

**IF3270 PEMBELAJARAN MESIN
LAPORAN TUGAS BESAR 2**

CNN, Simple RNN, dan LSTM



Disusun oleh:

Kelompok 16

- | | | |
|----|-------------------------|----------|
| 1. | Maulvi Ziadinda Maulana | 13522122 |
| 2. | Nicholas Reymond Sihite | 13522144 |
| 3. | Albert Ghazaly | 13522150 |

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025**

Daftar Isi

Daftar Isi.....	2
Daftar Tabel.....	3
Daftar Gambar.....	4
Bagian 1. Deskripsi Persoalan.....	5
Bagian 2. Pembahasan.....	6
2.1. Penjelasan Implementasi.....	6
2.1.1. Deskripsi Kelas, Atribut, dan Metode.....	6
2.1.1.1. CNN.....	6
2.1.1.2. Simple RNN.....	9
2.1.1.3. LSTM.....	15
2.1.2. Penjelasan Forward Propagation.....	18
2.1.2.1. CNN.....	18
2.1.2.2. Simple RNN.....	20
2.1.2.3. LSTM.....	22
2.2. Hasil Pengujian.....	25
2.2.1. Pengujian CNN.....	25
2.2.1.1. Pengaruh Jumlah Layer Konvolusi.....	25
2.2.1.2. Pengaruh Banyak Filter per Layer Konvolusi.....	27
2.2.1.3. Pengaruh Ukuran Filter per Layer Konvolusi.....	30
2.2.1.4. Pengaruh Jenis Pooling Layer.....	32
2.2.1.5. Perbandingan dengan CNN Scratch.....	34
2.2.2. Pengujian Simple RNN.....	35
2.2.2.1. Pengaruh Jumlah Layer RNN.....	35
2.2.2.2. Pengaruh Banyak Cell RNN per Layer.....	37
2.2.2.3. Pengaruh Jenis Layer RNN berdasarkan Arah.....	39
2.2.2.4. Perbandingan RNN Keras dengan Scratch Implementation.....	40
2.2.3. Pengujian LSTM.....	41
2.2.3.1. Pengaruh Jumlah Layer LSTM.....	41
2.2.3.2. Pengaruh Banyak Cell LSTM per Layer.....	43
2.2.3.3. Pengaruh Jenis Layer LSTM berdasarkan Arah.....	45
2.2.3.4. Perbandingan LSTM Keras dengan Scratch Implementation.....	47
Bagian 3. Kesimpulan dan Saran.....	50
3.1. Kesimpulan.....	50
3.2. Saran.....	50
Pembagian Tugas.....	52
Referensi.....	53

Daftar Tabel

Tabel 2.1.1.1.1. Deskripsi Kelas, Atribut, dan Metode CNN.....	6
Tabel 2.1.1.1.1. Deskripsi Kelas, Atribut, dan Metode Simple RNN.....	9
Tabel 2.1.1.3.1. Deskripsi Kelas, Atribut, dan Metode LSTM.....	15
Tabel 2.2.1.1.1. Pengujian Pengaruh Jumlah Layer Konvolusi.....	25
Tabel 2.2.1.2.1. Pengujian Pengaruh Banyak Filter per Layer Konvolusi.....	27
Tabel 2.2.1.3.1. Pengujian Pengaruh Ukuran Filter per Layer Konvolusi.....	30
Tabel 2.2.1.4.1. Pengujian Pengaruh Jenis Pooling Layer.....	32
Tabel 2.2.1.4.1. Perbandingan dengan CNN Scratch.....	34
Tabel 2.2.2.1.1. Pengujian Pengaruh Jumlah Layer RNN.....	35
Tabel 2.2.2.2.1. Pengujian Banyak Cell RNN per Layer.....	37
Tabel 2.2.2.3.1. Pengujian Pengaruh Jenis Layer RNN berdasarkan Arah.....	39
Tabel 2.2.2.4.1. Perbandingan Simple RNN Scratch dan Keras.....	41
Tabel 2.2.3.1.1. Pengujian Pengaruh Jumlah Layer LSTM.....	41
Tabel 2.2.3.2.1. Pengujian Pengaruh Banyak Cell LSTM per Layer.....	43
Tabel 2.2.3.3.1. Pengujian Pengaruh Jenis Layer LSTM berdasarkan Arah.....	45

Daftar Gambar

Gambar 2.2.2.1.1. Perbandingan Validation Loss dan Training Loss untuk Semua Model Variasi Banyak Layer Simple RNN.....	37
Gambar 2.2.2.3.1. Perbandingan Validation Loss dan Training Loss untuk Semua Model Variasi Directionality Simple RNN.....	40
Gambar 2.2.3.3.1. Loss dan Accuracy Curves Pengujian Pengaruh Jenis Layer LSTM berdasarkan Arah.....	46
Gambar 2.2.3.3.2. Performance Timing Comparison Pengujian Pengaruh Jenis Layer LSTM berdasarkan Arah.....	46
Gambar 2.2.3.3.3. Model Size Comparison Pengujian Pengaruh Jenis Layer LSTM berdasarkan Arah.....	47

Bagian 1. Deskripsi Persoalan

CNN adalah ANN yang dirancang khusus untuk data *grid* seperti gambar menggunakan *layer* konvolusi untuk mengekstrak fitur dan *pooling* untuk reduksi dimensi. *Neural network* ini memproses data untuk mengenali pola visual data tersebut. CNN sangat efektif untuk klasifikasi gambar dan deteksi objek.

RNN adalah ANN yang cocok untuk data sekuensial karena memiliki koneksi berulang yang berfungsi sebagai memori untuk informasi sebelumnya. Arsitektur ini memungkinkannya memproses *input* langkah demi langkah sambil mempertimbangkan konteks dari masa lalu. RNN umumnya digunakan dalam pemrosesan bahasa alami (NLP) dan analisis deret waktu.

LSTM adalah varian RNN yang mampu menangani dependensi jangka panjang menggunakan sel memori dan gerbang (*gates*). Gerbang-gerbang tersebut mengatur aliran informasi serta memutuskan apa yang harus diingat dan dilupakan. Karena kemampuan tersebut, LSTM unggul dalam tugas-tugas kompleks seperti penerjemahan bahasa dan pengenalan suara.

Pada Tugas Besar 2 IF3270 - Pembelajaran Mesin 2025, mahasiswa diminta untuk mengimplementasikan CNN, RNN, dan LSTM dari awal (*from scratch*) serta melakukan analisis terhadap penggunaan *hyperparameter*-nya. Pengerjaan tugas ini diharapkan dapat menambah wawasan dan pengetahuan mahasiswa mengenai CNN, RNN, dan LSTM.

Bagian 2. Pembahasan

2.1. Penjelasan Implementasi

2.1.1. Deskripsi Kelas, Atribut, dan Metode

2.1.1.1. CNN

Berikut merupakan tabel yang berisi deskripsi kelas, atribut, dan metode yang digunakan dalam implementasi CNN dari *scratch*.

Tabel 2.1.1.1.1. Deskripsi Kelas, Atribut, dan Metode CNN

Kelas CNNScratch	
Atribut	<div>layers</div> <div>Atribut ini berfungsi menyimpan data <i>layer-layer</i> yang digunakan pada model CNN.</div>
Metode	<div>transfer_weights(keras_model)</div> <div>Prosedur ini digunakan untuk memindahkan bobot hasil pelatihan model CNN Keras (<i>keras_model</i>) ke model CNN dari <i>scratch</i>.</div>
	<div>predict(input, batch_size)</div> <div>Fungsi ini digunakan untuk melakukan prediksi terhadap data <i>input</i> dengan <i>batch_size</i> tertentu (<i>default</i>-nya adalah 32).</div>
Kelas LayerScratch	
Atribut	-
Metode	<div>forward(input)</div> <div>Fungsi ini adalah fungsi abstrak yang bertujuan melakukan <i>forward propagation</i> dan akan diimplementasikan oleh turunan kelas ini.</div>
Kelas InputScratch	
Atribut	<div>input_shape</div> <div>Atribut ini berfungsi menyimpan bentuk data <i>input</i>.</div>
Metode	<div>forward(input)</div> <div>Fungsi ini adalah implementasi dari fungsi abstrak dari LayerScratch, tetapi hanya mengembalikan kembali <i>input</i>.</div>
Kelas Conv2DScratch	
Atribut	<div>filters</div> <div>Atribut ini berfungsi menyimpan berapa banyak <i>filter</i> yang akan digunakan pada saat <i>forward propagation</i>.</div>

	kernel_size
	Atribut ini berfungsi menyimpan ukuran kernel.
	strides
	Atribut ini berfungsi menyimpan <i>stride</i> yang akan digunakan pada saat <i>forward propagation</i> . Dapat menerima <i>tuple</i> (<i>int</i> , <i>int</i>) yang menyatakan <i>stride</i> dalam kolom dan baris.
	activation
	Atribut ini berfungsi menyimpan fungsi aktivasi yang akan digunakan pada hasil akhir <i>layer</i> ini. Fungsi aktivasi yang tersedia adalah 'relu' (ReLU), 'sigmoid' (<i>sigmoid</i>), dan 'softmax' (<i>softmax</i>).
	padding
Metode	Atribut ini berfungsi menyimpan <i>padding</i> yang dikenakan pada <i>input</i> pada saat <i>forward propagation</i> . Padding yang dapat diterima adalah 'valid' (tidak ada <i>padding</i>) dan 'same' (dikenakan <i>padding</i> sehingga dimensi <i>output</i> -nya sama dengan <i>input</i>).
	weights
	Atribut ini berfungsi menyimpan bobot <i>layer</i> ini.
	bias
	Atribut ini berfungsi menyimpan <i>bias layer</i> ini.
	set_weights(weights)
	Fungsi ini berfungsi memperbarui <i>layer</i> agar menggunakan bobot dari <i>weights</i> .
	forward(input)
	Fungsi ini adalah implementasi dari fungsi abstrak dari <i>LayerScratch</i> yang bertujuan melakukan <i>forward propagation</i> pada <i>convolution stage</i> .
Kelas MaxPooling2DScratch	
Atribut	pool_size
	Atribut ini berfungsi menyimpan ukuran <i>pooling</i> yang akan dilakukan.
	strides
	Atribut ini berfungsi menyimpan <i>stride</i> yang akan digunakan pada saat <i>forward propagation</i> .

	padding Atribut ini berfungsi menyimpan <i>padding</i> yang dikenakan pada <i>input</i> pada saat <i>forward propagation</i> .
Metode	forward(input) Fungsi ini adalah implementasi dari fungsi abstrak dari LayerScratch yang bertujuan melakukan <i>forward propagation</i> pada <i>pooling stage</i> , khususnya <i>max pooling</i> .
Kelas AvgPooling2DScratch	
Atribut	pool_size Atribut ini berfungsi menyimpan ukuran <i>pooling</i> yang akan dilakukan.
	strides Atribut ini berfungsi menyimpan <i>stride</i> yang akan digunakan pada saat <i>forward propagation</i> .
	padding Atribut ini berfungsi menyimpan <i>padding</i> yang dikenakan pada <i>input</i> pada saat <i>forward propagation</i> .
Metode	forward(input) Fungsi ini adalah implementasi dari fungsi abstrak dari LayerScratch yang bertujuan melakukan <i>forward propagation</i> pada <i>pooling stage</i> , khususnya <i>average pooling</i> .
Kelas FlattenScratch	
Atribut	-
Metode	forward(input) Fungsi ini adalah implementasi dari fungsi abstrak dari LayerScratch yang bertujuan ‘meratakan’ hasil dari <i>layer</i> CNN untuk dimasukkan ke <i>layer dense</i> .
Kelas DenseScratch	
Atribut	activation Atribut ini berfungsi menyimpan fungsi aktivasi yang akan digunakan pada hasil akhir <i>layer</i> ini.
	weights Atribut ini berfungsi menyimpan bobot <i>layer</i> ini.
	bias Atribut ini berfungsi menyimpan <i>bias layer</i> ini.

Metode	forward(input)
	Fungsi ini adalah implementasi dari fungsi abstrak dari LayerScratch yang bertujuan melakukan <i>forward propagation</i> pada <i>layer dense</i> .
Fungsi/Prosedur Tambahan	
F	image_to_col(input, kh, kw, sh, sw, out_h, out_w) Fungsi ini bertujuan mengubah bentuk input berdasarkan ukuran <i>kernel</i> (kh, kw), <i>stride</i> (sh, sw), dan dimensi <i>output</i> (out_h, out_w) menjadi kolom-kolom (vektor) untuk membuat komputasi lebih efisien.

2.1.1.2. Simple RNN

Berikut merupakan tabel yang berisi deskripsi kelas, atribut, dan metode yang digunakan dalam implementasi Simple RNN dari *scratch*

Tabel 2.1.1.1.1. Deskripsi Kelas, Atribut, dan Metode Simple RNN

Kelas RNNModel	
Atribut	vocab_size
	Atribut ini berfungsi untuk menyimpan ukuran kosakata (jumlah kata unik) yang akan digunakan oleh <i>embedding layer</i> .
	embedding_dim
	Atribut ini berfungsi untuk menyimpan dimensi vektor <i>embedding</i> untuk setiap kata.
	dropout_rate
	Atribut ini berfungsi untuk menyimpan <i>dropout rate</i> yang akan diterapkan pada <i>output RNN layer</i> untuk mencegah <i>overfitting</i> .
	rnn_units
	Atribut ini berfungsi untuk menyimpan jumlah unit (neuron) pada <i>RNN layer</i> .
	embedding_layer
	Atribut ini berfungsi untuk menyimpan objek <i>EmbeddingScratch</i> yang bertanggung jawab mengubah indeks kata menjadi vektor <i>embedding</i> .
	rnn_layer
	Atribut ini berfungsi untuk menyimpan objek <i>RNNScratch</i> yang mengimplementasikan logika RNN.

	<p>dense_layer</p> <p>Atribut ini berfungsi untuk menyimpan objek DenseScratch yang merupakan <i>output layer</i> model untuk klasifikasi.</p>
Metode	<p>forward_propagation(x)</p> <p>Fungsi ini melakukan <i>forward propagation</i> melalui seluruh model RNN. Fungsi ini akan memproses input <i>x</i> (yang merupakan urutan indeks kata), mengubahnya menjadi <i>embedding</i> menggunakan <i>embedding_layer</i>, kemudian memprosesnya melalui <i>rnn_layer</i>. Jika model merupakan model <i>bidirectional</i>, model akan melakukan <i>forward</i> dan <i>backward pass</i> secara terpisah dan hasilnya akan digabungkan. Setelah itu, model akan menerapkan <i>dropout</i>, dan hasil akhirnya akan melewati <i>dense_layer</i> untuk menghasilkan probabilitas <i>output</i>.</p>
	<p>predict(x)</p> <p>Fungsi ini digunakan untuk melakukan prediksi terhadap data input <i>x</i> yang akan memanggil <i>forward_propagation</i> untuk mendapatkan probabilitas <i>output</i> dan kemudian mengembalikan indeks kelas dengan probabilitas tertinggi sebagai hasil prediksi.</p>
	<p>evaluate(x, y)</p> <p>Fungsi ini digunakan untuk mengevaluasi kinerja model berdasarkan data input <i>x</i> dan label <i>y</i>. Fungsi ini akan memanggil <i>predict</i> untuk mendapatkan prediksi model dan kemudian menghitung <i>F1 score</i> (dengan rata-rata makro).</p>
	<p>load_weights(filepath)</p> <p>Prosedur ini digunakan untuk memuat bobot model dari <i>file</i> <i>filepath</i>. Bobot akan dimuat untuk seluruh layer yaitu <i>embedding layer</i>, <i>RNN layer</i>, dan <i>dense layer</i> dari <i>file</i> <i>.npy</i> yang disimpan dari hasil <i>training</i> model Keras.</p>
Kelas LayerScratch	
Atribut	-
Metode	<p>forward(input)</p> <p>Fungsi ini adalah fungsi abstrak yang bertujuan melakukan <i>forward propagation</i> dan akan diimplementasikan oleh turunan kelas ini.</p>

Kelas RNNScratch	
Atribut	<p>input_dim</p> <p>Atribut ini berfungsi menyimpan dimensi dari fitur input yang diterima layer.</p>
	<p>hidden_dim</p> <p>Atribut ini berfungsi menyimpan dimensi <i>hidden state</i> dari RNN.</p>
	<p>bidirectional</p> <p>Atribut ini berfungsi menyimpan nilai <i>boolean</i> yang menunjukkan apakah <i>layer</i> RNN akan melakukan <i>bidirectional pass</i> atau tidak.</p>
	<p>kernel</p> <p>Atribut ini berfungsi menyimpan bobot w_{xh} pada <i>hidden state</i>.</p>
	<p>recurrent_kernel</p> <p>Atribut ini berfungsi menyimpan bobot w_{hh} pada <i>hidden state</i> saat ini untuk melakukan perhitungan dengan <i>hidden state</i> sebelumnya.</p>
	<p>bias</p> <p>Atribut ini berfungsi menyimpan nilai bias yang akan ditambahkan dalam perhitungan <i>hidden state</i>.</p>
	<p>forward(x, h_prev, is_forward = True)</p> <p>Fungsi ini adalah implementasi dari fungsi abstrak dari LayerScratch, yang bertujuan untuk melakukan <i>forward propagation</i> pada RNN. Fungsi ini menerima input x pada t saat ini dan nilai <i>hidden state</i> sebelumnya. Jika model adalah <i>bidirectional</i>, maka parameter <i>is_forward</i> akan menentukan apakah perhitungan akan dilakukan secara <i>forward pass</i> atau <i>backward pass</i>.</p>
Kelas EmbeddingScratch	
Atribut	<p>vocab_size</p> <p>Atribut ini berfungsi untuk menyimpan ukuran kosakata (jumlah kata unik) yang akan digunakan oleh <i>embedding layer</i>.</p>
	<p>embedding_dim</p> <p>Atribut ini berfungsi untuk menyimpan dimensi vektor <i>embedding</i> untuk setiap kata.</p>

	<p><code>weights</code></p> <p>Atribut ini berfungsi menyimpan bobot <i>layer</i> ini.</p>
Metode	<p><code>set_weights(weights)</code></p> <p>Fungsi ini berfungsi memperbarui <i>layer</i> agar menggunakan bobot dari <code>weights</code>.</p>
	<p><code>forward(input)</code></p> <p>Fungsi ini adalah implementasi dari fungsi abstrak dari <code>LayerScratch</code> yang bertujuan melakukan <i>forward propagation</i> pada <i>embedding stage</i>.</p>
Kelas DenseScratch	
Atribut	<p><code>neurons</code></p> <p>Atribut ini berfungsi menyimpan jumlah neuron dalam <i>layer</i>.</p>
	<p><code>activation</code></p> <p>Atribut ini berfungsi menyimpan fungsi aktivasi yang akan digunakan pada hasil akhir <i>layer</i> ini.</p>
	<p><code>weights</code></p> <p>Atribut ini berfungsi menyimpan bobot <i>layer</i> ini.</p>
	<p><code>bias</code></p> <p>Atribut ini berfungsi menyimpan <i>bias layer</i> ini.</p>
Metode	<p><code>forward(input)</code></p> <p>Fungsi ini adalah implementasi dari fungsi abstrak dari <code>LayerScratch</code> yang bertujuan melakukan <i>forward propagation</i> pada <i>layer dense</i>.</p>
	<p><code>set_weights(weights)</code></p> <p>Fungsi ini adalah digunakan untuk mengatur bobot dan bias dari <i>layer</i>.</p>
Kelas RNNKerasModel	
Atribut	<p><code>vocab_size</code></p> <p>Atribut ini berfungsi untuk menyimpan ukuran kosakata (jumlah kata unik) yang akan digunakan oleh <i>embedding layer</i>.</p>
	<p><code>embedding_dim</code></p> <p>Atribut ini berfungsi untuk menyimpan menyimpan dimensi vektor <i>embedding</i> untuk setiap kata.</p>

	<p><code>dropout_rate</code></p> <p>Atribut ini berfungsi untuk menyimpan <i>dropout rate</i> yang akan diterapkan pada <i>output RNN layer</i> untuk mencegah <i>overfitting</i>.</p>
	<p><code>rnn_units</code></p> <p>Atribut ini berfungsi untuk menyimpan jumlah unit (neuron) pada <i>RNN layer</i>.</p>
	<p><code>embedding_layer</code></p> <p>Atribut ini berfungsi untuk menyimpan objek <code>EmbeddingScratch</code> yang bertanggung jawab mengubah indeks kata menjadi vektor <i>embedding</i>.</p>
	<p><code>rnn_layer</code></p> <p>Atribut ini berfungsi untuk menyimpan objek <code>RNNScratch</code> yang mengimplementasikan logika RNN.</p>
	<p><code>dense_layer</code></p> <p>Atribut ini berfungsi untuk menyimpan objek <code>DenseScratch</code> yang merupakan <i>output layer</i> model untuk klasifikasi.</p>
Metode	<p><code>forward_propagation(x)</code></p> <p>Fungsi ini melakukan <i>forward propagation</i> melalui seluruh model RNN. Fungsi ini akan memproses input <code>x</code> (yang merupakan urutan indeks kata), mengubahnya menjadi <i>embedding</i> menggunakan <code>embedding_layer</code>, kemudian memprosesnya melalui <code>rnn_layer</code>. Jika model merupakan model <i>bidirectional</i>, model akan melakukan <i>forward</i> dan <i>backward pass</i> secara terpisah dan hasilnya akan digabungkan. Setelah itu, model akan menerapkan <i>dropout</i>, dan hasil akhirnya akan melewati <code>dense_layer</code> untuk menghasilkan probabilitas <i>output</i>.</p>
	<p><code>predict(x)</code></p> <p>Fungsi ini digunakan untuk melakukan prediksi terhadap data input <code>x</code> yang akan memanggil <code>forward_propagation</code> untuk mendapatkan probabilitas <i>output</i> dan kemudian mengembalikan indeks kelas dengan probabilitas tertinggi sebagai hasil prediksi.</p>
	<p><code>evaluate(x, y)</code></p> <p>Fungsi ini digunakan untuk mengevaluasi kinerja model</p>

	berdasarkan data input x dan label y . Fungsi ini akan memanggil <code>predict</code> untuk mendapatkan prediksi model dan kemudian menghitung <i>F1 score</i> (dengan rata-rata makro).
	<code>save_weights(filepath)</code> Prosedur ini digunakan untuk menyimpan bobot model ke <i>file</i> <code>filepath</code> . File ini berisi bobot dari seluruh layer yaitu <i>embedding layer</i> , <i>RNN layer</i> , dan <i>dense layer</i> yang akan disimpan pada sebuah <i>file</i> <code>.npy</code> .
Kelas TextPreprocessor	
Atribut	<code>max_tokens</code> Atribut ini berfungsi untuk menyimpan jumlah maksimum <i>token</i> (kata unik) yang akan disimpan dalam kosakata.
	<code>output_sequence_length</code> Atribut ini berfungsi untuk menyimpan panjang urutan <i>output</i> dari teks yang telah diproses.
	<code>embedding_dim</code> Atribut ini berfungsi untuk menyimpan dimensi <i>embedding</i> dari setiap token.
	<code>vectorization_layer</code> Atribut ini berfungsi untuk menyimpan <code>TextVectorization</code> dari Keras.
	<code>fit(texts)</code> Membangun <i>vocabulary</i> dari kumpulan teks yang diberikan
	<code>preprocess(texts)</code> Mengubah kumpulan teks menjadi urutan angka sesuai <i>vocabulary</i> .
Metode	<code>get_vocabulary()</code> Mengembalikan <i>list</i> kata dalam <i>vocabulary</i> .
	<code>get_vocab_size()</code> Mengembalikan ukuran <i>vocabulary</i> yang digunakan.
Fungsi/Prosedur Tambahan	
F	<code>load_nusax_data(filepath)</code>

	<p>Fungsi ini bertujuan untuk memuat data dari <i>dataset</i> NuSaX yang disimpan dalam format CSV. Fungsi ini membaca <i>file</i> CSV dari lokasi <i>filepath</i> yang diberikan, mengekstrak kolom 'text' sebagai <i>array</i> teks, dan mengubah kolom 'label' menjadi representasi numerik (0 untuk 'neutral', 1 untuk 'negative', dan 2 untuk 'positive'). Hasilnya adalah <i>tuple</i> yang berisi <i>array</i> teks dan <i>array</i> label dalam bentuk numerik.</p>
--	---

2.1.1.3. LSTM

Berikut merupakan tabel yang berisi deskripsi kelas, atribut, dan metode yang digunakan dalam implementasi LSTM dari *scratch*.

Tabel 2.1.1.3.1. Deskripsi Kelas, Atribut, dan Metode LSTM

Kelas DropoutLayer	
Atribut	<p>rate</p> <p>Menyimpan nilai probabilitas <i>dropout</i>, yaitu proporsi <i>neuron</i> yang akan di-nol-kan selama pelatihan untuk mencegah <i>overfitting</i>.</p>
Metode	<p><code>forward(self, inputs, training=True)</code></p> <p>Melakukan proses dropout pada <i>input</i> jika sedang <i>training</i>, dan melewati input tanpa perubahan jika tidak.</p>
Kelas EmbeddingLayer	
Atribut	<p>input_dim</p> <p>Jumlah kata unik dalam <i>vocabulary</i> yang akan di-<i>embedding</i>.</p>
	<p>output_dim</p> <p>Ukuran vektor <i>embedding</i> untuk setiap kata.</p>
	<p>input_length</p> <p>Panjang urutan input yang diharapkan</p>
	<p>embeddings</p> <p>Matriks bobot <i>embedding</i> yang akan dipelajari selama pelatihan.</p>
	<p><code>forward(inputs)</code></p> <p>Mengubah urutan indeks kata menjadi urutan vektor <i>embedding</i> sesuai matriks <i>embedding</i>.</p>
Kelas LSTMLayer	

Atribut	units
	Jumlah unit (sel) LSTM dalam layer ini.
	input_dim
	Ukuran fitur input untuk setiap <i>timestep</i> .
Atribut	W_i, W_f, W_c, W_o
	Matriks bobot <i>input</i> untuk masing-masing <i>gate</i> (<i>input</i> , <i>forget</i> , <i>cell</i> , <i>output</i>).
	U_i, U_f, U_c, U_o
	Matriks bobot <i>recurrent</i> (<i>hidden state</i>) untuk masing-masing <i>gate</i> .
Atribut	b_i, b_f, b_c, b_o
	Vektor bias untuk masing-masing <i>gate</i> .
Metode	<code>forward(inputs, initial_state=None)</code>
	Melakukan propagasi maju pada <i>input</i> untuk menghasilkan <i>output</i> dan state LSTM
	<code>load_weights(weights)</code>
	Memuat bobot <i>layer</i> dari format eksternal, mengatur bobot dan bias ke struktur internal.
Metode	<code>get_weights()</code>
	Mengembalikan bobot dan bias layer dalam format yang dapat disimpan atau digunakan ulang.
Kelas LSTMModel	
Atribut	layers
	List berisi <i>layer-layer</i> (<i>embedding</i> , LSTM, <i>dense</i> , dsb) yang membentuk model.
	preprocessor
	Objek <i>preprocessor</i> teks untuk mengubah <i>input</i> mentah menjadi urutan angka.
Atribut	label_mapping
	Pemetaan label <i>string</i> ke angka untuk keperluan klasifikasi.
Metode	<code>add_embedding(vocab_size, embedding_dim, input_length)</code>

	Fungsi atau metode untuk standarisasi teks (misal: <i>lowercase</i> , hapus tanda baca).
	vocab List kata-kata dalam <i>vocabulary</i> yang telah dipelajari dari data.
Metode	fit(texts) Membangun <i>vocabulary</i> dari kumpulan teks yang diberikan
	preprocess(texts) Mengubah kumpulan teks menjadi urutan angka sesuai <i>vocabulary</i> .
	get_vocabulary() Mengembalikan <i>list</i> kata dalam <i>vocabulary</i> .
	get_vocab_size() Mengembalikan ukuran <i>vocabulary</i> yang digunakan.

2.1.2. Penjelasan Forward Propagation

2.1.2.1. CNN

Forward propagation pada CNN terdiri atas tiga tahap menurut *complex layer terminology*, yaitu *convolution*, *detector*, dan *pooling stage*.

Tahap *convolution* melibatkan perhitungan penjumlahan hasil perkalian antara area tertentu pada *input* (bergantung pada ukuran *kernel*) dengan *kernel*, lalu ditambah *bias*. Perhitungan tersebut akan dilakukan sampai seluruh data *input* dihitung dengan pergeseran (*stride*) tertentu. Selain itu, bisa juga ditambahkan *padding* pada *input*, yaitu angka 0 yang menjadi ‘lapisan’ terluar *input*.

Tahap *detector* melibatkan pengaplikasian fungsi aktivasi terhadap hasil dari tahap *convolution*. Fungsi aktivasi yang biasa digunakan adalah ReLU, *sigmoid*, dan *softmax*.

Tahap *pooling* sebenarnya bertujuan melakukan *downsampling* terhadap hasil tahap *detector*. Biasanya dilakukan dengan metode *max-pooling*, *average-pooling*, atau *L2-norm-pooling* per area tertentu dalam *input*, tergantung *stride* yang digunakan. Namun, *padding* juga dapat dikenakan pada tahap *pooling* sehingga bisa saja tujuan *downsampling* tidak tercapai.

Dimensi *output* dari tahap *convolution* dan *pooling* dapat dihitung dengan menggunakan rumus berikut:

$$\text{out_dim} = V * V * k$$

di mana

$$V = \frac{W - F + 2P}{S} + 1$$

k = banyaknya kernel

Pada implementasi yang telah dilakukan, tahap *convolution* dan *detector* diimplementasikan dalam metode ‘forward’ pada kelas *Conv2DScratch*. Selain itu, *input* yang awalnya memiliki dimensi lebih dari 1 dikenakan fungsi ‘*image_to_col*’ agar perkaliannya dapat dilakukan dengan efisien oleh *library* NumPy. Berikut merupakan potongan kodenya.

```
def forward(self, input):
    if self.weights is None or self.bias is None:
        raise ValueError("Weights must be set before
calling forward")

    batch, h, w, c = input.shape
    kh, kw = self.kernel_size
    sh, sw = self.strides

    if self.padding == 'same':
        pad_h = ((h - 1) * sh + kh - h) // 2
        pad_w = ((w - 1) * sw + kw - w) // 2
        input = np.pad(input, ((0,0), (pad_h, pad_h),
(pad_w, pad_w), (0,0)), mode='constant')

    out_h = (input.shape[1] - kh) // sh + 1
    out_w = (input.shape[2] - kw) // sw + 1

    cols = image_to_col(input, kh, kw, sh, sw, out_h,
out_w)
    w_col = self.weights.reshape(-1, self.filters)

    out = np.matmul(cols, w_col) + self.bias
    out = out.reshape(batch, out_h, out_w, self.filters)

    if self.activation == 'relu':
        out = np.maximum(0, out)
    elif self.activation == 'sigmoid':
        out = 1 / (1 + np.exp(-out))
    elif self.activation == 'softmax':
        exp_output = np.exp(out - np.max(out, axis=-1,
keepdims=True))
        out = exp_output / np.sum(exp_output, axis=-1,
keepdims=True)

    return out
```

Tahap *pooling* diimplementasikan dalam metode ‘forward’ pada kelas *MaxPooling2DScratch* dan *AvgPooling2DScratch*. Berikut merupakan potongan kodenya.

```
def forward(self, input):
    if input.ndim != 4:
        raise ValueError(f"Expected 4D input for
MaxPooling2D, got shape {input.shape}")
```

```

batch, h, w, c = input.shape
ph, pw = self.pool_size
sh, sw = self.strides

if self.padding == 'same':
    out_h = int(np.ceil(h / sh))
    out_w = int(np.ceil(w / sw))
    pad_h = max((out_h - 1) * sh + ph - h, 0)
    pad_w = max((out_w - 1) * sw + pw - w, 0)
    pad_top = pad_h // 2
    pad_bottom = pad_h - pad_top
    pad_left = pad_w // 2
    pad_right = pad_w - pad_left
    input = np.pad(input, ((0, 0), (pad_top,
pad_bottom), (pad_left, pad_right), (0, 0)),
mode='constant')
else:
    out_h = (h - ph) // sh + 1
    out_w = (w - pw) // sw + 1

    output = np.zeros((batch, out_h, out_w, c)) #
nyiapin tempat output

    for y in range(out_h):
        for x in range(out_w):
            region = input[:, y*sh:y*sh+ph,
x*sw:x*sw+pw, :]

{untuk max-pool}
            output[:, y, x, :] = np.max(region, axis=(1,
2))
{untuk average-pool}
            output[:, y, x, :] = np.mean(region,
axis=(1, 2))

    return output

```

2.1.2.2. Simple RNN

Simple RNN adalah arsitektur *neural network* yang digunakan untuk memproses data sekuensial. RNN memiliki *loop internal* yang memungkinkan informasi dari *timestep* sebelumnya dipertahankan dan diproses bersama input saat ini.

Misalkan:

$x_t \in \mathbb{R}^n$: input pada timestep ke- t

$h_t \in \mathbb{R}^m$: hidden state pada timestep ke- t

$h_{t-1} \in \mathbb{R}^m$: hidden state dari timestep sebelumnya

$W_{xh} \in \mathbb{R}^{n \times m}$: bobot input ke hidden

$W_{hh} \in \mathbb{R}^{m \times m}$: bobot recurrent (hidden ke hidden)

$b_h \in \mathbb{R}^m$: bias

$\phi(\cdot)$: fungsi aktivasi

Forward propagation untuk satu *timestep* didefinisikan sebagai

$$h_t = \phi(x_t W_{xh} + h_{t-1} W_{hh} + b_h)$$

Untuk *timestep* 1 nilai *hidden state* sebelumnya adalah $h_0 = \vec{0}$. Nilai *hidden state* untuk setiap *timestep* secara detail adalah:

$$\begin{aligned} h_1 &= \phi(x_1 W_{xh} + h_0 W_{hh} + b_h) \\ h_2 &= \phi(x_2 W_{xh} + h_1 W_{hh} + b_h) \\ &\vdots \\ h_T &= \phi(x_T W_{xh} + h_{T-1} W_{hh} + b_h) \end{aligned}$$

Jika menghasilkan output di setiap *timestep*, maka nilai *output* dari tiap *timestep* adalah:

$$y_t = \psi(h_t W_{hy} + b_y)$$

di mana:

W_{hy} : bobot *hidden* ke *output*

b_y : bias *output*

$\psi(\cdot)$: fungsi aktivasi *output*

Kami mengimplementasikan *forward propagation* dengan cukup sederhana yaitu:

```
def forward(
    self, x: np.ndarray, h_prev: np.ndarray,
    is_forward: bool = True
) -> np.ndarray:
    if any(w is None for w in [self.kernel,
self.recurrent_kernel, self.bias]):
        raise ValueError("Weights not set. Call
set_weights first.")

    if self.bidirectional:
        half_units = self.hidden_dim
        if is_forward:
```

```

        # Forward pass
        h = np.tanh(
            np.dot(x, self.kernel[:,
:half_units])
            + np.dot(h_prev,
self.recurrent_kernel[:,half_units, :half_units])
            + self.bias[:,half_units]
        )
    else:
        # Backward pass
        h = np.tanh(
            np.dot(x, self.kernel[:,
half_units:])
            + np.dot(h_prev,
self.recurrent_kernel[half_units:, half_units:])
            + self.bias[half_units:]
        )
    else:
        h = np.tanh(
            np.dot(x, self.kernel)
            + np.dot(h_prev, self.recurrent_kernel)
            + self.bias
        )

    return h

```

Pada implementasi tersebut kami menggunakan fungsi aktivasi *tanh*. Kami melakukan *forward propagation* secara sederhana dengan langsung menggunakan fungsi - fungsi yang disediakan oleh *numpy*. Yang membuat *forward propagation* yang kami buat terlihat cukup rumit adalah proses *forward pass* dan *backward pass* pada *bidirectional RNN*. Kami menyimpan *weights* untuk *unidirectional* dan *bidirectional* dalam satu variabel yang sama sehingga kode yang cukup rumit di atas sebenarnya adalah handling untuk *forward pass* dan *backward pass* dimana *forward pass* akan menggunakan setengah *weights* pertama, dan *backward* akan menggunakan setengah *weights* yang lain.

2.1.2.3. LSTM

Forward propagation pada LSTM (Long Short-Term Memory) merupakan proses utama di mana data input diproses secara berurutan, baik untuk tugas klasifikasi maupun prediksi urutan. Pada dasarnya, LSTM dirancang untuk mengatasi masalah *vanishing gradient problem* pada RNN standar dengan memperkenalkan mekanisme memori jangka panjang (*cell state*) dan tiga buah *gate* utama: *input gate*, *forget gate*, dan *output gate*.

Pada setiap *timestep* t , input berupa vektor x_t dan *hidden state* sebelumnya h_{t-1} serta *cell state* sebelumnya c_{t-1} digunakan untuk

menghitung nilai *gate* dan memperbarui *state*. Prosesnya sebagai berikut:

1. Input Gate (i_t): Mengontrol seberapa banyak informasi baru dari input yang akan dimasukkan ke cell state.

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

2. Forget Gate (f_t): Mengontrol seberapa banyak informasi lama pada cell state yang akan dilupakan.

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

3. Cell Candidate (\tilde{c}_t): Menghasilkan kandidat nilai baru yang akan ditambahkan ke cell state.

$$\tilde{c}_t = \tanh(W_c x_t + U_c h_{t-1} + b_c)$$

4. Output Gate (o_t): Mengontrol seberapa banyak informasi dari cell state yang akan dikeluarkan ke hidden state.

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

Setelah nilai *gate* dihitung, cell state dan hidden state diperbarui sebagai berikut:

1. Cell State Update:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

2. Hidden State Update:

$$h_t = o_t \odot \tanh(c_t)$$

Note: simbol \odot menyatakan operasi perkalian elemen-per-elemen (*element-wise*).

Pada implementasi *unidirectional*, proses ini dilakukan dari timestep pertama hingga terakhir (atau sebaliknya jika *gobackwards = True*). Jika parameter *returnsequences = True*, hidden state dari setiap timestep (h_1, h_2, \dots, h_T) dikumpulkan sebagai output sequence. Jika *returnsequences = False*, hanya hidden state terakhir (h_T) yang digunakan sebagai output, biasanya untuk klasifikasi.

Berikut merupakan potongan kode penerapan LSTM *unidirectional*:

```
def forward(self, inputs):
    if not self.weights_loaded or self.W_i is None
    or self.U_i is None:
        raise ValueError("Weights not loaded. Call
        load_weights() first.")

    batch_size, seq_length, input_dim = inputs.shape

    h = np.zeros((batch_size, self.units))
    c = np.zeros((batch_size, self.units))

    if self.return_sequences:
```

```

        outputs = np.zeros((batch_size, seq_length,
self.units))

        time_steps = range(seq_length)
        if self.go_backwards:
            time_steps = reversed(time_steps)

        for t in time_steps:
            x_t = inputs[:, t, :]

            i_gate = np.dot(x_t, self.W_i) + np.dot(h,
self.U_i) + self.b_i
            f_gate = np.dot(x_t, self.W_f) + np.dot(h,
self.U_f) + self.b_f
            c_gate = np.dot(x_t, self.W_c) + np.dot(h,
self.U_c) + self.b_c
            o_gate = np.dot(x_t, self.W_o) + np.dot(h,
self.U_o) + self.b_o

            i_t = self.sigmoid(i_gate)
            f_t = self.sigmoid(f_gate)
            c_tilde = self.tanh(c_gate)
            o_t = self.sigmoid(o_gate)

            c = f_t * c + i_t * c_tilde

            h = o_t * self.tanh(c)

            if self.return_sequences:
                if self.go_backwards:
                    outputs[:, seq_length - 1 - t, :] =
h
                else:
                    outputs[:, t, :] = h

            if self.return_sequences:
                return outputs
            else:
                return h

```

Pada *bidirectional* LSTM, proses *forward propagation* dilakukan dua kali: satu kali dari depan ke belakang (*forward*) dan satu kali dari belakang ke depan (*backward*). *Output* dari kedua arah kemudian digabungkan (*concatenate*) pada setiap timestep sehingga model dapat menangkap konteks dari kedua arah dalam *sequence*.

Berikut implementasi LSTM *bidirectional* yang memanfaatkan *forward function* dari LSTM *unidirectional*:

```

def forward(self, inputs):

    if not self.weights_loaded:
        raise ValueError("Bidirectional LSTM weights
not loaded. Call load_weights() first.")

    forward_out = self.forward_lstm.forward(inputs)
    backward_out =
self.backward_lstm.forward(inputs)

```



```

        output = np.concatenate([forward_out,
                                   backward_out], axis=-1)

        if not self.return_sequences:
            output = output[:, -1, :]

        return output

```

Dengan demikian, forward propagation pada LSTM memungkinkan model untuk secara dinamis memilih informasi mana yang perlu diingat, dilupakan, atau dikeluarkan pada setiap langkah sehingga sangat efektif untuk memproses data sekuensial seperti teks.

2.2. Hasil Pengujian

Pada pengujian, kami membuat sebuah fungsi sederhana untuk memastikan bahwa *seed* yang dipakai variasi model sama. Berikut adalah detail implementasi fungsi tersebut:

```

def reset_seeds(seed=42):
    import random as python_random
    import numpy as np
    import tensorflow as tf

    python_random.seed(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)

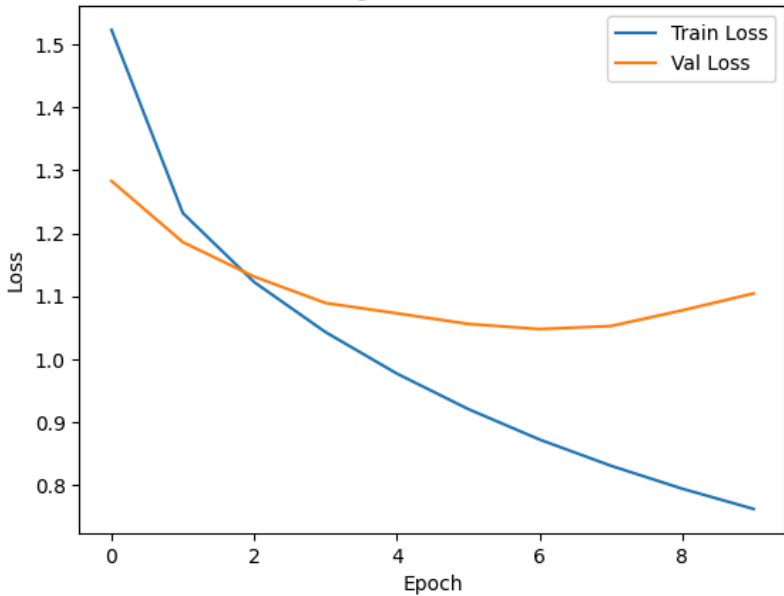
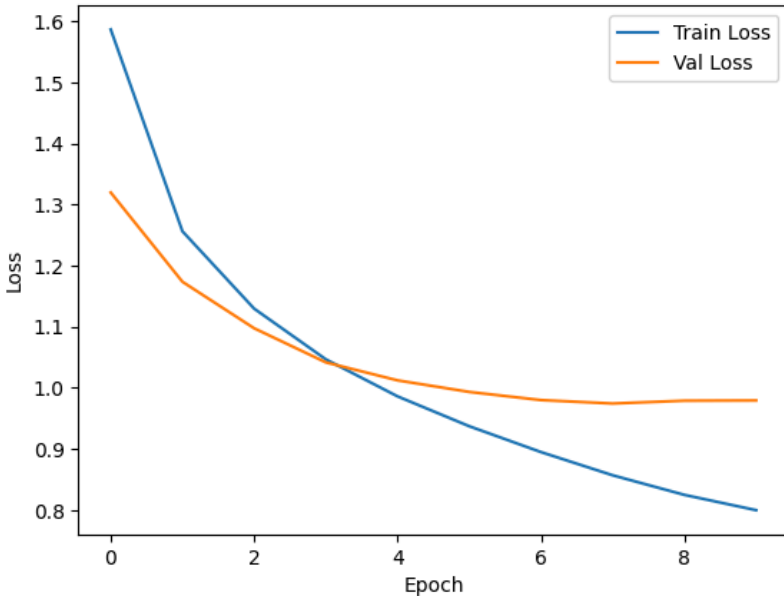
```

2.2.1. Pengujian CNN

2.2.1.1. Pengaruh Jumlah Layer Konvolusi

Tabel 2.2.1.1.1. Pengujian Pengaruh Jumlah Layer Konvolusi

No.	Hasil Pengujian
1.	<p><i>Hyperparameter:</i></p> <p>1 layer konvolusi: filter 32 dengan ukuran 3 x 3 dan aktivasi relu</p> <p>1 layer max pooling dengan ukuran 2 x 2</p> <p>1 flatten</p> <p>2 layer dense:</p> <ul style="list-style-type: none"> - 1 dengan 64 neuron dan aktivasi relu - 1 dengan 10 neuron dan aktivasi softmax
	Hasil:

	<div><p>Training vs Validation Loss</p><table><tr><th>Epoch</th><th>Train Loss</th><th>Val Loss</th></tr><tr><td>0</td><td>1.55</td><td>1.28</td></tr><tr><td>1</td><td>1.25</td><td>1.18</td></tr><tr><td>2</td><td>1.15</td><td>1.12</td></tr><tr><td>3</td><td>1.05</td><td>1.08</td></tr><tr><td>4</td><td>0.98</td><td>1.06</td></tr><tr><td>5</td><td>0.92</td><td>1.05</td></tr><tr><td>6</td><td>0.88</td><td>1.04</td></tr><tr><td>7</td><td>0.84</td><td>1.05</td></tr><tr><td>8</td><td>0.80</td><td>1.08</td></tr><tr><td>9</td><td>0.75</td><td>1.10</td></tr></table><p>Akurasi (<i>Macro F1 Score</i>): 0.6304</p></div>	Epoch	Train Loss	Val Loss	0	1.55	1.28	1	1.25	1.18	2	1.15	1.12	3	1.05	1.08	4	0.98	1.06	5	0.92	1.05	6	0.88	1.04	7	0.84	1.05	8	0.80	1.08	9	0.75	1.10
Epoch	Train Loss	Val Loss																																
0	1.55	1.28																																
1	1.25	1.18																																
2	1.15	1.12																																
3	1.05	1.08																																
4	0.98	1.06																																
5	0.92	1.05																																
6	0.88	1.04																																
7	0.84	1.05																																
8	0.80	1.08																																
9	0.75	1.10																																
2.	<div><p><i>Hyperparameter:</i></p><p>2 layer konvolusi: filter 32 dengan ukuran 3 x 3 dan aktivasi relu</p><p>2 layer max pooling dengan ukuran 2 x 2</p><p>1 flatten</p><p>2 layer dense:</p><ul style="list-style-type: none">- 1 dengan 64 neuron dan aktivasi relu- 1 dengan 10 neuron dan aktivasi softmax<p>Hasil:</p><div><p>Training vs Validation Loss</p><table><tr><th>Epoch</th><th>Train Loss</th><th>Val Loss</th></tr><tr><td>0</td><td>1.58</td><td>1.32</td></tr><tr><td>1</td><td>1.25</td><td>1.18</td></tr><tr><td>2</td><td>1.12</td><td>1.10</td></tr><tr><td>3</td><td>1.05</td><td>1.04</td></tr><tr><td>4</td><td>0.98</td><td>1.01</td></tr><tr><td>5</td><td>0.92</td><td>0.99</td></tr><tr><td>6</td><td>0.88</td><td>0.98</td></tr><tr><td>7</td><td>0.84</td><td>0.97</td></tr><tr><td>8</td><td>0.82</td><td>0.98</td></tr><tr><td>9</td><td>0.80</td><td>0.98</td></tr></table><p>Akurasi (<i>Macro F1 Score</i>):</p></div></div>	Epoch	Train Loss	Val Loss	0	1.58	1.32	1	1.25	1.18	2	1.12	1.10	3	1.05	1.04	4	0.98	1.01	5	0.92	0.99	6	0.88	0.98	7	0.84	0.97	8	0.82	0.98	9	0.80	0.98
Epoch	Train Loss	Val Loss																																
0	1.58	1.32																																
1	1.25	1.18																																
2	1.12	1.10																																
3	1.05	1.04																																
4	0.98	1.01																																
5	0.92	0.99																																
6	0.88	0.98																																
7	0.84	0.97																																
8	0.82	0.98																																
9	0.80	0.98																																

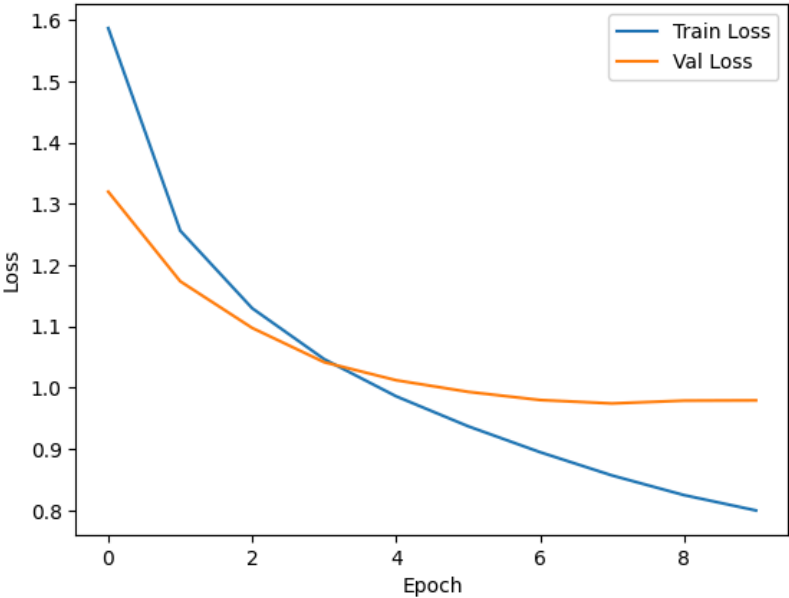
	0.6658																																	
3.	<p><i>Hyperparameter:</i></p> <p>3 layer konvolusi: filter 32 dengan ukuran 3 x 3 dan aktivasi relu</p> <p>3 layer max pooling dengan ukuran 2 x 2</p> <p>1 flatten</p> <p>2 layer dense:</p> <ul style="list-style-type: none">- 1 dengan 64 neuron dan aktivasi relu- 1 dengan 10 neuron dan aktivasi softmax <p>Hasil:</p> <div><p>Training vs Validation Loss</p><table border="1"><thead><tr><th>Epoch</th><th>Train Loss</th><th>Val Loss</th></tr></thead><tbody><tr><td>0</td><td>1.60</td><td>1.32</td></tr><tr><td>1</td><td>1.25</td><td>1.18</td></tr><tr><td>2</td><td>1.12</td><td>1.08</td></tr><tr><td>3</td><td>1.05</td><td>1.05</td></tr><tr><td>4</td><td>0.98</td><td>1.02</td></tr><tr><td>5</td><td>0.92</td><td>1.00</td></tr><tr><td>6</td><td>0.88</td><td>0.98</td></tr><tr><td>7</td><td>0.85</td><td>0.98</td></tr><tr><td>8</td><td>0.82</td><td>0.98</td></tr><tr><td>9</td><td>0.80</td><td>0.98</td></tr></tbody></table></div> <p>Akurasi (<i>Macro F1 Score</i>):</p> <p>0.6450</p>	Epoch	Train Loss	Val Loss	0	1.60	1.32	1	1.25	1.18	2	1.12	1.08	3	1.05	1.05	4	0.98	1.02	5	0.92	1.00	6	0.88	0.98	7	0.85	0.98	8	0.82	0.98	9	0.80	0.98
Epoch	Train Loss	Val Loss																																
0	1.60	1.32																																
1	1.25	1.18																																
2	1.12	1.08																																
3	1.05	1.05																																
4	0.98	1.02																																
5	0.92	1.00																																
6	0.88	0.98																																
7	0.85	0.98																																
8	0.82	0.98																																
9	0.80	0.98																																

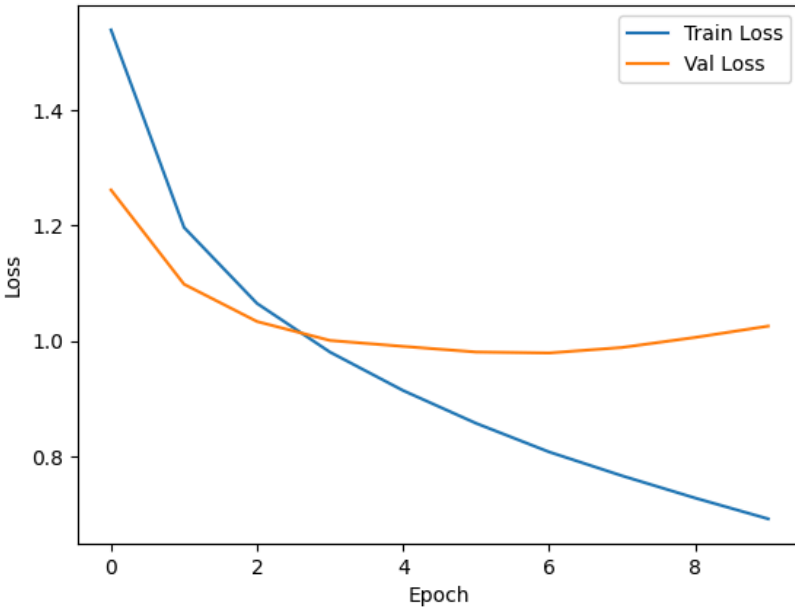
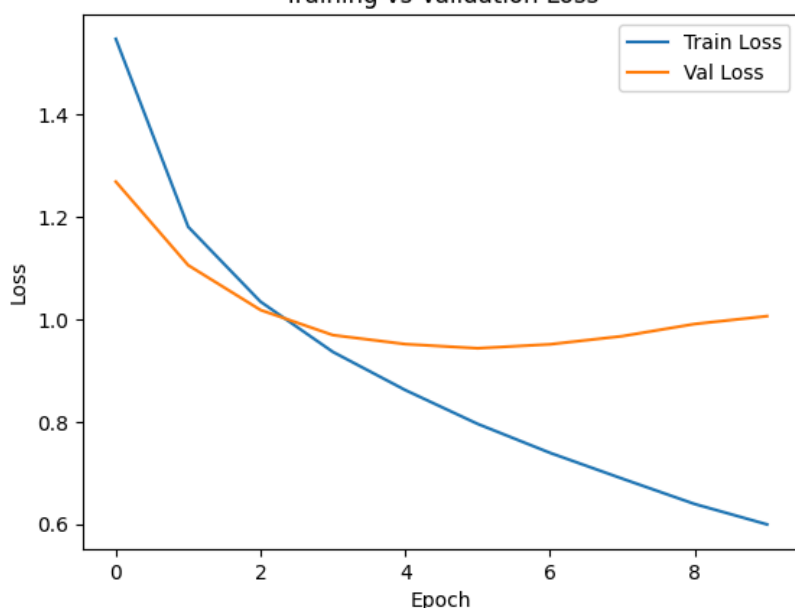
Berdasarkan hasil pengujian tersebut, jumlah *layer* konvolusi cukup berpengaruh terhadap akurasi model. Namun, penambahan/pengurangan jumlah *layer* konvolusi tidak selalu berarti peningkatan/penurunan akurasi. Perlu dilakukan eksplorasi terhadap *dataset* untuk menentukan jumlah *layer* konvolusi yang tepat untuk *dataset* tersebut. Untuk data *Cifar10*, dari ketiga pengujian tersebut, jumlah *layer* konvolusi yang terbaik dengan akurasi 0.6658 adalah 2.

2.2.1.2. Pengaruh Banyak Filter per Layer Konvolusi

Tabel 2.2.1.2.1. Pengujian Pengaruh Banyak Filter per Layer Konvolusi

No.	Hasil Pengujian
1.	<p><i>Hyperparameter:</i></p> <p>2 layer konvolusi: filter 32 dengan ukuran 3 x 3 dan aktivasi relu</p> <p>2 layer max pooling dengan ukuran 2 x 2</p> <p>1 flatten</p>

	<p>2 layer dense:</p> <ul style="list-style-type: none">- 1 dengan 64 neuron dan aktivasi relu- 1 dengan 10 neuron dan aktivasi softmax <p>Hasil:</p> <div><p>Training vs Validation Loss</p><table border="1"><thead><tr><th>Epoch</th><th>Train Loss</th><th>Val Loss</th></tr></thead><tbody><tr><td>0</td><td>1.58</td><td>1.32</td></tr><tr><td>1</td><td>1.25</td><td>1.18</td></tr><tr><td>2</td><td>1.13</td><td>1.09</td></tr><tr><td>3</td><td>1.04</td><td>1.04</td></tr><tr><td>4</td><td>0.98</td><td>1.01</td></tr><tr><td>5</td><td>0.92</td><td>0.99</td></tr><tr><td>6</td><td>0.88</td><td>0.98</td></tr><tr><td>7</td><td>0.85</td><td>0.97</td></tr><tr><td>8</td><td>0.83</td><td>0.98</td></tr><tr><td>9</td><td>0.81</td><td>0.98</td></tr></tbody></table></div> <p>Akurasi (<i>Macro F1 Score</i>): 0.6658</p>	Epoch	Train Loss	Val Loss	0	1.58	1.32	1	1.25	1.18	2	1.13	1.09	3	1.04	1.04	4	0.98	1.01	5	0.92	0.99	6	0.88	0.98	7	0.85	0.97	8	0.83	0.98	9	0.81	0.98
Epoch	Train Loss	Val Loss																																
0	1.58	1.32																																
1	1.25	1.18																																
2	1.13	1.09																																
3	1.04	1.04																																
4	0.98	1.01																																
5	0.92	0.99																																
6	0.88	0.98																																
7	0.85	0.97																																
8	0.83	0.98																																
9	0.81	0.98																																
2.	<p><i>Hyperparameter:</i></p> <p>2 layer konvolusi:</p> <ul style="list-style-type: none">- filter 32 dengan ukuran 3 x 3 dan aktivasi relu- filter 64 dengan ukuran 3 x 3 dan aktivasi relu <p>2 layer max pooling dengan ukuran 2 x 2</p> <p>1 flatten</p> <p>2 layer dense:</p> <ul style="list-style-type: none">- 1 dengan 64 neuron dan aktivasi relu- 1 dengan 10 neuron dan aktivasi softmax <p>Hasil:</p>																																	

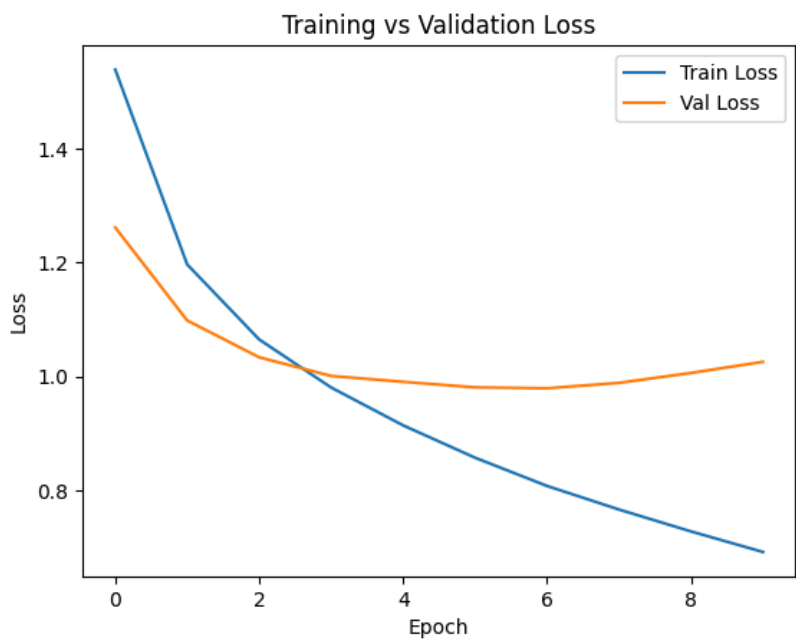
	<div><p>Training vs Validation Loss</p><table><tr><th>Epoch</th><th>Train Loss</th><th>Val Loss</th></tr><tr><td>0</td><td>1.55</td><td>1.25</td></tr><tr><td>1</td><td>1.20</td><td>1.10</td></tr><tr><td>2</td><td>1.05</td><td>1.05</td></tr><tr><td>3</td><td>1.00</td><td>1.02</td></tr><tr><td>4</td><td>0.92</td><td>1.00</td></tr><tr><td>5</td><td>0.85</td><td>0.98</td></tr><tr><td>6</td><td>0.80</td><td>0.98</td></tr><tr><td>7</td><td>0.75</td><td>0.99</td></tr><tr><td>8</td><td>0.72</td><td>1.01</td></tr><tr><td>9</td><td>0.70</td><td>1.03</td></tr></table><p>Akurasi (<i>Macro F1 Score</i>): 0.6756</p></div>	Epoch	Train Loss	Val Loss	0	1.55	1.25	1	1.20	1.10	2	1.05	1.05	3	1.00	1.02	4	0.92	1.00	5	0.85	0.98	6	0.80	0.98	7	0.75	0.99	8	0.72	1.01	9	0.70	1.03
Epoch	Train Loss	Val Loss																																
0	1.55	1.25																																
1	1.20	1.10																																
2	1.05	1.05																																
3	1.00	1.02																																
4	0.92	1.00																																
5	0.85	0.98																																
6	0.80	0.98																																
7	0.75	0.99																																
8	0.72	1.01																																
9	0.70	1.03																																
3.	<div><p><i>Hyperparameter:</i></p><p>2 layer konvolusi:</p><ul style="list-style-type: none">- filter 32 dengan ukuran 3 x 3 dan aktivasi relu- filter 128 dengan ukuran 3 x 3 dan aktivasi relu<p>2 layer max pooling dengan ukuran 2 x 2</p><p>1 flatten</p><p>2 layer dense:</p><ul style="list-style-type: none">- 1 dengan 64 neuron dan aktivasi relu- 1 dengan 10 neuron dan aktivasi softmax<p>Hasil:</p><div><p>Training vs Validation Loss</p><table><tr><th>Epoch</th><th>Train Loss</th><th>Val Loss</th></tr><tr><td>0</td><td>1.55</td><td>1.25</td></tr><tr><td>1</td><td>1.18</td><td>1.10</td></tr><tr><td>2</td><td>1.05</td><td>1.05</td></tr><tr><td>3</td><td>1.00</td><td>1.02</td></tr><tr><td>4</td><td>0.92</td><td>1.00</td></tr><tr><td>5</td><td>0.85</td><td>0.98</td></tr><tr><td>6</td><td>0.80</td><td>0.98</td></tr><tr><td>7</td><td>0.75</td><td>0.99</td></tr><tr><td>8</td><td>0.72</td><td>1.01</td></tr><tr><td>9</td><td>0.60</td><td>1.03</td></tr></table></div></div>	Epoch	Train Loss	Val Loss	0	1.55	1.25	1	1.18	1.10	2	1.05	1.05	3	1.00	1.02	4	0.92	1.00	5	0.85	0.98	6	0.80	0.98	7	0.75	0.99	8	0.72	1.01	9	0.60	1.03
Epoch	Train Loss	Val Loss																																
0	1.55	1.25																																
1	1.18	1.10																																
2	1.05	1.05																																
3	1.00	1.02																																
4	0.92	1.00																																
5	0.85	0.98																																
6	0.80	0.98																																
7	0.75	0.99																																
8	0.72	1.01																																
9	0.60	1.03																																

	Akurasi (<i>Macro F1 Score</i>): 0.6857
--	--

Berdasarkan hasil pengujian tersebut, banyak filter per *layer* konvolusi cukup berpengaruh terhadap akurasi. Pada kasus data Cifar10, semakin banyak jumlah filter yang digunakan pada *layer* konvolusi, semakin baik akurasinya. Hal ini cukup sesuai dengan teori CNN, yaitu filter yang semakin banyak memungkinkan model melihat lebih banyak fitur dari data *input* sehingga hasilnya semakin baik.

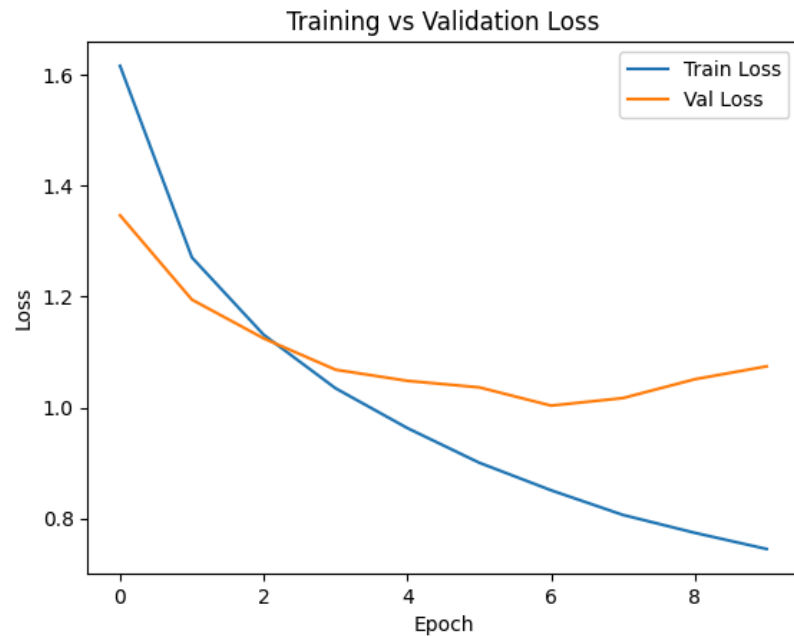
2.2.1.3. Pengaruh Ukuran Filter per Layer Konvolusi

Tabel 2.2.1.3.1. Pengujian Pengaruh Ukuran Filter per Layer Konvolusi

No.	Hasil Pengujian																																	
1.	<p><i>Hyperparameter:</i> 2 layer konvolusi: - filter 32 dengan ukuran 3 x 3 dan aktivasi relu - filter 64 dengan ukuran 3 x 3 dan aktivasi relu 2 layer max pooling dengan ukuran 2 x 2 1 flatten 2 layer dense: - 1 dengan 64 neuron dan aktivasi relu - 1 dengan 10 neuron dan aktivasi softmax</p> <p>Hasil:</p> <div><p>Training vs Validation Loss</p><table border="1"><caption>Approximate data points from the Training vs Validation Loss graph</caption><thead><tr><th>Epoch</th><th>Train Loss</th><th>Val Loss</th></tr></thead><tbody><tr><td>0</td><td>1.50</td><td>1.25</td></tr><tr><td>1</td><td>1.20</td><td>1.10</td></tr><tr><td>2</td><td>1.05</td><td>1.05</td></tr><tr><td>3</td><td>1.00</td><td>1.00</td></tr><tr><td>4</td><td>0.90</td><td>0.98</td></tr><tr><td>5</td><td>0.85</td><td>0.98</td></tr><tr><td>6</td><td>0.80</td><td>0.98</td></tr><tr><td>7</td><td>0.78</td><td>1.00</td></tr><tr><td>8</td><td>0.75</td><td>1.02</td></tr><tr><td>9</td><td>0.70</td><td>1.05</td></tr></tbody></table></div> <p>Akurasi (<i>Macro F1 Score</i>): 0.6756</p>	Epoch	Train Loss	Val Loss	0	1.50	1.25	1	1.20	1.10	2	1.05	1.05	3	1.00	1.00	4	0.90	0.98	5	0.85	0.98	6	0.80	0.98	7	0.78	1.00	8	0.75	1.02	9	0.70	1.05
Epoch	Train Loss	Val Loss																																
0	1.50	1.25																																
1	1.20	1.10																																
2	1.05	1.05																																
3	1.00	1.00																																
4	0.90	0.98																																
5	0.85	0.98																																
6	0.80	0.98																																
7	0.78	1.00																																
8	0.75	1.02																																
9	0.70	1.05																																
2.	<p><i>Hyperparameter:</i> 2 layer konvolusi: - filter 32 dengan ukuran 5 x 5 dan aktivasi relu - filter 64 dengan ukuran 5 x 5 dan aktivasi relu</p>																																	

2 layer max pooling dengan ukuran 2 x 2
 1 flatten
 2 layer dense:
 - 1 dengan 64 neuron dan aktivasi relu
 - 1 dengan 10 neuron dan aktivasi softmax

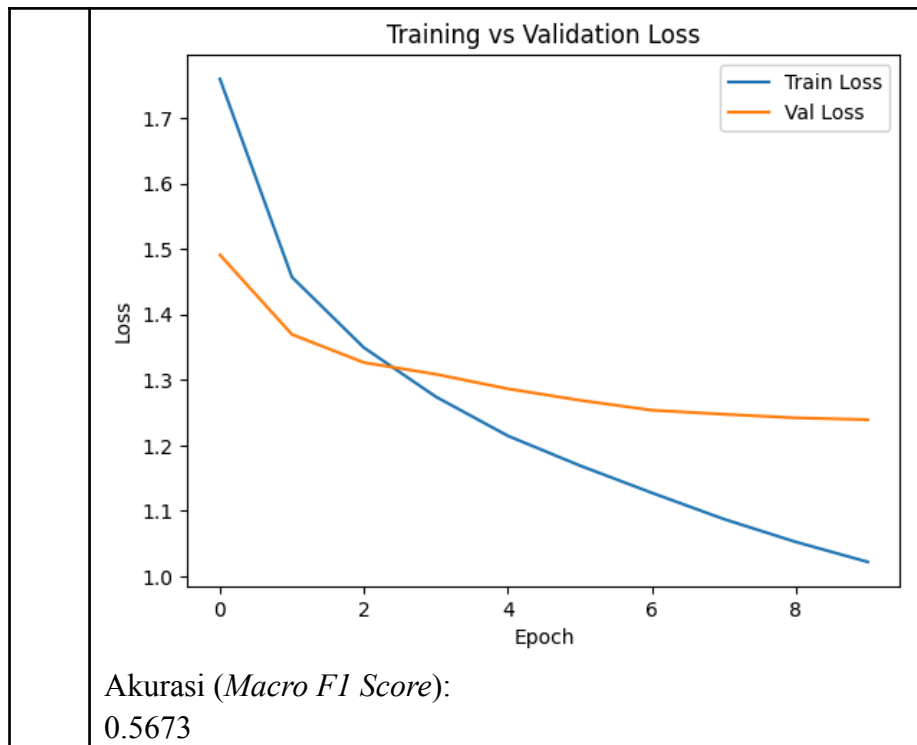
Hasil:



Akurasi (*Macro F1 Score*):
 0.6468

3. *Hyperparameter*:
 2 layer konvolusi:
 - filter 32 dengan ukuran 7 x 7 dan aktivasi relu
 - filter 64 dengan ukuran 7 x 7 dan aktivasi relu
 2 layer max pooling dengan ukuran 2 x 2
 1 flatten
 2 layer dense:
 - 1 dengan 64 neuron dan aktivasi relu
 - 1 dengan 10 neuron dan aktivasi softmax

Hasil:

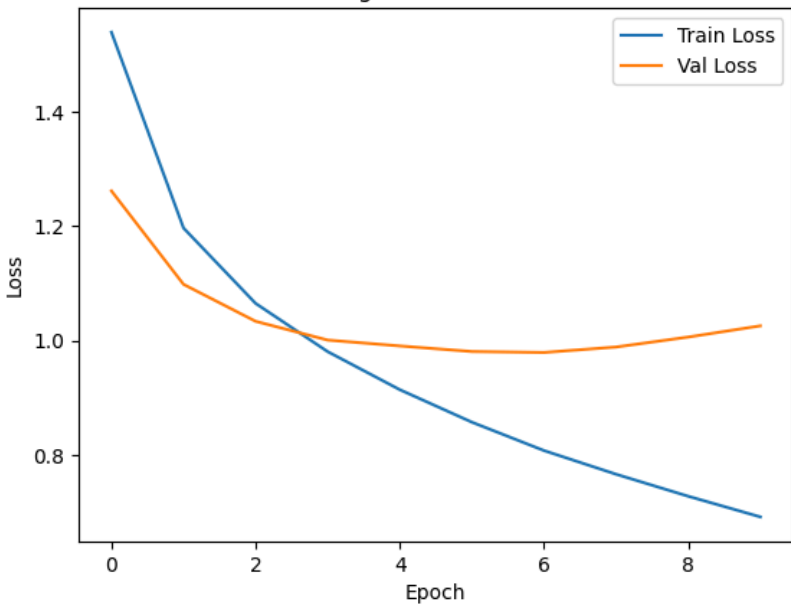
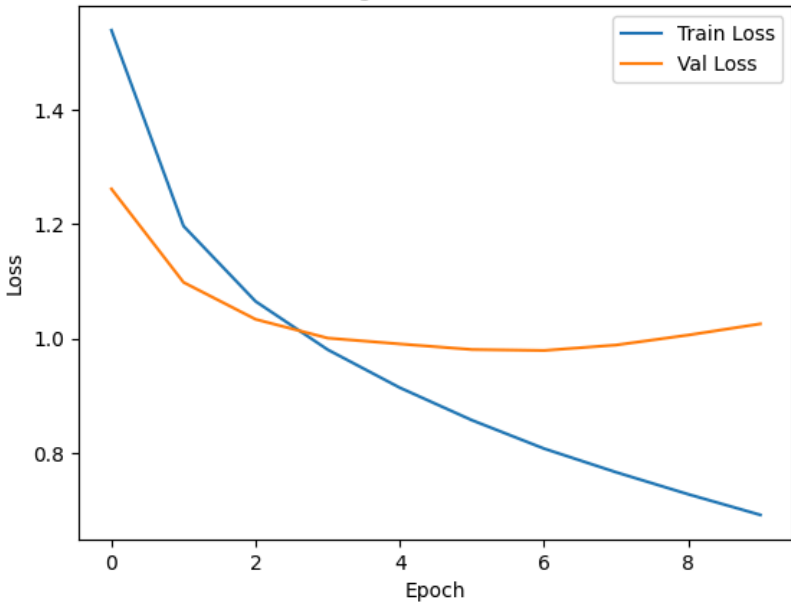


Berdasarkan hasil pengujian tersebut, ukuran filter per layer konvolusi cukup berpengaruh terhadap akurasi. Pada kasus data Cifar10, semakin besar ukuran filter yang digunakan pada layer konvolusi, semakin buruk akurasinya. Hal ini sesuai dengan teori CNN, yaitu ukuran filter yang semakin besar akan meningkatkan generalisasi pada *input* (kurang melihat detail) sehingga hasilnya semakin buruk.

2.2.1.4. Pengaruh Jenis Pooling Layer

Tabel 2.2.1.4.1. Pengujian Pengaruh Jenis Pooling Layer

No.	Hasil Pengujian
1.	<p><i>Hyperparameter:</i></p> <p>2 layer konvolusi:</p> <ul style="list-style-type: none"> - filter 32 dengan ukuran 3 x 3 dan aktivasi relu - filter 64 dengan ukuran 3 x 3 dan aktivasi relu <p>2 layer max pooling dengan ukuran 2 x 2</p> <p>1 flatten</p> <p>2 layer dense:</p> <ul style="list-style-type: none"> - 1 dengan 64 neuron dan aktivasi relu - 1 dengan 10 neuron dan aktivasi softmax <p>Hasil:</p>

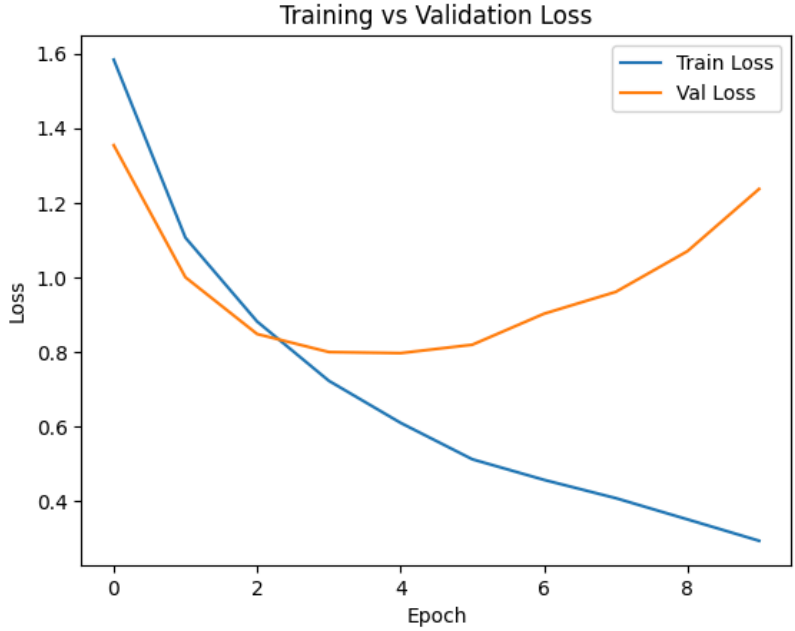
	<div><p>Training vs Validation Loss</p><table><tr><th>Epoch</th><th>Train Loss</th><th>Val Loss</th></tr><tr><td>0</td><td>1.50</td><td>1.25</td></tr><tr><td>1</td><td>1.20</td><td>1.10</td></tr><tr><td>2</td><td>1.05</td><td>1.05</td></tr><tr><td>3</td><td>0.95</td><td>1.00</td></tr><tr><td>4</td><td>0.90</td><td>0.98</td></tr><tr><td>5</td><td>0.85</td><td>0.98</td></tr><tr><td>6</td><td>0.80</td><td>0.98</td></tr><tr><td>7</td><td>0.75</td><td>0.99</td></tr><tr><td>8</td><td>0.70</td><td>1.00</td></tr><tr><td>9</td><td>0.67</td><td>1.02</td></tr></table><p>Akurasi (<i>Macro F1 Score</i>): 0.6756</p></div>	Epoch	Train Loss	Val Loss	0	1.50	1.25	1	1.20	1.10	2	1.05	1.05	3	0.95	1.00	4	0.90	0.98	5	0.85	0.98	6	0.80	0.98	7	0.75	0.99	8	0.70	1.00	9	0.67	1.02
Epoch	Train Loss	Val Loss																																
0	1.50	1.25																																
1	1.20	1.10																																
2	1.05	1.05																																
3	0.95	1.00																																
4	0.90	0.98																																
5	0.85	0.98																																
6	0.80	0.98																																
7	0.75	0.99																																
8	0.70	1.00																																
9	0.67	1.02																																
2.	<div><p><i>Hyperparameter:</i></p><p>2 layer konvolusi:</p><ul style="list-style-type: none">- filter 32 dengan ukuran 3 x 3 dan aktivasi relu- filter 64 dengan ukuran 3 x 3 dan aktivasi relu<p>2 layer average pooling dengan ukuran 2 x 2</p><p>1 flatten</p><p>2 layer dense:</p><ul style="list-style-type: none">- 1 dengan 64 neuron dan aktivasi relu- 1 dengan 10 neuron dan aktivasi softmax<p>Hasil:</p><div><p>Training vs Validation Loss</p><table><tr><th>Epoch</th><th>Train Loss</th><th>Val Loss</th></tr><tr><td>0</td><td>1.50</td><td>1.25</td></tr><tr><td>1</td><td>1.20</td><td>1.10</td></tr><tr><td>2</td><td>1.05</td><td>1.05</td></tr><tr><td>3</td><td>0.95</td><td>1.00</td></tr><tr><td>4</td><td>0.90</td><td>0.98</td></tr><tr><td>5</td><td>0.85</td><td>0.98</td></tr><tr><td>6</td><td>0.80</td><td>0.98</td></tr><tr><td>7</td><td>0.75</td><td>0.99</td></tr><tr><td>8</td><td>0.70</td><td>1.00</td></tr><tr><td>9</td><td>0.67</td><td>1.02</td></tr></table></div></div>	Epoch	Train Loss	Val Loss	0	1.50	1.25	1	1.20	1.10	2	1.05	1.05	3	0.95	1.00	4	0.90	0.98	5	0.85	0.98	6	0.80	0.98	7	0.75	0.99	8	0.70	1.00	9	0.67	1.02
Epoch	Train Loss	Val Loss																																
0	1.50	1.25																																
1	1.20	1.10																																
2	1.05	1.05																																
3	0.95	1.00																																
4	0.90	0.98																																
5	0.85	0.98																																
6	0.80	0.98																																
7	0.75	0.99																																
8	0.70	1.00																																
9	0.67	1.02																																

	Akurasi (<i>Macro F1 Score</i>): 0.6595
--	--

Berdasarkan hasil pengujian tersebut, jenis *pooling layer* berpengaruh pada akurasi model. Tidak dapat disimpulkan pada data lain jenis *pooling layer* yang terbaik. Namun, pada data Cifar10, jenis yang terbaik adalah *max-pooling* dengan akurasi 0.6756.

2.2.1.5. Perbandingan dengan CNN Scratch

Tabel 2.2.1.4.1. Perbandingan dengan CNN Scratch

No.	Hasil Pengujian
	<p><i>Hyperparameter:</i></p> <p>6 layer konvolusi:</p> <ul style="list-style-type: none"> - filter 32 dengan ukuran 3 x 3 dan aktivasi relu - filter 32 dengan ukuran 3 x 3 dan aktivasi relu - filter 64 dengan ukuran 3 x 3 dan aktivasi relu - filter 64 dengan ukuran 3 x 3 dan aktivasi relu - filter 128 dengan ukuran 3 x 3 dan aktivasi relu - filter 128 dengan ukuran 3 x 3 dan aktivasi relu <p>3 layer max pooling dengan ukuran 2 x 2</p> <p>1 flatten</p> <p>2 layer dense:</p> <ul style="list-style-type: none"> - 1 dengan 128 neuron dan aktivasi relu - 1 dengan 10 neuron dan aktivasi softmax
1.	<p>Model Keras</p> <p>Hasil:</p>  <p>Akurasi (<i>Macro F1 Score</i>): 0.7096</p>
2.	Model <i>from scratch</i>

	Akurasi (<i>Macro F1 Score</i>): 0.7096
--	--

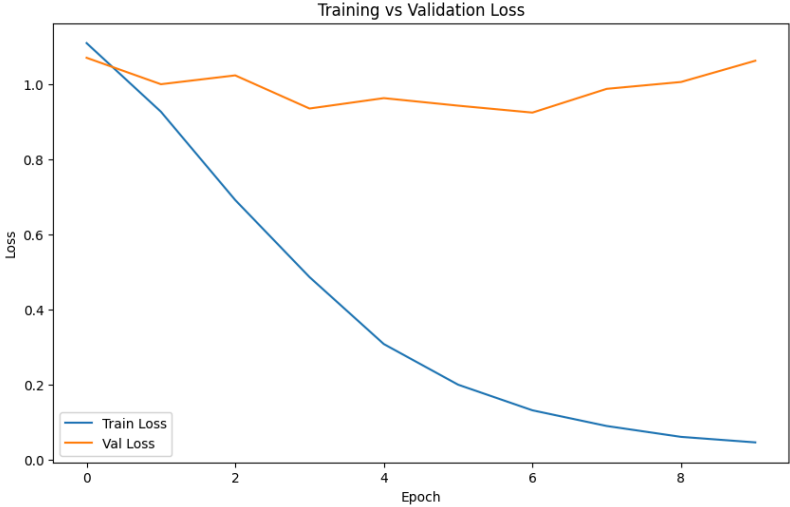
Berdasarkan hasil perbandingan model CNN dari *library* Keras dan *from scratch*, tidak ada perbedaan akurasi keduanya jika dihitung dengan *macro F1 score*. Hal ini disebabkan *weight* yang dipakai kedua model sama, sehingga hasil perhitungannya terhadap *input* yang sama adalah sama.

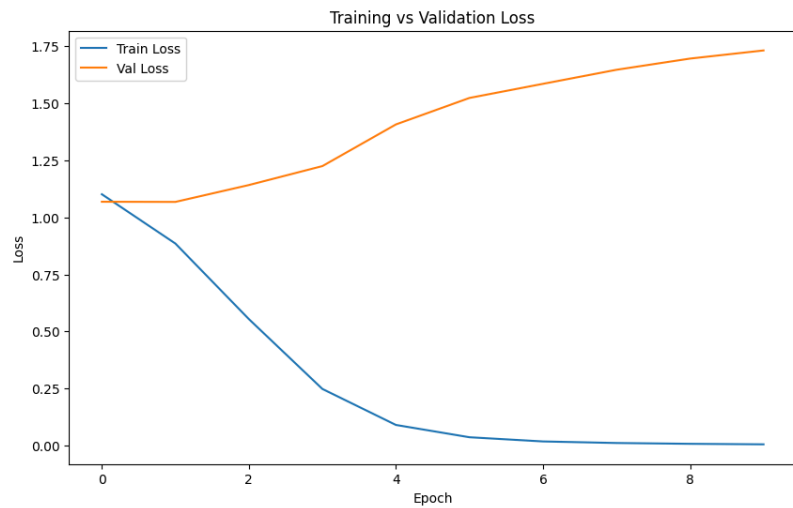
2.2.2. Pengujian Simple RNN

2.2.2.1. Pengaruh Jumlah Layer RNN

Dalam pengujian ini, ada beberapa parameter yang diterapkan untuk semua variasi model. Parameter tersebut adalah: *epochs* = 10, *batch_size* = 128, *Cell Size RNN* = 64, *Embedding Layer* dengan *input_dim* = 500, *output_dim* = 100, *input_length* = 534, *Dropout Layer* dengan *rate* = 0.2, dan *Dense Layer* dengan *units* = 3 dan fungsi aktivasi *softmax*.

Tabel 2.2.2.1.1. Pengujian Pengaruh Jumlah Layer RNN

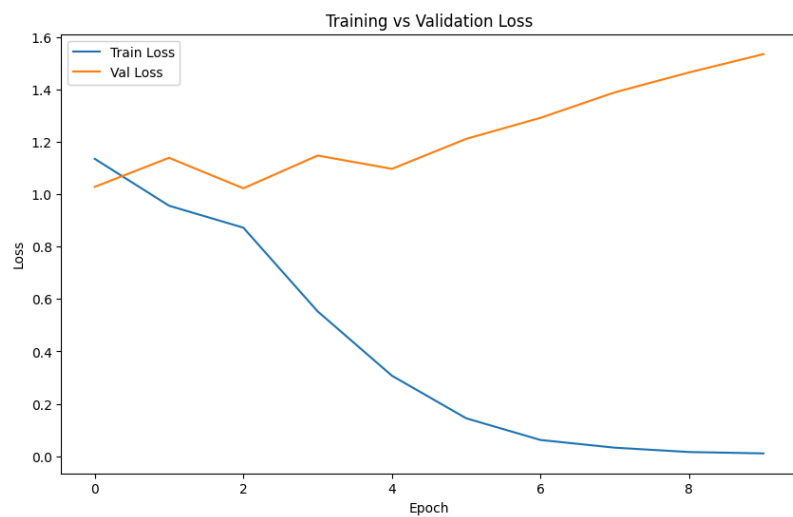
No.	Hasil Pengujian
1.	<p><i>Hyperparameter:</i> 1 Layer RNN</p> <p>Hasil:</p>  <p>Akurasi (<i>Macro F1 Score</i>): 0.5998</p>
2.	<p><i>Hyperparameter:</i> 2 Layer RNN</p> <p>Hasil:</p>



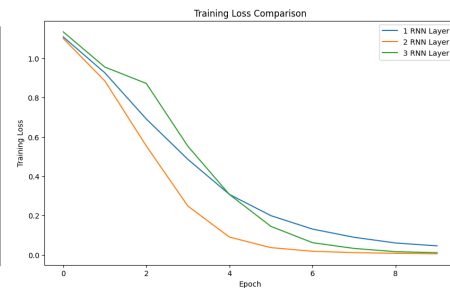
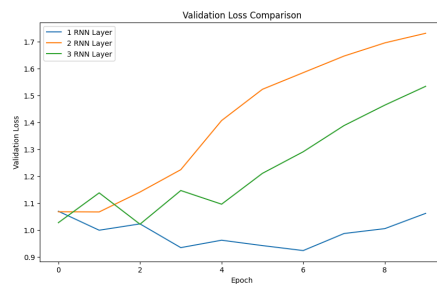
Akurasi (*Macro F1 Score*): 0.4805

3. *Hyperparameter:*
3 Layer RNN

Hasil:



Akurasi (*Macro F1 Score*):
0.5376




Gambar 2.2.2.1.1. Perbandingan Validation Loss dan Training Loss untuk Semua Model Variasi Banyak Layer Simple RNN

Berdasarkan hasil eksperimen dengan variasi jumlah layer RNN pada data NusaX, dapat disimpulkan bahwa model dengan **1 RNN layer** menunjukkan performa terbaik dengan Macro F1 *score* tertinggi pada data validasi yaitu 0.5998, walaupun jika dilihat dari grafik *validation loss*-nya, model ini *overfitting* seiring bertambahnya epoch. Sebaliknya, penambahan layer menjadi **2 dan 3 RNN layer** justru menghasilkan performa yang lebih buruk pada data validasi, ditandai dengan peningkatan validation loss dan skor Macro F1 yang lebih rendah yang menunjukkan kesulitan dalam generalisasi yang kemungkinan disebabkan oleh kompleksitas model yang berlebihan untuk ukuran dataset yang relatif kecil. Oleh karena itu, untuk tugas klasifikasi teks pada dataset NusaX dengan ukuran ini, arsitektur RNN yang lebih sederhana dengan satu layer tampaknya lebih efektif dibandingkan dengan model yang lebih dalam.

2.2.2.2. Pengaruh Banyak Cell RNN per Layer

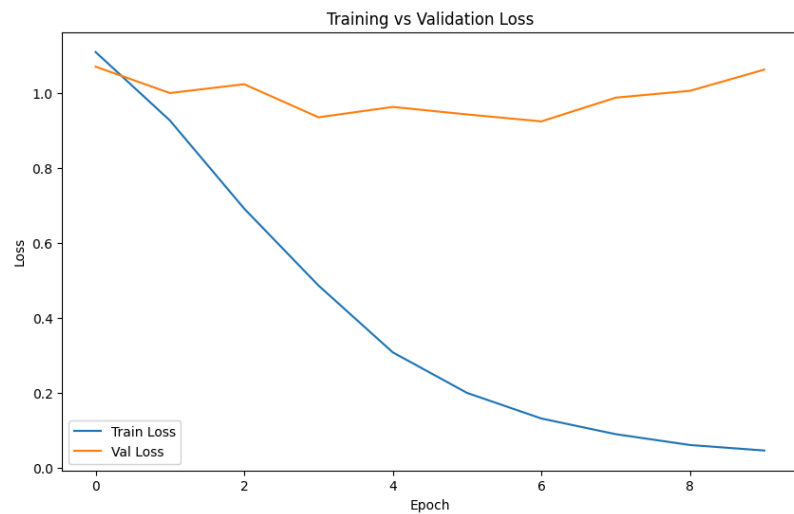
Dalam pengujian ini, ada beberapa parameter yang diterapkan untuk semua variasi model. Parameter tersebut adalah: *epochs* = 10, *batch_size* = 128, *Embedding Layer* dengan *input_dim* = 500, *output_dim* = 100, *input_length* = 534, *Dropout Layer* dengan *rate* = 0.2, dan *Dense Layer* dengan *units* = 3 dan fungsi aktivasi *softmax*.

Tabel 2.2.2.2.1. Pengujian Banyak Cell RNN per Layer

No.	Hasil Pengujian
1.	<p><i>Hyperparameter:</i> Jumlah <i>cell RNN</i>: 32</p> <p>Hasil:</p>  <p>Akurasi (<i>Macro F1 Score</i>): 0.4146</p>

2. *Hyperparameter:*
Jumlah *cell RNN*: 64

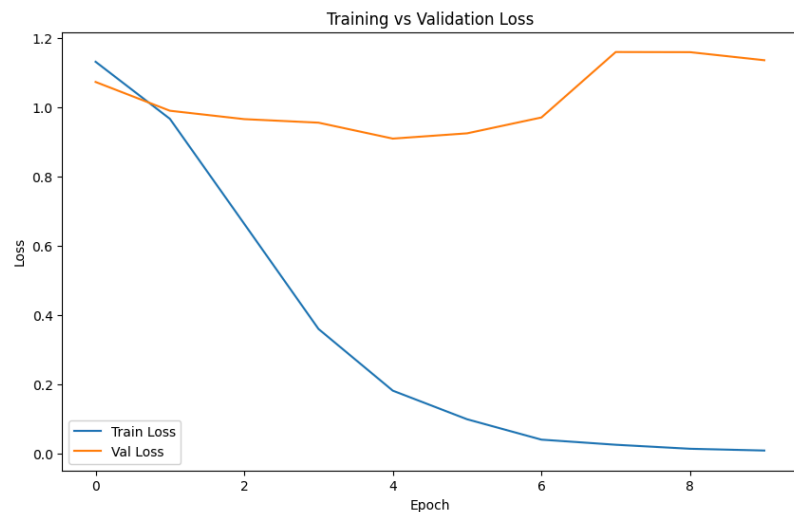
Hasil:



Akurasi (*Macro F1 Score*):
0.5998

3. *Hyperparameter:*
Jumlah *cell RNN*: 128

Hasil:



Akurasi (*Macro F1 Score*):
0.6238

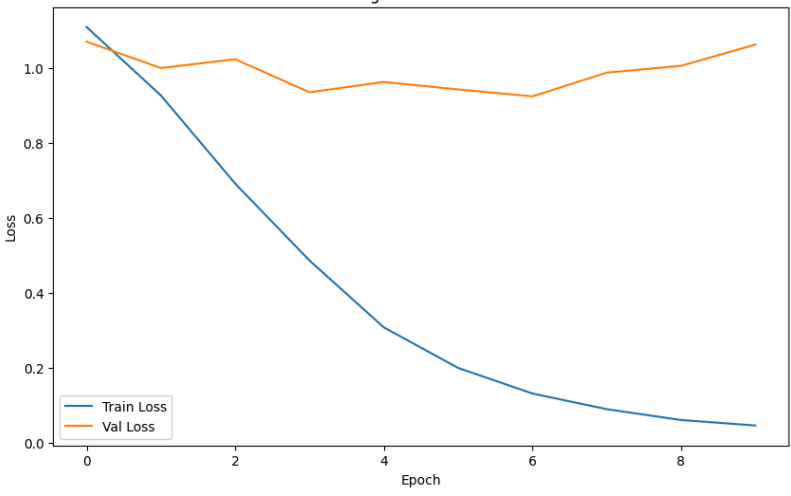
Hasil eksperimen variasi jumlah *cell* pada satu *layer* RNN menunjukkan bahwa peningkatan kapasitas model dari 32 ke 64 hingga 128 *cells* secara umum meningkatkan performa klasifikasi pada data NusaX, dengan model **128 cells** mencapai *Macro F1 score* tertinggi yaitu 0.6238. Namun, terdapat indikasi potensi *overfitting*, terutama pada model 128 *cells* yang menunjukkan peningkatan *validation loss* setelah beberapa *epoch*. Hal ini bisa terjadi karena ukuran *dataset* yang

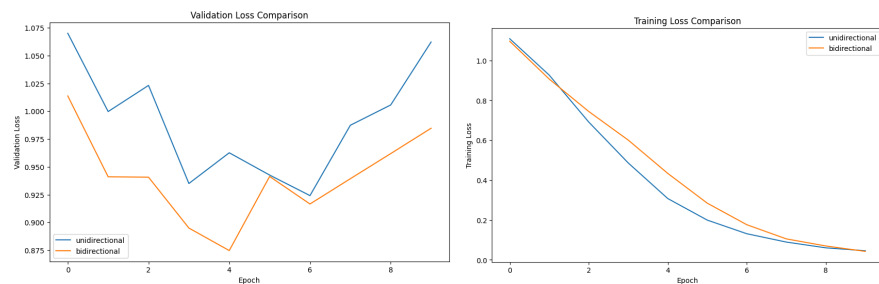
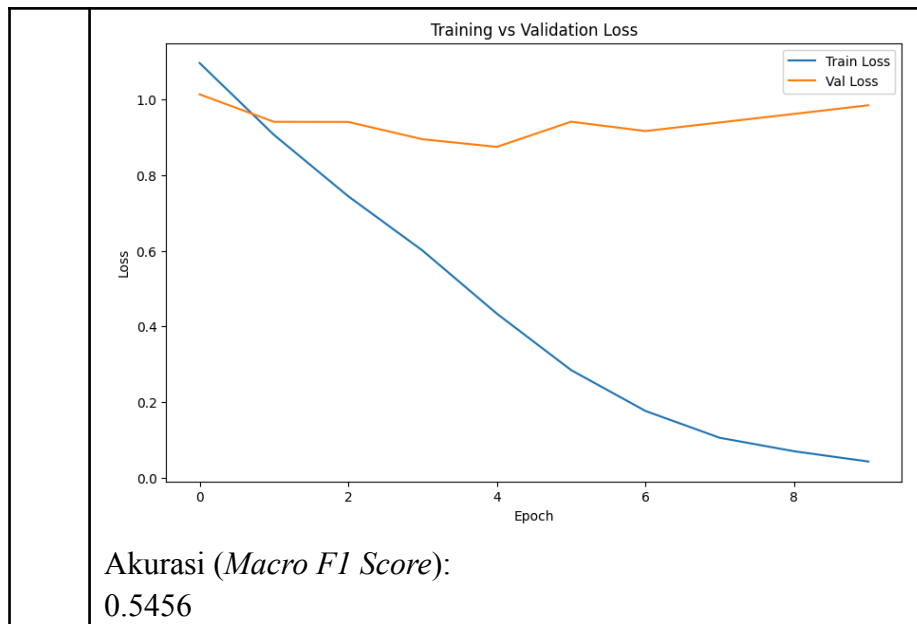
relatif kecil. Selain itu, peningkatan jumlah *cell* lebih lanjut juga belum tentu membuat peningkatan yang signifikan karena dengan jumlah *cell* 256, yang tidak kami lampirkan hasilnya di laporan ini, memiliki hasil *F1 score* yang lebih jelek dari 128 *cells*.

2.2.2.3. Pengaruh Jenis Layer RNN berdasarkan Arah

Dalam pengujian ini, ada beberapa parameter yang diterapkan untuk semua variasi model. Parameter tersebut adalah: *epochs* = 10, *batch_size* = 128, *Cell Size RNN* = 64, *Embedding Layer* dengan *input_dim* = 500, *output_dim* = 100, *input_length* = 534, *Dropout Layer* dengan *rate* = 0.2, dan *Dense Layer* dengan *units* = 3 dan fungsi aktivasi *softmax*.

Tabel 2.2.2.3.1. Pengujian Pengaruh Jenis Layer RNN berdasarkan Arah

No.	Hasil Pengujian
1.	<i>Hyperparameter:</i> Unidirectional
	Hasil:  <p>Akurasi (<i>Macro F1 Score</i>): 0.5938</p>
2.	<i>Hyperparameter:</i> Bidirectional
	Hasil:



Gambar 2.2.2.3.1. Perbandingan Validation Loss dan Training Loss untuk Semua Model Variasi Directionality Simple RNN


Berdasarkan hasil eksperimen, model **unidirectional** menunjukkan performa klasifikasi yang lebih baik dengan Macro F1 0.5998 dibandingkan dengan model **bidirectional** yang mencapai sekitar 0.53, meskipun model bidirectional menunjukkan tren validation loss yang lebih stabil, mengindikasikan potensi generalisasi yang lebih baik; namun, keunggulan performa model unidirectional, meskipun validation loss-nya lebih fluktuatif, menyiratkan bahwa untuk dataset ini, informasi dari arah sebelumnya dalam urutan teks mungkin lebih dominan dalam menentukan label, atau model bidirectional menjadi terlalu kompleks dan rentan terhadap overfitting dengan ukuran dataset yang kecil.

2.2.2.4. Perbandingan RNN Keras dengan Scratch Implementation

Dalam pengujian ini, ada beberapa parameter yang diterapkan untuk semua variasi model. Parameter tersebut adalah: *epochs* = 10, *batch_size* = 128, *Cell Size RNN* = 128, *Embedding Layer* dengan *input_dim* = 500, *output_dim* = 100, *input_length* = 534, *Dropout*

Layer dengan $rate = 0.2$, dan Dense Layer dengan $units = 3$ dan fungsi aktivasi *softmax*. Model yang digunakan juga adalah model *bidirectional*.

Tabel 2.2.2.4.1. Perbandingan Simple RNN Scratch dan Keras

No	Hasil Pengujian
1.	<p><i>Keras Model</i></p> <p>Hasil:</p>  <p>Akurasi (<i>Macro F1 Score</i>): 0.5420991558764415</p>
2.	<p><i>Scratch Model</i></p> <p>Akurasi (<i>Macro F1 Score</i>): 0.5420991558764415</p>

Berdasarkan hasil perbandingan model Simple RNN dari *library* Keras dan *from scratch*, tidak ada perbedaan akurasi keduanya jika dihitung dengan *macro F1 score*. Hal ini disebabkan *weight* yang dipakai kedua model sama, sehingga hasil perhitungannya terhadap *input* yang sama adalah sama.

2.2.3. Pengujian LSTM

2.2.3.1. Pengaruh Jumlah Layer LSTM

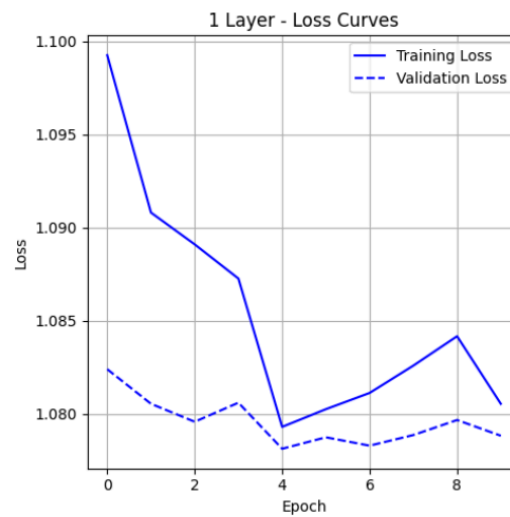
Tabel 2.2.3.1.1. Pengujian Pengaruh Jumlah Layer LSTM

No.	Hasil Pengujian
	<p><i>Hyperparameter default:</i></p> <ul style="list-style-type: none"> - <i>LSTM Units</i>: 64 unit/lapisan - <i>Embedding</i>: 100 dimensi - <i>Dropout</i>: <ul style="list-style-type: none"> 1. Setelah LSTM: 0.3 2. Setelah dense: 0.5 - <i>Dense Layer</i>: 32 unit (<i>ReLU</i>) - <i>Output</i>: <i>softmax</i> (sesuai kelas) - <i>Optimizer</i>: <i>Adam</i> ($rate=0.001$) - <i>Loss</i>: <i>Sparse categorical cross-entropy</i>

- *Batch*: 32
- *Epochs*: 10

1. *Hyperparameter*:
LSTM *layers* = 1

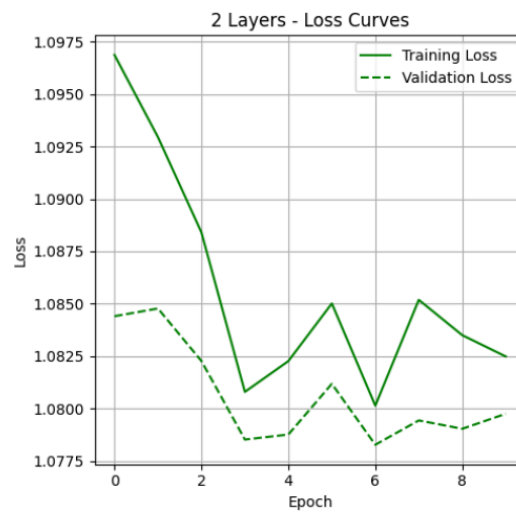
Hasil:



Akurasi (*Macro F1 Score*):
0.1844

2. *Hyperparameter*:
LSTM *layers* = 2

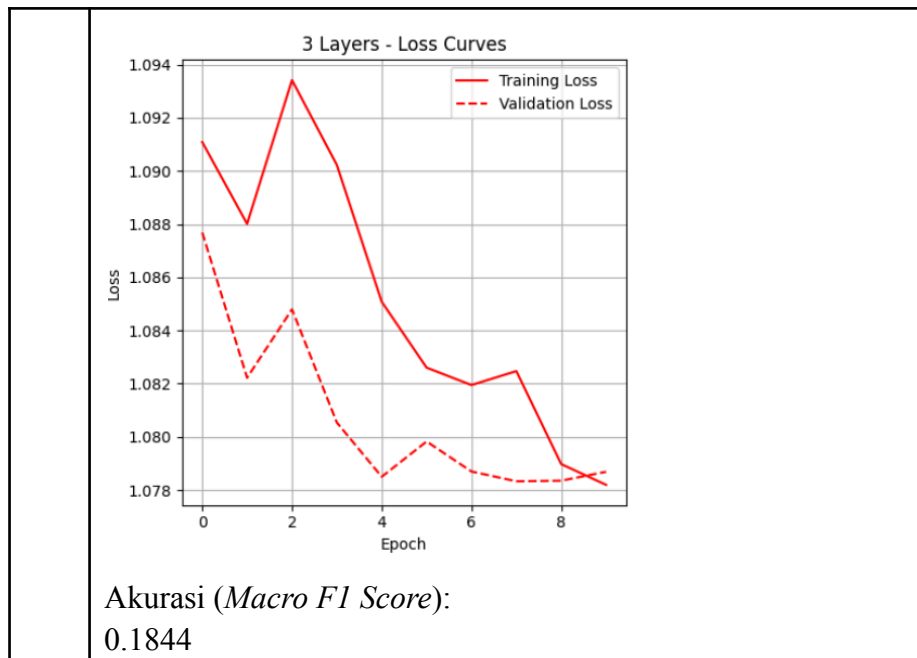
Hasil:



Akurasi (*Macro F1 Score*):
0.1827

3. *Hyperparameter*:
LSTM *layers* = 3

Hasil:



Berdasarkan hasil yang divisualisasikan, penambahan jumlah *layer* LSTM dari satu menjadi dua *layer* memberikan sedikit **penurunan** pada *Macro F1-Score*, dari 0.1844 menjadi 0.18127. Ini mengindikasikan bahwa penambahan kedalaman hingga dua *layer* memungkinkan model untuk menangkap sedikit lebih banyak kompleksitas dalam data tetapi menyebabkan **overfit**. Namun, penambahan *layer* ketiga **menghasilkan** peningkatan kinerja, dengan *F1-score* menjadi 0.1844. Hal ini menunjukkan bahwa untuk *dataset* dan konfigurasi *hyperparameter* lainnya yang digunakan, dua *layer* LSTM maupun tiga *layer* LSTM akan memperbesar kompleksitas model yang berujung pada risiko **overfit** dan **ketidakkonsistenan** hasil model.

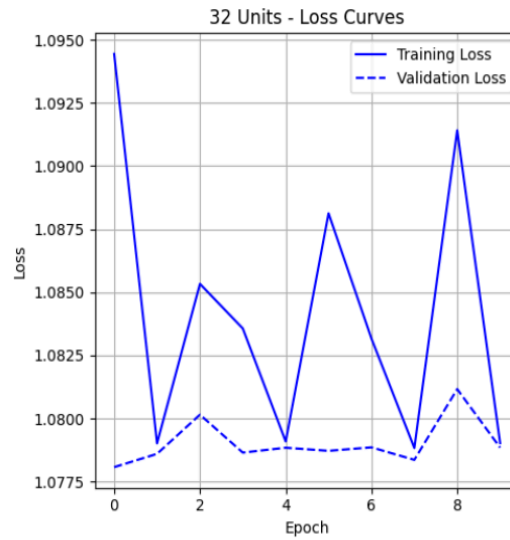
2.2.3.2. Pengaruh Banyak Cell LSTM per Layer

Tabel 2.2.3.2.1. Pengujian Pengaruh Banyak Cell LSTM per Layer

No.	Hasil Pengujian
	<p><i>Hyperparameter default:</i></p> <ul style="list-style-type: none"> - <i>Embedding</i>: 100 dimensi - <i>Direction</i>: <i>Unidirectional</i> - <i>Dropout</i>: <ul style="list-style-type: none"> 1. Setelah <i>LSTM</i>: 0.3 2. Setelah <i>dense</i>: 0.5 - <i>Dense Layer</i>: 32 unit (<i>ReLU</i>) - <i>Output</i>: <i>softmax</i> (sesuai kelas) - <i>Optimizer</i>: <i>Adam</i> (<i>rate</i>=0.001) - <i>Loss</i>: <i>Sparse categorical cross-entropy</i> - <i>Batch</i>: 32 - <i>Epochs</i>: 10

1. *Hyperparameter:*
cell LSTM per layers = 32

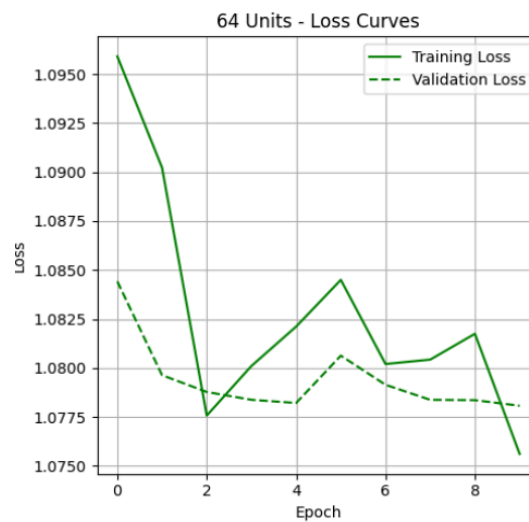
Hasil:



Akurasi (*Macro F1 Score*):
0.1827

2. *Hyperparameter:*
cell LSTM per layers = 64

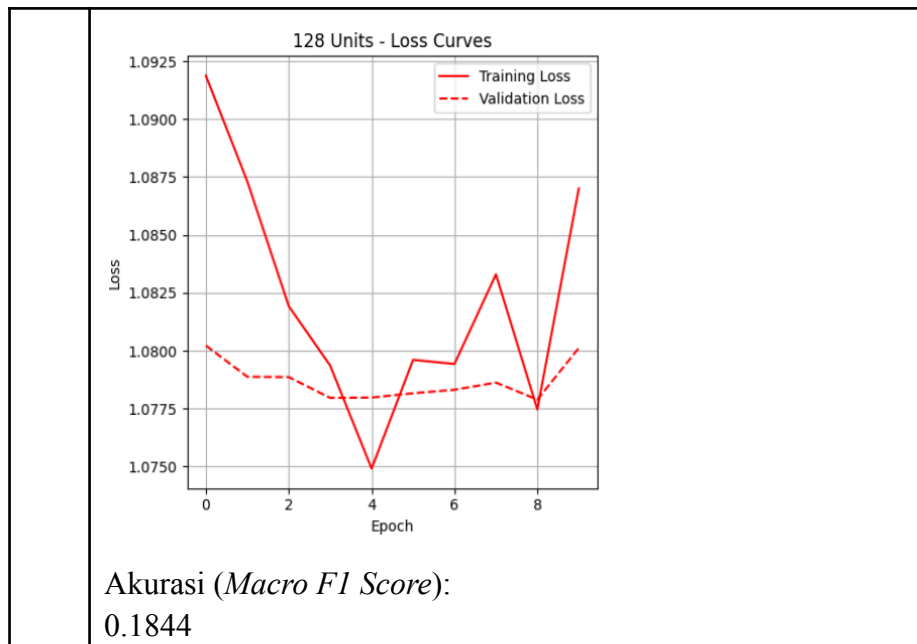
Hasil:



Akurasi (*Macro F1 Score*):
0.1844

3. *Hyperparameter:*
cell LSTM per layers = 128

Hasil:



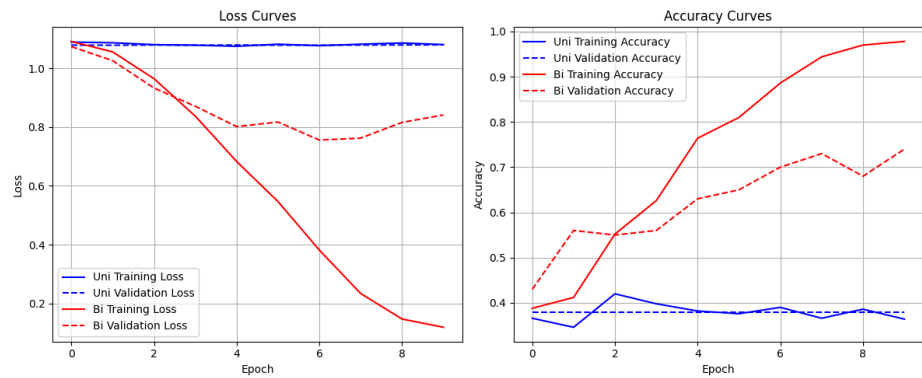
Berdasarkan hasil visual, peningkatan jumlah unit LSTM dari 32 menjadi 64 unit memberikan sedikit kenaikan pada *Macro F1-Score*, dari 0.1827 menjadi 0.1844. Ini mengindikasikan bahwa penambahan kapasitas model hingga 64 unit memungkinkan penangkapan pola yang sedikit **lebih baik**. Namun, peningkatan lebih lanjut jumlah unit menjadi 128 **tidak menghasilkan perbaikan** kinerja tambahan dalam hal *F1-score*, meskipun jumlah parameter model meningkat signifikan. Hal ini menunjukkan bahwa untuk dataset dan konfigurasi arsitektur saat ini, 64 unit LSTM mungkin sudah mencapai batas manfaat dari penambahan ukuran sel, dan penambahan lebih lanjut tidak efektif meningkatkan performa prediktif model

2.2.3.3. Pengaruh Jenis Layer LSTM berdasarkan Arah

Tabel 2.2.3.3.1. Pengujian Pengaruh Jenis Layer LSTM berdasarkan Arah

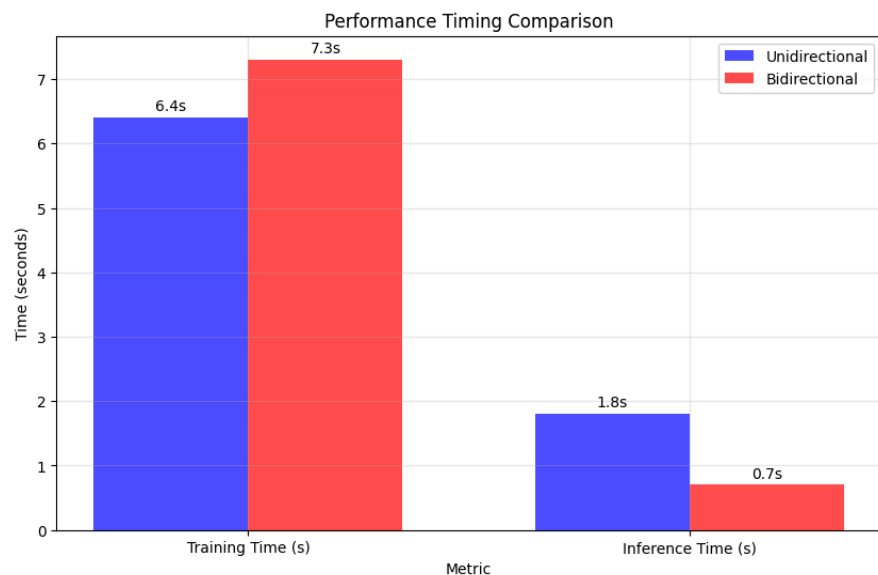
No.	Hasil Pengujian
	<p><i>Hyperparameter default:</i></p> <ul style="list-style-type: none"> - <i>LSTM Units</i>: 32 dan 64 unit - <i>Embedding</i>: 100 dimensi - <i>Dropout</i>: <ul style="list-style-type: none"> 1. Setelah <i>LSTM</i>: 0.3 2. Setelah <i>dense</i>: 0.5 - <i>Dense Layer</i>: 32 unit (<i>ReLU</i>) - <i>Output</i>: <i>softmax</i> (sesuai kelas) - <i>Optimizer</i>: <i>Adam</i> (<i>rate</i>=0.001) - <i>Loss Function</i>: <i>Sparse categorical cross-entropy</i> - <i>Batch Size</i>: 32 - <i>Epochs</i>: 10
1.	<p><i>Hyperparameter:</i></p> <p><i>LSTM layer</i>: <i>unidirectional</i></p>

	Akurasi (<i>Macro F1 Score</i>): 0.1844
2.	<i>Hyperparameter:</i> <i>LSTM layer: bidirectional</i>
	Akurasi (<i>Macro F1 Score</i>): 0.4920



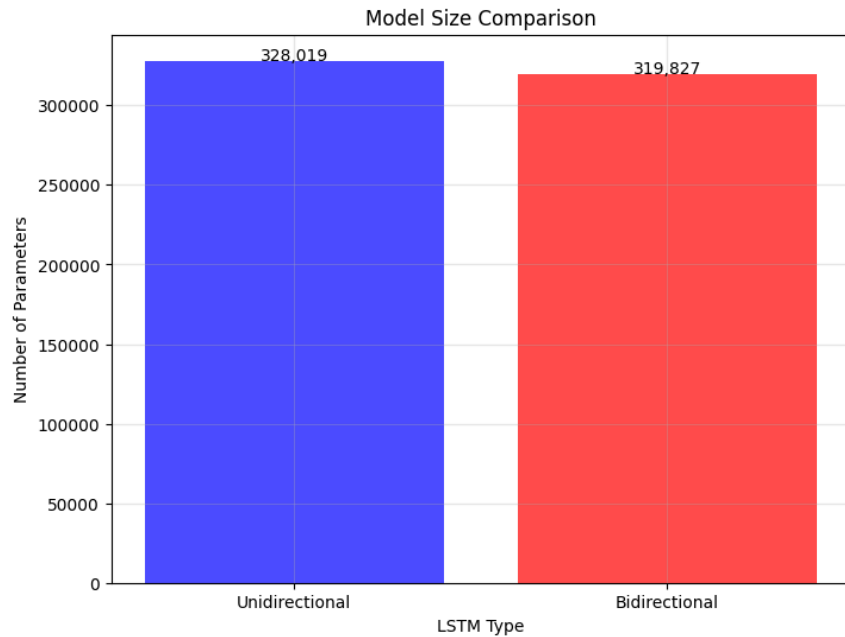
Gambar 2.2.3.3.1. Loss dan Accuracy Curves Pengujian Pengaruh Jenis Layer LSTM berdasarkan Arah

Model *Bidirectional* menunjukkan penurunan *loss* yang jauh lebih signifikan dan peningkatan akurasi yang lebih tinggi pada data *training* dan *validasi* dibandingkan model *Unidirectional* yang *loss* dan akurasinya cenderung stagnan dan buruk.



Gambar 2.2.3.3.2. Performance Timing Comparison Pengujian Pengaruh Jenis Layer LSTM berdasarkan Arah

Model *Bidirectional* membutuhkan waktu pelatihan yang lebih lama (6.4 detik dibanding 7.3 detik), tetapi waktu inferensinya sedikit lebih cepat (1.8 detik dibanding 0.7 detik) dalam kasus ini.



Gambar 2.2.3.3.3. Model Size Comparison Pengujian Pengaruh Jenis Layer LSTM berdasarkan Arah

Model *Unidirectional* memiliki sekitar 328,019 parameter, sedangkan model *Bidirectional* memiliki jumlah parameter yang sedikit lebih rendah, yaitu sekitar 319,827 parameter.

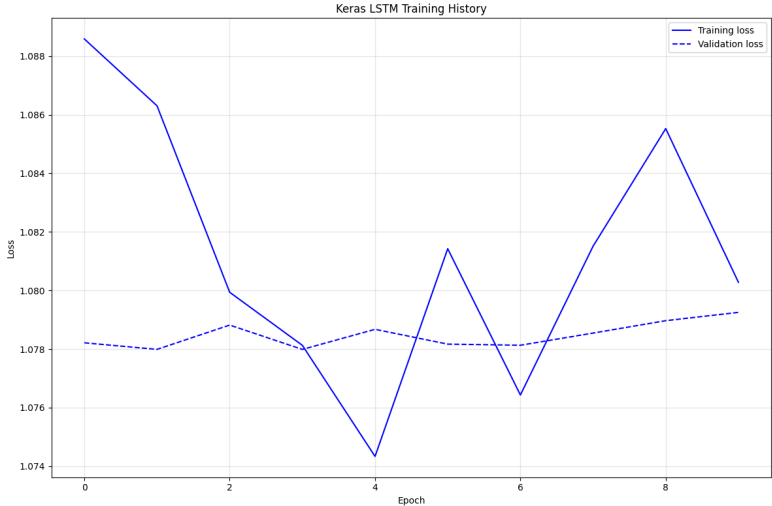
Catatan: Biasanya, model *bidirectional* memiliki parameter yang lebih banyak jika jumlah unit LSTM per layer sama. Perbedaan ini mungkin disebabkan oleh cara penghitungan atau konfigurasi spesifik dalam kode yang menghasilkan model Keras untuk penghitungan parameter.

Penggunaan arsitektur *Bidirectional* LSTM menunjukkan **pengaruh positif yang sangat signifikan** terhadap kinerja prediktif, dengan peningkatan *F1-score* yang drastis dari 0.1844 menjadi 0.6931, karena kemampuannya memproses konteks dari kedua arah sekuens. Namun, hal ini datang dengan **pengaruh negatif pada waktu pelatihan** yang menjadi lebih lama. Secara menarik, dalam kasus ini, waktu inferensi untuk model *Bidirectional* sedikit lebih cepat, menunjukkan pengaruh **positif kecil**, meskipun biasanya diharapkan sebanding atau sedikit lebih lama. Terkait ukuran model, hasil spesifik ini menunjukkan model *Bidirectional* memiliki sedikit parameter lebih sedikit, yang merupakan pengaruh **positif kecil** pada ukuran, meskipun secara teoritis Bidirectional LSTM dengan konfigurasi unit yang sama biasanya memiliki lebih banyak parameter.

2.2.3.4. Perbandingan LSTM Keras dengan Scratch Implementation

Tabel 2.2.3.4.1. Perbandingan LSTM Keras dengan Scratch Implementation

No.	Hasil Pengujian
	Model Hyperparameters

	<ul style="list-style-type: none"> - <i>LSTM Units</i> : 64 - <i>Embedding Dimension</i>: 100 - <i>Direction</i> : <i>unidirectional</i> - <i>Dropout Rates</i>: <ol style="list-style-type: none"> 1. Setelah <i>LSTM layer</i>: 0.3 2. Setelah <i>first Dense layer</i>: 0.5 - <i>Dense Layer Size</i>: 32 units (<i>ReLU activation</i>) - <i>Output Layer</i>: 3 units (<i>softmax activation</i>). - <i>Optimizer</i>: <i>Adam</i> (rate 0.001) - <i>Loss Function</i>: <i>Sparse categorical cross-entropy</i>. - <i>Batch Size</i>: 32 - <i>Epochs</i>: 10 - <i>Max Vocab Size</i>: 10000 - <i>Max Sequence Length</i>: 100
1.	<p><i>Keras Model</i></p> <p>Hasil:</p>  <p>Akurasi (<i>Macro F1 Score</i>): 0.184448</p>
2.	<p><i>Scratch Model</i></p> <p>Akurasi (<i>Macro F1 Score</i>): 0.184448</p>

Berdasarkan perbandingan *forward pass* antara model LSTM Keras dan model LSTM yang dikembangkan dari awal (*scratch*), diperoleh hasil performa yang identik pada data uji. Setelah bobot dari model Keras yang telah dilatih ditransfer ke model *scratch*, kedua model **secara konsisten** mencapai akurasi skor F1 makro 0.184448. Kesamaan nilai metrik ini memberikan indikasi kuat bahwa implementasi logika komputasi *forward pass* pada model LSTM *scratch* telah berhasil dilakukan, karena mampu menghasilkan prediksi yang sama persis dengan model Keras ketika menggunakan parameter

bobot yang serupa. Meskipun kurva loss pelatihan model Keras (seperti yang divisualisasikan pada grafik histori pelatihan) dan skor F1 yang relatif rendah menunjukkan bahwa model belum mencapai performa optimal, kesesuaian hasil prediksi antara kedua model ini tetap menjadi poin validasi yang menunjukkan bahwa logika inferensi pada model LSTM yang dikembangkan dari awal telah berfungsi sebagaimana mestinya.

Bagian 3. Kesimpulan dan Saran

3.1. Kesimpulan

Pengujian terhadap berbagai *hyperparameter* dalam CNN membuktikan bahwa banyak dan ukuran filter memiliki pengaruh yang linear terhadap hasil akurasi model, yaitu semakin banyak filter yang digunakan, semakin baik akurasinya, sementara semakin besar ukuran filter yang digunakan, semakin buruk akurasinya. Jumlah *layer* konvolusi dan jenis *pooling layer* memang memberikan akurasi yang berbeda, tetapi tidak ada kesimpulan definitif untuk jenis data secara general (perlu disesuaikan dengan konteks data). Tidak ada perbedaan antara hasil prediksi model dari *library* Keras dengan yang dibuat sendiri karena *input*, *weight*, dan *bias* yang digunakan keduanya pada saat prediksi sama persis sehingga hasil perhitungannya pasti sama.

Berdasarkan eksperimen pengujian *hyperparameter* pada *Simple RNN*, dapat disimpulkan bahwa arsitektur dengan satu layer RNN menunjukkan performa terbaik untuk dataset NusaX, dibanding model dengan dua atau tiga layer yang cenderung mengalami penurunan perform. Peningkatan jumlah *cell* RNN dalam satu layer memiliki korelasi positif dengan peningkatan akurasi hingga mencapai titik tertentu (128 *cells*), di mana penambahan *cell* lebih lanjut tidak lagi memberikan peningkatan signifikan dan bahkan dapat menurunkan performa. Selain itu, model *unidirectional* terbukti lebih efektif dibandingkan *bidirectional* untuk tugas klasifikasi pada dataset ini, meskipun model *bidirectional* menunjukkan tren *validation loss* yang lebih stabil. Terakhir, implementasi model *Simple RNN* menggunakan *library* Keras dan dari *scratch* menghasilkan akurasi yang identik ketika diberikan *weights* dan *bias* yang sama.

Berdasarkan analisis *hyperparameter* LSTM yang telah dilakukan, terbukti bahwa direksionalitas adalah faktor paling berpengaruh, dengan *Bidirectional LSTM* secara signifikan meningkatkan *F1-score* (misalnya, dari 0.1844 ke 0.6931) karena pemahaman konteks dua arah yang superior. Selanjutnya, jumlah unit LSTM memberikan peningkatan kinerja yang lebih moderat, di mana penambahan dari 32 ke 64 unit sedikit menaikkan *F1-score*, namun tidak ada manfaat lebih lanjut dengan 128 unit. Selanjutnya, pengaruh jumlah *layer* pada LSTM yang menunjukkan inkonsistensi dalam hasil *F1-score macro* yakni turun saat *layer* 1 menuju *layer* 2 (dari 0.1844 ke 0.1827) dan naik kembali saat *layer* 2 ke *layer* 3 (dari 0.1827 ke 0.1844). Hal tersebut menunjukkan penambahan *layer* menyebabkan peningkatan kompleksitas yang jika disediakan *dataset* yang sederhana hanya akan memperbesar kemungkinan *overfit*.

3.2. Saran

1. Eksplorasi Hyperparameter Lebih Lanjut

Pengujian yang dilakukan telah memberikan gambaran awal mengenai pengaruh beberapa *hyperparameter*. Namun, masih banyak kombinasi dan nilai *hyperparameter* yang belum dieksplorasi. Disarankan untuk melakukan pengujian yang lebih dalam, termasuk menggunakan teknik optimasi *hyperparameter*, seperti *grid search* atau *random search*, untuk menemukan konfigurasi terbaik untuk setiap model (CNN, *Simple RNN*, dan LSTM).

2. Penggunaan Teknik Regularisasi

Dalam beberapa pengujian, terlihat indikasi *overfitting* (misalnya pada model RNN dengan 1 layer). Untuk mengatasi hal ini, disarankan untuk menambahkan teknik regularisasi seperti *L1/L2 regularization* pada *layer dense* atau *batch normalization* pada *layer* konvolusi dan RNN.

3. Evaluasi Metrik yang Lebih Komprehensif

Pengujian saat ini fokus pada *Macro F1 Score*. Disarankan untuk menambahkan metrik evaluasi lain seperti *precision*, *recall*, *accuracy* per kelas, dan *confusion matrix* untuk mendapatkan pemahaman yang lebih mendalam mengenai performa model.

4. Penggunaan Arsitektur Model yang Lebih Kompleks

Untuk penelitian yang lebih kompleks, disarankan untuk menggunakan arsitektur model yang lebih canggih, seperti model *Transformer* untuk data sekuensial atau arsitektur ResNet/Inception untuk data gambar.

5. Pengujian dengan Dataset yang Berbeda

Hasil pengujian sangat bergantung pada dataset yang digunakan. Disarankan untuk menguji model pada dataset yang berbeda untuk melihat generalisasi model.

Pembagian Tugas

Tabel 4.1. Pembagian Tugas

NIM	Tugas
13522122	Implementasi dan analisis RNN
13522144	Implementasi dan analisis CNN
13522150	Implementasi dan analisis LSTM

Referensi

3Blue1Brown. (2025, 18 Februari). Neural networks [Playlist]. YouTube.
https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi.