

Tugas Besar 2 IF3270 Machine Learning
Semester II tahun 2024/2025
Convolutional Neural Network dan Recurrent Neural Network



Oleh:

Samy Muhammad Haikal (13522151)

Muhammad Roihan (13522152)

Muhammad Dzaki Arta (13522149)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

DAFTAR ISI

DAFTAR ISI	2
DESKRIPSI PERSOALAN	3
1.1 Convolutional Neural Network (CNN)	3
Eksperimen Hyperparameter:	3
Forward Propagation from Scratch:	3
2. Simple Recurrent Neural Network (Simple RNN)	3
Eksperimen Hyperparameter:	4
Forward Propagation from Scratch:	4
3. Long Short-Term Memory Network (LSTM)	4
Eksperimen Hyperparameter:	5
Forward Propagation from Scratch:	5
PEMBAHASAN	6
1.1 CNN	6
1.2 Simple RNN	6
1.2.1 Unidirectional	6
1.2.2 Bidirectional	7
Ilustrasi Singkat:	7
1.3 LSTM	8
1.4 Kelas dan Objek	11
Tabel 1.3.1	11
HASIL PENGUJIAN	12
2.1 CNN	12
Tabel 2.1 Perbandingan CNN	12
2.1.1 Pengaruh jumlah layer konvolusi	12
2.1.2 Pengaruh banyak filter per layer konvolusi	12
2.1.3 Pengaruh ukuran filter per layer konvolusi	12
2.1.4 Pengaruh jenis pooling layer	13
2.2 Simple RNN	13
Tabel 2.2 Perbandingan Simple RNN	13
2.2.1 Pengaruh jumlah layer RNN	13
2.2.2 Pengaruh banyak cell RNN per layer	14
2.2.3 Pengaruh jenis layer RNN berdasarkan arah	15
2.3 LSTM	16
Tabel 2.3 Perbandingan LSTM	16
2.3.1 Pengaruh jumlah layer RNN	16
2.3.2 Pengaruh banyak cell LSTM per layer	18
2.3.3 Pengaruh jenis layer LSTM berdasarkan arah	20

2.3.4 Perbandingan Forward Propagation Manual Dan Keras	22
KESIMPULAN DAN SARAN	24
PEMBAGIAN TUGAS	25
REFERENSI	26
LAMPIRAN	27

DESKRIPSI PERSOALAN

Implementasikan modul forward propagation dari tiga jenis arsitektur jaringan saraf: **CNN (Convolutional Neural Network)**, **Simple RNN (Recurrent Neural Network)**, dan **LSTM (Long Short-Term Memory Network)**, dengan ketentuan sebagai berikut:

1.1 Convolutional Neural Network (CNN)

Model CNN digunakan untuk menyelesaikan tugas **klasifikasi citra** pada dataset **CIFAR-10**, dan harus memenuhi syarat berikut:

- Model harus terdiri dari kombinasi layer berikut:
 - Conv2D
 - Pooling (MaxPooling atau AveragePooling)
 - Flatten atau GlobalPooling
 - Dense
- Loss function: Sparse Categorical Crossentropy
- Optimizer: Adam
- Dataset CIFAR-10 yang tersedia hanya memiliki dua split (train dan test), sehingga perlu dilakukan **split ulang** untuk menghasilkan:
 - 40.000 data untuk training
 - 10.000 data untuk validasi
 - 10.000 data untuk testing

Eksperimen Hyperparameter:

- **Jumlah layer konvolusi** (3 variasi)
- **Jumlah filter per layer konvolusi** (3 variasi)
- **Ukuran filter per layer konvolusi** (3 variasi)
- **Jenis pooling** (2 variasi: max dan average pooling)

Untuk masing-masing variasi:

- Bandingkan akurasi akhir, macro f1-score, serta grafik training loss dan validation loss tiap epoch
- Berikan kesimpulan bagaimana masing-masing parameter mempengaruhi performa model

Forward Propagation from Scratch:

- Implementasi modul forward propagation secara modular, satu method per layer
- Harus mampu membaca dan menggunakan bobot hasil pelatihan dari Keras
- Hasil forward propagation harus dibandingkan dengan output Keras menggunakan data test
- Metrik evaluasi: **macro f1-score**

1.2. Simple Recurrent Neural Network (Simple RNN)

Model RNN digunakan untuk **klasifikasi sentimen** pada dataset **NusaX-Sentiment** (Bahasa Indonesia). Langkah preprocessing dan pelatihan model meliputi:

- **Tokenisasi** menggunakan TextVectorization untuk mengubah teks menjadi deretan token integer
- **Embedding** menggunakan Embedding Layer dari Keras
- Model harus memiliki:
 - Embedding layer
 - RNN layer (bidirectional atau unidirectional)
 - Dropout layer
 - Dense layer
- Loss function: Sparse Categorical Crossentropy
- Optimizer: Adam

Eksperimen Hyperparameter:

- **Jumlah layer RNN** (3 variasi)
- **Jumlah unit RNN per layer** (3 variasi)
- **Arah RNN** (2 variasi: unidirectional dan bidirectional)

Untuk masing-masing variasi:

- Bandingkan hasil akhir, macro f1-score, serta grafik training loss dan validation loss tiap epoch
- Berikan kesimpulan bagaimana masing-masing parameter mempengaruhi performa model

Forward Propagation from Scratch:

- Implementasi forward propagation modular, satu method per layer
- Harus mampu membaca bobot hasil pelatihan dari Keras
- Bandingkan hasil output dengan implementasi Keras pada data test
- Metrik evaluasi: **macro f1-score**

1.3. Long Short-Term Memory Network (LSTM)

Model LSTM juga digunakan untuk **klasifikasi sentimen** pada dataset **NusaX-Sentiment**, dengan alur preprocessing dan pelatihan yang mirip dengan RNN:

- Tokenisasi menggunakan TextVectorization
- Embedding menggunakan Embedding Layer dari Keras
- Model harus terdiri dari:
 - Embedding layer
 - LSTM layer (bidirectional dan/atau unidirectional)
 - Dropout layer
 - Dense layer
- Loss function: Sparse Categorical Crossentropy

- Optimizer: Adam

Eksperimen Hyperparameter:

- **Jumlah layer LSTM** (3 variasi)
- **Jumlah unit LSTM per layer** (3 variasi)
- **Jenis arah LSTM** (2 variasi: unidirectional dan bidirectional)

Untuk masing-masing variasi:

- Bandingkan hasil akhir, macro f1-score, serta grafik training loss dan validation loss tiap epoch
- Berikan kesimpulan bagaimana masing-masing parameter mempengaruhi performa model

Forward Propagation from Scratch:

- Implementasi modular, method terpisah untuk tiap layer
- Harus dapat memanfaatkan bobot hasil pelatihan dari Keras
- Uji hasil forward propagation dengan implementasi Keras menggunakan data test
- Metrik evaluasi: **macro f1-score**

PEMBAHASAN

1.1 CNN

Pada bagian ini, kami mengimplementasikan Convolutional Neural Network (CNN) untuk tugas klasifikasi gambar pada dataset CIFAR-10. Implementasi dilakukan dalam dua tahap utama: pertama menggunakan library Keras untuk pelatihan dan eksperimen hyperparameter, dan kedua mengimplementasikan fungsi *forward propagation* secara manual (*from scratch*) menggunakan NumPy dan memvalidasinya dengan bobot dari model Keras.

1.1.1 Arsitektur Dasar Model Keras (Baseline Model)

Sebagai titik awal dan acuan untuk eksperimen hyperparameter, kami merancang sebuah model CNN dasar dengan Keras. Arsitektur model dasar ini adalah sebagai berikut:

- Input Layer: Menerima gambar CIFAR-10 dengan shape (32, 32, 3).
- Blok Konvolusi 1:
 - `Conv2D`: 32 filter, kernel size (3,3), aktivasi ReLU, padding 'same'
 - `MaxPooling2D`: Pool size (2,2).
- Blok Konvolusi 2:
 - `Conv2D`: 64 filter, kernel size (3,3).
 - `MaxPooling2D`: Pool size (2,2).
- Flatten Layer: Mengubah output dari layer konvolusi menjadi vektor 1D.
- Dense Layer (Hidden): 128 unit, aktivasi ReLU..
- Output Layer (Dense): 10 unit (sesuai jumlah kelas CIFAR-10), aktivasi Softmax.
- Model ini di-compile menggunakan optimizer Adam dan loss function Sparse Categorical Crossentropy.

1.1.2 Implementasi Layer From Scratch

Untuk implementasi *forward propagation from scratch*, kami membuat kelas-kelas modular untuk setiap jenis layer yang digunakan dalam arsitektur CNN kami. Kelas-kelas ini disimpan dalam direktori `src/CNN/`.

- `Conv2DLayerFS`: Mengimplementasikan operasi konvolusi 2D, termasuk padding dan stride. Menerima input feature map, filter (kernel), dan bias.
- `ReLULayerFS`: Mengimplementasikan fungsi aktivasi ReLU.

- ``MaxPooling2DLayerFS`` / ``AveragePooling2DLayerFS``: Mengimplementasikan operasi max pooling atau average pooling.
- ``FlattenLayerFS``: Mengubah input tensor multidimensi menjadi vektor satu dimensi.
- ``DenseLayerFS``: Mengimplementasikan layer fully connected, termasuk aktivasi opsional (ReLU/Softmax).

1.1.3 Penjelasan Forward Propagation CNN From Scratch

Proses *forward propagation* pada model CNN *from scratch* (``CNNModelFS``) dilakukan dengan melewati data input secara sekuensial melalui setiap layer yang telah ditambahkan ke model. Setiap objek layer dalam ``CNNModelFS`` memiliki method ``forward()`` sendiri yang melakukan perhitungan spesifik untuk layer tersebut. Alur umumnya mengikuti arsitektur Keras yang telah ditentukan, dimulai dari layer konvolusi, aktivasi, pooling, hingga flatten dan dense layer untuk klasifikasi.

1.1.4 Cara Memuat Bobot Keras ke Model From Scratch

Untuk validasi, model ``CNNModelFS`` *from scratch* menggunakan bobot yang sama dengan model Keras yang telah dilatih. Method ``load_keras_weights(keras_model)`` di dalam kelas ``CNNModelFS`` bertanggung jawab untuk ini. Method tersebut mengiterasi layer-layer pada model Keras dan model *from scratch*, mencocokkan layer yang sesuai tipenya (misalnya, ``Conv2DLayerFS`` dengan ``tf.keras.layers.Conv2D``), kemudian mengambil bobot (``weights`` dan ``biases``) dari layer Keras menggunakan ``keras_layer.get_weights()`` dan memuatnya ke layer *from scratch* menggunakan method ``fs_layer.load_weights(weights, biases)``.

1.2 Simple RNN

1.2.1 Unidirectional

Langkah awal dalam fungsi ini adalah Embedding, yaitu mengubah input (urutan token) menjadi vektor embedding berdimensi tetap. Setelah tahap embedding, data masukan diproses oleh satu atau lebih lapisan Simple RNN. Proses utama dilakukan oleh fungsi `rnn_layer_forward`.

Di dalam fungsi ini, hidden state (h_t) diinisialisasi (biasanya dengan nol). Untuk setiap timestep dalam sekuens masukan, fungsi `rnn_cell_forward` dipanggil.

Fungsi `rnn_cell_forward` adalah inti dari operasi Simple RNN pada satu timestep. Fungsi ini menerima masukan pada timestep saat ini (x_t) dan hidden state dari timestep sebelumnya (h_{prev}), beserta bobot-bobot (W_x , W_h , b) yang terkait. Perhitungan pada tiap timestep dilakukan sebagai berikut:

- **Input Processing:**

Input pada waktu ke- t (x_t) dikalikan dengan bobot input-hidden (W_x), dan hidden state sebelumnya (h_{prev}) dikalikan dengan bobot hidden-hidden (W_h).

- **Penjumlahan dan Aktivasi:**

Hasil perkalian tadi dijumlahkan bersama bias (b), kemudian dilewatkan ke fungsi aktivasi tanh:

Code

- $h_t = \tanh(x_t @ W_x + h_{prev} @ W_h + b)$

- **Pengumpulan Output:**

Jika `return_sequences=True`, maka seluruh urutan hidden state (h_t dari setiap timestep) disimpan. Jika `return_sequences=False`, hanya hidden state terakhir yang diambil.

Jika terdapat beberapa lapisan Simple RNN yang ditumpuk (stacked RNNs), output dari satu lapisan Simple RNN akan menjadi input untuk lapisan berikutnya. Pada lapisan Simple RNN terakhir, hanya hidden state terakhir yang digunakan untuk tugas selanjutnya.

Tahap terakhir dalam forward propagation adalah Dense Layer. Output dari lapisan Simple RNN terakhir dimasukkan ke dalam lapisan dense, yang melakukan transformasi linear (mengalikan dengan bobot, menambahkan bias), lalu hasilnya dilewatkan ke fungsi aktivasi softmax untuk menghasilkan prediksi akhir model.

1.2.2 Bidirectional

Pada arsitektur Bidirectional RNN, setelah tahap embedding, input diproses oleh satu atau lebih lapisan Bidirectional Simple RNN.

Setiap lapisan Bidirectional Simple RNN bekerja sebagai berikut:

- **Arah Maju (Forward):**

Lapisan Simple RNN memproses sekuens masukan dari awal ke akhir, menghasilkan urutan hidden state (`forward_outputs`) dan hidden state terakhir (`forward_h`).

- **Arah Mundur (Backward):**

Urutan input dibalik sepanjang dimensi waktu, lalu diproses oleh lapisan Simple RNN kedua (arah mundur) dengan bobot terpisah, menghasilkan `backward_outputs` dan `backward_h`. Output `backward_outputs` kemudian dibalik kembali agar sejajar dengan urutan waktu asli.

- **Penggabungan Hasil (Concatenation):**

`forward_outputs` dan `backward_outputs` digabungkan secara konkatenasi pada dimensi fitur, sehingga setiap timestep mengandung informasi dari konteks masa lalu (`forward`) dan masa depan (`backward`). Hidden state terakhir dari kedua arah (`forward_h` dan `backward_h`) juga dikombinasikan.

Output gabungan dari lapisan Bidirectional RNN terakhir kemudian dimasukkan ke Dense Layer (proses seperti di atas, diakhiri dengan softmax).

Ilustrasi Singkat:

- RNN mengingat informasi urutan sebelumnya lewat hidden state.
- Proses ini diulang untuk setiap kata/token pada urutan input.
- Pada bidirectional RNN, model juga mengingat konteks dari masa depan (bukan hanya masa lalu).
- Model dapat belajar pola urutan dalam data sekuensial, seperti kalimat atau urutan kata.

Implementasi Simple RNN from scratch meniru persis operasi layer SimpleRNN dari Keras, termasuk versi bidirectional jika digunakan. Semua bobot (W_x , W_h , b) diambil dari model Keras hasil training, sehingga hasil seharusnya identik selama perhitungan dilakukan dengan benar. Forward propagation dilakukan eksplisit per time step, sehingga mudah untuk memahami dan mempelajari konsep dasar RNN.

1.3 LSTM

1.3.1 Unidirectional

Langkah awal dalam fungsi ini adalah Embedding. Tahap ini mengubah input menjadi vektor embedding yang padat. Setelah data masukan diubah menjadi embedding, data tersebut kemudian diproses oleh satu atau lebih lapisan LSTM. Fungsi `lstm_layer_forward` mengelola pemrosesan sekuensial data oleh lapisan LSTM. Di dalam fungsi ini, hidden state (h_t) dan cell state (c_t) diinisialisasi (dengan nilai nol). Kemudian, untuk setiap timestep dalam sekuens masukan, fungsi `lstm_cell_forward` dipanggil.

Fungsi `lstm_cell_forward` adalah inti dari operasi LSTM pada satu timestep. Fungsi ini menerima masukan pada timestep saat ini (x_t), hidden state dari timestep sebelumnya (h_{prev}), dan cell state dari timestep sebelumnya (c_{prev}), beserta bobot-bobot (weights) yang terkait dengan sel tersebut. Di dalam sel LSTM, terdapat beberapa "gerbang" (gates) yang mengatur aliran informasi:

A. Input Gate (i_t): Menentukan seberapa banyak informasi baru dari masukan saat ini (x_t) dan hidden state sebelumnya (h_{prev}) yang akan disimpan dalam cell state. Gerbang ini menggunakan fungsi aktivasi sigmoid.

B. Forget Gate (f_t): Menentukan seberapa banyak informasi dari cell state sebelumnya (c_{prev}) yang akan dilupakan atau dibuang. Gerbang ini juga menggunakan fungsi aktivasi sigmoid.

C. Candidate Values (\tilde{c}_t): Menghasilkan nilai-nilai kandidat baru yang berpotensi ditambahkan ke cell state. Nilai ini dihitung berdasarkan masukan saat ini (x_t) dan hidden state sebelumnya (h_{prev}), dan biasanya menggunakan fungsi aktivasi tanh.

D. Output Gate (o_t): Menentukan seberapa banyak informasi dari cell state saat ini yang akan dikeluarkan sebagai hidden state (h_t). Gerbang ini menggunakan fungsi aktivasi sigmoid.

Setelah nilai-nilai dari ketiga gerbang dan nilai kandidat dihitung, cell state diperbarui ($c_t = f_t * c_{prev} + i_t * \tilde{c}_t$). Ini adalah kombinasi dari informasi yang dipertahankan dari cell state sebelumnya (dikontrol oleh forget gate) dan informasi baru yang ditambahkan (dikontrol oleh input gate dan nilai kandidat). Selanjutnya, hidden state juga diperbarui ($h_t = o_t * \text{self.tanh}(c_t)$). Hidden state ini merupakan output dari sel LSTM pada timestep tersebut dan

akan diteruskan ke timestep berikutnya atau lapisan selanjutnya. Fungsi `lstm_layer_forward` mengumpulkan hidden state dari setiap timestep untuk membentuk output dari seluruh lapisan LSTM. Jika terdapat beberapa lapisan LSTM yang ditumpuk (stacked LSTMs), output dari satu lapisan LSTM (berupa semua hidden state dari setiap timestep) akan menjadi input untuk lapisan LSTM berikutnya. Namun, untuk lapisan LSTM terakhir, biasanya hanya hidden state terakhir (`final_h`) atau representasi agregat lainnya yang digunakan untuk tugas selanjutnya.

Tahap terakhir dalam `forward_propagation` adalah Dense Layer. Output dari lapisan LSTM terakhir kemudian dimasukkan ke dalam lapisan dense. Fungsi `dense_forward` melakukan transformasi linear pada inputnya dengan mengalikannya dengan matriks bobot dan menambahkan bias. Hasil dari transformasi linear ini kemudian dilewatkan melalui fungsi aktivasi softmax. Output dari fungsi softmax ini adalah prediksi akhir dari model.

1.3.2 Bidirectional

Langkah pertama, sama seperti sebelumnya, adalah Embedding. Setelah tahap embedding, data masukan yang sudah berupa vektor embedding (embedded) diproses oleh satu atau lebih lapisan Bidirectional LSTM. Perulangan `for i, layer_weights in enumerate(self.weights['lstm_layers'])`: mengiterasi setiap lapisan BiLSTM yang ada dalam model. Di setiap iterasi, fungsi `bidirectional_lstm_layer_forward` dipanggil.

Fungsi `bidirectional_lstm_layer_forward` adalah inti dari pemrosesan BiLSTM. Cara kerjanya adalah sebagai berikut:

A. Pemrosesan Arah Maju (Forward Direction):

Lapisan LSTM standar dipanggil untuk memproses sekuens masukan (inputs) dari awal hingga akhir (arah maju). Proses ini menggunakan bobot khusus untuk arah maju. Hasil dari pemrosesan ini adalah `forward_outputs` (hidden state untuk setiap timestep dari arah maju), `forward_h` (hidden state terakhir dari arah maju), dan `forward_c` (cell state terakhir dari arah maju).

B. Pemrosesan Arah Mundur (Backward Direction):

Sekuens masukan dibalik urutannya sepanjang dimensi waktu. Ini berarti LSTM akan memproses sekuens dari akhir ke awal. Lapisan LSTM standar kembali dipanggil,

kali ini dengan sekuens yang sudah dibalik dan menggunakan bobot khusus untuk arah mundur. Hasilnya adalah `backward_outputs`, `backward_h`, dan `backward_c` dari arah mundur. Penting untuk dicatat bahwa `backward_outputs` kemudian dibalik kembali urutannya agar sesuai dengan urutan waktu asli dari sekuens masukan. Dengan demikian, `backward_outputs` pada timestep t berisi informasi dari konteks "masa depan" relatif terhadap timestep t tersebut.

C. Penggabungan Hasil (Concatenation):

`forward_outputs` dan `backward_outputs` digabungkan (dikontatenasi). Hasilnya, `combined_outputs`, untuk setiap timestep, kini berisi informasi dari kedua arah pemrosesan (konteks masa lalu dari LSTM maju dan konteks masa depan dari LSTM mundur). Dimensi fitur dari `combined_outputs` akan menjadi dua kali lipat dimensi hidden unit LSTM tunggal. Demikian pula, hidden state terakhir dari arah maju (`forward_h`) dan hidden state terakhir dari arah mundur (`backward_h`) digabungkan untuk menghasilkan `combined_final_h`. Hidden state gabungan ini merepresentasikan seluruh sekuens dengan mempertimbangkan informasi dari kedua arah.

Tahap terakhir adalah Dense Layer. `lstm_output` (yang merupakan `combined_final_h` dari lapisan BiLSTM terakhir) dimasukkan ke dalam lapisan dense. Lapisan ini melakukan transformasi linear dan kemudian menerapkan fungsi aktivasi softmax untuk menghasilkan distribusi probabilitas atas kelas-kelas target. Output dari fungsi softmax ini adalah prediksi akhir dari model.

1.4 Kelas dan Objek

Berikut adalah kelas-kelas yang diimplementasikan.

Tabel 1.4.1 Kelas Conv2DLayerFS

CLASS		
	Conv2DLayerFS	Kelas untuk layer konvolusi 2D, melakukan operasi konvolusi dengan filter/kernel pada input feature map.
ATTRIBUTE		

	num_filters (int): Jumlah filter/kernel yang digunakan. kernel_size (tuple): Ukuran kernel konvolusi (height, width). stride (tuple): Langkah pergeseran kernel. padding (str): Jenis padding ('same' atau 'valid'). weights (np.ndarray): Bobot kernel berukuran (kernel_h, kernel_w, input_channels, num_filters). biases (np.ndarray): Bias untuk setiap filter.	
METHOD		
	forward(x)	Melakukan operasi konvolusi pada input data dan menghasilkan feature map.
	load_weights(weights, biases)	Memuat bobot dan bias dari model Keras.

Tabel 1.4.2 Kelas ReLULayerFS

CLASS		
	ReLULayerFS	Kelas untuk layer aktivasi ReLU (Rectified Linear Unit).
ATTRIBUTE		
	(tidak ada atribut khusus)	
METHOD		
	forward(x)	Menerapkan fungsi aktivasi ReLU: $f(x) = \max(0, x)$.

Tabel 1.4.3 Kelas MaxPooling2DLayerFS

CLASS		
	MaxPooling2DLayerFS	Kelas untuk layer max pooling 2D, mengambil nilai maksimum dari setiap region pooling.
ATTRIBUTE		

	pool_size (tuple): Ukuran pooling window (height, width). stride (tuple): Langkah pergeseran pooling window.	
METHOD		
	forward(x)	Melakukan operasi max pooling pada input data.

Tabel 1.4.4 Kelas AveragePooling2DLayerFS

CLASS		
	AveragePooling2DLayerFS	Kelas untuk layer average pooling 2D, menghitung rata-rata nilai dari setiap region pooling.
ATTRIBUTE		
	pool_size (tuple): Ukuran pooling window (height, width). stride (tuple): Langkah pergeseran pooling window.	
METHOD		
	forward(x)	Melakukan operasi average pooling pada input data.

Tabel 1.4.5 Kelas FlattenLayerFS

Tugas 1: Neural Networks - Flatten Layer, GPT-2		
CLASS		
	FlattenLayerFS	Kelas untuk layer flatten, mengubah tensor multidimensi menjadi vektor 1D.
ATTRIBUTE		
	(tidak ada atribut khusus)	
METHOD		

	forward(x)	Meratakan (flatten) input tensor menjadi shape (batch_size, -1).

Tabel 1.4.6 Kelas DenseLayerFS

CLASS		
	DenseLayerFS	Kelas untuk layer fully connected (dense), melakukan transformasi linear dengan aktivasi opsional.
ATTRIBUTE		
	num_units (int): Jumlah unit/neuron dalam layer. activation_name (str): Nama fungsi aktivasi ('relu', 'softmax', atau None). weights (np.ndarray): Matriks bobot berukuran (input_dim, num_units). biases (np.ndarray): Vektor bias berukuran (num_units,).	
METHOD		
	forward(x)	forward(input_data): Melakukan transformasi linear input (weights + biases), diikuti aktivasi.
	load_weights(weights, biases)	Memuat bobot dan bias dari model Keras.
	_apply_activation(x)	Menerapkan fungsi aktivasi sesuai activation_name.

Tabel 1.4.7 Kelas CNNModelFS

CLASS		
	CNNModelFS	Kelas utama untuk model CNN from scratch yang menggabungkan semua layer individual.
ATTRIBUTE		
	layers (list): Daftar semua layer yang membentuk model CNN secara berurutan.	

METHOD		
	add_layer(layer)	Menambahkan layer baru ke dalam model.
	forward(input_data, verbose=False)	Melakukan forward propagation lengkap melalui semua layer.
	predict_proba_batch(input_data, batch_size_fs=32)	Melakukan prediksi probabilitas dengan batch processing.
	predict_classes_batch(input_data, batch_size_fs=32)	Melakukan prediksi kelas dengan batch processing.
	load_keras_weights(keras_model)	Memuat bobot dari model Keras yang telah dilatih ke semua layer yang sesuai.

Tabel 1.4.8 DataLoaderCIFAR10

CLASS		
	LoaderCIFAR10	Kelas untuk memuat dan memproses dataset CIFAR-10 dengan pembagian train/validation/test.
ATTRIBUTE		
	num_classes (int): Jumlah kelas dalam CIFAR-10 (10 kelas). random_state_split (int): Seed untuk reproducible data splitting.	
METHOD		
	get_train_data():	Mengembalikan data training (40k sampel).
	get_validation_data()	Mengembalikan data validasi (10k sampel).

	get_test_data()	Mengembalikan data testing (10k sampel).
	get_input_shape()	Mengembalikan shape input CIFAR-10 (32, 32, 3).
	_split_validation():	Membagi data training menjadi train dan validation dengan rasio 4:1.

Tabel 1.4.9 Kelas MyEmbedding

CLASS		
	MyEmbedding	Kelas untuk layer embedding, mengubah indeks kata menjadi vektor representasi berdimensi tetap.
ATTRIBUTE		
	weights (np.ndarray): Matriks bobot embedding berukuran (vocab_size, embed_dim).	
METHOD		
	forward(x)	Mengubah indeks kata menjadi representasi vektor (lookup embedding).

Tabel 1.4.10 Kelas MySimpleEmbedding

CLASS		
	MySimpleRNN	Kelas untuk layer RNN satu arah (forward only).
ATTRIBUTE		
	Wx (np.ndarray): Bobot untuk input saat ini. Wh (np.ndarray): Bobot untuk hidden state sebelumnya. b (np.ndarray): Vektor bias. return_sequences (bool): Jika True, mengembalikan semua output per timestep, jika False hanya timestep terakhir.	
METHOD		

	forward(x)	Melakukan proses forward pass untuk urutan input, menghasilkan hidden state per timestep atau hanya timestep terakhir.

Tabel 1.4.11 Kelas MyBidirectionalSimpleRNN

CLASS		
	MyBidirectionalSimpleRNN	Kelas untuk layer RNN dua arah (bidirectional).
ATTRIBUTE		
	forward_rnn (MySimpleRNN): RNN arah maju. backward_rnn (MySimpleRNN): RNN arah mundur. return_sequences (bool): Sama seperti di MySimpleRNN.	
METHOD		
	forward(x)	Menjalankan RNN dua arah, kemudian menggabungkan hasilnya dari kedua arah pada dimensi fitur.

Tabel 1.4.12 Kelas MyDense

CLASS		
	MyDense	Kelas untuk layer fully-connected (dense).
ATTRIBUTE		
	W (np.ndarray): Matriks bobot. b (np.ndarray): Vektor bias.	
METHOD		
	forward(x)	Melakukan transformasi linear $x @ W + b$.

Tabel 1.4.13 Kelas MyDropout

CLASS		
	MyDropout	Kelas untuk menerapkan fungsi softmax.
ATTRIBUTE		
	(tidak ada atribut)	
METHOD		
	forward(x)	Menghitung softmax dari input x secara stabil (menghindari overflow).

Tabel 1.4.14 Kelas MyRNNModel

CLASS		
	MyRNNModel	Kelas utama yang menyusun keseluruhan model RNN dari layer-layer individual.
ATTRIBUTE		
	<p>embedding (MyEmbedding): Layer embedding.</p> <p>rnn_layers (list): Daftar layer RNN (simple atau bidirectional).</p> <p>dropout_layers (list): Daftar layer dropout.</p> <p>ordered_layers (list): Urutan layer sesuai model Keras asli.</p> <p>dense (MyDense): Layer fully-connected terakhir.</p> <p>softmax (MySoftmax): Layer softmax untuk output klasifikasi.</p>	
METHOD		
	forward(x)	Melakukan forward pass lengkap: embedding → RNN → dropout → dense → softmax.

CLASS		
	DataPreprocessor	Kelas untuk melakukan preprocessing data
ATTRIBUTE		
	max_vocab_size : ukuran vocab maksimum max_sequence_length : panjang maksimum sequence tokenizer : Tokenizer Text_vectorizer : objek TensorFlow TextVectorization layer label_encoder : menyimpan label encoder	
METHOD		
	load_nusax_data(self)	Load dataset nusax
	preprocess_with_keras_tokenizer(self, train_texts, val_texts, test_texts)	Melakukan preprocessing data menggunakan tokenizer
	preprocess_with_text_vectorization(self, train_texts, val_texts, test_texts):	Mengkonversi teks mentah menjadi representasi numerik
	encode_labels(self, train_labels, val_labels, test_labels):	Mengubah label kategorikal menjadi numerikal
	get_vocab_size(self)	Mengembalikan ukuran vocab maksimum
	get_num_classes(self)	Mengembalikan jumlah kelas

CLASS		
	LSTMForwardPropagation	Kelas untuk melakukan forward propagation manual pada model LSTM dengan mengekstrak dan menggunakan bobot dari model Keras yang telah

		dilatih.
ATTRIBUTE		
	weights : dictionary yang menyimpan semua bobot model (LSTM layers dan dense layer) embedding_weights : matriks bobot embedding layer vocab_size : ukuran vocabulary embedding_dim : dimensi embedding lstm_units : jumlah unit dalam layer LSTM num_classes : jumlah kelas output	
METHOD		
	load_keras_weights(self, model_path)	Memuat bobot dari model Keras yang tersimpan dan mengekstraknya untuk digunakan dalam implementasi manual
	sigmoid(self, x)	Implementasi fungsi aktivasi sigmoid dengan clipping untuk stabilitas numerik
	tanh(self, x)	Implementasi fungsi aktivasi tanh dengan clipping untuk stabilitas numerik
	softmax(self, x)	Implementasi fungsi aktivasi softmax untuk layer output
	embedding_forward(self, input_ids)	Melakukan forward pass pada embedding layer untuk mengkonversi token ID menjadi representasi vektor
	lstm_cell_forward(self, x_t, h_prev, c_prev, weights)	Implementasi manual satu sel LSTM dengan perhitungan gates (input, forget, cell, output)
	lstm_layer_forward(self, inputs, layer_weights)	Melakukan forward pass pada satu layer LSTM lengkap untuk seluruh sequence
	dense_forward(self, inputs)	Melakukan forward pass pada dense layer dengan transformasi linear dan softmax

	forward_propagation(self, input_ids)	Melakukan forward propagation lengkap dari input hingga output prediksi
	predict(self, X)	Melakukan prediksi pada data input dan mengembalikan kelas prediksi beserta probabilitasnya
	evaluate(self, X_test, y_test)	Mengevaluasi performa model pada data test dengan menghitung F1 score macro

CLASS		
	BidirectionalLSTMForward Propagation	Kelas turunan dari LSTMForwardPropagation untuk melakukan forward propagation manual pada model Bidirectional LSTM.
ATTRIBUTE		
	(sama seperti LSTMForwardPropagation)	
METHOD (Tambahan)		
	bidirectional_lstm_layer_forward(self, inputs, layer_weights)	Melakukan forward pass pada layer Bidirectional LSTM dengan menggabungkan output dari arah forward dan backward

HASIL PENGUJIAN

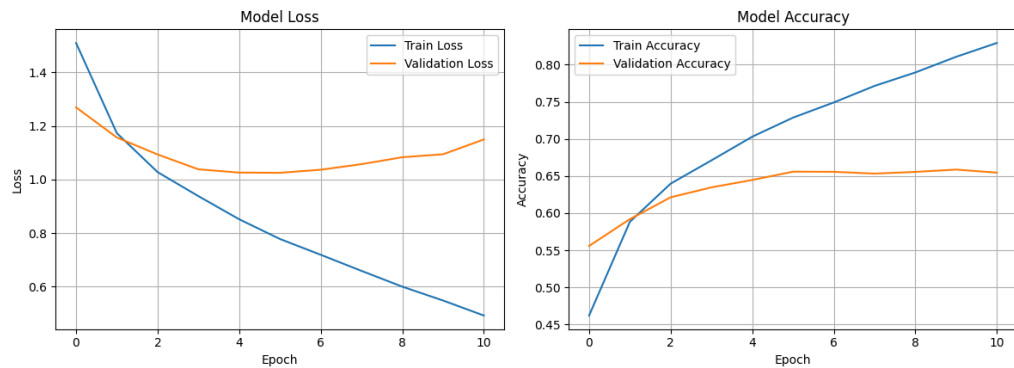
2.1 CNN

Tabel 2.1 Perbandingan CNN

2.1.1 Pengaruh jumlah layer konvolusi	
Layer Konvolusi	Loss graph

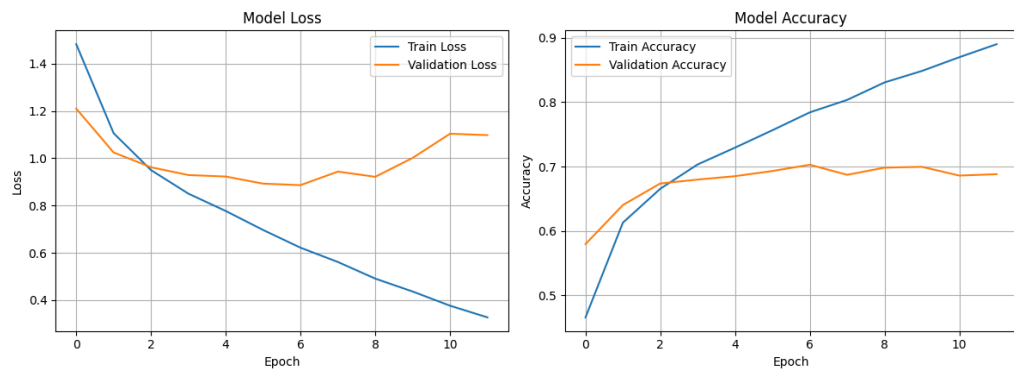
1

Pengaruh Jumlah Layer Konvolusi - 1Blok_1Conv



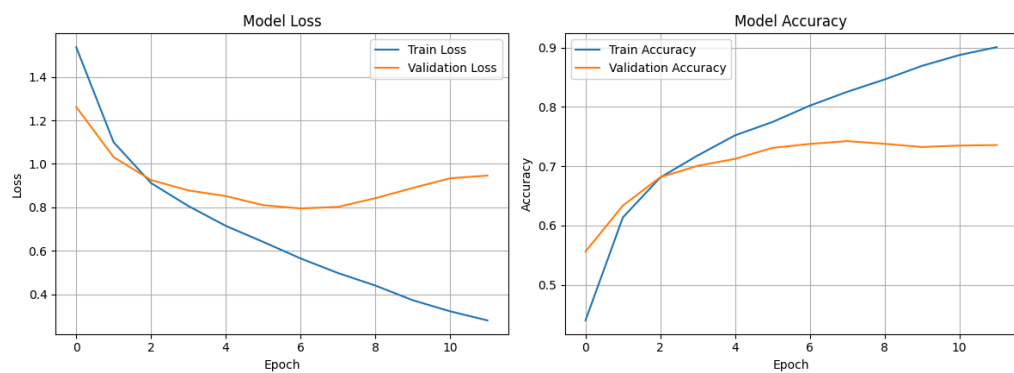
2

Pengaruh Jumlah Layer Konvolusi - 2Blok_1ConvPer



3

Pengaruh Jumlah Layer Konvolusi - 3Blok_1ConvPer



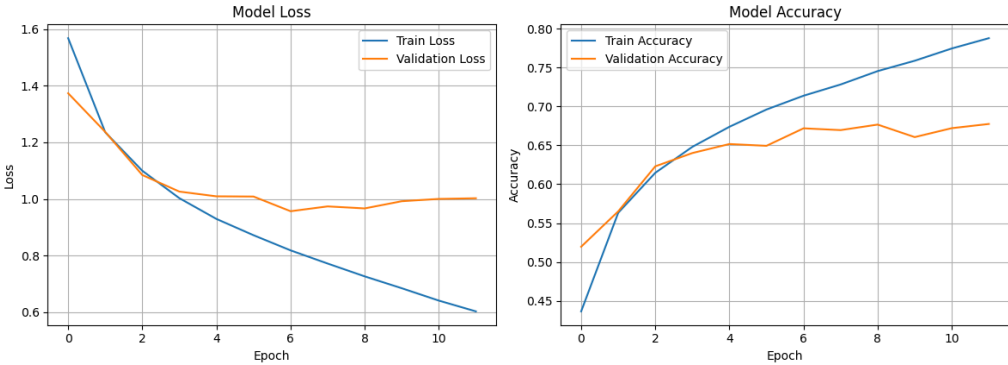
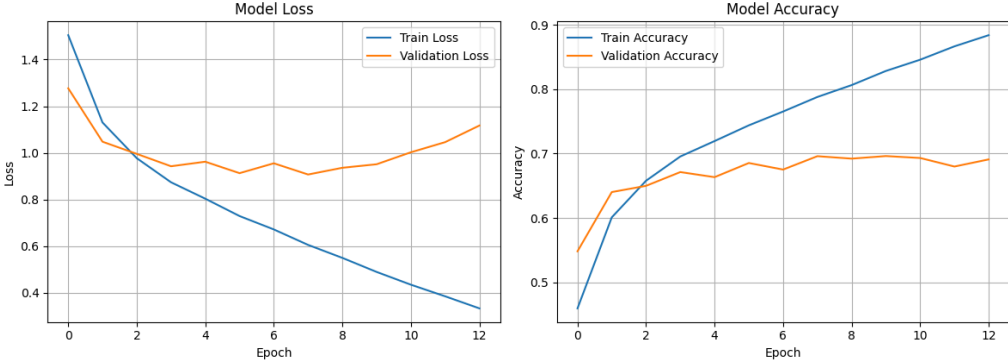
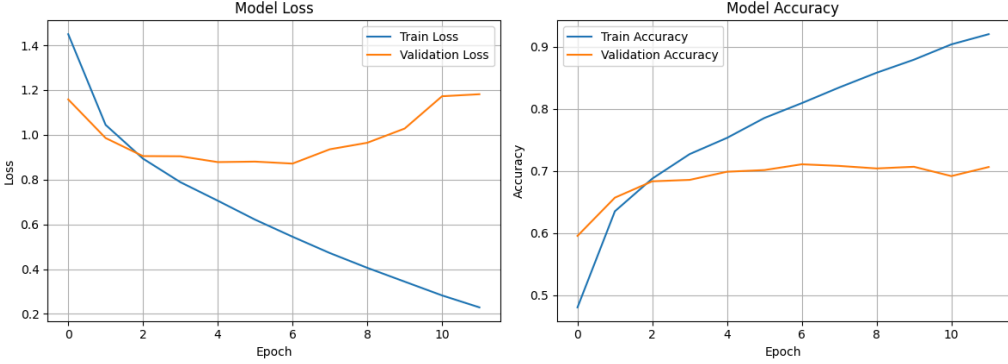
Perbandingan Akurasi

```

--- Hasil Akhir untuk Eksperimen: Pengaruh Jumlah Layer Konvolusi ---
Pengaruh Jumlah Layer Konvolusi - 1Blok_1Conv: Macro F1-Score = 0.6390, Accuracy = 0.6411
Pengaruh Jumlah Layer Konvolusi - 2Blok_1ConvPer: Macro F1-Score = 0.7014, Accuracy = 0.7059
Pengaruh Jumlah Layer Konvolusi - 3Blok_1ConvPer: Macro F1-Score = 0.7290, Accuracy = 0.7308

```

2.1.2 Pengaruh banyak filter per layer konvolusi

Banyak Filter	Loss graph
16, 32	<p data-bbox="711 382 1114 407">Pengaruh Banyak Filter - FilterKecil_16_32</p> <div data-bbox="412 436 1414 798">  <p>The graphs for 16 and 32 filters show a steady decrease in loss and a corresponding increase in accuracy over 10 epochs. The training loss starts at approximately 1.55 and drops to 0.6, while validation loss starts at 1.35 and drops to 1.0. Training accuracy starts at 0.45 and rises to 0.78, with validation accuracy starting at 0.52 and rising to 0.68.</p> </div>
32,64	<p data-bbox="699 848 1125 873">Pengaruh Banyak Filter - FilterSedang_32_64</p> <div data-bbox="412 903 1414 1264">  <p>The graphs for 32 and 64 filters show a decrease in loss and an increase in accuracy over 12 epochs. Training loss starts at 1.5 and drops to 0.35, while validation loss starts at 1.25 and drops to 1.1. Training accuracy starts at 0.45 and rises to 0.88, with validation accuracy starting at 0.55 and rising to 0.69.</p> </div>
64,128	<p data-bbox="699 1314 1125 1339">Pengaruh Banyak Filter - FilterBesar_64_128</p> <div data-bbox="412 1369 1414 1730">  <p>The graphs for 64 and 128 filters show a decrease in loss and an increase in accuracy over 10 epochs. Training loss starts at 1.45 and drops to 0.25, while validation loss starts at 1.15 and drops to 1.18. Training accuracy starts at 0.48 and rises to 0.92, with validation accuracy starting at 0.59 and rising to 0.71.</p> </div>
Perbandingan Akurasi	

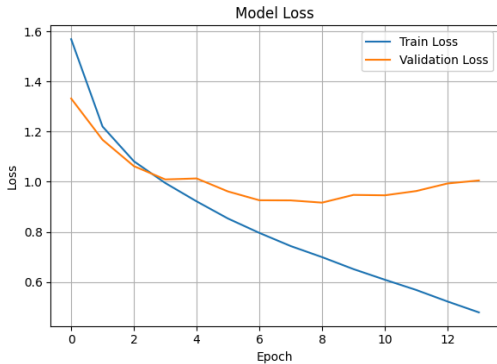
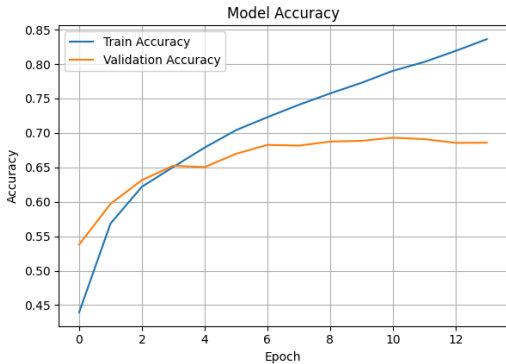
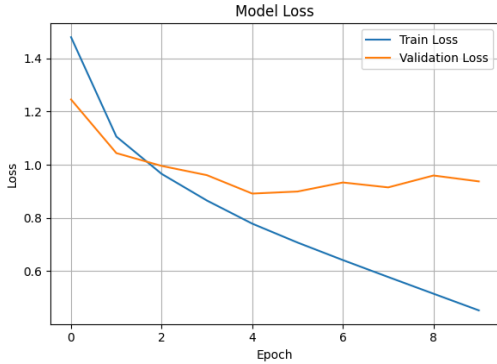
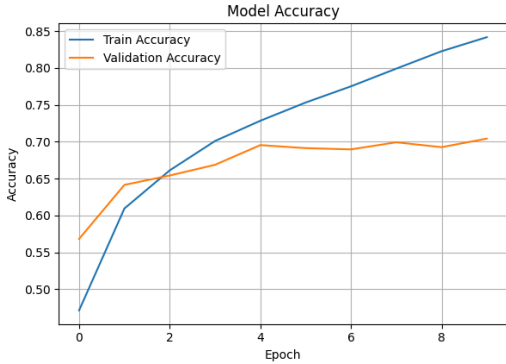
--- Hasil Akhir untuk Eksperimen: Pengaruh Banyak Filter ---

Pengaruh Banyak Filter - FilterKecil_16_32: Macro F1-Score = 0.6724, Accuracy = 0.6748

Pengaruh Banyak Filter - FilterSedang_32_64: Macro F1-Score = 0.7041, Accuracy = 0.7035

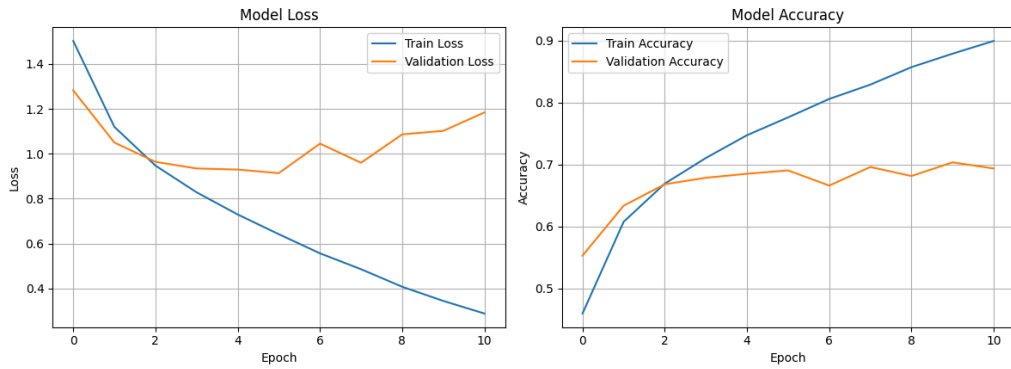
Pengaruh Banyak Filter - FilterBesar_64_128: Macro F1-Score = 0.7107, Accuracy = 0.7086

2.1.3 Pengaruh ukuran filter per layer konvolusi

Ukuran Filter	Loss graph
2x2	<p>Pengaruh Ukuran Filter - Kernel_2x2</p> <div>   </div>
3x3	<p>Pengaruh Ukuran Filter - Kernel_3x3</p> <div>   </div>

5x5

Pengaruh Ukuran Filter - Kernel_5x5



Perbandingan Akurasi

```
--- Hasil Akhir untuk Eksperimen: Pengaruh Ukuran Filter ---
Pengaruh Ukuran Filter - Kernel_2x2: Macro F1-Score = 0.6821, Accuracy = 0.6837
Pengaruh Ukuran Filter - Kernel_3x3: Macro F1-Score = 0.6915, Accuracy = 0.6917
Pengaruh Ukuran Filter - Kernel_5x5: Macro F1-Score = 0.6859, Accuracy = 0.6900
```

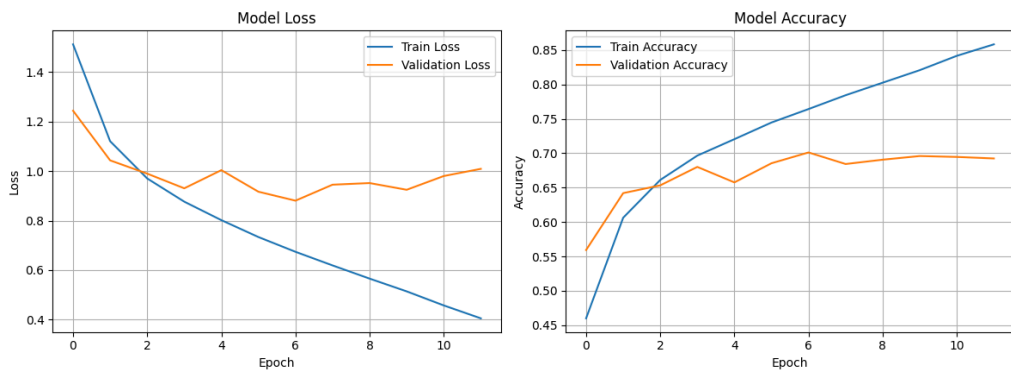
2.1.4 Pengaruh jenis pooling layer

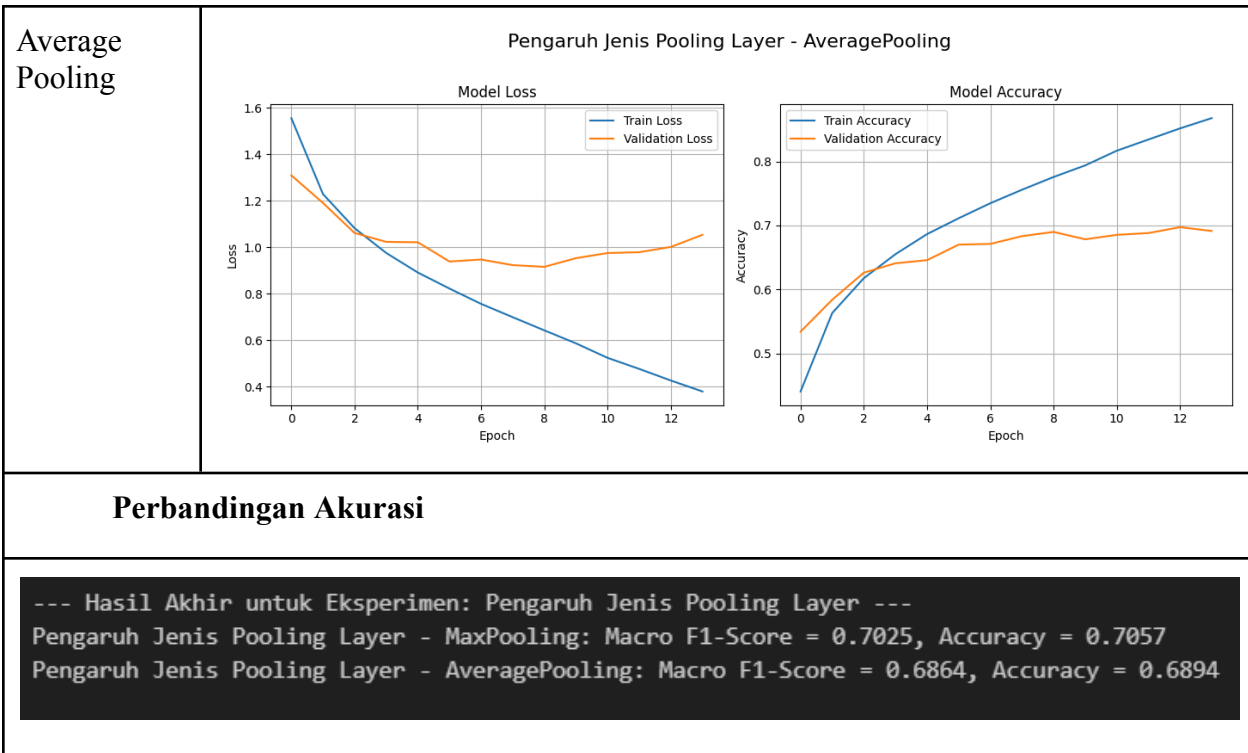
Pooling Layer

Macro F1 Score

Max Pooling

Pengaruh Jenis Pooling Layer - MaxPooling

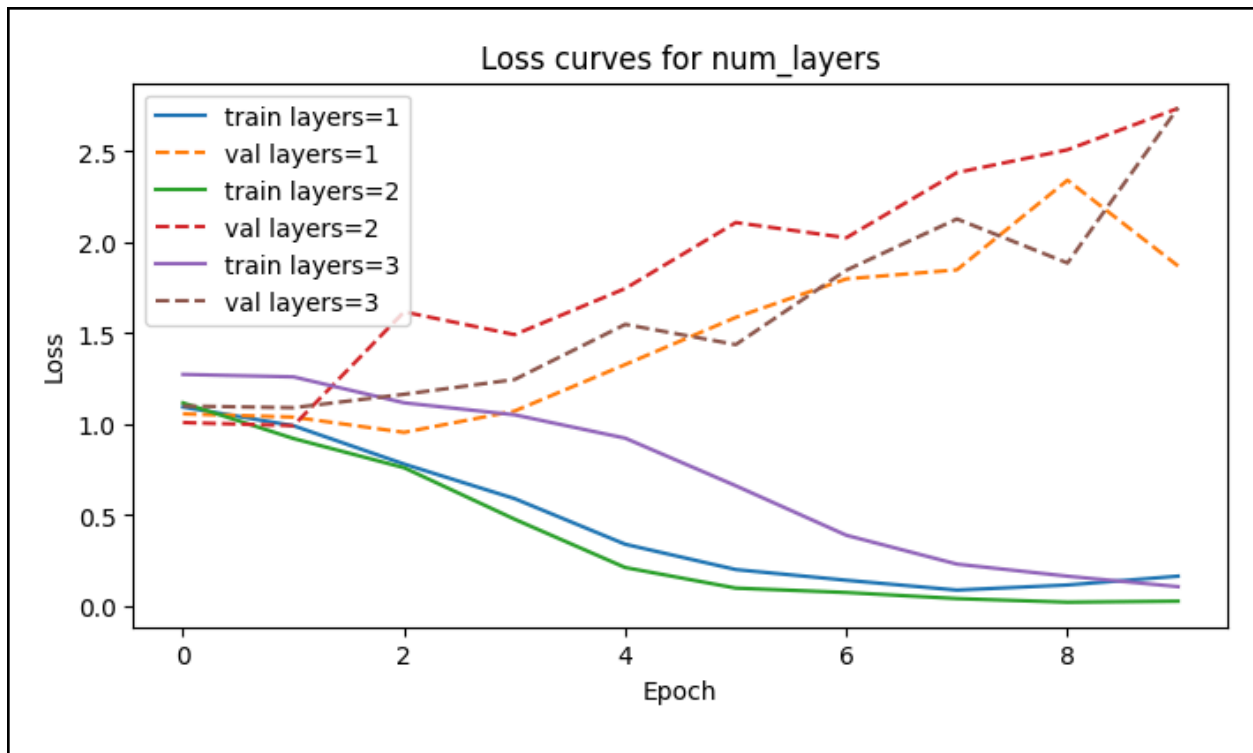




2.2 Simple RNN

Tabel 2.2 Perbandingan Simple RNN

2.2.1 Pengaruh jumlah layer RNN RNN unit: 64 Bidirectional: False	
Jumlah Layer	Macro F1 Score
1	0.4408
2	0.4570
3	0.3577
Perbandingan Loss	

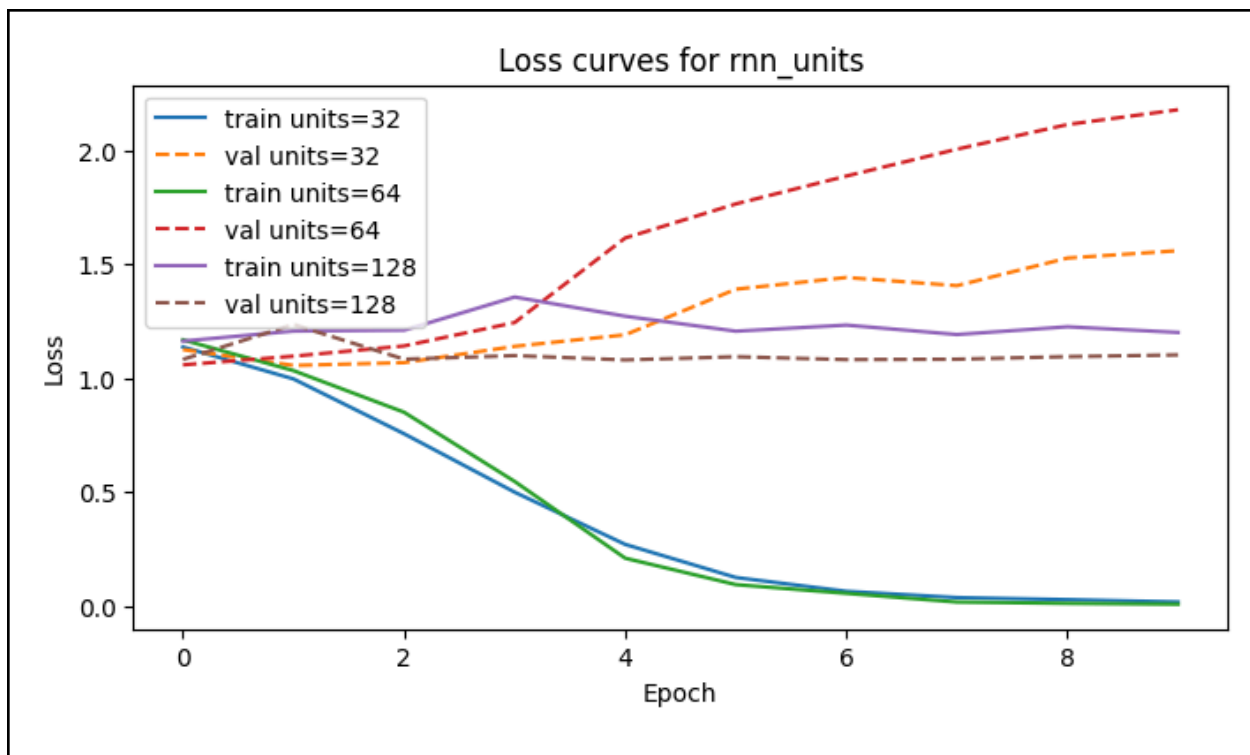


2.2.2 Pengaruh banyak cell RNN per layer

Banyak Layer: 2

Bidirectional: False

Banyak cell RNN	Macro F1 Score
32	0.4363
64	0.4677
128	0.1844
Perbandingan Loss	

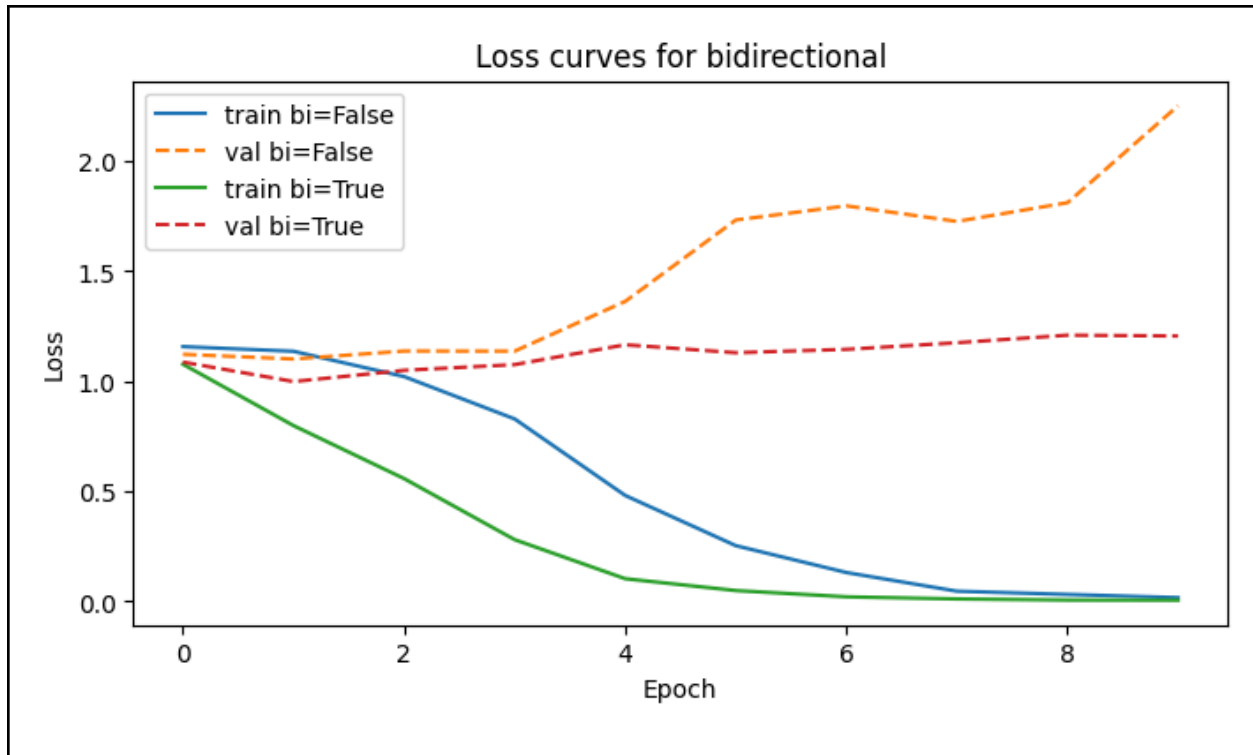


2.2.3 Pengaruh jenis layer RNN berdasarkan arah

Banyak Layer: 2

RNN unit: 64

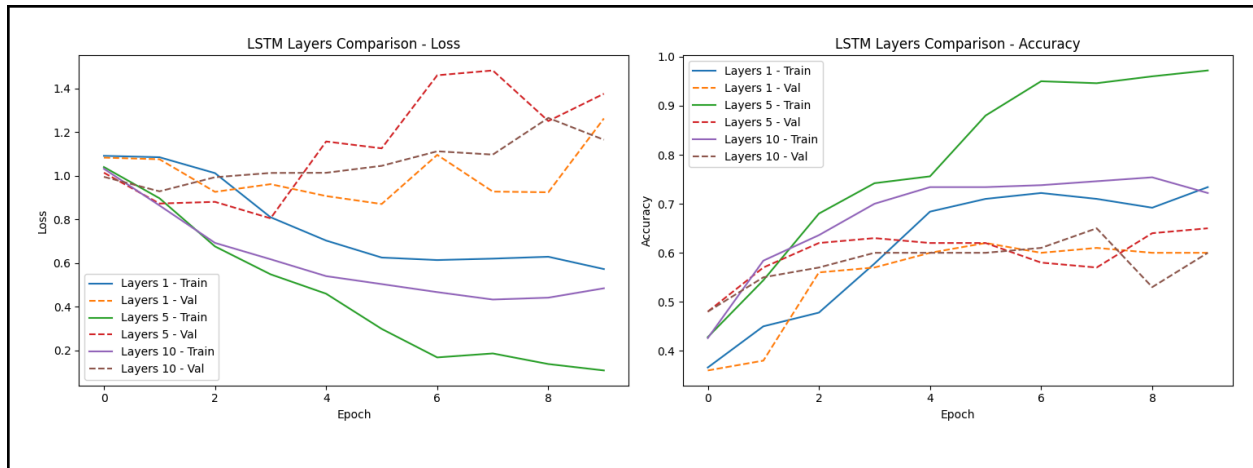
Arah	Macro F1 Score
Unidirectional	0.5710
Bidirectional	0.4601
Perbandingan Loss	



2.3 LSTM

Tabel 2.3 Perbandingan LSTM

2.3.1 Pengaruh jumlah layer LSTM	
Jumlah Layer	Macro F1 Score
1	0.4598
5	0.5976
10	0.4614
Perbandingan Akurasi Dan Loss	



Penjelasan

Model dengan 5 Layer (Performa Terbaik): Konfigurasi ini mencapai skor Macro F1 tertinggi (0.5976).

- Grafik:** Pada grafik akurasi, "Layers 5 - Train" (garis hijau solid) menunjukkan peningkatan yang stabil dan mencapai akurasi training tertinggi. Lebih penting lagi, "Layers 5 - Val" (garis merah putus-putus) pada grafik akurasi juga menunjukkan tren yang umumnya baik, mencapai puncak lalu stabil atau sedikit menurun. Ini mengindikasikan model belajar dengan baik dari data training tanpa langsung mengalami *overfitting* separah konfigurasi lain. Grafik *loss* untuk 5 layer juga menunjukkan *training loss* yang menurun secara konsisten.

Model dengan 1 Layer: Model ini memiliki skor Macro F1 terendah (0.4598).

- Grafik:** Akurasi "Layers 1 - Train" (garis biru solid) meningkat tetapi tetap lebih rendah dibandingkan model 5-layer. Akurasi "Layers 1 - Val" (garis oranye putus-putus) juga relatif rendah dan berfluktuasi, yang mengindikasikan kemungkinan terjadinya *underfitting* – artinya model terlalu sederhana untuk menangkap kompleksitas data. *Loss* untuk 1 layer, meskipun menurun, tidak mencapai serendah model 5-layer untuk data training.

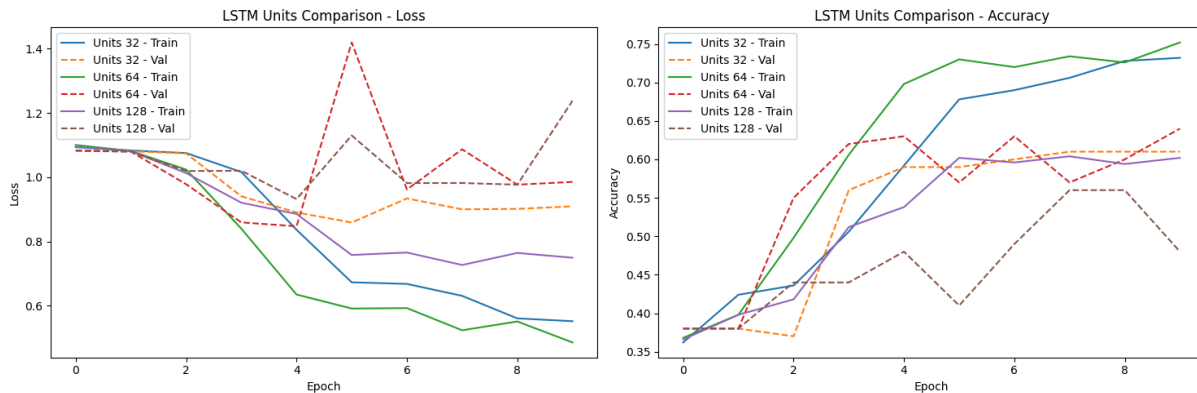
Model dengan 10 Layer: Skor Macro F1 model ini (0.4614) hanya sedikit lebih baik dari model 1-layer dan jauh lebih buruk dari model 5-layer.

- **Grafik:** Akurasi "Layers 10 - Train" (garis ungu solid) menunjukkan peningkatan tetapi tidak melampaui model 5-layer. Yang krusial, akurasi "Layers 10 - Val" (garis cokelat putus-putus) tidak stabil dan umumnya lebih rendah dari akurasi trainingnya, serta lebih rendah dari akurasi validasi model 5-layer. Hal ini, ditambah dengan *validation loss* (garis cokelat putus-putus pada grafik *loss*) yang meningkat signifikan setelah penurunan awal, sangat mengindikasikan terjadinya **overfitting**. Model mempelajari data training terlalu baik, termasuk *noise*-nya, sehingga gagal melakukan generalisasi pada data validasi yang belum pernah dilihat.

Untuk analisis dataset NusaX-Sentiment dengan konfigurasi hyperparameter yang diuji, model LSTM dengan **5 layer memberikan performa terbaik**. Menambah jumlah lapisan pada LSTM memungkinkan model untuk belajar representasi data yang lebih kompleks dan hierarkis. Setiap lapisan tambahan dapat menangkap pola yang lebih abstrak dari lapisan sebelumnya. Namun, Menambah jumlah layer lebih dari itu (menjadi 10) menyebabkan *overfitting* dan menurunkan performa, sementara menggunakan terlalu sedikit layer (1 layer) mengakibatkan *underfitting*.

2.3.2 Pengaruh banyak cell LSTM per layer

Banyak cell LSTM	Macro F1 Score
32	0.5039
64	0.4917
128	0.3864
Perbandingan Akurasi Dan Loss	



Penjelasan

Model dengan 32 Unit (Performa Terbaik di antara yang diuji): Konfigurasi ini mencapai skor Macro F1 tertinggi (0.5039) dibandingkan dua konfigurasi unit lainnya.

- Grafik:** Pada grafik akurasi, "Units 32 - Train" (garis biru solid) menunjukkan peningkatan akurasi yang baik dan relatif stabil. "Units 32 - Val" (garis oranye putus-putus) juga menunjukkan tren peningkatan akurasi validasi, meskipun ada fluktuasi, dan tampaknya memiliki *gap* yang lebih kecil dengan akurasi trainingnya dibandingkan konfigurasi lain, terutama di akhir epoch. Pada grafik *loss*, "Units 32 - Val" menunjukkan *loss* validasi yang relatif stabil setelah penurunan awal, tidak melonjak drastis seperti pada konfigurasi unit yang lebih banyak. Ini mengindikasikan keseimbangan yang lebih baik antara belajar dan generalisasi.

Model dengan 64 Unit: Model ini memiliki skor Macro F1 (0.4917) yang sedikit lebih rendah dari model dengan 32 unit.

- Grafik:** "Units 64 - Train" (garis hijau solid) pada grafik akurasi menunjukkan akurasi training tertinggi di antara ketiganya, yang mengindikasikan model ini belajar dengan sangat baik dari data training. Namun, "Units 64 - Val" (garis merah putus-putus) menunjukkan fluktuasi yang signifikan dan *gap* yang lebih besar dengan akurasi trainingnya, terutama terlihat pada lonjakan *validation loss* ("Units 64 - Val" pada grafik *loss*) di sekitar epoch 4-5 dan setelahnya. Ini adalah indikasi kuat adanya

overfitting. Model mulai menghafal data training sehingga performanya pada data validasi menurun atau tidak stabil.

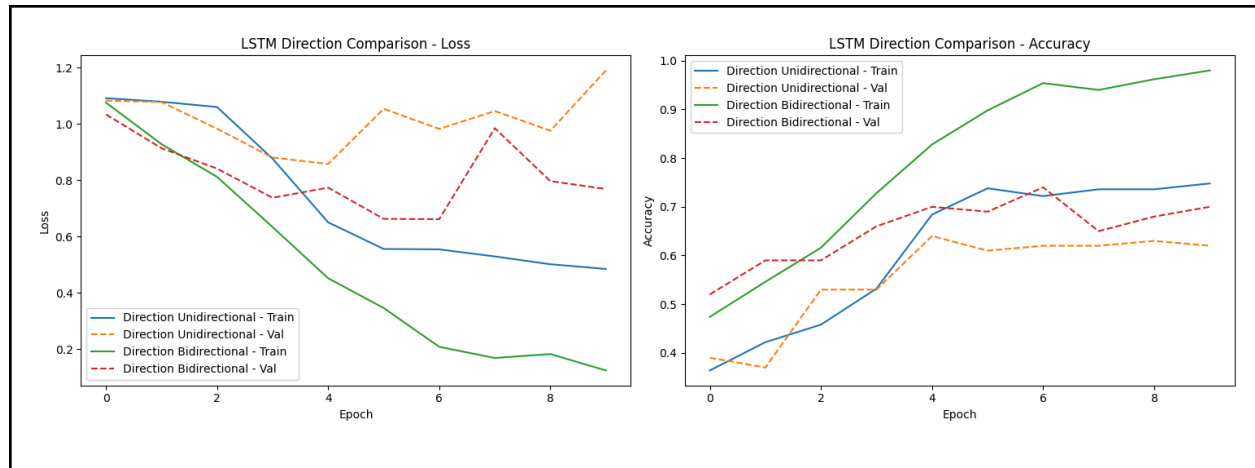
Model dengan 128 Unit (Performa Terendah): Model ini menghasilkan skor Macro F1 terendah (0.3864).

- **Grafik:** "Units 128 - Train" (garis ungu solid) pada grafik akurasi menunjukkan peningkatan, namun akurasi trainingnya tidak setinggi model 64 unit. Yang lebih signifikan, "Units 128 - Val" (garis coklat putus-putus) menunjukkan akurasi validasi yang paling rendah dan paling tidak stabil. Pada grafik *loss*, "Units 128 - Val" menunjukkan *validation loss* yang tinggi dan cenderung meningkat setelah beberapa epoch awal. Ini adalah tanda **overfitting yang parah**. Dengan 128 unit, model menjadi terlalu kompleks untuk jumlah data yang ada atau untuk tugas yang diberikan, sehingga sangat buruk dalam generalisasi ke data baru.

Untuk dataset NusaX-Sentiment dengan arsitektur dan hyperparameter lain yang tetap, **jumlah unit LSTM sebanyak 32 memberikan hasil terbaik** dari ketiga konfigurasi yang di uji. Menambah jumlah unit (ke 64 dan terutama ke 128) tampaknya menyebabkan model menjadi semakin rentan terhadap *overfitting*, yang pada akhirnya menurunkan performa generalisasinya pada data yang belum pernah dilihat.

2.3.3 Pengaruh jenis layer LSTM berdasarkan arah

Arah	Macro F1 Score
Unidirectional	0.4822
Bidirectional	0.7286
Perbandingan Akurasi Dan Loss	



Penjelasan

Performa Unggul Bidirectional LSTM: Bidirectional LSTM (BiLSTM) menunjukkan performa yang jauh lebih unggul dengan Macro F1 Score 0.7286, dibandingkan Unidirectional LSTM (UniLSTM) dengan skor 0.4822. Ini adalah perbedaan yang signifikan.

Grafik Akurasi:

- "Direction Bidirectional - Train" (garis hijau solid) menunjukkan akurasi training yang paling tinggi dan meningkat secara konsisten, mencapai hampir 1.0 (atau 100%).
- "Direction Bidirectional - Val" (garis merah putus-putus) menunjukkan akurasi validasi yang juga secara signifikan lebih tinggi dibandingkan UniLSTM. Meskipun ada sedikit fluktuasi dan *gap* dengan akurasi trainingnya (menandakan sedikit *overfitting*), akurasi validasi BiLSTM secara keseluruhan tetap jauh di atas UniLSTM.
- "Direction Unidirectional - Train" (garis biru solid) dan "Direction Unidirectional - Val" (garis oranye putus-putus) keduanya menunjukkan akurasi yang lebih rendah. Akurasi validasi UniLSTM tampak stagnan di level yang lebih rendah setelah beberapa epoch awal.

Grafik Loss:

- "Direction Bidirectional - Train" (garis hijau solid) menunjukkan *loss* training yang turun paling tajam dan mencapai nilai terendah, yang konsisten dengan akurasi training tertinggi.
- "Direction Bidirectional - Val" (garis merah putus-putus) menunjukkan *validation loss* yang awalnya turun namun kemudian sedikit naik atau berfluktuasi, mengindikasikan bahwa model BiLSTM mungkin mulai sedikit *overfit*. Namun, nilai *loss* validasinya secara umum masih lebih baik (atau setidaknya tidak lebih buruk secara drastis pada titik tertentu) dibandingkan UniLSTM jika kita melihat tren akurasinya yang jauh lebih superior.
- "Direction Unidirectional - Train" (garis biru solid) menunjukkan penurunan *loss* training, tetapi tidak serendah BiLSTM.
- "Direction Unidirectional - Val" (garis oranye putus-putus) menunjukkan *validation loss* yang cenderung lebih tinggi dan lebih fluktuatif atau bahkan meningkat di beberapa titik, yang berkorelasi dengan performa akurasi validasi yang lebih rendah.

Berdasarkan hasil eksperimen, **Bidirectional LSTM secara signifikan lebih efektif daripada Unidirectional LSTM untuk analisis sentimen pada dataset NusaX-Sentimen.** Kemampuannya untuk memanfaatkan konteks dari kedua arah dalam sekuens teks memberikan keunggulan performa yang jelas. Meskipun ada tanda-tanda *overfitting* ringan pada model BiLSTM, manfaat dari representasi kontekstual yang lebih kaya jauh lebih besar daripada dampak negatif *overfitting* tersebut dalam perbandingan ini.

2.3.4 Perbandingan Forward Propagation Manual Dan Keras

Hasil

```
=== COMPARISON RESULTS ===  
Keras Model F1 Score: 0.4648  
Manual Implementation F1 Score: 0.4648  
Difference: 0.0000  
Prediction Agreement: 1.0000  
Unidirectional Results:  
  Keras F1: 0.4648  
  Manual F1: 0.4648  
  Agreement: 1.0000
```

```
=== COMPARISON RESULTS ===  
Keras Model F1 Score: 0.7286  
Manual Implementation F1 Score: 0.7286  
Difference: 0.0000  
Prediction Agreement: 1.0000  
Bidirectional Results:  
  Keras F1: 0.7286  
  Manual F1: 0.7286  
  Agreement: 1.0000
```

Penjelasan

Hasil forward propagation manual dan menggunakan Keras sama mengindikasikan algoritma LSTM yang diimplementasikan secara manual sudah benar.

KESIMPULAN DAN SARAN

A. CNN

Berdasarkan hasil eksperimen hyperparameter dan implementasi forward propagation CNN from scratch pada dataset CIFAR-10, diperoleh beberapa temuan penting:

1. Pengaruh Jumlah Layer Konvolusi:

- Model dengan 3 blok konvolusi memberikan performa terbaik dibandingkan 1 dan 2 blok
- Peningkatan kedalaman network secara konsisten meningkatkan kemampuan ekstraksi fitur hierarkis
- Model dengan 1 blok mengalami underfitting, sementara 3 blok mencapai keseimbangan optimal antara kompleksitas dan generalisasi

2. Pengaruh Jumlah Filter:

- Konfigurasi filter besar (64-128) memberikan performa superior dibandingkan filter kecil (16-32)
- Peningkatan jumlah filter memberikan improvement dengan diminishing returns
- Trade-off antara performa dan kompleksitas komputasi perlu dipertimbangkan

3. Pengaruh Ukuran Kernel:

- Kernel 3x3 terbukti optimal untuk dataset CIFAR-10, sejalan dengan best practice CNN modern
- Kernel 2x2 terlalu kecil untuk menangkap fitur spasial yang kompleks
- Kernel 5x5 cenderung mengalami overfitting dan inefisiensi parameter

4. Pengaruh Jenis Pooling:

- Max pooling secara konsisten outperform average pooling
- Max pooling lebih efektif dalam mempertahankan fitur diskriminatif yang penting untuk klasifikasi
- Average pooling cenderung mengaburkan detail penting antar kelas

B. RNN

Berdasarkan hasil evaluasi model dengan berbagai konfigurasi pada dataset NusaX, diperoleh beberapa temuan penting:

1. **Jumlah lapisan optimal** untuk RNN adalah **2**. Lebih dari itu dapat menurunkan performa, saat ditambah menjadi 3 lapisan, performa menurun drastis (**0.3577**), mengindikasikan kemungkinan overfitting atau kesulitan dalam pelatihan model yang lebih dalam.
2. **Jumlah unit optimal** adalah **64**, di mana keseimbangan kompleksitas dan generalisasi tercapai. Terlalu banyak unit (**128**) menyebabkan penurunan tajam dalam performa (**0.1844**), kemungkinan besar karena **overfitting** atau kesulitan dalam konvergensi.
3. **Bidirectional RNN** memberikan peningkatan signifikan dalam performa karena kemampuan memproses konteks dua arah dalam sekuens teks.

C. LSTM

1. Menambah jumlah lapisan pada LSTM memungkinkan model untuk belajar representasi data yang lebih kompleks dan hierarkis. Setiap lapisan tambahan dapat menangkap pola yang lebih abstrak dari lapisan sebelumnya. Namun, Menambah jumlah layer terlalu banyak dapat menyebabkan overfitting dan menurunkan performa, sementara menggunakan terlalu sedikit layer mengakibatkan underfitting.
2. Menambah jumlah unit LSTM terlalu banyak dapat menyebabkan model menjadi semakin rentan terhadap *overfitting*, yang pada akhirnya menurunkan performa generalisasinya pada data yang belum pernah dilihat.
3. Kemampuan Bidirectional untuk memanfaatkan konteks dari kedua arah dalam sekuens teks memberikan keunggulan performa yang jelas.

PEMBAGIAN TUGAS

NIM	Tugas
13522151	RNN, docs
13522152	LSTM, docs
13522149	CNN, docs

REFERENSI

- https://d2l.ai/chapter_recurrent-modern/lstm.html
- https://d2l.ai/chapter_recurrent-modern/deep-rnn.html
- https://d2l.ai/chapter_recurrent-modern/bi-rnn.html
- https://d2l.ai/chapter_recurrent-neural-networks/index.html
- https://d2l.ai/chapter_convolutional-neural-networks/index.html
- <https://numpy.org/doc/2.1/reference/generated/numpy.einsum.html>

LAMPIRAN

Pranala GitHub: <https://github.com/mroiin/Tubes-2-ML>