatihan dan mengurangi risiko *vanishing gradient*, tetapi dapat kehilangan konteks jangka panjang yang sering kali penting dalam mendeteksi manipulasi *video* yang halus. Sebaliknya, sekuens panjang (16 dan 24 frame) memberikan model kesempatan untuk mengamati pola-pola jangka panjang dalam *video*, seperti sinkronisasi gerakan bibir dengan suara atau konsistensi ekspresi wajah, yang sering kali menjadi titik lemah *video deepfake*. Namun, penggunaan sekuens panjang juga meningkatkan dimensi input, memperpanjang waktu pelatihan, dan meningkatkan kompleksitas perhitungan gradien, yang dapat menghambat konvergensi [36].

2.5.3 Jumlah LSTM Layers (1, 2, 3)

Menambahkan lebih dari satu lapisan LSTM memungkinkan model untuk membangun representasi hierarkis dari data sekuensial, di mana lapisan bawah menangkap pola-pola lokal sementara lapisan atas dapat mengabstraksikan dinamika temporal tingkat tinggi. Dalam eksperimen ini, diuji penggunaan 1, 2, dan 3 lapisan untuk mengevaluasi kontribusi kedalaman terhadap kemampuan generalisasi model dalam klasifikasi *deepfake*. Satu lapisan LSTM sudah cukup untuk menangani pola temporal dasar, tetapi mungkin tidak mampu menangkap hubungan yang lebih kompleks seperti ekspresi emosional atau perubahan halus dalam mikro-ekspresi. Dua hingga tiga lapisan memberi ruang bagi model untuk memproses informasi dalam beberapa tingkat kompleksitas, meskipun penambahan kedalaman yang berlebihan juga dapat menghambat pembelajaran efektif, memperbesar risiko *overfitting*, serta memerlukan teknik regularisasi dan normalisasi tambahan [37].

2.5.4 Dropout Rate (0.3, 0.4, 0.5)

Dropout merupakan teknik regularisasi penting dalam jaringan saraf yang berfungsi untuk mencegah *overfitting* dengan menonaktifkan sejumlah neuron secara acak selama pelatihan. Eksplorasi tingkat *dropout* sebesar 0.3, 0.4, dan 0.5 ditujukan untuk mengidentifikasi seberapa besar regularisasi dibutuhkan oleh model LSTM dalam menghindari ketergantungan berlebih terhadap pola dalam data pelatihan. Tingkat *dropout* yang rendah (seperti 0.3) masih menjaga arsitektur tetap padat dan efisien, sedangkan tingkat yang lebih tinggi (0.5) memberikan tekanan regularisasi lebih kuat yang bermanfaat dalam *dataset* berukuran terbatas atau dengan distribusi kelas tidak seimbang. Dalam domain *deepfake detection*, *dropout* juga penting untuk menjaga generalisasi model terhadap berbagai jenis manipulasi visual yang tidak terdapat dalam data pelatihan [38].

2.5.5 Scheduler (ReduceLROnPlateau, CosineAnnealingLR, OneCycleLR)

Penggunaan *scheduler* dalam pelatihan model berperan penting dalam mengatur dinamika *learning rate*, yang berdampak langsung pada kecepatan dan kualitas konvergensi. *Scheduler ReduceLROnPlateau* digunakan untuk menurunkan *learning rate* secara adaptif jika metrik validasi tidak membaik dalam beberapa *epoch*, sehingga membantu model keluar dari stagnasi lokal. Sementara

itu, *CosineAnnealingLR* menurunkan *learning rate* secara bertahap mengikuti fungsi kosinus, yang bertujuan menjaga stabilitas pelatihan dan memaksimalkan eksplorasi parameter pada awal pelatihan. Adapun *OneCycleLR* dirancang berdasarkan prinsip bahwa peningkatan agresif *learning rate* di awal pelatihan, diikuti penurunan tajam menjelang akhir, dapat mempercepat generalisasi dan konvergensi. Eksplorasi ketiga *scheduler* ini dilakukan untuk menentukan pola penurunan *learning rate* mana yang paling sesuai dengan arsitektur model dan karakteristik dinamika loss dalam deteksi deepfake [39].

2.5.6 Learning Rate (1e-3, 1e-4, 1e-5)

Learning rate adalah *hyperparameter* fundamental yang menentukan seberapa besar langkah pembaruan parameter selama proses *backpropagation*. Rentang nilai 1e-3, 1e-4, dan 1e-5 digunakan untuk mengevaluasi sensitivitas model terhadap laju pembelajaran, terutama dalam pelatihan arsitektur besar seperti LSTM. Nilai yang terlalu tinggi seperti 1e-3 memungkinkan konvergensi cepat, namun rentan terhadap osilasi dan pelatihan yang tidak stabil. Sebaliknya, nilai rendah seperti 1e-5 menawarkan pelatihan yang lebih stabil, tetapi memperlambat konvergensi secara signifikan dan membutuhkan lebih banyak *epoch*. Oleh karena itu, pemilihan *learning rate* optimal menjadi kunci untuk memperoleh keseimbangan antara stabilitas dan efisiensi pelatihan, khususnya ketika digunakan bersama dengan *scheduler* dinamis [40].

2.5.7 Weight Decay (1e-3, 1e-4, 1e-5)

Weight decay atau regularisasi L2 membantu membatasi pertumbuhan berlebihan bobot jaringan dengan menambahkan penalti terhadap besar bobot dalam fungsi kerugian. Dalam konteks *deepfake detection*, *weight decay* penting untuk mencegah model menghafal *noise* atau artefak yang tidak konsisten. Nilai 1e-3 memberikan regularisasi yang kuat dan cocok digunakan saat model menunjukkan gejala *overfitting*. Sementara itu, nilai 1e-5 memberikan fleksibilitas lebih besar dalam eksplorasi parameter tetapi berisiko mengurangi kemampuan generalisasi. Eksperimen ini dilakukan untuk memahami sejauh mana penalti bobot memengaruhi performa akhir model dan bagaimana interaksinya dengan *dropout* serta *learning rate* [41].

2.5.8 Batch Size (8, 16, 24)

Ukuran *batch* memiliki dampak besar terhadap dinamika pembelajaran, stabilitas gradien, dan efisiensi GPU. *Batch* kecil seperti 8 memberikan gradien yang lebih bervariasi, sehingga cenderung membantu generalisasi, tetapi menyebabkan fluktuasi *loss* yang tinggi. *Batch* sedang seperti 16 atau 24 memberi *trade-off* yang seimbang antara akurasi gradien dan kecepatan pelatihan. Dalam eksperimen ini, *batch size* disesuaikan dengan kapasitas GPU dan kebutuhan komputasi LSTM yang sensitif terhadap urutan data. Ukuran *batch* juga memengaruhi efektivitas optimisasi dan regularisasi, sehingga eksplorasinya penting dalam membentuk dinamika pelatihan yang stabil dan adaptif terhadap distribusi data video [42].

2.5.9 Loss Function (CrossEntropyLoss, MSE, Focal Loss)

Pemilihan fungsi kerugian (*loss function*) mencerminkan asumsi terhadap distribusi data dan objektif pelatihan. CrossEntropyLoss merupakan pilihan standar untuk klasifikasi biner dan memiliki formulasi *log-likelihood* yang sesuai untuk probabilitas *output softmax*. MSE (*Mean Squared Error*), meskipun biasanya digunakan untuk regresi, dieksplorasi untuk menguji apakah pendekatan probabilistik terhadap prediksi dapat menghasilkan manfaat dalam mendeteksi ambiguitas manipulasi video. Sementara itu, Focal *Loss* diperkenalkan sebagai solusi atas ketidakseimbangan kelas, dengan menurunkan kontribusi contoh mudah dan menekankan pada contoh yang sulit diklasifikasikan. Ini sangat berguna dalam *dataset deepfake* yang mungkin mengandung distribusi kelas tidak seimbang atau variasi kualitas manipulasi yang besar [43].

2.5.10 Optimizer (RAdam, RMSprop, AdamW)

Pemilihan algoritma optimisasi sangat menentukan efektivitas proses pelatihan. RAdam (*Rectified* Adam) menawarkan keunggulan dalam fase awal pelatihan karena mengatasi ketidakstabilan varian dalam momentum adaptif, menjadikannya cocok untuk model besar dan *dataset* kompleks. RMSprop dikenal sebagai pilihan populer untuk RNN karena kemampuannya menyesuaikan langkah pembelajaran berdasarkan rata-rata kuadrat gradien sebelumnya, sehingga stabil untuk data urutan. AdamW adalah varian dari Adam yang memisahkan *weight decay* dari pembaruan gradien, membuatnya lebih efektif untuk regularisasi

eksplisit. Ketiga *optimizer* ini dievaluasi untuk menentukan mana yang paling seimbang dalam hal kecepatan konvergensi, stabilitas pelatihan, dan daya generalisasi akhir [44].

2.6 *Random Search*

Random Search adalah salah satu metode optimasi untuk pemilihan *hyperparameter* dalam pembelajaran mesin yang bekerja dengan cara mengambil sampel acak dari ruang parameter yang telah ditentukan sebelumnya. Metode ini bertujuan untuk menemukan kombinasi parameter yang memberikan performa terbaik pada model tanpa perlu menguji semua kemungkinan secara menyeluruh. Alih-alih memeriksa setiap kombinasi seperti pada *Grid Search*, *Random Search* secara acak mengevaluasi sejumlah kombinasi parameter dalam jumlah iterasi tertentu [45]. Meskipun bersifat acak, penelitian menunjukkan bahwa metode ini cenderung lebih efisien dibandingkan *Grid Search*, terutama ketika hanya sebagian kecil dari *hyperparameter* yang benar-benar berpengaruh terhadap performa model secara signifikan [45, 46].

Dalam konteks pembelajaran mendalam (*deep learning*), di mana pelatihan model memerlukan sumber daya komputasi yang besar, *Random Search* menjadi solusi yang praktis karena mampu mengeksplorasi lebih banyak kombinasi parameter dalam waktu yang lebih singkat. Efektivitas *Random Search* dalam mengoptimalkan parameter juga didukung oleh berbagai studi yang menunjukkan bahwa metode ini dapat menemukan hasil mendekati optimal dengan waktu yang lebih efisien [47].

Dalam penelitian ini, pemilihan *hyperparameter* dilakukan menggunakan pendekatan *Random Search*. Beberapa *hyperparameter* penting yang diatur meliputi jumlah unit dalam setiap sel LSTM, panjang urutan input (*sequence length*), jumlah lapisan tersembunyi (*LSTM layers*), tingkat *dropout*, jenis *scheduler* untuk pengaturan laju pembelajaran, nilai *learning rate*, *weight decay*, ukuran *batch*, jenis fungsi kerugian (*loss function*), serta algoritma optimisasi (*optimizer*).

Untuk setiap eksperimen, nilai-nilai *hyperparameter* tersebut diambil secara acak dari set nilai yang telah ditentukan sebelumnya. Sebagai contoh, jumlah unit LSTM diacak dari {512, 1024, 2048}, panjang sekuens dari {8, 16, 24}, serta nilai lain seperti *dropout rate*, *learning rate*, dan *batch size* juga disampling dari rentang nilai yang sudah ditetapkan. Beberapa fungsi kerugian yang dievaluasi meliputi CrossEntropyLoss, Mean Squared Error (MSE), dan Focal *Loss*, sedangkan

algoritma optimisasi yang digunakan mencakup RAdam, RMSprop, dan AdamW.

Pendekatan ini tidak hanya mempercepat proses pencarian dibandingkan pencarian menyeluruh, tetapi juga memungkinkan eksplorasi kombinasi yang lebih luas dalam ruang parameter, yang pada akhirnya dapat menghasilkan model dengan kinerja yang lebih optimal.

2.7 Confusion Matrix

Confusion Matrix adalah metode yang digunakan dalam pembelajaran mesin untuk menilai performa model klasifikasi [48]. Matriks ini berbentuk tabel persegi, di mana baris merepresentasikan kelas yang sebenarnya, sedangkan kolom menunjukkan kelas yang diprediksi oleh model. Dalam kasus klasifikasi biner, ukuran *Confusion Matrix* adalah 2×2, sementara dalam klasifikasi multi-kelas, ukurannya menjadi $k \times k$ sesuai dengan jumlah kelas yang ada. Dalam bidang rekayasa perangkat lunak, Font dkk. memanfaatkan *Confusion Matrix* untuk membandingkan hasil prediksi dengan nilai sebenarnya dari elemen model. Matriks ini terdiri dari empat komponen utama: *true positive* (TP), *false positive* (FP), *true negative* (TN), dan *false negative* (FN), yang menjadi dasar dalam menghitung berbagai metrik evaluasi seperti *precision*, *recall*, dan *F1-score* [49].

Rumus 2.14 menunjukkan *Precision* yang merupakan ukuran seberapa banyak dari seluruh prediksi positif yang benar.

$$\text{Precision} = \frac{TP}{TP+FP} \tag{2.14}$$

Rumus 2.15 menunjukkan *Recall* yang merupakan ukuran seberapa banyak data positif yang berhasil dikenali oleh model secara benar.

$$\text{Recall} = \frac{TP}{TP+FN} \tag{2.15}$$

Rumus 2.16 menunjukkan *F1-Score* yang merupakan rata-rata antara *precision* dan *recall*

$$F1\text{-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.16}$$

2.8 Evaluasi Model

Evaluasi model dilakukan dengan mempertimbangkan berbagai metrik yang mencerminkan performa model dalam tugas deteksi objek menggunakan ResNet-50. Beberapa metrik yang digunakan dalam evaluasi model meliputi:

- *Accuracy*: Akurasi mengukur proporsi prediksi yang benar terhadap seluruh prediksi yang dilakukan. Ini merupakan indikator umum untuk menilai seberapa baik model mengklasifikasikan data secara keseluruhan [50].
- *Losses*: Metrik ini mengukur kerugian selama pelatihan dan validasi, yang mencerminkan seberapa baik model belajar. Penurunan *loss* secara konsisten menunjukkan peningkatan akurasi model [51].
- *Confusion Matrix*: Digunakan untuk menilai performa model dengan membandingkan prediksi dengan nilai *ground truth*. Matriks ini memberikan wawasan tentang distribusi kesalahan yang dilakukan model [52].
- *Precision, Recall*, dan *F1-score*: *Precision* menunjukkan seberapa banyak prediksi positif yang benar, *recall* menunjukkan seberapa banyak data positif yang berhasil dideteksi, dan *F1-score* merupakan keseimbangan antara *precision* dan *recall*. Metrik ini penting untuk mengevaluasi performa model secara komprehensif [53].

Metrik-metrik ini membantu dalam memahami kekuatan dan kelemahan model serta memungkinkan perbandingan dengan model lain yang dikembangkan.

UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA