# Laporan Tugas Besar 2 IF3270 Pembelajaran Mesin

# Convolutional Neural Network dan Recurrent Neural Network

**Semester 2 Tahun 2024/2025**


**Disusun oleh:**

| | |
|---|---|
| Hugo Sabam Augusto | 13522129 |
| Muhamad Zaki | 13522136 |
| Ahmad Rafi Maliki | 13522137 |

**Teknik Informatika**

**Institut Teknologi Bandung**

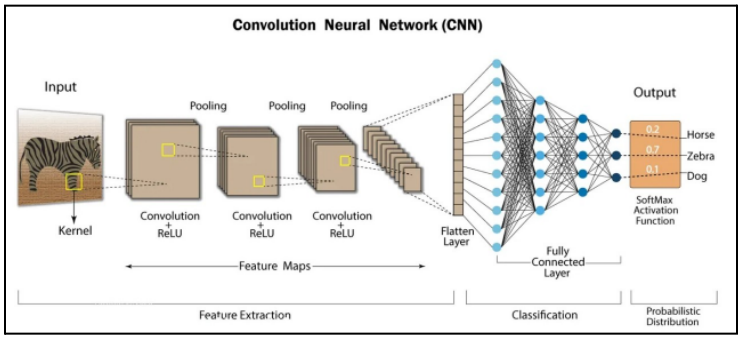
**2025**

# Daftar Isi

# Bagian 1.

# Deskripsi Persoalan

## 1.1. Convolutional Neural Networks (CNN)

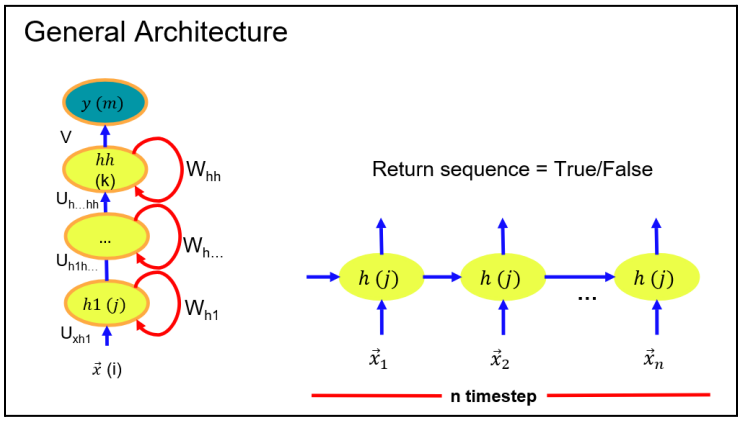



**Gambar 1.1.1** Arsitektur Convolutional Neural Network

Convolution Neural Network (CNN) adalah jenis dari neural network yang khusus dirancang untuk memproses data yang memiliki struktur grid, seperti gambar atau video. Ciri khas dari CNN yaitu terdapat beberapa layer. Layer yang paling umum dimiliki adalah convolutional layer dimana layer ini menggunakan operasi konvolusi dengan filter (kernel) untuk mengekstrak fitur atau pola tertentu dari gambar. Lalu ada pooling layers dimana layer ini mengecilkan dimensi gambar untuk mengurangi komputasi dan membuat model lebih robust terhadap perubahan kecil. Selanjutnya ada Fully Connected Layers dimana layer ini menghubungkan semua neuron untuk klasifikasi.

Convolutional Neural Network (CNN) mempelajari fitur dari gambar melalui proses pelatihan berlapis yang dimulai dengan inisialisasi filter , dimana filter pada lapisan awal belajar mendeteksi pola sederhana seperti tepi, sementara filter di lapisan lebih dalam mengenali fitur kompleks. Lapisan pooling berperan menyederhanakan representasi fitur dengan mengurangi dimensi spasial sehingga model lebih robust terhadap variasi kecil, sementara lapisan fully connected mengintegrasikan seluruh fitur yang telah diekstrak untuk klasifikasi akhir melalui fungsi softmax yang menghasilkan probabilitas kategori, dengan seluruh proses ini didukung efisiensi komputasi berkat konsep parameter sharing dalam operasi konvolusi.

## 1.2. Simple Recurrent Neural Networks (Simple RNN)

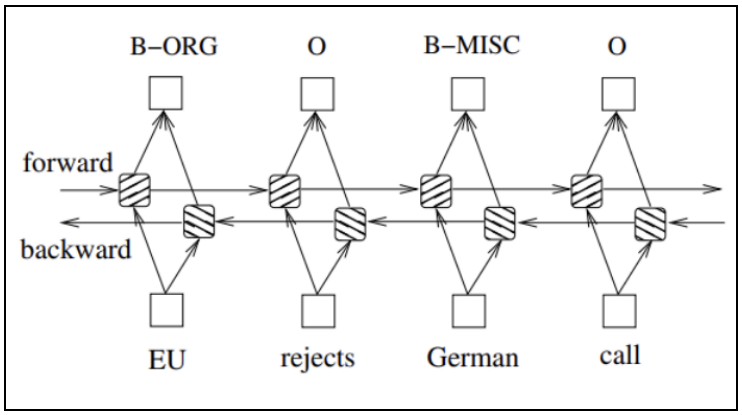



**Gambar 1.2.1** *Folded Simple* RNN (kiri), *Unfolded Simple* RNN (Kanan)

Recurrent Neural Networks (RNN) adalah variasi dari Neural Network yang didesain untuk bekerja dengan data yang memiliki urutan sekuens. RNN bekerja dengan cara menerima input berupa data yang berurutan kemudian memproses input satu langkah waktu (*time step*) pada satu waktu, dan menyimpan informasi dari langkah sebelumnya dalam bentuk *hidden state* yang terus diperbaharui selama proses berjalan.

Jika dijabarkan, RNN sebenarnya terdiri dari beberapa *fully connected neurons* untuk tiap *time step*-nya yang cara kerjanya mirip dengan FFNN. Hanya saja tiap *time step* memiliki *input* tambahan yaitu nilai *hidden state* dari *time step* sebelumnya. Gambar 1.3.1 (Kanan) merupakan gambaran dari sebuah *layer* RNN dengan *n time step* dan *j* neuron untuk tiap *time step*-nya.

RNN sangat bagus untuk memproses data yang memiliki urutan atau sekuens, seperti teks, suara, atau data deret waktu, karena kemampuannya untuk menyimpan informasi dari langkah waktu (*time step*) sebelumnya melalui *hidden state*. Dengan demikian, RNN mampu menangkap pola temporal dan ketergantungan jangka panjang dalam data, membuatnya cocok untuk permasalahan seperti prediksi teks, analisis sentimen, pengenalan suara, atau peramalan deret waktu.

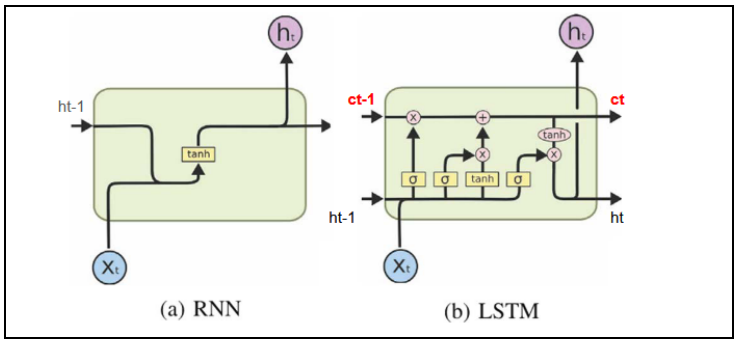



**Gambar 1.2.2** *Bidirectional RNN*

Adapula variasi lain dari RNN yaitu *Bidirectional* RNN, yaitu RNN yang memproses data sekuens secara dua arah, yaitu dari maju (*forward*) dan mundur (*backward*) secara bersamaan. Model ini meningkatkan kemampuan *Simple* RNN dengan memungkinkan model memahami konteks secara lebih lengkap dengan melihat informasi masa depan dan masa lalu dalam urutan data. *Bidirectional* RNN sangat efektif untuk masalah yang memerlukan konteks lengkap, seperti penerjemahan bahasa, penandaan bagian kata (POS *tagging*), dan pengenalan ucapan, di mana makna kata atau sinyal dapat dipengaruhi oleh konteks sebelum dan sesudahnya.

### 1.3. Long Short Term Memory (LSTM)

LSTM memiliki arsitektur yang mirip dengan RNN, menggunakan struktur dasar yang serupa seperti bobot input (U), bobot output (V), dan bobot rekuren (W) pada hidden node yang berulang dari waktu ke waktu. Namun perbedaan utamanya terletak pada adanya cell state dalam LSTM.




**Gambar 1.3.1** Perbandingan *Cell* RNN dan LSTM

Cell state dalam LSTM berfungsi sebagai jalur memori jangka panjang dan memungkinkan informasi mengalir hampir tanpa adanya gangguan antar langkah waktu. Untuk mengatur aliran informasi ini, LSTM menggunakan 3 jenis gerbang yaitu forget gate, input gate, dan output gate. ‘Gerbang’ ini secara dinamis mengontrol informasi mana yang disimpan, diperbarui, atau dibuang. Dengan struktur ini, LSTM bisa lebih unggul dibanding RNN biasa dalam mengatasi masalah vanishing gradient dan lebih mampu mengingat konteks dari urutan yang panjang. Oleh karena itu, LSTM banyak digunakan dalam berbagai aplikasi seperti pemrosesan bahasa alami (misalnya penerjemahan dan analisis sentimen), prediksi deret waktu (seperti harga saham dan cuaca), pengenalan suara, hingga analisis video, di mana pemahaman terhadap urutan data sangat krusial.

# Bagian 2.

# Pembahasan

## 2.1. Penjelasan Implementasi

### 2.1.1. Convolutional Neural Networks (CNN)

#### 2.1.1.1. Deskripsi Kelas

**Tabel 2.1.1.1.1** Atribut Kelas Conv2DLayer

| Atribut | Tipe Data | Deskripsi |
|---|---|---|
| weights | np.ndarray | Bobot Filter Konvolusi |
| biases | np.ndarray | Bias untuk setiap filter |
| strides | tuple(int,int) | Pergeseran konvolusi (x,y) |
| padding | str | Jenis padding |
| k_h | int | Tinggi kernel |
| k_w | int | Tinggi kernel |
| in_ch | int | Jumlah channel input |
| num_filters | int | Jumlah filters |
| s_y,s_x | int | Stride dalam x dan y |

**Tabel 2.1.1.1.2** Metode Kelas Conv2DLayer

| Metode | Deskripsi |
|---|---|
| __init__ | Konstruktor |
| _pad_input | Untuk menambahkan padding (same atau valid) |
| forward | Melakukan forward propagation pada layer konvolusi |

**Tabel 2.1.1.1.3** Atribut Kelas ActivationLayer

| Atribut | Tipe Data | Deskripsi |
|---|---|---|
| activation_type | str | Jenis Aktivasi (Relu/Softmax) |

**Tabel 2.1.1.1.4** Metode Kelas Activation Layer

| Metode | Deskripsi |
|---|---|

| | |
|---|---|
| __init__ | Inisialisasi |
| forward | Menerapkan fungsi aktivasi ke input |

**Tabel 2.1.1.1.5** Atribut Kelas Pooling Layer

| Atribut | Tipe Data | Deskripsi |
|---|---|---|
| pool_h | int | Tinggi window pooling |
| pool_w | int | Lebar window pooling |
| s_y | int | Stride y axis |
| s_x | int | Stride x axis |
| mode | str | Jenis pooling |

**Tabel 2.1.1.1.6** Metode Kelas Pooling Layer

| Metode | Deskripsi |
|---|---|
| __init__ | Inisialisai |
| forward | Operasi pooling pada input |

**Tabel 2.1.1.1.7** Atribut Kelas FlattenLayer

| Atribut | Tipe Data | Deskripsi |
|---|---|---|
| - | - | - |

**Tabel 2.1.1.1.8** Metode Kelas FlattenLayer

| Metode | Deskripsi |
|---|---|
| forward | Meratakan input menjadi 1D shape |

**Tabel 2.1.1.1.9** Atribut Kelas GlobalAveragePooling2Dlayer

| Atribut | Tipe Data | Deskripsi |
|---|---|---|
| - | - | - |

**Tabel 2.1.1.1.10** Metode Kelas GlobalAveragePooling2Dlayer

| Metode | Deskripsi |
|---|---|

| | |
|---|---|
| forward | Melakukan forward propagation |

**Tabel 2.1.1.1.11** Atribut Kelas DenseLayer

| Atribut | Tipe Data | Deskripsi |
|---|---|---|
| weights | np.ndarray | Bobot FC |
| bias | np.ndarray | bias |

**Tabel 2.1.1.1.12** Metode Kelas DenseLayer

| Metode | Deskripsi |
|---|---|
| __init__ | inisialisasi |
| forward | Menghitung denselayer untuk input |

**Tabel 2.1.1.1.13** Atribut Kelas CustomCNN

| Atribut | Tipe Data | Deskripsi |
|---|---|---|
| layers | list | Daftar Layer Model |

**Tabel 2.1.1.1.14** Metode Kelas CustomCNN

| Metode | Deskripsi |
|---|---|
| __init__ | inisialisasi |
| _load_weights_and_build_layers | Ekstrak bobot dari model keras |
| predict | Melakukan prediksi dengan forward pass pada semua layer |

### 2.1.1.2. Forward Propagation

Forward Propagation pada CNN yang diimplementasikan dimulai dengan memuat bobot atau weight yang sudah dibuat sebelumnya menggunakan keras. Layer akan di rekonstruksi ulang dengan fungsi _load_weights_and_build_layers pada kelas CNN. Disini model yang digunakan sudah predefined karena kita tidak menyimpan arsitektur modelnya tetapi hanya menyimpan bobotnya. Maka dari itu kita perlu mendefinisikan model yang dibuat agar nantinya tidak terjadi kesalahan pada perhitungan.

Setelah semua layer dimuat, langkah selanjutnya adalah melakukan prediksi dengan forward propagation. Prediksi ini dijalankan pada kelas CustomCNN, dimana kita melakukan iterasi pada setiap layernya. Berikut adalah penjelasan yang dilakukan pada tiap layernya.

1. Jika Layer adalah Convolusi, maka langkah yang dilakukan adalah memanggil class Convolution2D dengan metode forward. Metode ini akan menghitung dimensi output, disini ada 2 kemungkinan jika menggunakan padding same atau valid, padding valid adalah dimana paddingnya adalah 0. Setelahnya akan dilakukan loop pada tiap batch input, tiap filter/kernel dan tiap posisi. Untuk setiap posisi (i,j) maka akan mengambil bagian dari input dengan ukuran yang sama dengan filter, lalu akan dikalikan antara bagian tersebut dengan filter, setelahnya akan dijumlahkan pada satu nilai lalu ditambahkan bias.

2. Jika layer adalah Aktivasi maka akan memanggil class Activation Layer dimana akan melakukan cek tipe aktivasi, jika tipenya adalah ReLU maka akan mengembalikan nilai maksimum antara 0 dan input. Jika softmax akan dilakukan perhitungan dengan rumus softmax.

3. Jika layer adalah pooling layer maka akan dilakukan reshape atau menurunkan dimensi, Untuk tiap posisi (i, j), diambil patch kecil (misal 2x2), dihitung nilai maksimum (untuk max pooling) atau rata-rata (untuk average pooling), lalu hasilnya disimpan pada output.

4. Jika layer adalah Flatten Layer maka akan dilakukan pemanggilan terhadap class FlattenLayer dimana disini akan me reduksi input menjadi 1D.

5. Jika layer adalah GlobalAveragePooling jika layer adalah global average pooling maka dihitung rata-rata dari setiap channel yang sering digunakan sebelum dense layer.

6. Jika layer adalah denselayer maka akan melakukan perhitungan output dengan melakukan perkalian dota antara input dan weights ditambah dengan bias.

### 2.1.2. Simple Recurrent Neural Networks (Simple RNN)

#### 2.1.2.1. Deskripsi Kelas

**Tabel 2.1.2.1.1** Atribut Kelas 'RNNCell'

| Atribut | Tipe Data | Deskripsi |
|---|---|---|
| units | int | Banyaknya *hidden units* pada sel RNN |
| W | np.ndarray | Matriks bobot input (input_dim x units) |
| U | np.ndarray | Matriks bobot rekuren (units x units) |
| b | np.ndarray | Vektor bias (units) |

**Tabel 2.1.2.1.2** Metode Kelas 'RNNCell'

| Metode | Deskripsi |
|---|---|
| __init__(weights, units) | Menginisialisasi kelas dengan bobot dan *hidden units* |
| forward(x_t, h_prev) | Menghitung nilai *hidden state* untuk time step |

Kedua Tabel 2.1.2.1.1 dan 2.1.2.1.2 mendefinisikan atribut dan metode yang dimiliki kelas 'RNNCell' yaitu kelas yang mewakili sel dasar dari RNN yang memproses satu langkah waktu (*time step*). Sel ini menerima input saat ini dan *hidden state* sebelumnya untuk menghasilkan *hidden state* baru menggunakan fungsi aktivasi tanh.

**Tabel 2.1.2.1.3** Atribut Kelas 'RNNLayer'

| Atribut | Tipe Data | Deskripsi |
|---|---|---|
| units | int | Banyaknya *hidden units* pada sel RNN |
| return_sequences | bool | Jika 'True', mengembalikan sekuens penuh, jika tidak maka mengembalikan nilai terakhir pada *hidden state* |
| activation | str | Fungsi aktivasi yang digunakan |
| bidirectional | bool | Mengindikasikan apakah *layer* merupakan *bidirectional* |
| rnn_cell | RNNCell | Sel RNN yang digunakan pada *layer* |

**Tabel 2.1.2.1.4** Metode Kelas 'RNNLayer'

| Metode | Deskripsi |
|---|---|
| __init__(weights, config) | Menginisialisasi *layer* RNN dengan bobot dan konfigurasi |
| forward(inputs) | *Forward propagation,* melakukan iterasi untuk tiap *time step* dan memanggil metode forward() pada sel RNN, kemudian mengembalikan *output* berupa sekuens *hidden state* atau nilai terakhir dari *hidden state* |

Kedua Tabel 2.1.2.1.3 dan 2.1.2.1.4 mendefinisikan atribut dan metode yang dimiliki kelas 'RNNLayer' yaitu kelas yang menjalankan RNNCell secara berurutan untuk setiap *time step* dalam input. Kelas ini dapat mengembalikan seluruh urutan *output* atau hanya *output* dari langkah terakhir.

**Tabel 2.1.2.1.5** Atribut Kelas 'BidirectionalRNNLayer'

| Atribut | Tipe Data | Deskripsi |
|---|---|---|
| units | int | Banyaknya *hidden units* pada sel RNN |
| return_sequences | bool | Jika 'True', mengembalikan sekuens penuh, jika tidak maka mengembalikan nilai terakhir pada *hidden state* |
| activation | str | Fungsi aktivasi yang digunakan |
| bidirectional | bool | Mengindikasikan apakah *layer* merupakan *bidirectional* |
| rnn_cell_forward | RNNCell | Sel RNN arah maju yang digunakan pada *layer* |
| rnn_cell_backward | RNNCell | Sel RNN arah mundur yang digunakan pada *layer* |

**Tabel 2.1.2.1.6** Metode Kelas 'BidirectionalRNNLayer'

| Metode | Deskripsi |
|---|---|
| __init__(weights, config) | Menginisialisasi *layer* RNN dengan bobot dan konfigurasi |
| forward(inputs) | *Forward propagation,* melakukan iterasi untuk tiap *time step* dan memanggil metode forward() pada sel RNN, menggabungkan hasil |

| | |
|---|---|
| | kedua arah dengan konkatenasi, kemudian mengembalikan *output* berupa sekuens *hidden state* atau nilai terakhir dari *hidden state* |

Kedua Tabel 2.1.2.1.5 dan 2.1.2.1.6 mendefinisikan atribut dan metode yang dimiliki kelas 'BidirectionalRNNLayer' yaitu kelas yang merupakan versi dua arah dari RNNLayer. Kelas ini melibatkan dua RNNCell, satu untuk membaca urutan dari depan ke belakang (*forward*), dan satu lagi untuk dari belakang ke depan (*backward*). Hasil keduanya digabungkan untuk menangkap konteks dari masa lalu dan masa depan.

**Tabel 2.1.2.1.7** Atribut Kelas 'RNNModel'

| Atribut | Tipe Data | Deskripsi |
|---|---|---|
| layers | list[BaseLayer] | List berisikan *layer* yang digunakan pada model (Embedding, RNN, Dropout, Dense) |
| (inherited) weights | list[np.ndarray] | Bobot tiap *layer* yang diperoleh dari model keras. |
| (inherited) layer_types | list[str] | Nama/tipe dari tiap *layer* |
| (inherited) layer_configs | list[dict] | Konfigurasi tiap *layer* |

**Tabel 2.1.2.1.8** Metode Kelas 'RNNModel'

| Metode | Deskripsi |
|---|---|
| __init__(model_input) | Menginisialisasi kelas kustom RNNModel menggunakan model/data dari kelas RNN milik Keras |
| _build_layers() | Mengonstruksi *custom layers* berdasarkan tipe dan konfigurasi milik model Keras |
| forward(inputs) | *Forward propagation* |

Kedua Tabel 2.1.2.1.7 dan 2.1.2.1.8 mendefinisikan atribut dan metode yang dimiliki kelas 'RNNModel' yaitu model Neural Network yang terdiri dari beberapa *layer* seperti Embedding, RNN, Dropout, dan Dense. Struktur model dibangun secara dinamis dari model Keras yang sudah dilatih. Kelas ini digunakan untuk proses inferensi data *input.*

#### 2.1.2.2. Forward Propagation

Pada *forward propagation,* data *input* diproses secara bertahap melewati berbagai *layer* dalam model untuk menghasilkan *output* akhir. *Forward propagation* dimulai dari objek kelas RNNModel, lalu setiap *layer* akan melakukan proses komputasi menggunakan metode forward() nya masing. Jika *layer* adalah RNNLayer atau BidirectionalLayer, maka akan dilakukan pemanggilan metode forward() untuk setiap *timestep* melalui sel pada RNNCell, yang bertugas menghitung *hidden state* pada tiap *time step*.

Berikut merupakan proses yang terjadi saat *forward propagation* secara lebih rinci:

1. Pemanggilan RNNModel.forward(inputs)
   - Input berupa array 3D dengan format (*batch_size, timesteps, input_dim*) diterima oleh model.
   - Model kemudian melakukan iterasi terhadap semua *layer* yang telah dibangun dan tersimpan di self.layers.
2. Setiap *layer* memanggil metode forward(x)
   - Fungsi forward(x) dari setiap *instance* objek *layer* dipanggil secara berurutan.
   - Hasil *output* dari satu layer menjadi *input* untuk layer berikutnya.
3. Jika *layer* adalah RNNLayer
   - *Layer* memanggil RNNCell.forward(x_t, h_prev) untuk setiap *time step t.*
   - *Hidden state h_t* dikumpulkan menjadi *output* sepanjang waktu.
   - Nilai *h_t* dihitung menggunakan persamaan berikut

     $$h_t = \tanh(W * x_t + U * h_{prev} + b)$$

4. Jika *layer* adalah BidirectionalRNNLayer
   - Dua RNNCell digunakan, satu untuk arah maju (*forward*) dan satu untuk arah mundur (*backward*).
   - Pada arah maju proses dilakukan dari $t$=0 sampai $t$=$T$-1.
   - Pada arah mundur proses dilakukan dari $t$=$T$-1 sampai $t$=0.
   - Hasil kedua arah digabungkan dengan cara konkatenasi menggunakan np.concatenate().

5. *Output*
    - Jika *return_sequences* = True, maka seluruh *output* di setiap *time step* dikembalikan dengan bentuk (*batch_size, timesteps, units* atau *2*units*)
    - Jika *return_sequences* = False, hanya *output* dari *time step* terakhir yang dikembalikan dengan bentuk (*batch_size,, units* atau *2*units*)

### 2.1.3. Long Short Term Memory (LSTM)

#### 2.1.3.1. Deskripsi Kelas

**Tabel 2.1.3.1.1** kelas 'LSTMCell'

| Atribut | Tipe Data | Deskripsi |
|---|---|---|
| input_size | int | Jumlah unit input (fitur / timestep) |
| hidden_size | int | Ukuran dari hidden state dan cell state |
| W_f | np.ndarray | Matrix bobot untuk *forget gate*. |
| W_i | np.ndarray | Matrix bobot untuk *gate input* |
| W_c | np.ndarray | Matrix bobot untuk *candidate cell state* |
| W_o | np.ndarray | Matrix bobot untuk *gate output* |
| b_f | np.ndarray | Bias untuk *forget gate* |
| b_i | np.ndarray | Bias untuk *input gate* |
| b_c | np.ndarray | Bias untuk *candidate sell state* |
| b_o | np.ndarray | Bias untuk *output gate* |
| **Metode** | **Deskripsi** | |
| __init__() | Konstruktor yang menginisialisasi param dan bobot | |
| forward(x,h,c) | Fungsi ini melakukan 1 langkah waktu (timestep) dari LSTM cell, input (x), h (hidden), c(cell), lalu hitung gate f,i,o dan cell candidate state (ĉ) dan menghasilkan h_t,c_t | |

Kelas ini memiliki peran dan fungsi sebagai konsep dari cell yang sudah dijelaskan pada deskripsi permasalahan LSTM. Kelas ini menjadi satuan terkecil untuk forward prop, yang nanti akan dipanggil sekuensial berdasarkan timestep oleh LSTMLayer

**Tabel 2.1.3.1.2** kelas 'LSTMLayer'

| Atribut | Tipe Data | Deskripsi |
|---|---|---|

| | | |
|---|---|---|
| input_size | int | Jumlah fitur pada tiap input timestep. |
| hidden_size | int | Ukuran hidden state dan cell state. |
| sequence_length | int | Panjang urutan input (jumlah timestep). |
| cell | LSTMCell | Objek LSTMCell yang digunakan pada setiap timestep. |
| **Metode** | **Deskripsi** | |
| __init__() | Konstruktor. Menginisialisasi cell dan parameter layer. | |
| forward(x) | Memproses seluruh sequence x (batch_size, sequence_length, input_size) menggunakan LSTMCell secara time step. Output: hiddens dan last_cell. | |

Kelas ini memiliki peran sebagai eksekutor dari tiap cell yang ada di layer tersebut. Dengan memanggil fungsi forward, akan diproses seluruh sekuens menggunakan LSTMCell secara *time step*.

**Tabel 2.1.3.1.3** Kelas 'BidirectionalLSTMLayer'

| **Atribut** | **Tipe Data** | **Deskripsi** |
|---|---|---|
| forward_layer | LSTMLayer | Layer LSTM untuk arah maju (forward). |
| backward_layer | LSTMLayer | Layer LSTM untuk arah mundur (backward). |
| **Metode** | **Deskripsi** | |
| __init__() | Konstruktor. Membuat dua instance LSTMLayer: forward dan backward. | |
| forward(x) | Memproses input x dari kedua arah (maju dan mundur), lalu menggabungkan output hidden state dari keduanya di setiap timestep. Output: (batch_size, seq_len, 2 * hidden_size) | |

Kelas ini meng-*handle* apabila digunakan sebuah LSTM dengan versi 2 arah atau Bidirectional. Cara kerja fungsi ini cukup mirip dengan yang ada di RNN (menggunakan 2 layer LSTM, backward dan forward). Yang membedakan adalah konsep dari cell state, yang mana tidak dimiliki oleh RNN.

**Tabel 2.1.2.1.4** Kelas 'LSTMModel'

| Atribut | Tipe Data | Deskripsi |
|---|---|---|
| layers | list[BaseLayer] | List berisikan *layer* yang digunakan pada model (Embedding, LSTM, Dropout, Dense) |
| (inherited) weights | list[np.ndarray] | Bobot tiap *layer* yang diperoleh dari model keras. |
| (inherited) layer_types | list[str] | Nama/tipe dari tiap *layer* |
| (inherited) layer_configs | list[dict] | Konfigurasi tiap *layer* |
| **Metode** | **Deskripsi** | |
| __init__(model_input) | Menginisialisasi kelas kustom LSTMModel menggunakan model/data dari kelas LSTM milik Keras | |
| _build_layers() | Mengonstruksi *custom layers* berdasarkan tipe dan konfigurasi milik model Keras | |
| forward(inputs) | *Forward propagation* | |

Kelas ini digunakan untuk pembentukan layer layer yang ada di config model milik pustaka *Keras*. Sama seperti yang dimiliki RNN, model ini menginisialisasi kelas kustom LSTMModel menggunakan model/data dari kelas LSTM milik *Keras*, lalu konstruksi *layers* berdasarkan tipe dan konfigurasi milik model *Keras*

#### 2.1.3.2. Forward Propagation

Seperti Forward Propagation pada milik pustaka *Keras*, untuk implementasi forward propagation dari scratch dimulai dengan inisialisasi objek LSTMModel, dimana disini layer akan diciptakan beserta bobot-bobot yang sudah dihasilkan melalui training model *Keras* melalui konfigurasi model tersebut. Begitu fungsi *forward(inputs)* dipanggil. Maka akan dilakukan forward propagation sesuai urutan layer.

Berikut prosesnya:

1. Pemanggilan fungsi forward dari LSTMModel.*forward*(*input)*
    - Input berupa token 2D (batch_size, timesteps) diterima oleh model.
    - Model melakukan iterasi terhadap semua layer yang telah dibangun dan disimpan di self.layers.

- Setiap layer memanggil metode forward(x) secara berurutan

2. Jika *layer* adalah LSTMLayer
   - Layer ini menerima input 3D (batch_size, timesteps, input_dim)
   - Untuk setiap timestep t, layer memanggil LSTMCell.forward(x_t, h_prev, c_prev)
   - Nilai hidden state h_t dan cell state c_t dihitung menggunakan persamaan LSTM:

$$
\begin{aligned}
\_t &= \sigma(W\_f * x\_t + U\_f * h\_prev + b\_f) \\
i\_t &= \sigma(W\_i * x\_t + U\_i * h\_prev + b\_i) \\
o\_t &= \sigma(W\_o * x\_t + U\_o * h\_prev + b\_o) \\
g\_t &= \tanh(W\_c * x\_t + U\_c * h\_prev + b\_c) \\
c\_t &= f\_t * c\_\{t-1\} + i\_t * g\_t \\
h\_t &= o\_t * \tanh(c\_t)
\end{aligned}
$$

3. Jika *layer* adalah BidirectionalLSTMLayer
   - Menggunakan dua LSTMLayer: satu untuk arah maju ($t = 0 \rightarrow T-1$) dan satu untuk arah mundur ($t = T-1 \rightarrow 0$).
   - Kedua arah memproses input dan menghasilkan hidden state masing-masing.
   - Output dari kedua arah digabung dengan konkatenasi
4. *Output*
   - (batch_size, timesteps, hidden_size) jika return_sequences=True
   - (batch_size, hidden_size) jika return_sequences=False

## 2.2. Hasil Pengujian

### 2.2.1. Convolutional Neural Networks (CNN)

#### 2.2.1.1. Pengaruh Jumlah Layer Konvolusi

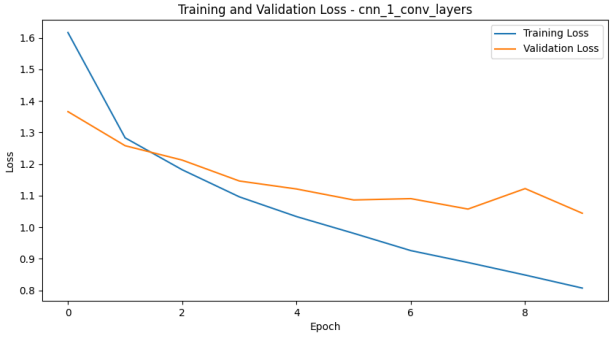
**Tabel 2.2.2.1.1** Hyperparameter Kontrol Pengujian Jumlah *Layer* Konvolusi Berbeda

| Parameter | Nilai |
|---|---|

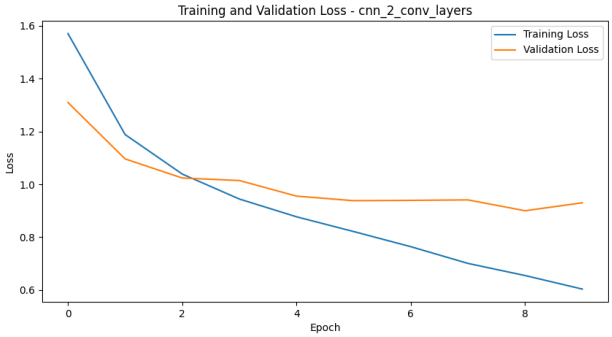
| | |
|---|---|
| Pooling | MaxPooling |
| Kernel Size | (3,3) |
| Filter Awal | 32 |
| Aktivasi | ReLU |
| *Batch Size* | 128 |
| Jumlah Epoch | 10 |
| Optimizer | Adam |
| Jumlah Kelas | 10 |

**Tabel 2.2.2.1.2** Hasil Pengujian Jumlah *Layer* Konvolusi Berbeda

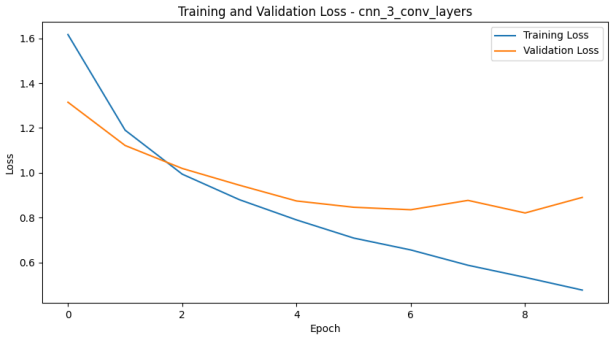
| **Jumlah Layer** | ***Test Loss*** | ***Test Accuracy*** | **F1-*Score*** |
|---|---|---|---|
| 1 | 1.0469 | 0.6391 | 0.1955 |
| 2 | 0.9284 | 0.6899 | 0.1955 |
| 3 | 0.7168 | 0.8863 | 0.1953 |




**Gambar 2.2.1.1.1** Grafik 1 Layer




**Gambar 2.2.1.1.2** Grafik 2 Layer




**Gambar 2.2.1.1.3** Grafik 3 Layer

Berdasarkan hasil pengujian, dapat disimpulkan bahwa dengan menggunakan 3 layer konvolusi akan mendapatkan akurasi yang lebih baik, walaupun F1-Score antara ke-3 nya tidak terlalu jauh, hanya berbeda sedikit saja. Dari grafik juga ditunjukkan bahwa ketiganya menunjukkan adanya overfitting yaitu dimana training loss turun akan tetapi validation lossnya mendatar.

### 2.2.1.2. Pengaruh Banyak Filter per Layer Konvolusi

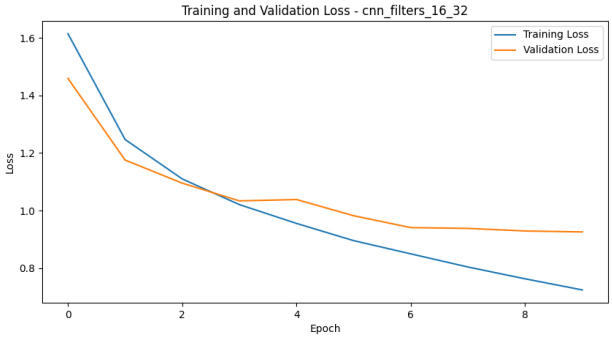
**Tabel 2.2.2.2.1** Hyperparameter Kontrol Pengujian banyak filter per layer Konvolusi Berbeda

| Parameter | Nilai |
|---|---|
| Jumlah Layer Konvolusi | 2 |
| Pooling | MaxPooling |
| Kernel Size | (3,3) |
| Variasi Filter | [16, 32], [32, 64], [64, 128] |
| Aktivasi | ReLU |
| *Batch Size* | 128 |
| Jumlah Epoch | 10 |
| Optimizer | Adam |
| Jumlah Kelas | 10 |

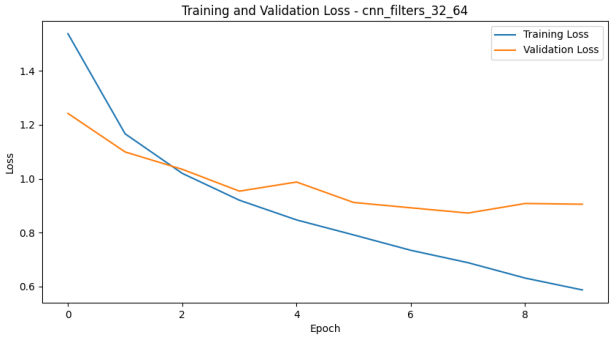
**Tabel 2.2.2.2.2** Hasil Pengujian banyak filter per layer Konvolusi Konvolusi Berbeda

| Kombinasi Filter | *Test Loss* | *Test Accuracy* | F1-*Score* |
|---|---|---|---|
| 16-32 | 0.9330 | 0.6764 | 0.1954 |

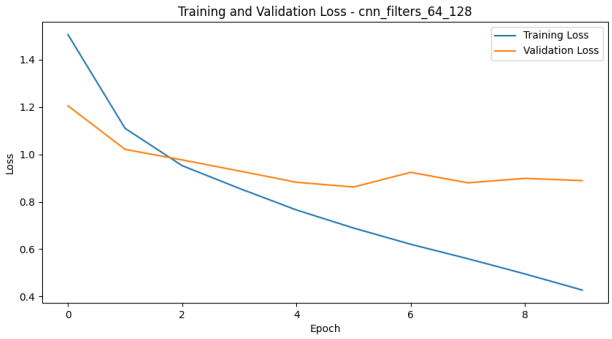
| | | | |
|---|---|---|---|
| 32-64 | 0.9053 | 0.6980 | 0.1956 |
| 64-128 | 0.9144 | 0.7169 | 0.1956 |




**Gambar 2.2.2.2.1** Grafik Konfigurasi 1




**Gambar 2.2.2.2.2** Grafik Konfigurasi 2




**Gambar 2.2.2.2.3** Grafik Konfigurasi 3

Dari data diatas kombinasi filter 32-64 menunjukkan performa terbaik dalam hal test loss, sedangkan kombinasi 64-128 unggul dalam akurasi. Namun, perbedaan antara kedua kombinasi ini tidak signifikan, menunjukkan bahwa peningkatan jumlah filter tidak selalu memberikan peningkatan yang berarti pada semua metrik evaluasi. Pada grafik pun

ditunjukkan bahwa adanya overfitting disetiap kita menambah banyak filter per layer konvolusi dimana pada konfigurasi terakhir gap antara val loss dan train loss nya sangat tinggi

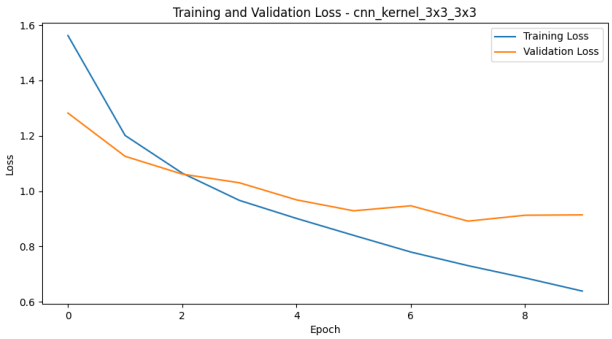
### 2.2.1.3. Pengaruh Ukuran Filter per Layer Konvolusi

**Tabel 2.2.2.3.1** Hyperparameter Kontrol Pengujian Ukuran Filter per Layer Konvolusi Berbeda

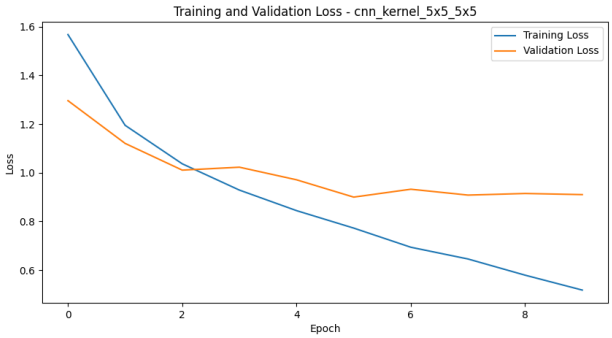
| Parameter | Nilai |
|---|---|
| Jumlah Layer Konvolusi | 2 |
| Pooling | MaxPooling |
| Variasi Kernel | (3x3)-(3x3), (5x5)-(5x5), (3x3)-(5x5) |
| Filter | [32, 64] |
| Aktivasi | ReLU |
| *Batch Size* | 128 |
| Jumlah Epoch | 10 |
| Optimizer | Adam |
| Jumlah Kelas | 10 |

**Tabel 2.2.2.3.2** Hasil Pengujian Ukuran Filter per Layer Konvolusi Berbeda

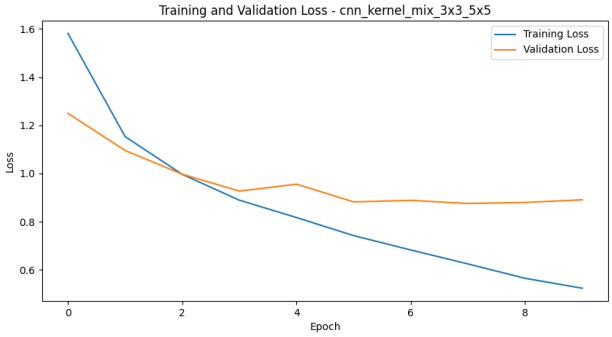
| Kombinasi Variasi | *Test Loss* | *Test Accuracy* | F1-*Score* |
|---|---|---|---|
| (3x3)-(3x3) | 0.9096 | 0.6989 | 0.1954 |
| (5x5)-(5x5) | 0.9093 | 0.7083 | 0.1955 |
| (3x3)-(5x5) | 0.9093 | 0.7082 | 0.1955 |




**Gambar 2.2.2.3.1** Grafik Konfigurasi 1




**Gambar 2.2.2.3.2** Grafik Konfigurasi 2




**Gambar 2.2.2.3.3** Grafik Konfigurasi 3

Berdasarkan Hasil pengujian ukuran filter 3x3 dan 5x5 (dalam 2 layer konvolusi) menunjukkan pengaruh yang minimal terhadap performa model. Ukuran filter 3x3 dan 5x5 pun menunjukkan hanya ada sedikit perubahan pada hasil. Grafik dari ketiganya pun menunjukkan kemiripan. Pada kasus ini ukuran filter tidak terlalu mempengaruhi performa model

### 2.2.1.4. Pengaruh Jenis Pooling Layer

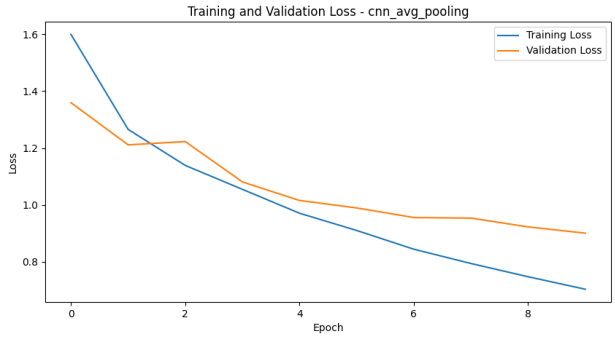
**Tabel 2.2.2.4.1** Hyperparameter Kontrol Pengujian Jenis Pooling Layer Konvolusi Berbeda

| Parameter | Nilai |
|---|---|
| Jumlah Layer Konvolusi | 2 |
| Pooling | MaxPooling, AvgPooling |
| Kernel Size | (3,3) |
| Filter | [32, 64] |
| Aktivasi | ReLU |
| *Batch Size* | 128 |

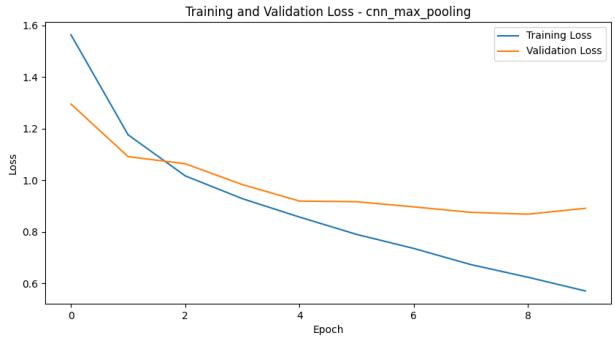
| | |
|---|---|
| Jumlah Epoch | 10 |
| Optimizer | Adam |
| Jumlah Kelas | 10 |

**Tabel 2.2.2.4.2** Hasil Pengujian Jenis Pooling LayerKonvolusi Berbeda

| **Jenis Pooling** | ***Test Loss*** | ***Test Accuracy*** | **F1-*Score*** |
|---|---|---|---|
| Average | 0.9056 | 0.6878 | 0.1956 |
| Max | 0.8969 | 0.6994 | 0.1956 |




**Gambar 2.2.2.4.1** Grafik Konfigurasi 1




**Gambar 2.2.2.4.2** Grafik Konfigurasi 2

Berdasarkan hasil pengujian dan analisis grafik training/validation loss, dapat disimpulkan bahwa max pooling menunjukkan performa yang sedikit lebih unggul dibanding average pooling. Dari segi proses training, max pooling mencapai loss akhir yang lebih rendah dengan kurva yang lebih stabil, menunjukkan kemampuan generalisasi yang lebih baik. Hasil evaluasi akhir juga mengkonfirmasi keunggulan max pooling dengan test loss 0.8969 dan akurasi 0.6994, dibanding average pooling yang mencapai 0.9056 dan 0.6878. Namun, kedua

metode pooling menunjukkan F1-score yang identik (0.1956), mengindikasikan bahwa masalah klasifikasi untuk kelas minoritas belum teratasi oleh perubahan jenis pooling layer ini.

### 2.2.2. Simple Recurrent Neural Networks (Simple RNN)

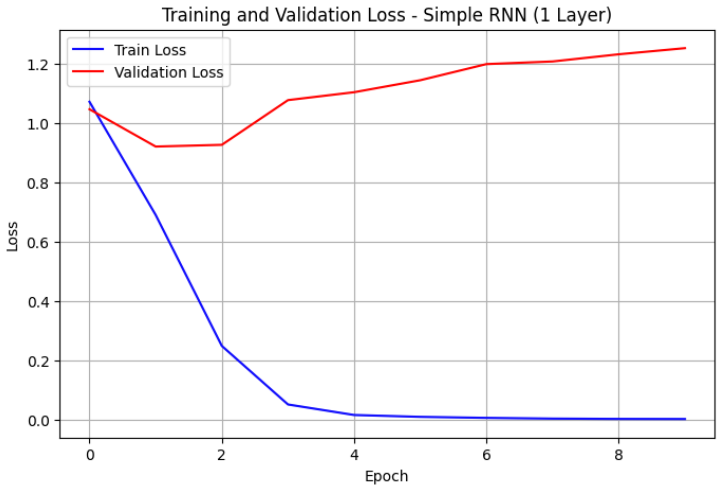
#### 2.2.2.1. Pengaruh Jumlah Layer RNN

**Tabel 2.2.2.1.1** Hyperparameter Kontrol Pengujian Jumlah *Layer* RNN Berbeda

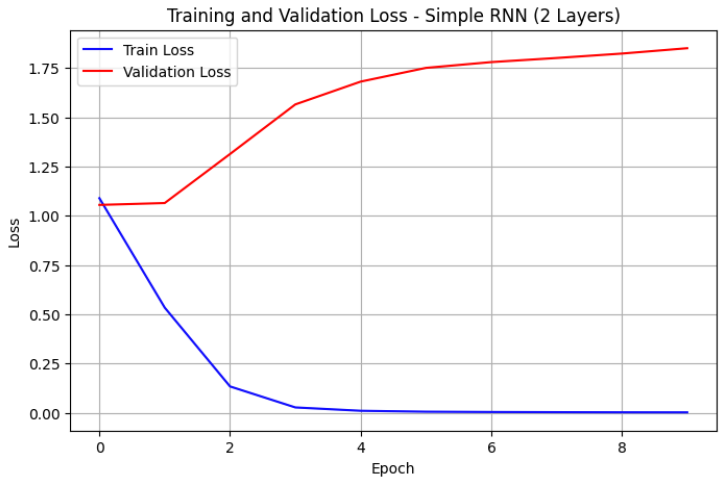
| Parameter | Nilai |
|---|---|
| Jenis *Layer* | *Unidirectional* |
| Banyak *Cell* | 64 |
| *Embeddings* | 128 |
| Jumlah *Epoch* | 10 |
| *Batch Size* | 32 |
| *Time step* | 10 |

**Tabel 2.2.2.1.2** Hasil Pengujian Jumlah *Layer* RNN Berbeda

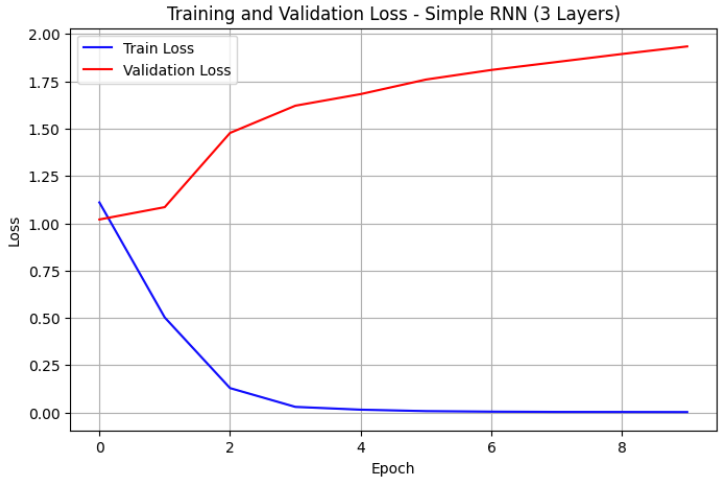
| Jumlah Layer | *Test Loss* | *Test Accuracy* | F1-*Score* |
|---|---|---|---|
| 1 | 1.4573 | 0.5150 | 0.4771 |
| 2 | 1.5559 | 0.4650 | 0.4430 |
| 3 | 1.4984 | 0.5100 | 0.4999 |




**Gambar 2.2.2.1.1** Grafik *Training* and *Validation Loss* - *Simple* RNN (1 *Layer*) Tiap Epoch




**Gambar 2.2.2.1.2** Grafik *Training* and *Validation Loss* - *Simple* RNN (2 *Layers*) Tiap Epoch




**Gambar 2.2.2.1.3** Grafik *Training* and *Validation Loss* - *Simple* RNN (3 *Layers*) Tiap Epoch

Secara teori, semakin banyak *layer* yang dimiliki oleh model maka model tersebut akan semakin baik dalam mempelajari pola hirarkikal yang kompleks. Namun, semakin banyak *layer* yang dimiliki, model akan semakin rentan untuk mengalami *overfitting*.

Berdasarkan percobaan yang kami lakukan terhadap jumlah *layer* satu, dua, dan tiga, didapati bahwa tidak ditemukan adanya korelasi linear antara jumlah *layer* dengan skor F1 yang didapatkan. Nilai skor F1 paling rendah dimiliki oleh model dengan jumlah *layer* dua. Bisa diperhatikan juga dari grafik *training* dan *validation loss* bahwa seluruh model mengalami *overfitting* sehingga menghasilkan performa buruk pada dataset tes. Kami menyimpulkan hal tersebut terjadi karena dataset latih yang digunakan jumlahnya sangat sedikit yaitu hanya 500.

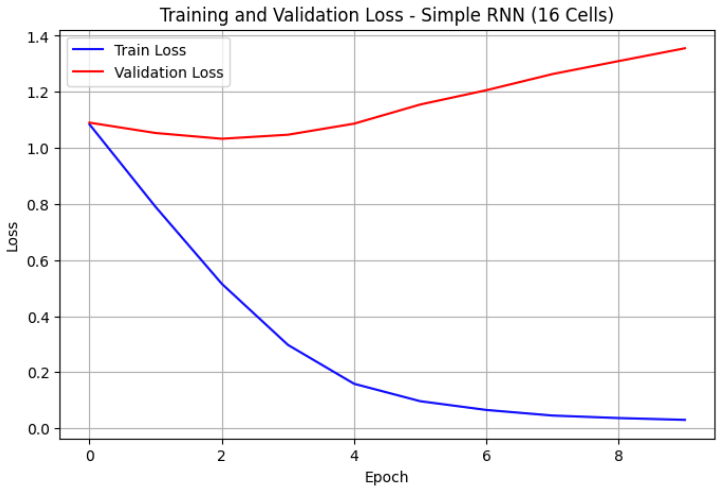
### 2.2.2.2. Pengaruh Banyak Cell RNN per Layer

**Tabel 2.2.2.2.1** Hyperparameter Kontrol Pengujian Banyak *Cell* per *Layer* Berbeda

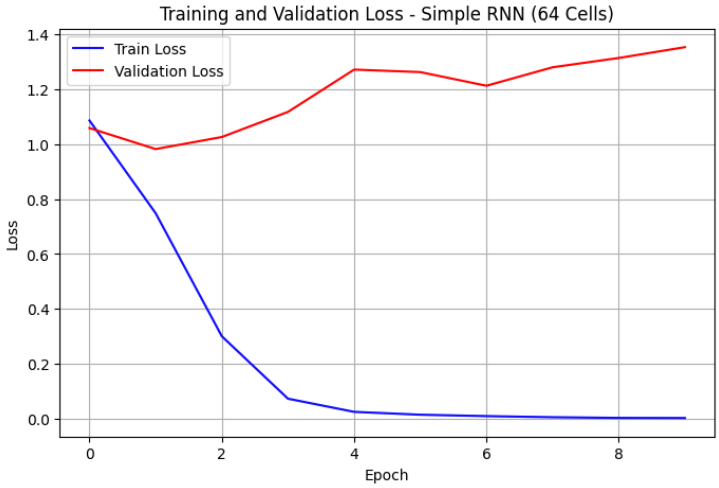
| Parameter | Nilai |
|---|---|
| Jenis *Layer* | *Unidirectional* |
| Jumlah *Layer* | 1 |
| *Embeddings* | 128 |
| Jumlah *Epoch* | 10 |
| *Batch Size* | 32 |
| *Time step* | 10 |

**Tabel 2.2.2.2.2** Hasil Pengujian  Pengujian Banyak *Cell* per *Layer* Berbeda

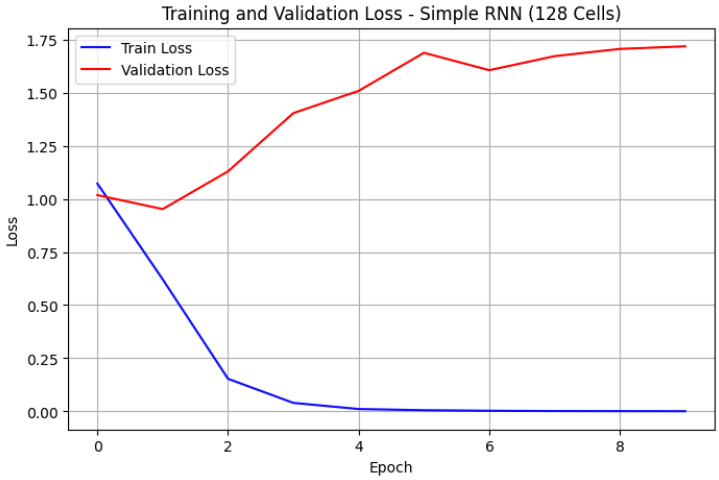
| Banyak *Cell* | *Test Loss* | *Test Accuracy* | F1-*Score* |
|---|---|---|---|
| 16 | 1.2547 | 0.4350 | 0.4120 |
| 64 | 1.2629 | 0.5350 | 0.5259 |
| 128 | 1.5893 | 0.5025 | 0.4682 |




**Gambar 2.2.2.2.1** Grafik *Training* and *Validation Loss* - *Simple* RNN (16 *Cells*) Tiap Epoch




**Gambar 2.2.2.2.2** Grafik *Training* and *Validation Loss* - *Simple* RNN (64 *Cells*) Tiap Epoch




**Gambar 2.2.2.2.3** Grafik *Training* and *Validation Loss* - *Simple* RNN (64 *Cells*) Tiap Epoch

Secara teori, semakin banyak *cell* atau *hidden units* yang dimiliki oleh model maka model tersebut akan semakin baik dalam mempelajari data dengan sekuens yang lebih panjang. Model akan lebih baik dalam mengenali hubungan antara tiap kata pada dataset. Namun, masalah *overfitting* juga rentan terjadi dengan semakin banyaknya jumlah *cell*, terlebih lagi jika dataset latihnya sedikit.

Berdasarkan percobaan yang kami lakukan terhadap jumlah *cell* enam belas, enam puluh empat, dan seratus dua puluh delapan, didapati bahwa tidak ditemukan adanya korelasi linear antara jumlah *layer* dengan skor F1 yang didapatkan. Namun, skor F1 paling tinggi diperoleh oleh model dengan jumlah *cell* enam puluh empat. Kami menyimpulkan bahwa

untuk dataset ini, performa model akan meningkat seiring meningkatnya jumlah *cell* atau *hidden state* sampai taraf tertentu, setelah itu performa model akan menurun karena potensi terjadinya *overfitting*.

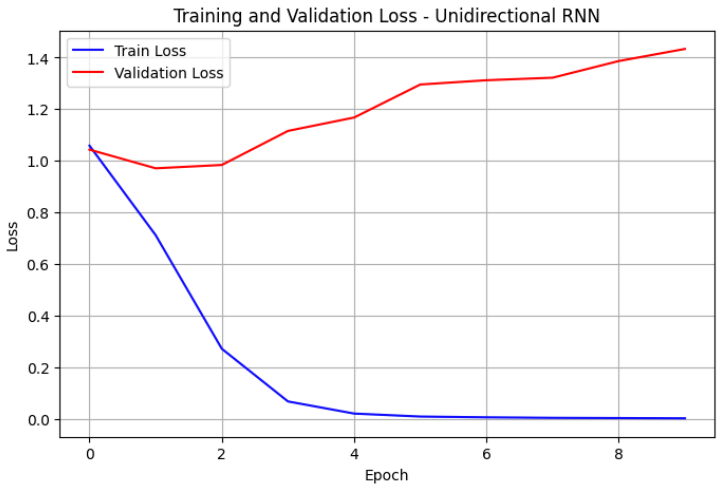
### 2.2.2.3. Pengaruh Jenis Layer RNN Berdasarkan Arah

**Tabel 2.2.2.3.1** Hyperparameter Kontrol Pengujian Jenis *Layer* RNN Berbeda

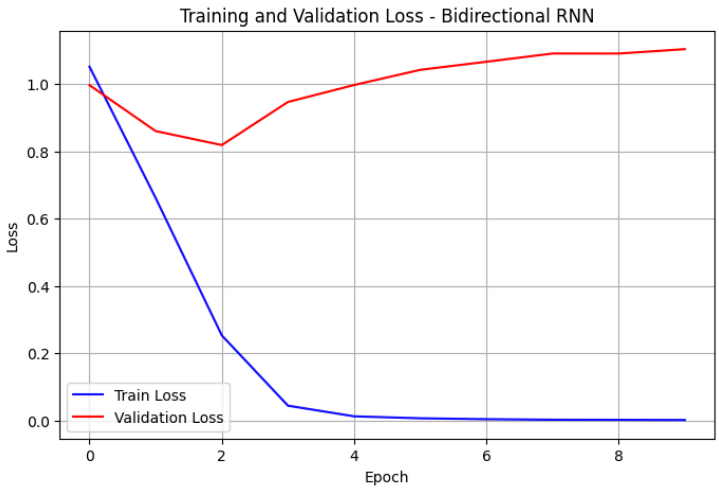
| Parameter | Nilai |
|---|---|
| Jumlah *Layer* | 1 |
| Banyak *Cell* | 64 |
| *Embeddings* | 128 |
| Jumlah *Epoch* | 10 |
| *Batch Size* | 32 |
| *Time step* | 10 |

**Tabel 2.2.2.3.2** Hasil Pengujian  Pengujian Jenis *Layer* RNN Berbeda

| Jenis *Layer* | *Test Loss* | *Test Accuracy* | F1-*Score* |
|---|---|---|---|
| Unidirectional | 1.4139 | 0.5075 | 0.4927 |
| Bidirectional | 1.1158 | 0.5750 | 0.5426 |




**Gambar 2.2.2.3.1** Grafik *Training* and *Validation Loss* - *Simple* RNN (Unidirectional) Tiap Epoch




**Gambar 2.2.2.3.2** Grafik *Training* and *Validation Loss* - *Simple* RNN (Bidirectional) Tiap Epoch

Secara teori, model *bidirectional* akan lebih cocok dalam kasus ini karena relasi antar kata bekerja dua arah. Sebuah kata yang muncul terlebih dahulu bisa saja mempengaruhi kemunculan kata setelahnya atau kemunculannya yang dipengaruhi oleh kata setelahnya. Model *bidirectional* yang mampu menangkap konteks dari kedua arah diprediksi akan menghasilkan performa yang lebih baik.

Berdasarkan percobaan yang kami lakukan terhadap model *unidirectional* dan juga *bidirectional*, didapati bahwa model *bidirectional* menghasilkan skor F1 dan juga akurasi yang lebih tinggi, hal ini sesuai dengan teorinya. Dapat disimpulkan bahwa model *bidirectional* menghasilkan hasil yang lebih baik untuk kasus analisis sentimen karena bentuk data yang berupa kata dalam kalimat.

### 2.2.3. Long Short Term Memory (LSTM)

#### 2.2.3.1. Pengaruh Jumlah Layer LSTM

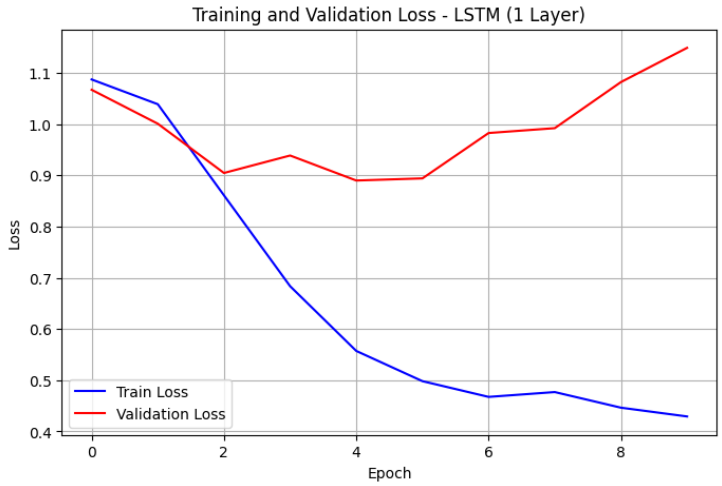
**Tabel 2.2.3.1.1** Hyperparameter Kontrol Pengujian Jumlah *Layer* LSTM Berbeda

| Parameter | Nilai |
|---|---|
| Jenis *Layer* | *Unidirectional* |
| Banyak *Cell* | 32 |
| *Embeddings* | 50 |
| Jumlah *Epoch* | 10 |

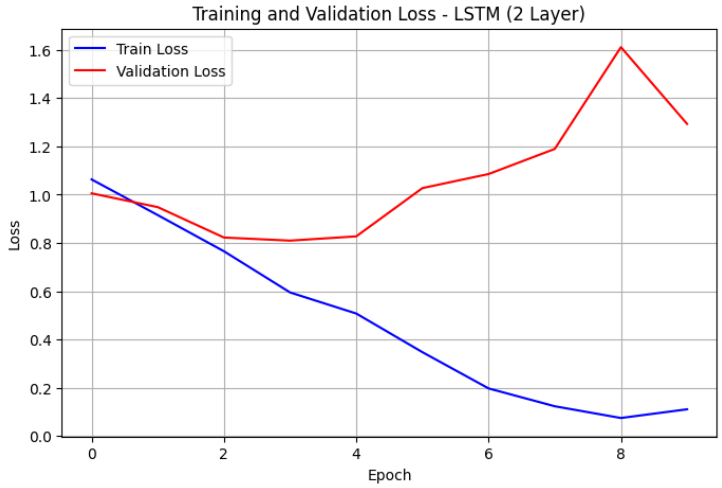
| | |
|---|---|
| *Batch Size* | 32 |
| *Time step* | 30 |

**Tabel 2.2.3.3.2** Hasil Pengujian Jumlah *Layer* LSTM Berbeda

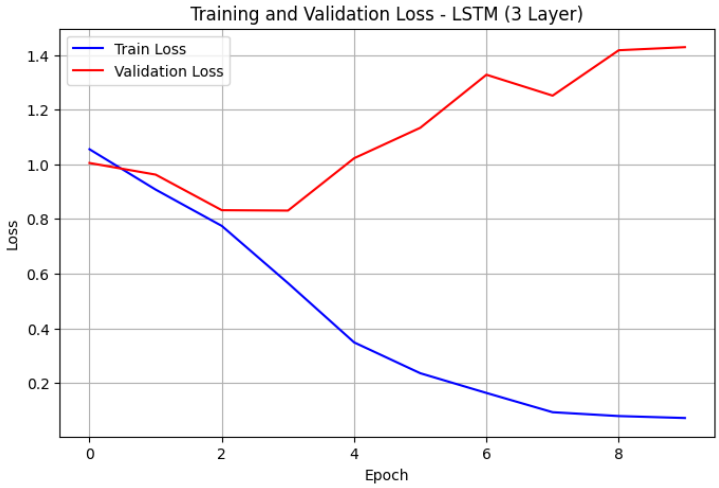
| Jumlah Layer | *Test Loss* | *Test Accuracy* | F1-*Score* |
|---|---|---|---|
| 1 | 0.7830 | 0.6250 | 0.4812 |
| 2 | 1.2800 | 0.4850 | 0.4878 |
| 3 | 1.0573 | 0.5500 | 0.5583 |




**Gambar 2.2.3.1.1** Grafik *Training* dan *Validation Loss* - LSTM 1 Layer




**Gambar 2.2.3.1.2** Grafik *Training* dan *Validation Loss* - LSTM 2 Layer




**Gambar 2.2.3.1.3** Grafik *Training* dan *Validation Loss* - LSTM 3 Layer

Hasil menunjukkan bahwa model dengan 1 layer LSTM memberikan akurasi dan F1-score yang relatif baik (0.6250 dan 0.4812), namun performa justru menurun saat jumlah layer ditambah menjadi 2 (F1-score turun menjadi 0.4878) dan baru sedikit membaik pada 3 layer (F1-score 0.5583). Secara teori, penambahan layer seharusnya memperkaya kemampuan representasi model terhadap urutan data, namun hal ini juga meningkatkan kompleksitas dan risiko overfitting, terutama jika jumlah data terbatas atau jumlah epoch rendah.

Jumlah data latih yang terbatas (hanya 500 sampel) membuat model sulit mempelajari pola secara menyeluruh, sementara ukuran data uji yang hampir setara menyebabkan evaluasi menjadi lebih sensitif terhadap kesalahan prediksi. Idealnya, penambahan layer perlu diimbangi dengan strategi pelatihan yang tepat seperti penyesuaian learning rate, penambahan dropout, peningkatan jumlah epoch, serta penggunaan dataset yang lebih besar dan seimbang.

### 2.2.3.2. Pengaruh Banyak Cell LSTM per Layer

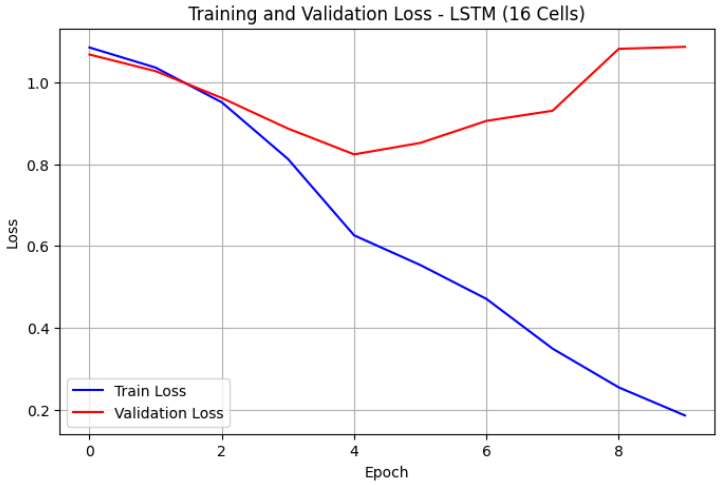
**Tabel 2.2.3.2.1** Hyperparameter Kontrol Pengujian Banyak *Cell* per *Layer* Berbeda

| Parameter | Nilai |
|---|---|
| Jenis *Layer* | *Unidirectional* |
| Jumlah *Layer* | 1 |

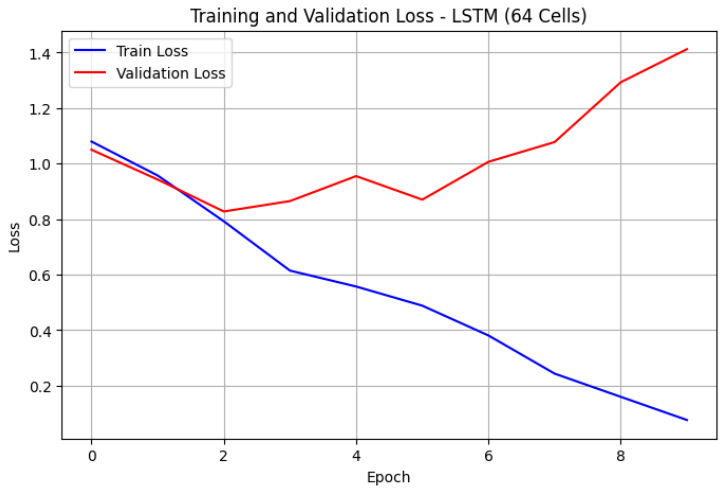
| | |
|---|---|
| *Embeddings* | 50 |
| Jumlah *Epoch* | 10 |
| *Batch Size* | 32 |
| *Time step* | 30 |

**Tabel 2.2.3.2.2** Hasil Pengujian  Pengujian Banyak *Cell* per *Layer* Berbeda

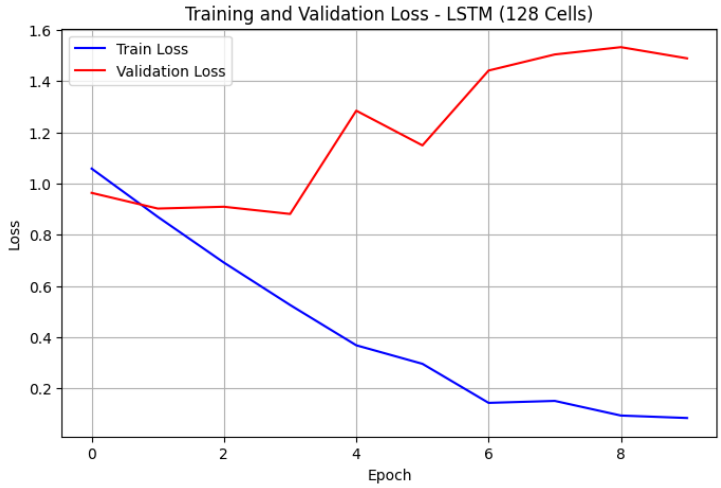
| **Banyak *Cell*** | ***Test Loss*** | ***Test Accuracy*** | **F1-*Score*** |
|---|---|---|---|
| 16 | 0.9258 | 0.5875 | 0.5706 |
| 64 | 1.0033 | 0.5425 | 0.5365 |
| 128 | 1.1055 | 0.5250 | 0.5153 |




**Gambar 2.2.3.2.1** Grafik *Training* dan *Validation Loss* - LSTM 16 Cells




**Gambar 2.2.3.2.2** Grafik *Training* dan *Validation Loss* - LSTM 64 Cells




**Gambar 2.2.3.2.3** Grafik *Training* dan *Validation Loss* - LSTM 128 Cells

Hasil menunjukkan bahwa peningkatan jumlah cell dari 16 ke 128 justru menyebabkan penurunan performa model, di mana F1-score tertinggi dicapai saat menggunakan 16 cell (0.5706), sementara performa menurun pada 64 dan 128 cell (menjadi 0.5365 dan 0.5153). Secara teori, penambahan jumlah cell seharusnya meningkatkan kapasitas model dalam menangkap pola urutan yang kompleks karena semakin banyak memori internal yang tersedia. Namun, hal ini juga meningkatkan jumlah parameter yang harus dipelajari, sehingga tanpa data pelatihan yang cukup, model cenderung mengalami overfitting. Oleh karena itu, performa terbaik justru dicapai dengan model yang lebih sederhana, menunjukkan pentingnya

keseimbangan antara kapasitas model dan ketersediaan data serta strategi pelatihan yang memadai.

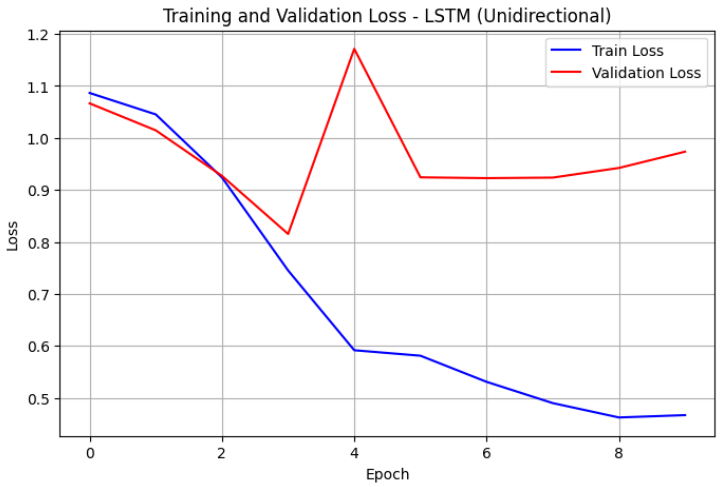
### 2.2.3.3. Pengaruh Jenis Layer LSTM Berdasarkan Arah

**Tabel 2.2.3.3.1** Hyperparameter Kontrol Pengujian Jenis *Layer* LSTM Berbeda

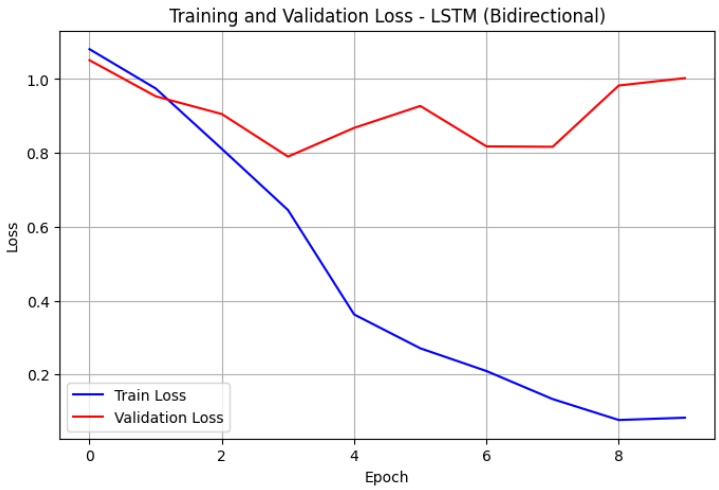
| Parameter | Nilai |
|---|---|
| Jumlah *Layer* | 1 |
| Banyak *Cell* | 32 |
| *Embeddings* | 50 |
| Jumlah *Epoch* | 10 |
| *Batch Size* | 32 |
| *Time step* | 30 |

**Tabel 2.2.3.3.2** Hasil Pengujian  Pengujian Jenis *Layer* LSTM Berbeda

| Jenis *Layer* | *Train Loss* | *Test Accuracy* | F1-*Score* |
|---|---|---|---|
| Unidirectional | 0.8428 | 0.6125 | 0.4727 |
| Bidirectional | 0.7597 | 0.6850 | 0.6833 |




**Gambar 2.2.3.3.1** Grafik *Training* dan *Validation Loss* - LSTM Unidirectional




**Gambar 2.2.3.3.2** Grafik *Training* dan *Validation Loss* - LSTM Bidirectional

Hasil menunjukkan bahwa model bidirectional secara signifikan mengungguli unidirectional, dengan F1-score meningkat dari 0.4727 menjadi 0.6833 dan akurasi dari 0.6125 menjadi 0.6850. Secara teori, LSTM bidirectional mampu memproses informasi dari dua arah (maju dan mundur), sehingga dapat menangkap konteks sekuens yang lebih kaya dibandingkan LSTM unidirectional yang hanya membaca dari satu arah. Ini sangat berguna terutama untuk tugas-tugas di mana informasi masa depan dalam urutan juga relevan terhadap prediksi saat ini. Penurunan train loss pada bidirectional (dari 0.8428 menjadi 0.7597) juga menunjukkan bahwa model belajar lebih baik terhadap pola dalam data. Meskipun model bidirectional memiliki kompleksitas dan jumlah parameter yang lebih besar, hasil ini menunjukkan bahwa dalam konteks data dan konfigurasi saat ini, kompleksitas tersebut justru memberikan manfaat dalam meningkatkan generalisasi model terhadap data uji, tanpa overfitting yang signifikan.

# Bagian 3.
# Kesimpulan dan Saran

## 3.1. Kesimpulan

### a. Convolutional Neural Networks (CNN)

Performa model CNN sangat dipengaruhi oleh arsitekturnya, walaupun hyperparameter mempengaruhi performa akan tetapi dari pengujian yang dilakukan dengan F1-Score sebagai benchmark didapatkan bahwa tuning hyperparameter tidak membuat performa dari model naik secara signifikan.

Meskipun terjadi peningkatan pada nilai *test accuracy* dan penurunan *test loss* saat jumlah layer konvolusi atau jumlah filter per layer ditambah, namun F1-Score cenderung stagnan di angka sekitar 0.1955.Penambahan jumlah layer konvolusi hingga tiga lapis memang memberikan peningkatan akurasi yang lebih signifikan, namun juga menyebabkan overfitting. Ukuran kernel yang digunakan pada layer konvolusi juga tidak menunjukkan pengaruh besar terhadap performa. Baik penggunaan kernel (3x3), (5x5), maupun kombinasi keduanya memberikan hasil F1-Score yang hampir identik, dengan *test loss* dan *accuracy* yang sangat mirip. Ini menunjukkan bahwa variasi ukuran filter saja tidak cukup untuk meningkatkan kinerja model secara signifikan. Terakhir, jenis pooling layer memberikan pengaruh kecil terhadap performa. MaxPooling sedikit lebih unggul dibanding AveragePooling dalam hal *test loss* dan *accuracy*, namun keduanya tetap memberikan nilai F1-Score yang identik.

Secara keseluruhan, hasil ini mengindikasikan bahwa meskipun tuning arsitektur CNN seperti jumlah layer, jumlah filter, ukuran kernel, dan jenis pooling dapat memberikan peningkatan kecil pada akurasi, hal tersebut belum cukup untuk meningkatkan F1-Score secara signifikan.

### b. Recurrent Neural Networks (RNN)

Performa model RNN sangat dipengaruhi oleh struktur arsitektur, khususnya jumlah *layer* dan jumlah *cell* (*hidden units*). Meskipun secara teori penambahan *layer* dan *cell* dapat membantu model mempelajari pola yang lebih kompleks dan hubungan sekuensial yang panjang, percobaan menunjukan bahwa peningkatan tersebut tidak

selalu berbanding lurus dengan peningkatan performa. Dalam konteks dataset yang terbatas, model dengan dua *layer* dan jumlah cell yang terlalu banyak justru rentan mengalami *overfitting* sehingga menurunkan akurasi pada data uji. Model dengan jumlah *cell* moderat (sekitar 64) memberikan hasil terbaik, menunjukan bahwa ada titik optimal di mana model mampu belajar efektif tanpa mengalami *overfitting*.

Selain itu, model RNN *bidirectional* terbukti lebih unggul dibandingkan model *unidirectional*, terutama dalam tugas yang membutuhkan pemahaman konteks dua arah seperti analisis sentimen. Kemampuan *bidirectional* untuk menangkap informasi dari kedua arah sekuens membuatnya lebih tepat dalam menangani hubungan antar kaya yang saling mempengaruhi satu sama lain. Oleh karena itu, arsitektur RNN yang ringan namun *bidirectional* dengan jumlah *layer* dan *cell* yang seimbang menjadi pilihan paling optimal untuk kondisi dengan data pelatihan terbatas dan kompleksitas tugas yang memerlukan pemahaman konteks menyeluruh.

**c. Long Short Term Memory (LSTM)**

LSTM sangat dipengaruhi oleh arsitektur model. Jumlah layer dan cell yang terlalu banyak justru menurunkan performa akibat overfitting, terutama dengan data latih yang terbatas. Sebaliknya, model sederhana dengan 1 layer dan sedikit cell (16) menunjukkan hasil terbaik. Selain itu, LSTM bidirectional secara konsisten unggul dibandingkan unidirectional karena kemampuannya menangkap konteks dari dua arah. Secara keseluruhan, arsitektur yang ringan namun efektif, seperti LSTM bidirectional dengan kompleksitas minimal, lebih optimal dalam kondisi data dan pelatihan terbatas.

## 3.2. Saran

Sebagai pengembangan lebih lanjut dari tugas ini, disarankan untuk mengimplementasikan proses *backpropagation* dan optimasi parameter guna melatih model secara *end-to-end*, sehingga performa model dapat diuji secara realistis. Selain itu, pengujian terhadap berbagai jenis data (seperti citra, teks, dan data deret waktu) dapat memberikan pemahaman yang lebih komprehensif mengenai keunggulan dan

keterbatasan masing-masing arsitektur (CNN, RNN, LSTM) dalam konteks yang berbeda.

Untuk mendalami topik, eksplorasi lebih lanjut mengenai model dan arsitektur lain seperti GRU, ResNet, dan Transformer, serta menerapkan teknik regularisasi yang lebih baik untuk mencegah *overfitting* sangat disarankan. Evaluasi performa menggunakan metrik yang lebih beragam juga akan memberikan gambaran yang lebih utuh terhadap efektivitas model. Penelitian lanjutan juga dapat diarahkan pada aspek efisiensi komputasi dan interpretabilitas model, yang semakin penting dalam pengembangan aplikasi nyata berbasis *deep learning*.

# Pembagian Tugas

Tabel 4.1 Pembagian Tugas

| NIM | Nama | Tugas |
|---|---|---|
| 13522129 | Hugo Sabam Augusto | LSTM |
| 13522136 | Muhammad Zaki | CNN |
| 13522137 | Ahmad Rafi Maliki | RNN |

# Referensi


[1] TensorFlow, *Dense layer – Keras*. [Daring]. Tersedia: https://keras.io/api/layers/core_layers/dense/ [Diakses: 29 Mei 2025].

[2] TensorFlow, *Dropout layer – Keras*. [Daring]. Tersedia: https://keras.io/api/layers/regularization_layers/dropout/ [Diakses: 29 Mei 2025].

[3] TensorFlow, *LSTM layer – Keras*. [Daring]. Tersedia: https://keras.io/api/layers/recurrent_layers/lstm/ [Diakses: 29 Mei 2025].

[4] TensorFlow, *Embedding layer – Keras*. [Daring]. Tersedia: https://keras.io/api/layers/core_layers/embedding/ [Diakses: 29 Mei 2025].

[5] Datacamp, "What is tokenization?" 19 Juli 2022. [Daring]. Tersedia: https://www.datacamp.com/blog/what-is-tokenization [Diakses: 29 Mei 2025].

[6] Tim Pengajar IF 3270, "Recurrent Neural Network," *Edunex*, [Daring]. Tersedia: https://edunexcontentprodhot.blob.core.windows.net/edunex/2025/71372-Machine-Learning/343463-ANN-RNN/file/1745204854393_IF3270-Mgg09-RNN-part1?sv=2024-11-04&spr=https&st=2025-04-21T03%3A07%3A34Z&se=2027-04-21T03%3A07%3A34Z&sr=b&sp=r&sig=UhQ0BG6Fm4ar0SEu8raY2Vxbmj%2B6PxdTGE%2Fs7K0JyFM%3D&rsct=application%2Fpdf [Diakses: 29 Mei 2025].

# Lampiran

Pranala *repository* GitHub

https://github.com/miannetopokki/IF3270_Tubes2_MachineLearning