



# **TUGAS BESAR 2**

## **Convolutional Neural Network dan Recurrent Neural Network**

**Dibuat oleh:**

Kelompok 49

**Anggota:**

1. 13521049 - Brian Kheng
2. 13522013 - Denise Felicia Tiowanni
3. 13522053 - Erdianti Wiga Putri Andini

**IF3270 - Pembelajaran Mesin**

**2025**

# Daftar Isi

<b>Daftar Isi.....</b>	<b>1</b>
<b>Bab I. Deskripsi Persoalan.....</b>	<b>3</b>
1.1. Latar Belakang.....	3
1.2. Ruang Lingkup Implementasi.....	3
<b>Bab II. Pembahasan.....</b>	<b>6</b>
2.1. Penjelasan Implementasi.....	6
2.1.1. Deskripsi Kelas beserta Deskripsi Atribut dan Methodnya.....	6
2.1.1.1. CNN.....	6
2.1.1.2. Simple RNN.....	11
2.1.1.3. LSTM.....	18
2.1.2. Penjelasan Forward Propagation.....	25
2.1.2.1. CNN.....	25
2.1.2.2. Simple RNN.....	26
2.1.2.3. LSTM.....	29
2.1.3. Penjelasan Backward Propagation.....	33
2.1.3.1. CNN.....	33
2.1.3.2. Simple RNN.....	33
2.1.3.3. LSTM.....	35
2.2. Hasil pengujian.....	38
2.2.1. CNN.....	38
2.2.1.1. Pengaruh jumlah layer konvolusi.....	39
2.2.1.2. Pengaruh banyak filter per layer konvolusi.....	41
2.2.1.3. Pengaruh ukuran filter per layer konvolusi.....	43
2.2.1.4. Pengaruh jenis pooling layer.....	45
2.2.1.5. Perbandingan Keras Model vs Scratch Model.....	46
2.2.2. Simple RNN.....	46
2.2.2.1. Pengaruh jumlah layer RNN.....	47
2.2.2.2. Pengaruh banyak cell RNN per layer.....	50
2.2.2.3. Pengaruh jenis layer RNN berdasarkan arah.....	52
2.2.2.4. Perbandingan Forward Propagation Keras Model vs. Scratch Model....	54
2.2.2.5. Perbandingan Scratch Model dengan hanya Forward Propagation vs. dengan Forward + Backward Propagation.....	59
2.2.3. LSTM.....	61
2.2.3.1. Pengaruh jumlah layer LSTM.....	62
2.2.3.2. Pengaruh banyak cell LSTM per layer.....	64
2.2.3.3. Pengaruh jenis layer LSTM berdasarkan arah.....	66

2.2.3.4. Perbandingan Keras Model vs Scratch Model.....	67
2.2.2.6. Pengujian Implementasi Backward Propagation LSTM.....	72
2.2.3.5. Pengujian Implementasi Batch Inference LSTM.....	74
<b>Bab III. Kesimpulan dan Saran.....</b>	<b>76</b>
3.1. Kesimpulan.....	76
3.2. Saran.....	76
<b>Pembagian Tugas.....</b>	<b>78</b>
<b>Referensi.....</b>	<b>79</b>

# Bab I. Deskripsi Persoalan

## 1.1. Latar Belakang

Dalam era modern pembelajaran mesin, kemampuan untuk memahami dan mengimplementasikan model deep learning merupakan keahlian fundamental yang sangat dibutuhkan, khususnya dalam bidang seperti visi komputer dan pemrosesan bahasa alami. Dua arsitektur yang secara luas digunakan dalam bidang ini adalah **Convolutional Neural Network (CNN)** dan **Recurrent Neural Network (RNN)**, beserta varian lanjutannya yaitu **Long Short-Term Memory (LSTM)**. CNN terbukti efektif dalam tugas-tugas klasifikasi dan pengenalan gambar, sedangkan RNN dan LSTM sangat berguna untuk menangani data sekuensial, seperti analisis sentimen atau prediksi teks.

Pemahaman terhadap algoritma forward propagation menjadi hal penting karena merupakan inti dari proses prediksi dalam jaringan saraf. Proses ini menentukan bagaimana data masukan ditransformasikan melalui jaringan hingga menghasilkan keluaran. Dengan mengimplementasikan forward propagation tanpa bantuan framework high-level seperti Keras atau PyTorch, kami diharapkan dapat memahami dengan lebih mendalam proses internal dari setiap layer dalam model deep learning, serta bagaimana bobot yang telah dilatih digunakan dalam inferensi.

Tugas besar ini memberikan kesempatan bagi kami untuk tidak hanya menggunakan model siap pakai, tetapi juga membangun dan menguji model dari dasar. Selain itu, eksperimen terhadap berbagai hyperparameter memungkinkan kami untuk mengevaluasi pengaruh struktur model terhadap performa akhir, sehingga memperkuat kemampuan analisis kritis terhadap desain arsitektur jaringan saraf.

## 1.2. Ruang Lingkup Implementasi

Adapun ruang lingkup implementasi tugas besar ini mencakup:

### 1. Pelatihan Model CNN dengan CIFAR-10

- Membangun dan melatih model klasifikasi gambar menggunakan arsitektur CNN pada dataset CIFAR-10 dengan library Keras.
- Mencakup layer Conv2D, Pooling, Flatten/GlobalPooling, dan Dense.
- Melakukan eksperimen terhadap variasi hyperparameter: jumlah layer konvolusi, banyak filter, ukuran filter, dan jenis pooling.
- Evaluasi performa menggunakan macro F1-score.

## **2. Implementasi Forward Propagation CNN from Scratch**

- Mengimplementasikan modul forward propagation CNN dari awal menggunakan NumPy.
- Membaca bobot model hasil pelatihan Keras.
- Menguji akurasi inferensi dibandingkan dengan Keras menggunakan data test.

## **3. Pelatihan Model RNN dengan NusaX-Sentiment**

- Melatih model klasifikasi teks Bahasa Indonesia menggunakan Simple RNN.
- Preprocessing teks dengan TextVectorization dan Embedding.
- Eksperimen terhadap jumlah layer RNN, banyaknya cell, dan arah (bidirectional/unidirectional).

## **4. Implementasi Forward Propagation RNN from Scratch**

- Membangun forward propagation Simple RNN menggunakan NumPy.
- Menggunakan bobot hasil pelatihan Keras dan membandingkan hasil inferensi pada data uji.

## **5. Pelatihan Model LSTM dengan NusaX-Sentiment**

- Melatih model klasifikasi teks menggunakan LSTM.
- Melakukan eksperimen terhadap jumlah layer, jumlah cell per layer, dan arah layer.

## **6. Implementasi Forward Propagation LSTM from Scratch**

- Mengimplementasikan forward propagation dari awal untuk LSTM menggunakan NumPy.
- Menguji kesesuaian output dengan model Keras dan mengevaluasi performa menggunakan macro F1-score.

## **7. Implementasi Backward Propagation**

- Mengembangkan backward propagation untuk seluruh jenis layer yang digunakan pada implementasi RNN dan LSTM.

## Bab II. Pembahasan

### 2.1. Penjelasan Implementasi

#### 2.1.1. Deskripsi Kelas beserta Deskripsi Atribut dan Methodnya

Dalam pengerjaan tugas besar ini, kami mengimplementasikan tiga jenis model, yaitu:

##### 2.1.1.1. CNN

Model ini dibangun menggunakan method utama `build_cnn_from_keras` yang merupakan implementasi manual dari arsitektur Convolutional Neural Network (CNN) dengan berbagai kelas pendukung seperti `Conv2D`, `MaxPooling2D`, `AveragePooling2D`, `Flatten`, `Dense`, `ReLU`, `Softmax`, `Dropout`, dan `CNNModel`.

##### a. Kelas Conv2D

Kelas `Conv2D` adalah layer yang melakukan konvolusi 2D.

Berikut adalah atribut dalam kelas `Conv2D`:

Atribut	Deskripsi
<code>weights</code>	Bobot dari <i>pre-trained</i> dengan ukuran ( <code>kernel_size</code> , <code>kernel_size</code> , <code>in_channels</code> , <code>out_channels</code> )
<code>biases</code>	Bias dari <i>pre-trained</i> dengan ukuran ( <code>out_channels</code> ,)
<code>stride</code>	Stride dari operasi konvolusi

dan berikut adalah method dalam kelas `Conv2D`:

Method	Deskripsi
<code>__init__(self, weights, biases, stride=1)</code>	Melakukan inisiasi objek dengan parameter <code>weights</code> , <code>biases</code> , dan <code>stride</code>

forward(self, inputs)	Forward pass layer Conv2D, melakukan proses konvolusi 2D terhadap input (batch_size, height, width, in_channels) dan menghasilkan output (batch_size, output_height, output_width, out_channels)
-----------------------	--

#### b. Kelas MaxPooling2D

Kelas MaxPooling2D adalah layer yang melakukan pooling berdasarkan nilai maksimum untuk mereduksi ukuran *feature maps*.

Atribut	Deskripsi
pool_size	Ukuran dari pooling window (height, width)
strides	Stride dari operasi pooling

dan berikut adalah method dalam kelas MaxPooling2D:

Method	Deskripsi
__init__(self, pool_size=(2, 2), strides=None)	Melakukan inisiasi objek dengan parameter pool_size dan strides
forward(self, inputs)	Forward pass layer MaxPooling2D, melakukan proses max pooling berdasarkan ukuran window dan strides

#### c. Kelas AveragePooling2D

Kelas AveragePooling2D adalah layer yang melakukan pooling berdasarkan nilai rata-rata untuk mereduksi ukuran *feature maps*.



Atribut	Deskripsi
pool_size	Ukuran dari pooling window (height, width)
strides	Stride dari operasi pooling

dan berikut adalah method dalam kelas MaxPooling2D:

Method	Deskripsi
<code>__init__(self, pool_size=(2, 2), strides=None)</code>	Melakukan inisiasi objek dengan parameter pool_size dan strides
<code>forward(self, inputs)</code>	Forward pass layer AveragePooling2D, melakukan proses average pooling berdasarkan ukuran window dan strides

#### d. Kelas Flatten

Kelas Flatten adalah layer yang melakukan flatten terhadap feature maps (tensor 3D) menjadi tensor 1D.

Method	Deskripsi
<code>forward(self, inputs)</code>	Forward pass layer Flatten, mengubah ukuran tensor 3D menjadi 1D

#### e. Kelas Dense

Kelas Dense adalah layer yang terdiri atas fully connected layer.

Atribut	Deskripsi
weights	Bobot dari <i>pre-trained</i> dengan ukuran (input_dim,

	output_dim)
biases	Bias dari <i>pre-trained</i> dengan ukuran (output_dim,)

dan berikut adalah method dalam kelas Dense:

Method	Deskripsi
<code>__init__(self, weights, biases)</code>	Melakukan inisiasi objek dengan parameter weights dan biases
<code>forward(self, inputs)</code>	Forward pass layer Dense berdasarkan parameter weights dan biases

#### f. Kelas ReLU

Kelas ReLU adalah fungsi aktivasi ReLU.

Method	Deskripsi
<code>forward(self, inputs)</code>	Proses transformasi inputs dengan fungsi aktivasi ReLU

#### g. Kelas Softmax

Kelas Softmax adalah fungsi aktivasi softmax.

Method	Deskripsi
<code>forward(self, inputs)</code>	Proses transformasi inputs dengan fungsi aktivasi softmax

#### h. Kelas Dropout

Kelas Dropout adalah layer yang digunakan untuk regularisasi.

Atribut	Deskripsi
---------	-----------

rate	dropout rate.
------	---------------

dan berikut adalah method dalam kelas Dropout:

Method	Deskripsi
<code>__init__(self, rate)</code>	Melakukan inisiasi objek dengan parameter rate
<code>forward(self, inputs)</code>	Forward pass layer Dropout berdasarkan rate yang telah ditentukan

#### i. Kelas CNNModel

Kelas CNNModel adalah kelas untuk menginisiasi CNN model.

Atribut	Deskripsi
layers	layer-layer yang digunakan untuk membuat CNN model

dan berikut adalah method dalam kelas Dropout:

Method	Deskripsi
<code>__init__(self)</code>	Melakukan inisiasi objek dengan layers mula-mula kosong
<code>add_layer(self, layer)</code>	Menambahkan layer baru ke model
<code>forward(self, inputs)</code>	Forward propagation ke CNN model
<code>predict(self, inputs,</code>	Melakukan prediksi terhadap input data dengan jumlah batch yang ditentukan

batch_size=32)	
----------------	--

#### j. Method `build_cnn_from_keras`

Method `build_cnn_from_keras` adalah kelas untuk menginisiasi CNN model dari *pre-trained* model keras.

#### 2.1.1.2. Simple RNN

Model ini dibangun menggunakan kelas utama `RNNModelFromScratch` yang merupakan implementasi manual dari arsitektur Recurrent Neural Network (RNN) dengan berbagai kelas pendukung seperti Embedding Layer, Bidirectional RNN, Dropout, dan Dense Layer.

##### a. Kelas `EmbeddingLayer`

Kelas `EmbeddingLayer` adalah layer yang berfungsi untuk mengkonversi token ID menjadi representasi vektor dense (embedding) dengan dimensi tertentu.

Berikut adalah atribut dalam kelas `EmbeddingLayer`:

Atribut	Deskripsi
<code>weights</code>	Matrix bobot embedding dengan dimensi (vocab_size, embedding_dim)
<code>vocab_size</code>	Ukuran vocabulary (jumlah token unik)
<code>embedding_dim</code>	Dimensi vector embedding
<code>input_sequences</code>	Menyimpan input sequences untuk keperluan backward propagation

dan berikut adalah method dalam kelas `EmbeddingLayer`:

Method	Deskripsi
--------	-----------

<code>__init__(weights)</code>	Konstruktor untuk inisialisasi layer dengan matrix bobot embedding
<code>forward(input_sequences)</code>	Melakukan forward pass untuk mengkonversi token ID menjadi embedding vectors
<code>backward(grad_output)</code>	Melakukan backward pass untuk menghitung gradien terhadap bobot embedding

#### b. Kelas SimpleRNNCell

Kelas SimpleRNNCell adalah implementasi sel RNN dasar yang memproses satu timestep input dan menghasilkan hidden state.

Berikut adalah atribut dalam kelas SimpleRNNCell:

Atribut	Deskripsi
<code>W_input</code>	Matrix bobot untuk input ke hidden state
<code>W_recurrent</code>	Matrix bobot untuk koneksi recurrent (hidden state sebelumnya)
<code>bias</code>	Vector bias untuk perhitungan linear
<code>units</code>	Jumlah unit/neuron dalam sel RNN
<code>x_sequence</code>	Menyimpan sequence input untuk backward pass
<code>h_sequence</code>	Menyimpan sequence hidden states untuk backward pass
<code>tanh_outputs</code>	Menyimpan output linear sebelum aktivasi tanh

dan berikut adalah method dalam kelas SimpleRNNCell:

Method	Deskripsi
--------	-----------

<code>__init__(input_weights, recurrent_weights, bias)</code>	Konstruktor untuk inisialisasi sel RNN dengan bobot dan bias
<code>forward_step(x_t, h_prev)</code>	Melakukan forward pass untuk satu timestep
<code>backward_step(grad_h_t, x_t, h_prev, linear_output)</code>	Melakukan backward pass untuk satu timestep dan menghitung gradien

### c. Kelas SimpleRNNLayer

Kelas SimpleRNNLayer adalah wrapper yang mengelola SimpleRNNCell untuk memproses seluruh sequence input.

Berikut adalah atribut dalam kelas SimpleRNNLayer:

Atribut	Deskripsi
<code>rnn_cell</code>	Instance dari SimpleRNNCell yang digunakan
<code>return_sequences</code>	Boolean untuk menentukan apakah mengembalikan seluruh sequence atau hanya output terakhir
<code>go_backwards</code>	Boolean untuk memproses sequence secara mundur
<code>inputs</code>	Menyimpan input sequence untuk backward pass
<code>h_sequence</code>	Menyimpan sequence hidden states

dan berikut adalah method dalam kelas SimpleRNNLayer:

Method	Deskripsi
<code>__init__(rnn_cell, return_sequences, go_backwards)</code>	Konstruktor untuk inisialisasi layer RNN
<code>forward(inputs)</code>	Memproses seluruh sequence input melalui RNN cell
<code>backward(grad_output)</code>	Melakukan backpropagation through time (BPTT)

**d. Kelas BidirectionalRNNLayer**

Kelas BidirectionalRNNLayer menggabungkan dua SimpleRNNLayer (forward dan backward) untuk memproses sequence dari kedua arah.

Berikut adalah atribut dalam kelas BidirectionalRNNLayer:

Atribut	Deskripsi
<code>forward_rnn</code>	Instance SimpleRNNLayer untuk pemrosesan forward
<code>backward_rnn</code>	Instance SimpleRNNLayer untuk pemrosesan backward
<code>return_sequences</code>	Boolean untuk menentukan format output

dan berikut adalah method dalam kelas BidirectionalRNNLayer:

Method	Deskripsi
--------	-----------

<code>__init__(forward_rnn, backward_rnn, return_sequences)</code>	Konstruktor untuk inisialisasi bidirectional RNN
<code>forward(inputs)</code>	Memproses input melalui kedua RNN dan menggabungkan outputnya
<code>backward(grad_output)</code>	Melakukan backward pass untuk kedua arah RNN

#### e. Kelas DropoutLayer

Kelas DropoutLayer implementasi teknik regularisasi dropout untuk mencegah overfitting.

Berikut adalah atribut dalam kelas DropoutLayer:

Atribut	Deskripsi
rate	Tingkat dropout (probabilitas neuron yang di-drop)
mask	Binary mask yang digunakan untuk dropout

dan berikut adalah method dalam kelas DropoutLayer:

Method	Deskripsi
<code>__init__(rate)</code>	Konstruktor untuk inisialisasi layer dropout
<code>forward(inputs, training)</code>	Menerapkan dropout jika dalam mode training
<code>backward(grad_output)</code>	Melakukan backward pass dengan menerapkan mask yang sama



#### f. Kelas DenseLayer

Kelas DenseLayer adalah implementasi fully connected layer dengan berbagai fungsi aktivasi.

Berikut adalah atribut dalam kelas DenseLayer:

Atribut	Deskripsi
weights	Matrix bobot koneksi antar neuron
bias	Vector bias untuk setiap neuron
activation	Jenis fungsi aktivasi ('linear', 'relu', 'softmax')
inputs	Menyimpan input untuk backward pass
linear_output	Menyimpan output linear sebelum aktivasi

dan berikut adalah method dalam kelas DenseLayer:

Method	Deskripsi
__init__(weights, bias, activation)	Konstruktor untuk inisialisasi dense layer
forward(inputs)	Melakukan transformasi linear dan menerapkan fungsi aktivasi
backward(grad_output, y_true)	Menghitung gradien terhadap bobot, bias, dan input

#### g. Kelas RNNModelFromScratch

Kelas RNNModelFromScratch adalah kelas utama yang menggabungkan semua komponen untuk membentuk model RNN lengkap.

Berikut adalah atribut dalam kelas RNNModelFromScratch:

Atribut	Deskripsi
keras_model	Model Keras yang dimuat untuk ekstraksi bobot
layers	List berisi semua layer dalam model
layer_names	List nama-nama layer
vectorizer	TextVectorization untuk preprocessing teks

dan berikut adalah method dalam kelas RNNModelFromScratch:

Method	Deskripsi
<code>__init__(model_path, vectorizer_path, training_texts)</code>	Konstruktor untuk memuat model dan vectorizer
<code>_extract_layers()</code>	Method private untuk mengekstrak layer dari model Keras
<code>forward(texts, training)</code>	Melakukan forward pass untuk prediksi
<code>backward(grad_output, y_true)</code>	Melakukan backward pass untuk menghitung gradien
<code>compute_loss(predictions, y_true)</code>	Menghitung sparse categorical cross-entropy
<code>predict(texts, training)</code>	Method untuk melakukan prediksi (wrapper dari forward)

### 2.1.1.3. LSTM

Model ini dibangun menggunakan kelas utama LSTMFromScratch yang merupakan implementasi manual dari arsitektur Long Short-Term Memory (LSTM). LSTM adalah jenis RNN yang mampu mempelajari dependensi jangka panjang dengan menggunakan mekanisme gating. Implementasi ini terdiri dari berbagai kelas pendukung seperti LSTMCell, UnidirectionalLSTMLayer, BidirectionalLSTMLayer, EmbeddingLayer, DropoutLayer, dan DenseLayer. Model ini memungkinkan pembuatan arsitektur LSTM yang fleksibel dengan kemampuan untuk memuat bobot dari model Keras yang sudah ada.

#### a. Kelas LSTMCell

Kelas LSTMCell merupakan implementasi dari satu sel LSTM yang memproses satu time step input.

Berikut adalah atribut dalam kelas LSTMCell:

Atribut	Deskripsi
input_dim	Dimensi vektor input
hidden_dim	Dimensi hidden state/output
Wi	Matriks bobot untuk input ke semua gates, berukuran [input_dim, 4×hidden_dim]
Ui	Matriks bobot recurrent untuk semua gates, berukuran [hidden_dim, 4×hidden_dim]
bi	Vektor bias untuk semua gates, berukuran [4×hidden_dim]
cache	Dictionary untuk menyimpan nilai-nilai intermediate untuk backward pass

dan berikut adalah method dalam kelas EmbeddingLayer:

Method	Deskripsi
load_weights(weights, bias)	Memuat bobot dari model Keras ke dalam sel LSTM
forward(x, h_prev, c_prev)	Melakukan forward pass untuk satu time step dan mengembalikan hidden state serta cell state baru
backward(dh_next, dc_next)	Melakukan backward pass untuk satu time step dan menghitung gradien

#### b. Kelas UnidirectionalLSTMLayer

Kelas UnidirectionalLSTMLayer mengatur LSTMCell untuk memproses seluruh sequence input secara berurutan.

Berikut adalah atribut dalam kelas UnidirectionalLSTMLayer:

Atribut	Deskripsi
lstm_cell	Instance dari kelas LSTMCell
hidden_dim	Dimensi hidden state pada LSTMCell
cache	Dictionary untuk menyimpan nilai-nilai intermediate (state, gate, dll) selama forward pass, digunakan untuk backward pass

dan berikut adalah method dalam kelas UnidirectionalLSTMLayer:

Method	Deskripsi
load_weights(keras_layer)	Memuat bobot dari layer LSTM Keras ke dalam sel LSTM custom atau implementasi lain.

	Biasanya digunakan untuk menginisialisasi bobot dari model Keras yang sudah dilatih.
<code>forward(x, return_sequences=False)</code>	Memproses seluruh sequence input x melalui LSTM dan mengembalikan output. Jika <code>return_sequences=True</code> , mengembalikan output untuk setiap time step; jika <code>False</code> , hanya output terakhir yang dikembalikan.
<code>backward(dout)</code>	Melakukan backpropagation through time (BPTT) untuk seluruh sequence, menghitung gradien berdasarkan error dout yang diterima dari layer berikutnya.

### c. Kelas `BidirectionalLSTMLayer`

Kelas `BidirectionalLSTMLayer` mengimplementasikan layer LSTM bidirectional yang memproses sequence dari arah forward dan backward.

Berikut adalah atribut dalam kelas `BidirectionalLSTMLayer`:

Atribut	Deskripsi
<code>forward_lstm</code>	Instance <code>LSTMCell</code> untuk arah forward
<code>backward_lstm</code>	Instance <code>LSTMCell</code> untuk arah backward
<code>hidden_dim</code>	Dimensi hidden state untuk setiap arah
<code>cache</code>	Dictionary untuk menyimpan nilai-nilai intermediate

dan berikut adalah method dalam kelas BidirectionalLSTMLayer:

Method	Deskripsi
load_weights(keras_layer)	Memuat bobot dari layer Bidirectional LSTM Keras ke dalam model custom atau implementasi lain.
forward(x, return_sequences=False)	Memproses input melalui kedua arah LSTM (forward dan backward) dan menggabungkan hasilnya. Jika return_sequences=True, mengembalikan output untuk setiap time step; jika False, hanya output terakhir yang dikembalikan.
backward(dout)	Melakukan backward pass untuk kedua arah LSTM, menghitung gradien berdasarkan error dout yang diterima dari layer berikutnya.

#### d. Kelas EmbeddingLayer

Kelas EmbeddingLayer mengkonversi token ID menjadi representasi vektor dense (embedding).

Berikut adalah atribut dalam kelas EmbeddingLayer:

Atribut	Deskripsi
vocab_size	Ukuran vocabulary, yaitu jumlah token unik dalam kamus
embedding_dim	Dimensi vektor embedding yang merepresentasikan setiap token
weights	Matriks bobot embedding dengan dimensi

	(vocab_size, embedding_dim)
cache	Dictionary untuk menyimpan input yang digunakan pada backward pass

dan berikut adalah method dalam kelas EmbeddingLayer:

Method	Deskripsi
load_weights(ke ras_layer)	Memuat bobot embedding dari layer Embedding Keras ke dalam model custom
forward(x)	Mengubah input token ID menjadi embedding vectors sesuai bobot yang tersimpan
backward(dout)	Menghitung gradien terhadap bobot embedding berdasarkan error yang diterima dari layer berikutnya

#### e. Kelas DenseLayer

Kelas DenseLayer mengimplementasikan fully connected layer dengan berbagai fungsi aktivasi.

Berikut adalah atribut dalam kelas DenseLayer:

Atribut	Deskripsi
input_dim	Dimensi input ke layer
output_dim	Dimensi output dari layer
activation	Jenis fungsi aktivasi yang digunakan, misalnya 'relu', 'softmax', atau None

W	Matriks bobot dengan dimensi (input_dim, output_dim)
b	Vektor bias dengan dimensi (output_dim)
cache	Dictionary untuk menyimpan nilai-nilai intermediate selama forward pass

dan berikut adalah method dalam kelas DenseLayer:

Method	Deskripsi
load_weights(ke ras_layer)	Memuat bobot dari layer Dense Keras ke dalam model custom atau implementasi lain.
forward(x)	Melakukan transformasi linear pada input x dan menerapkan fungsi aktivasi yang dipilih.
backward(dout)	Menghitung gradien terhadap bobot, bias, dan input berdasarkan error dout dari layer berikutnya.

#### f. Kelas DropoutLayer

Kelas DropoutLayer mengimplementasikan teknik regularisasi dropout untuk mencegah overfitting.

Berikut adalah atribut dalam kelas DropoutLayer:

Atribut	Deskripsi
rate	Tingkat dropout, yaitu probabilitas neuron yang di-drop saat training (misal 0.3 artinya 30%)
cache	Dictionary untuk menyimpan mask (topeng) biner



	yang digunakan saat forward pass untuk backward pass
--	--

dan berikut adalah method dalam kelas DropoutLayer:

Method	Deskripsi
forward(x, training=False)	Menerapkan dropout pada input x jika dalam mode training
backward(dout)	Menerapkan mask yang sama pada gradien dout selama backward pass

#### g. Kelas LSTMFromScratch

Kelas LSTMFromScratch adalah kelas utama yang menggabungkan semua komponen untuk membentuk model LSTM lengkap.

Berikut adalah atribut dalam kelas LSTMFromScratch:

Atribut	Deskripsi
layers	List berisi semua layer dalam model

dan berikut adalah method dalam kelas LSTMFromScratch:

Method	Deskripsi
add_layer(layer)	Menambahkan layer baru ke dalam model
forward(x, training=False)	Melakukan forward pass melalui semua layer dalam model, memproses input x secara berurutan dari layer pertama hingga terakhir. Parameter training menentukan apakah mode

	training atau inference.
<code>backward(dout)</code>	Melakukan backward pass untuk menghitung gradien dari semua layer berdasarkan error dout yang diterima dari layer berikutnya
<code>train_step(x, y, learning_rate=0.01)</code>	Melakukan satu langkah training, meliputi forward pass, menghitung loss, backward pass, dan update parameter dengan learning rate yang diberikan
<code>batch_inference(data, batch_size=32)</code>	Melakukan inferensi pada batch data dengan ukuran batch tertentu untuk efisiensi komputasi dan memori
<code>create_from_scratch_model(keras_model_path, bidirectional=False)</code>	Method static untuk membuat model dari awal berdasarkan model Keras yang sudah ada, dengan opsi menggunakan layer bidirectional atau tidak

## 2.1.2. Penjelasan Forward Propagation

### 2.1.2.1. CNN

Input image CIFAR-10 memiliki shape (32, 32, 3).

#### 1. Conv2D

Layer konvolusi 2D dilakukan terhadap input (32, 32, 3) dengan stride=1, kernel size=3, dan filter=64 akan menghasilkan output shape (30, 30, 64).

#### 2. ReLU

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

Melakukan transformasi input menggunakan fungsi aktivasi ReLU.

### 3. MaxPooling2D

Melakukan proses pooling dengan pooling window (2, 2) dan stride=2 terhadap input shape (30, 30, 64), menghasilkan output shape (15, 15, 64).

### 4. Flatten

Melakukan flatten terhadap input (15, 15, 64) menghasilkan (14.400,1).

### 5. Dense

$$z = \mathbf{W}h + \mathbf{b}$$

Fully connected layer menerima input (14.400,1) menghasilkan output shape (10,1).

### 6. Softmax

$$\hat{y}_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Fungsi aktivasi softmax untuk menentukan kelas dengan probabilitas tertinggi.

## 2.1.2.2. Simple RNN

### 1. Embedding Layer

EmbeddingLayer akan mengubah token ID integer (hasil dari TextVectorization) menjadi vektor embedding berdimensi tetap.

- Proses Forward:

```
embedded[i, j] = self.weights[token_id]
```

- Rumus:

Jika x adalah input (token ID), dan E adalah matriks embedding (vocab\_size × embedding\_dim), maka:

$$\mathbf{e}_t = E[x_t]$$

## 2. Simple RNN Cell

SimpleRNNCell akan memproses satu langkah waktu (time-step) dari urutan input dan menyimpan state tersembunyi (hidden state).

- Proses Forward:

```
linear_output = np.dot(x_t, self.W_input) +  
np.dot(h_prev, self.W_recurrent) + self.bias  
h_t = np.tanh(linear_output)
```

- Rumus:

$$\tilde{h}_t = \mathbf{W}_x x_t + \mathbf{W}_h h_{t-1} + \mathbf{b}$$
$$h_t = \tanh(\tilde{h}_t)$$

Keterangan:

- $x_t$ : input pada waktu t
- $h_{t-1}$ : hidden state sebelumnya
- $\mathbf{W}_x$ : bobot input  $\rightarrow$  hidden
- $\mathbf{W}_h$ : bobot hidden  $\rightarrow$  hidden
- $\mathbf{b}$ : bias
- $h_t$ : hidden state saat ini

## 3. Simple RNN Layer

SimpleRNNLayer menyusun SimpleRNNCell menjadi layer yang bisa memproses seluruh urutan waktu (sequence) secara batch.

- Proses Forward:

```
for t in time_steps:  
    x_t = inputs[:, t, :]  
    h = self.rnn_cell.forward_step(x_t, h)
```

- Alur:
  - Untuk setiap timestep t dalam sequence:

- Ambil input  $x_t$
- Hitung  $h_t$  menggunakan SimpleRNNCell
- Simpan  $h_t$  untuk keperluan training/backward
- Output:
  - Jika `return_sequences=True`: seluruh urutan hidden state (batch, time, units\_)
  - Jika `False`: hanya hidden state terakhir (batch, units)

#### 4. Bidirectional RNN Layer

BidirectionalRNNLayer menjalankan dua RNN (maju dan mundur) lalu menggabungkan outputnya.

- Proses Forward:

```
self.forward_output = self.forward_rnn.forward(inputs)
self.backward_output =
self.backward_rnn.forward(inputs)
outputs = np.concatenate([self.forward_output,
self.backward_output], axis=-1)
```

- Penjelasan:
  - Dia menggunakan dua SimpleRNNLayer, satu dengan `go_backwards=False`, satu lagi `True`
  - Hasil hidden dari dua arah digabung:

$$\text{output}_t = [h_t^{\rightarrow}; h_t^{\leftarrow}]$$

#### 5. Dense Layer

Dense Layer akhirnya mengubah output RNN menjadi prediksi akhir.

- Proses Forward:
 

```
self.linear_output = np.dot(inputs, self.weights) +
self.bias
```
- Rumus:

$$z = \mathbf{W}h + \mathbf{b}$$

Jika menggunakan softmax, aktivasi output menjadi:

$$\hat{y}_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

### 2.1.2.3. LSTM

#### 1. Embedding Layer

EmbeddingLayer akan mengubah token ID integer menjadi vektor embedding berdimensi tetap.

- Proses Forward:

```
embedded = self.weights[x]
```

- Rumus:

Jika x adalah input (token ID), dan E adalah matriks embedding (vocab\_size × embedding\_dim), maka:

$$\text{embedded} = E[x]$$

#### 2. LSTM Cell

LSTMCell akan memproses satu langkah waktu (time-step) dari urutan input dan menyimpan state tersembunyi (hidden state) dan cell state.

- Proses Forward:

```
gates = np.dot(x, self.Wi) + np.dot(h_prev, self.Ui) + self.bi
i, f, c, o = np.split(gates, 4, axis=1)
i_gate = sigmoid(i)
f_gate = sigmoid(f)
c_tilde = np.tanh(c)
o_gate = sigmoid(o)
c_next = f_gate * c_prev + i_gate * c_tilde
h_next = o_gate * np.tanh(c_next)
```

- Rumus:

$$\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}$$

Keterangan:

- $f_t$  : Forget gate
- $i_t$  : Input gate
- $o_t$  : Output gate
- $c_t$  : Cell state
- $h_t$  : Hidden state saat ini
- $h_{t-1}$  : Hidden state sebelumnya dari timestep t-1
- $x_t$  : Input pada timestep t
- $W$  : Bobot untuk input
- $U$  : Bobot untuk hidden state sebelumnya
- $b$  : Bias
- $\sigma_g$  : Fungsi sigmoid (0-1)
- $\sigma_c, \sigma_h$  : Fungsi tanh (-1 sampai 1)

### 3. Unidirectional LSTM Layer

UnidirectionalLSTMLayer menyusun LSTMCell menjadi layer yang bisa memproses seluruh urutan waktu (sequence) secara batch.

- Proses Forward:

```

for t in range(time_steps):
    h, c = self.lstm_cell.forward(x[:, t, :], h, c)
    if return_sequences:
        outputs[:, t, :] = h

```

- Alur:
  - Untuk setiap timestep t dalam sequence:
    - Ambil input  $x_t$

- Hitung  $h_t$  dan  $c_t$  menggunakan LSTMCell
- Simpan  $h_t$  dan  $c_t$  untuk keperluan backward
- Output:
  - Jika `return_sequences=True`: seluruh urutan hidden state (batch, time, hidden\_dim)
  - Jika `False`: hanya hidden state terakhir (batch, hidden\_dim)

#### 4. Bidirectional LSTM Layer

`BidirectionalLSTMLayer` menjalankan dua LSTM (maju dan mundur) lalu menggabungkan outputnya.

- Proses Forward:

```
# Forward direction
for t in range(time_steps):
    h_forward, c_forward =
self.forward_lstm.forward(x[:, t, :], h_forward,
c_forward)
    forward_outputs[:, t, :] = h_forward

# Backward direction
for t in reversed(range(time_steps)):
    h_backward, c_backward =
self.backward_lstm.forward(x[:, t, :], h_backward,
c_backward)
    backward_outputs[:, t, :] = h_backward
```

- Output:
  - Jika `return_sequences=True`: gabungan dari seluruh hidden state forward dan backward (batch, time,  $2 \times \text{hidden\_dim}$ )
  - Jika `False`: gabungan dari hidden state terakhir forward dan pertama backward (batch,  $2 \times \text{hidden\_dim}$ )



## 5. Dropout Layer

DropoutLayer menerapkan regularisasi dropout selama training untuk mencegah overfitting.

- Proses Forward:

```
if training:
    mask = np.random.binomial(1, 1 - self.rate,
size=x.shape) / (1 - self.rate)
    output = x * mask
else:
    output = x
```

- Rumus:

- Selama training:  $\text{output} = x * \text{mask}$ , dimana mask adalah tensor biner (0 atau  $1/(1-\text{rate})$ )
- Selama inferensi:  $\text{output} = x$

## 6. Dense Layer

DenseLayer mengubah output LSTM menjadi prediksi akhir.

- Proses Forward:

```
z = np.dot(x, self.W) + self.b
if self.activation == 'relu':
    output = relu(z)
elif self.activation == 'softmax':
    output = softmax(z)
else:
    output = z
```

- Rumus:

$$z = \mathbf{W}h + \mathbf{b}$$

Jika menggunakan softmax, aktivasi output menjadi:

$$\hat{y}_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Jika menggunakan relu:  $\text{output} = \max(0, z)$

Jika linear:  $\text{output} = z$

### 2.1.3. Penjelasan Backward Propagation

#### 2.1.3.1. CNN

##### 1. Conv2D Layer

Melakukan perhitungan gradien output dengan weights yang di transpose.

##### 2. MaxPooling2D Layer

Menggunakan nilai mask yang disimpan saat forward pass untuk meneruskan gradien hanya ke posisi nilai maksimum, posisi lain mendapat gradien 0.

##### 3. Flatten Layer

Dilakukan reshape gradien kembali ke dimensi input asli.

##### 4. Dense Layer

Menghitung gradien dari dense layer.

##### 5. ReLU

Gradien diteruskan utuh jika  $\text{input} > 0$ , diset 0 jika  $\text{input} \leq 0$ .

##### 6. Softmax

Menggunakan jacobian matrix karena setiap output softmax bergantung pada semua input.

##### 7. CrossEntropyLoss Function

Menghitung turunan dari cross entropy loss.

$$H = - \sum p(x) \log p(x)$$

#### 2.1.3.2. Simple RNN

##### 1. Dense Layer

Jika output layer menggunakan softmax dengan sparse categorical cross-entropy loss, maka gradiennya cukup simpel dan efisien dihitung.

- Dengan sparse categorical cross-entropy loss:

$$\mathcal{L} = -\log(\hat{y}_{\text{true}})$$

- Gradient dari loss terhadap z (output dari Dense sebelum softmax):

$$\frac{\partial \mathcal{L}}{\partial z} = \hat{y} - y_{\text{one-hot}}$$

Karena menggunakan sparse label (bukan one-hot), maka cukup:

$$\frac{\partial \mathcal{L}}{\partial z_i} = \begin{cases} \hat{y}_i - 1 & \text{jika } i = \text{label\_benar} \\ \hat{y}_i & \text{lainnya} \end{cases}$$

- Kemudian dihitung:

$$\frac{\partial \mathcal{L}}{\partial W} = x^T \cdot \delta$$

$$\frac{\partial \mathcal{L}}{\partial b} = \sum \delta$$

$$\frac{\partial \mathcal{L}}{\partial x} = \delta \cdot W^T$$

## 2. Dropout Layer

- Mengalikan gradien dengan mask:

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial \text{dropout\_output}} \times \text{mask}$$

## 3. Bidirectional RNN Layer

- Gradien dibagi dua:
  - Forward RNN menerima separuh pertama
  - Backward RNN menerima separuh kedua
- Gradien output gabungan:

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial x_{\text{fwd}}} + \frac{\partial \mathcal{L}}{\partial x_{\text{bwd}}}$$

#### 4. Simple RNN Layer ((Backward Pass through Time)

Proses backward mengikuti Backpropagation Through Time (BPTT).  
Setiap waktu  $t$  dihitung secara mundur.

- Gradien dihitung untuk setiap timestep:

Untuk  $h_t = \tanh(W_{in}x_t + W_{rec}h_{t-1} + b)$ , turunan dari tanh:

$$\frac{d \tanh(z)}{dz} = 1 - \tanh^2(z)$$

- Rumus utama backward step:

1. Gradien terhadap aktivasi sebelum tanh:

$$\delta_t = \frac{\partial \mathcal{L}}{\partial h_t} \cdot (1 - \tanh^2(z_t))$$

2. Gradien terhadap input, hidden sebelumnya, dan parameter:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_{in}} &= x_t^T \cdot \delta_t \\ \frac{\partial \mathcal{L}}{\partial W_{rec}} &= h_{t-1}^T \cdot \delta_t \\ \frac{\partial \mathcal{L}}{\partial b} &= \delta_t \\ \frac{\partial \mathcal{L}}{\partial x_t} &= \delta_t \cdot W_{in}^T \\ \frac{\partial \mathcal{L}}{\partial h_{t-1}} &= \delta_t \cdot W_{rec}^T\end{aligned}$$

3. Ini dilakukan dari waktu  $T \rightarrow 0$

4. Gradien disimpan dan diakumulasi untuk semua timestep

#### 5. Embedding Layer

- Gradien hanya muncul untuk token yang digunakan:

$$\frac{\partial \mathcal{L}}{\partial \text{embedding}[i]} + = \text{grad\_output}[t]$$

### 2.1.3.3. LSTM

Proses backward mengikuti Backpropagation Through Time (BPTT). Setiap waktu  $t$  dihitung secara mundur untuk memperbaiki parameter model berdasarkan gradien error.

#### 1. LSTMCell

LSTM Cell adalah unit dasar yang memproses satu timestep dengan empat gate (input, forget, cell, output). Propagasi mundur menghitung gradien untuk setiap gate dan parameter berdasarkan rantai turunan dari loss function.

Gradien terhadap output gate:

$$\frac{\partial \mathcal{L}}{\partial o_t} = \frac{\partial \mathcal{L}}{\partial h_t} \cdot \tanh(c_t)$$

Gradient terhadap cell state:

$$\frac{\partial \mathcal{L}}{\partial c_t} = \frac{\partial \mathcal{L}}{\partial c_{t+1}} + \frac{\partial \mathcal{L}}{\partial h_t} \cdot o_t \cdot (1 - \tanh^2(c_t))$$

Gradient terhadap forget gate:

$$\frac{\partial \mathcal{L}}{\partial f_t} = \frac{\partial \mathcal{L}}{\partial c_t} \cdot c_{t-1}$$

Gradient terhadap input gate:

$$\frac{\partial \mathcal{L}}{\partial i_t} = \frac{\partial \mathcal{L}}{\partial c_t} \cdot \tilde{c}_t$$

Gradient terhadap cell input:

$$\frac{\partial \mathcal{L}}{\partial \tilde{c}_t} = \frac{\partial \mathcal{L}}{\partial c_t} \cdot i_t$$

Gradient terhadap cell state sebelumnya:

$$\frac{\partial \mathcal{L}}{\partial c_{t-1}} = \frac{\partial \mathcal{L}}{\partial c_t} \cdot f_t$$

Gradient untuk gate sebelum aktivasi:

$$\frac{\partial \mathcal{L}}{\partial z_i} = \frac{\partial \mathcal{L}}{\partial i_t} \cdot i_t \cdot (1 - i_t)$$

$$\frac{\partial \mathcal{L}}{\partial z_f} = \frac{\partial \mathcal{L}}{\partial f_t} \cdot f_t \cdot (1 - f_t)$$

$$\frac{\partial \mathcal{L}}{\partial z_c} = \frac{\partial \mathcal{L}}{\partial \tilde{c}_t} \cdot (1 - \tilde{c}_t^2)$$

$$\frac{\partial \mathcal{L}}{\partial z_o} = \frac{\partial \mathcal{L}}{\partial o_t} \cdot o_t \cdot (1 - o_t)$$

Gradient terhadap parameter:

$$\frac{\partial \mathcal{L}}{\partial W_i} = x_t^T \cdot \delta_t$$

$$\frac{\partial \mathcal{L}}{\partial U_i} = h_{t-1}^T \cdot \delta_t$$

$$\frac{\partial \mathcal{L}}{\partial b_i} = \sum \delta_t$$

dimana

$$\delta_t = \left[ \frac{\partial \mathcal{L}}{\partial z_i}, \frac{\partial \mathcal{L}}{\partial z_f}, \frac{\partial \mathcal{L}}{\partial z_c}, \frac{\partial \mathcal{L}}{\partial z_o} \right]$$

## 2. UnidirectionalLSTM

UnidirectionalLSTM menjalankan LSTM Cell untuk serangkaian timestep secara berurutan. Propagasi mundur berjalan dari timestep

terakhir ke awal, mengumpulkan dan meneruskan gradien sesuai dengan aliran informasi melalui waktu.

Proses backward mengiterasi semua timestep secara terbalik dengan tahap:

- a. Untuk setiap timestep  $t$  dari  $T$  hingga  $0$ :

$$\frac{\partial \mathcal{L}}{\partial h_t} = \begin{cases} \frac{\partial \mathcal{L}}{\partial y_t} + \frac{\partial \mathcal{L}}{\partial h_{t+1}} & \text{jika return\_sequences} \\ \frac{\partial \mathcal{L}}{\partial y_T} & \text{jika } t = T \text{ dan tidak return\_sequences} \\ 0 & \text{lainnya} \end{cases}$$

- b. Hitung gradient menggunakan LSTM cell
- c. Akumulasi gradient:

$$\frac{\partial \mathcal{L}}{\partial W_i} = \sum_{t=0}^T \frac{\partial \mathcal{L}}{\partial W_i^t}$$

$$\frac{\partial \mathcal{L}}{\partial U_i} = \sum_{t=0}^T \frac{\partial \mathcal{L}}{\partial U_i^t}$$

$$\frac{\partial \mathcal{L}}{\partial b_i} = \sum_{t=0}^T \frac{\partial \mathcal{L}}{\partial b_i^t}$$

### 3. BidirectionalLSTMLayer

BidirectionalLSTM menggunakan dua LSTM yang berjalan dalam arah berlawanan untuk menangkap konteks dari masa lalu dan masa depan. Propagasi mundur harus menangani kedua arah ini secara terpisah, kemudian menggabungkan gradien mereka.

- Gradient dibagi dua:
  - Forward LSTM menerima separuh pertama
  - Backward LSTM menerima separuh kedua
- Gradient output gabungan:

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial x_{fwd}} + \frac{\partial \mathcal{L}}{\partial x_{bwd}}$$

#### 4. Embedding Layer

Embedding Layer mengubah indeks token menjadi vektor. Selama propagasi mundur, hanya embedding untuk token yang muncul dalam batch yang diupdate, yang membuat proses menjadi lebih efisien karena kebanyakan token tidak digunakan dalam setiap batch.

Gradient hanya muncul untuk token yang digunakan:

$$\frac{\partial \mathcal{L}}{\partial embedding[i]} = \sum_{(batch, pos) \text{ dimana } token=i} \frac{\partial \mathcal{L}}{\partial output[batch, pos]}$$

#### 5. Dropout Layer

Dropout Layer menonaktifkan neuron secara acak selama training untuk mencegah overfitting. Selama propagasi mundur, gradien hanya diteruskan melalui neuron yang aktif selama forward pass, dengan menggunakan mask yang sama. Mengalikan gradien dengan mask:

$$\frac{\partial \mathcal{L}}{\partial x} = \frac{\partial \mathcal{L}}{\partial dropout\_output} \times mask$$

## 2.2. Hasil pengujian

### 2.2.1. CNN

Ukuran filter default = 3x3

Pooling layer default = Max Pooling

Activation function default = ReLU

Banyak filter default = [32, 64]

Arsitektur model secara general:

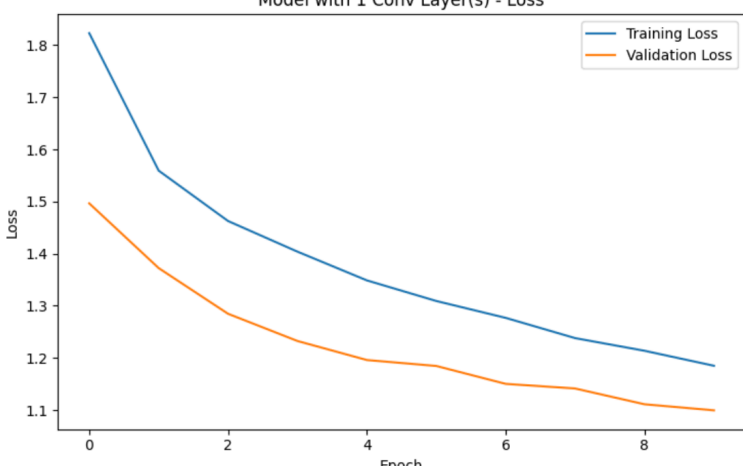
- Conv2D layer
- ReLU layer
- MaxPooling/AvgPooling layer
- Conv2D layer



- ReLU layer
- MaxPooling/AvgPooling layer
- Flatten layer
- Dense layer(128)
- Dropout layer(0.5)
- Dense layer(10)
- Softmax layer

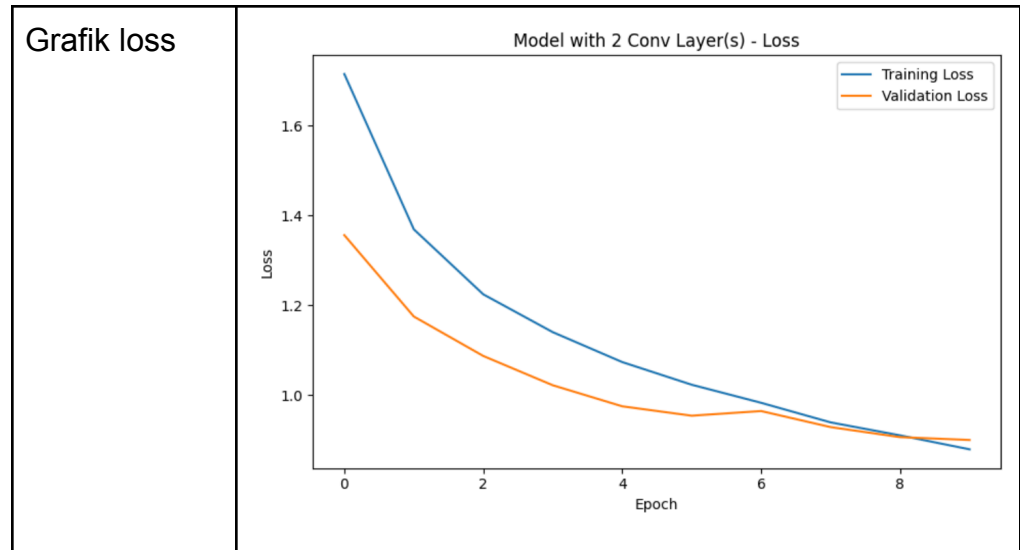
### 2.2.1.1. Pengaruh jumlah layer konvolusi

a. Layer konvolusi = 3 buah

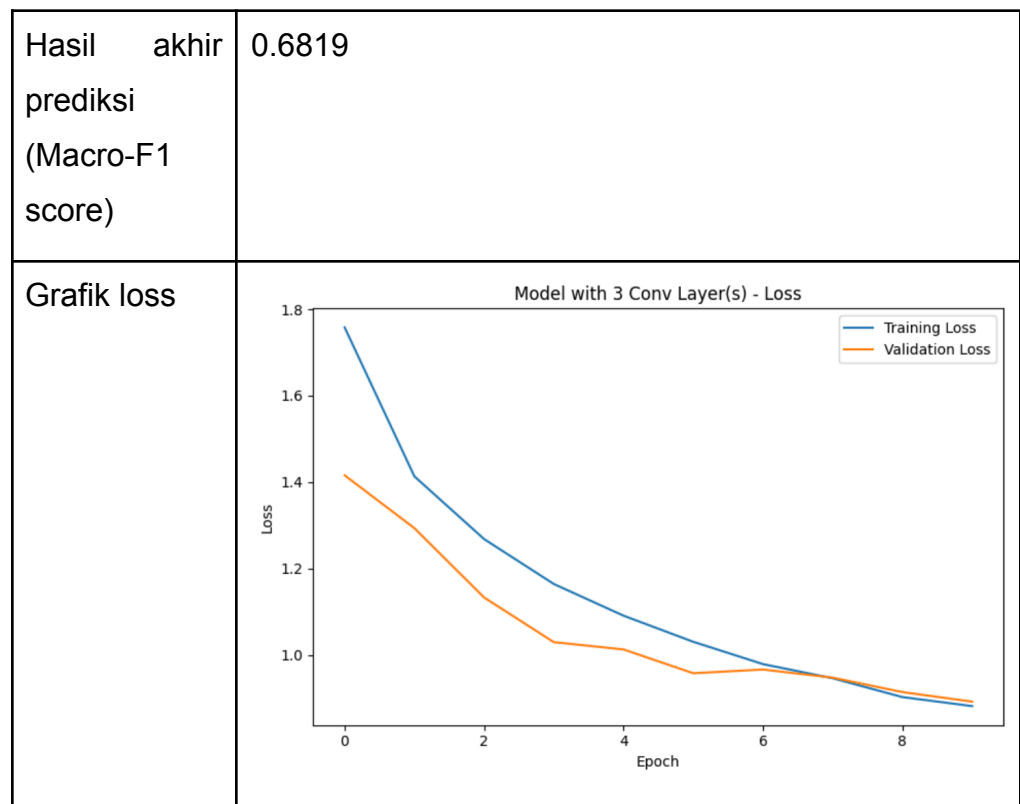
Hasil akhir prediksi (Macro-F1 score)	0.6067																																	
Grafik loss	<div><p>Model with 1 Conv Layer(s) - Loss</p><p>The graph displays the loss curves for a model with 1 convolutional layer. The x-axis represents the number of epochs (0 to 9), and the y-axis represents the loss (1.1 to 1.8). The training loss (blue line) starts at approximately 1.82 at epoch 0 and decreases steadily to about 1.18 by epoch 9. The validation loss (orange line) starts at approximately 1.50 at epoch 0 and decreases to about 1.10 by epoch 9. Both losses show a consistent downward trend over the 9 epochs.</p><table><thead><tr><th>Epoch</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.82</td><td>1.50</td></tr><tr><td>1</td><td>1.58</td><td>1.38</td></tr><tr><td>2</td><td>1.48</td><td>1.30</td></tr><tr><td>3</td><td>1.42</td><td>1.25</td></tr><tr><td>4</td><td>1.35</td><td>1.20</td></tr><tr><td>5</td><td>1.30</td><td>1.18</td></tr><tr><td>6</td><td>1.28</td><td>1.15</td></tr><tr><td>7</td><td>1.25</td><td>1.14</td></tr><tr><td>8</td><td>1.22</td><td>1.12</td></tr><tr><td>9</td><td>1.18</td><td>1.10</td></tr></tbody></table></div>	Epoch	Training Loss	Validation Loss	0	1.82	1.50	1	1.58	1.38	2	1.48	1.30	3	1.42	1.25	4	1.35	1.20	5	1.30	1.18	6	1.28	1.15	7	1.25	1.14	8	1.22	1.12	9	1.18	1.10
Epoch	Training Loss	Validation Loss																																
0	1.82	1.50																																
1	1.58	1.38																																
2	1.48	1.30																																
3	1.42	1.25																																
4	1.35	1.20																																
5	1.30	1.18																																
6	1.28	1.15																																
7	1.25	1.14																																
8	1.22	1.12																																
9	1.18	1.10																																

b. Layer konvolusi = 2 buah

Hasil akhir prediksi (Macro-F1 score)	0.6794
---------------------------------------	--------



c. Layer konvolusi = 3 buah



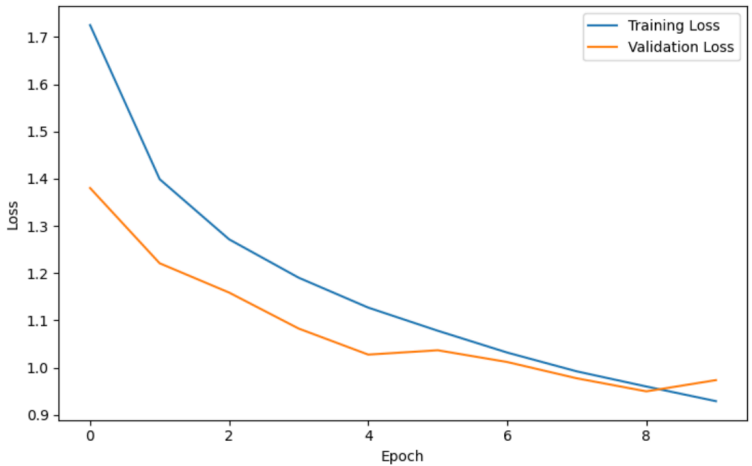
d. Kesimpulan

Semakin banyak jumlah layer konvolusi dapat menghasilkan ekstraksi fitur yang lebih baik (*low - high level feature*) juga meningkatkan performa model. Namun, layer yang terlalu dalam dapat menyebabkan

*overfitting* maupun *vanishing/exploding gradient problem* tanpa melakukan *proper regularization* ataupun *residual connection*.

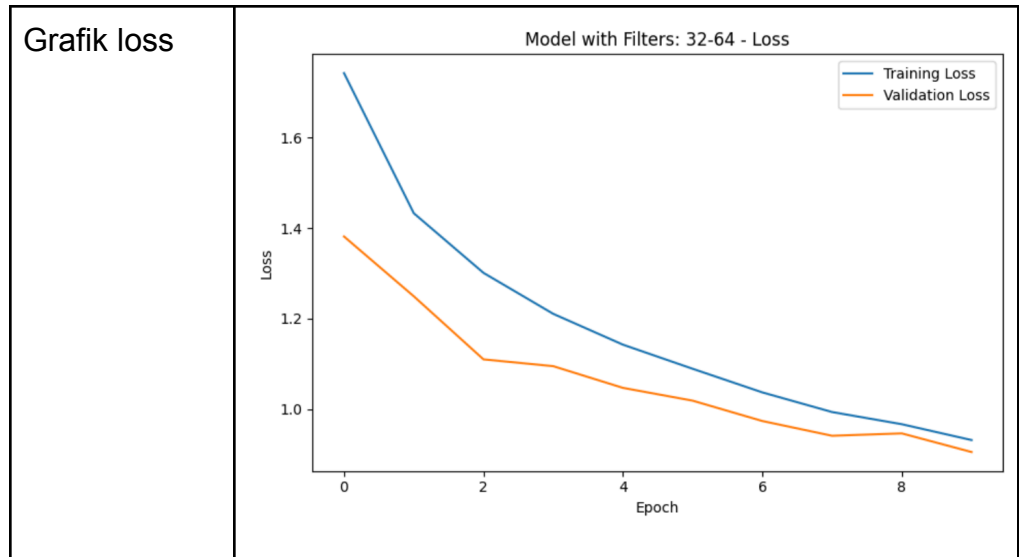
### 2.2.1.2. Pengaruh banyak filter per layer konvolusi

a. Banyak filter = [16, 32]

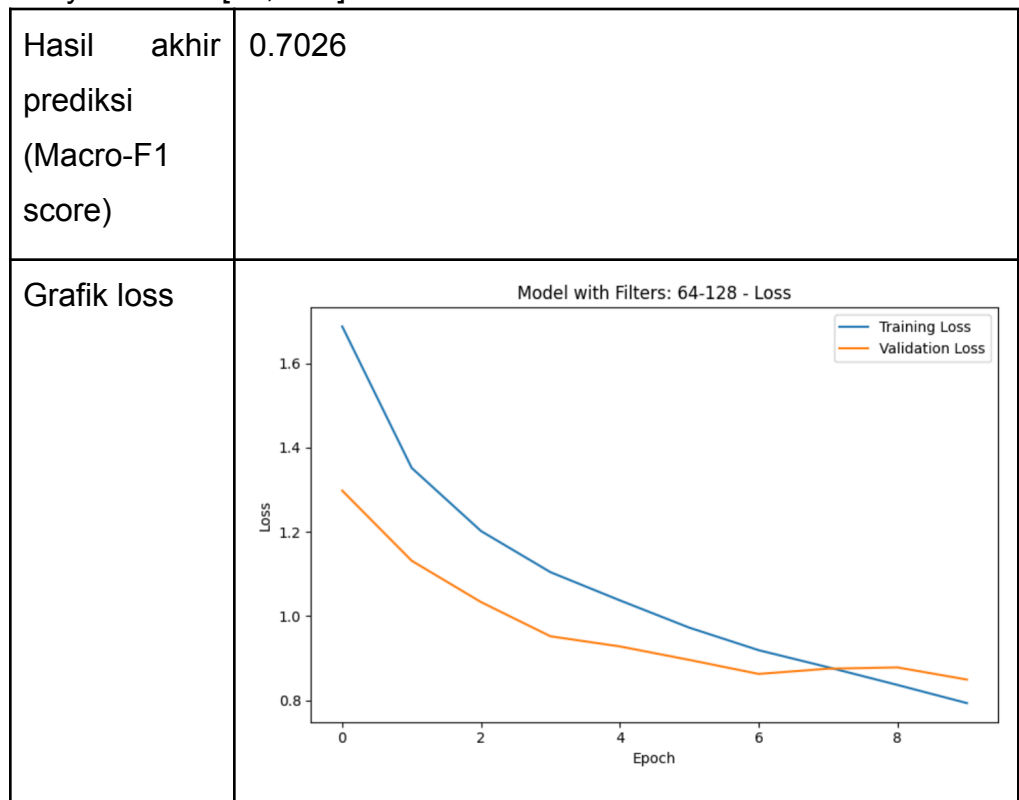
Hasil akhir prediksi (Macro-F1 score)	0.6677																																	
Grafik loss	<div><p>Model with Filters: 16-32 - Loss</p><table border="1"><thead><tr><th>Epoch</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.72</td><td>1.38</td></tr><tr><td>1</td><td>1.40</td><td>1.22</td></tr><tr><td>2</td><td>1.28</td><td>1.16</td></tr><tr><td>3</td><td>1.18</td><td>1.08</td></tr><tr><td>4</td><td>1.12</td><td>1.03</td></tr><tr><td>5</td><td>1.08</td><td>1.04</td></tr><tr><td>6</td><td>1.04</td><td>1.02</td></tr><tr><td>7</td><td>1.00</td><td>0.99</td></tr><tr><td>8</td><td>0.96</td><td>0.96</td></tr><tr><td>9</td><td>0.94</td><td>0.98</td></tr></tbody></table></div>	Epoch	Training Loss	Validation Loss	0	1.72	1.38	1	1.40	1.22	2	1.28	1.16	3	1.18	1.08	4	1.12	1.03	5	1.08	1.04	6	1.04	1.02	7	1.00	0.99	8	0.96	0.96	9	0.94	0.98
Epoch	Training Loss	Validation Loss																																
0	1.72	1.38																																
1	1.40	1.22																																
2	1.28	1.16																																
3	1.18	1.08																																
4	1.12	1.03																																
5	1.08	1.04																																
6	1.04	1.02																																
7	1.00	0.99																																
8	0.96	0.96																																
9	0.94	0.98																																

b. Banyak filter = [32, 64]

Hasil akhir prediksi (Macro-F1 score)	0.6825
---------------------------------------	--------



c. Banyak filter = [64, 128]



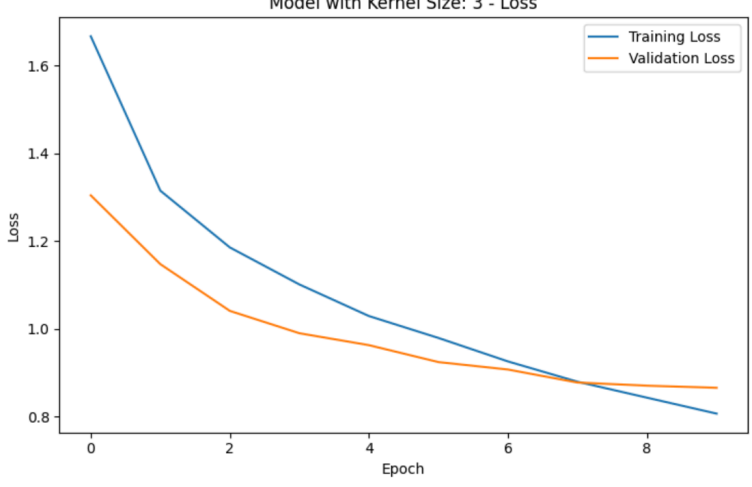
d. Kesimpulan

Meningkatnya jumlah filter selaras dengan peningkatan performa model. Hal ini disebabkan karena semakin banyak filter yang digunakan maka akan semakin banyak pula fitur yang dapat

diekstraksi. Namun semakin banyak filters yang digunakan dapat meningkatkan *cost* dari komputasi.

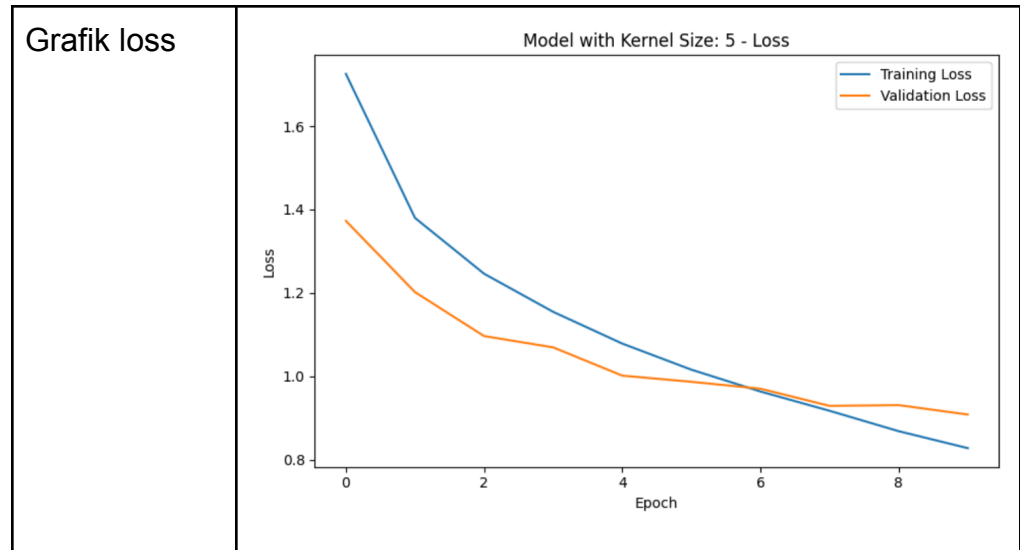
### 2.2.1.3. Pengaruh ukuran filter per layer konvolusi

#### a. Ukuran filter = 3x3

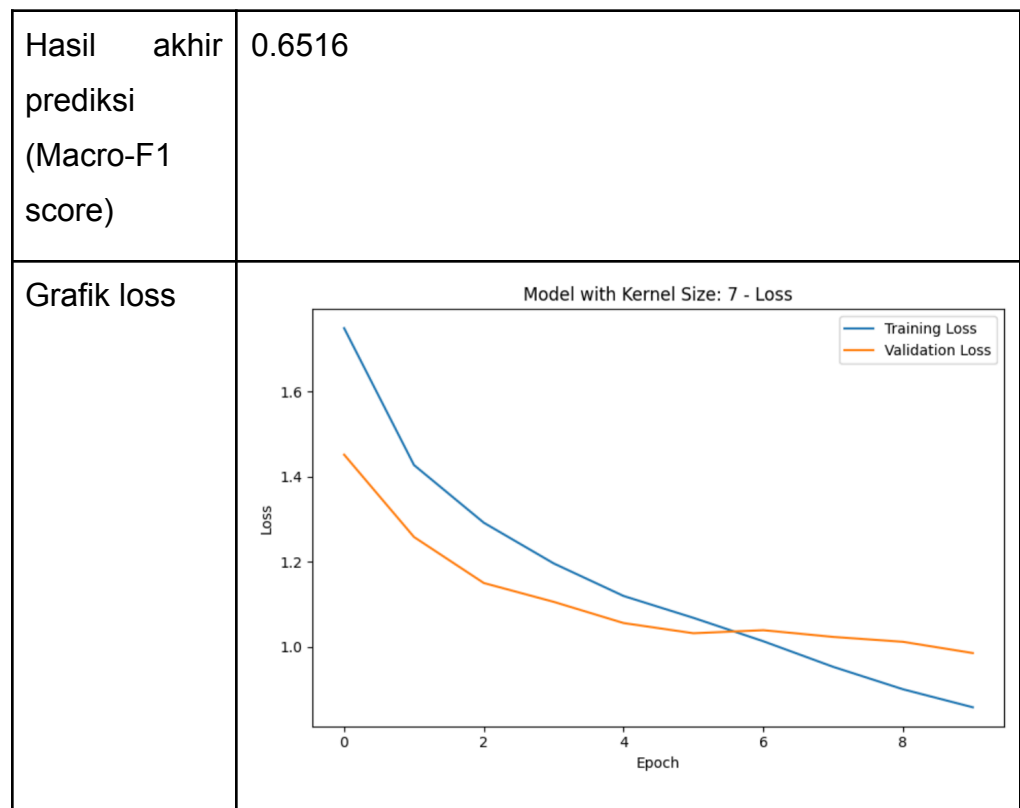
Hasil akhir prediksi (Macro-F1 score)	0.6939																																	
Grafik loss	<div><p>Model with Kernel Size: 3 - Loss</p><table border="1"><thead><tr><th>Epoch</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.65</td><td>1.30</td></tr><tr><td>1</td><td>1.31</td><td>1.15</td></tr><tr><td>2</td><td>1.18</td><td>1.04</td></tr><tr><td>3</td><td>1.10</td><td>0.99</td></tr><tr><td>4</td><td>1.02</td><td>0.96</td></tr><tr><td>5</td><td>0.95</td><td>0.92</td></tr><tr><td>6</td><td>0.90</td><td>0.90</td></tr><tr><td>7</td><td>0.85</td><td>0.87</td></tr><tr><td>8</td><td>0.82</td><td>0.87</td></tr><tr><td>9</td><td>0.81</td><td>0.87</td></tr></tbody></table></div>	Epoch	Training Loss	Validation Loss	0	1.65	1.30	1	1.31	1.15	2	1.18	1.04	3	1.10	0.99	4	1.02	0.96	5	0.95	0.92	6	0.90	0.90	7	0.85	0.87	8	0.82	0.87	9	0.81	0.87
Epoch	Training Loss	Validation Loss																																
0	1.65	1.30																																
1	1.31	1.15																																
2	1.18	1.04																																
3	1.10	0.99																																
4	1.02	0.96																																
5	0.95	0.92																																
6	0.90	0.90																																
7	0.85	0.87																																
8	0.82	0.87																																
9	0.81	0.87																																

#### b. Ukuran filter = 5x5

Hasil akhir prediksi (Macro-F1 score)	0.6765
---------------------------------------	--------



c. Ukuran filter = 7x7



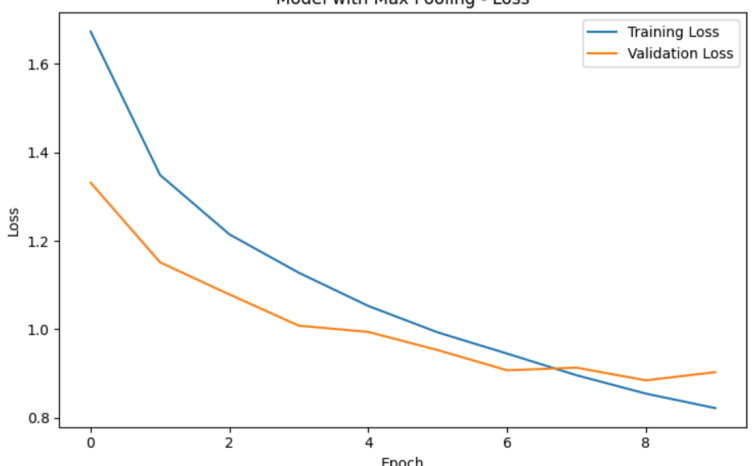
d. Kesimpulan

Filter yang lebih kecil (3x3) lebih bagus untuk menangkap fine-grained details yang terdapat pada gambar, sedangkan filter yang berukuran lebih besar berguna untuk menangkap konteks informasi secara global

namun dapat kehilangan presisi akan fitur yang kecil. Dan pada kasus ini karena konteks image yang dimasukan berukuran kecil (32, 32, 3) maka filter yang lebih kecil akan lebih unggul untuk menangkap detail fitur yang kecil juga.

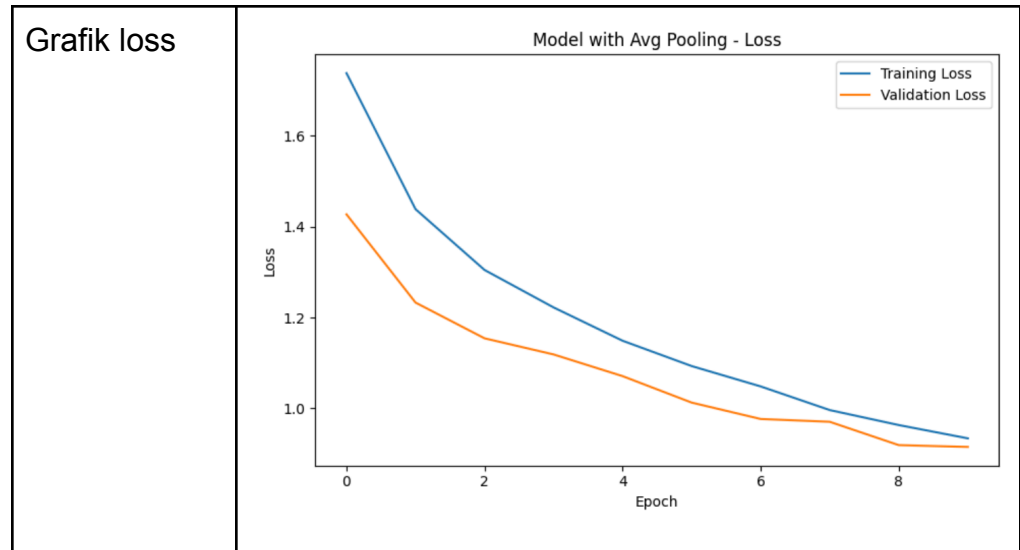
#### 2.2.1.4. Pengaruh jenis pooling layer

a. Pooling = max

Hasil akhir prediksi (Macro-F1 score)	0.6891																																	
Grafik loss	<div><p>Model with Max Pooling - Loss</p><p>The graph displays the training and validation loss for a model using Max Pooling over 9 epochs. The training loss (blue line) starts at approximately 1.65 and decreases steadily to about 0.82. The validation loss (orange line) starts at approximately 1.35 and decreases to about 0.90, showing a slight increase after epoch 8.</p><table><thead><tr><th>Epoch</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.65</td><td>1.35</td></tr><tr><td>1</td><td>1.35</td><td>1.15</td></tr><tr><td>2</td><td>1.22</td><td>1.08</td></tr><tr><td>3</td><td>1.12</td><td>1.02</td></tr><tr><td>4</td><td>1.05</td><td>1.00</td></tr><tr><td>5</td><td>0.98</td><td>0.95</td></tr><tr><td>6</td><td>0.92</td><td>0.90</td></tr><tr><td>7</td><td>0.88</td><td>0.92</td></tr><tr><td>8</td><td>0.85</td><td>0.88</td></tr><tr><td>9</td><td>0.82</td><td>0.90</td></tr></tbody></table></div>	Epoch	Training Loss	Validation Loss	0	1.65	1.35	1	1.35	1.15	2	1.22	1.08	3	1.12	1.02	4	1.05	1.00	5	0.98	0.95	6	0.92	0.90	7	0.88	0.92	8	0.85	0.88	9	0.82	0.90
Epoch	Training Loss	Validation Loss																																
0	1.65	1.35																																
1	1.35	1.15																																
2	1.22	1.08																																
3	1.12	1.02																																
4	1.05	1.00																																
5	0.98	0.95																																
6	0.92	0.90																																
7	0.88	0.92																																
8	0.85	0.88																																
9	0.82	0.90																																

b. Pooling = average

Hasil akhir prediksi (Macro-F1 score)	0.6744
---------------------------------------	--------



### c. Kesimpulan

Max pooling cenderung mengambil perwakilan fitur yang paling signifikan saja, sedangkan avg pooling mempertahankan semua konteks fitur yang ada di area tersebut. Dalam kasus ini, max pooling memiliki performa yang lebih bagus.

#### 2.2.1.5. Perbandingan Keras Model vs Scratch Model

	Keras	Scratch	Kesimpulan
<b>Forward Propagation</b>	Macro-F1: 0.7004	Macro-F1: 0.7004	Kecocokan: 100%
<b>Backward Propagation</b>	-	Terjadi penurunan nilai loss dari 295 → 286 → 276	Backpropagation berhasil dilakukan

#### 2.2.2. Simple RNN

Parameter *preprocessing* yang digunakan dalam pengujian adalah sebagai berikut:

- **MAX\_FEATURES** = 3000. Parameter ini menentukan jumlah maksimum fitur/kata unik yang digunakan dalam pemodelan.
- **MAX\_LENGTH** = 50. Parameter ini menetapkan panjang maksimum setiap input teks dalam bentuk urutan token.



- EMBEDDING\_DIM = 32 → Parameter ini menentukan jumlah dimensi dalam representasi vektor embedding untuk setiap kata.

Konfigurasi parameter ini dipilih dengan mempertimbangkan ukuran dataset yang cukup kecil.

Berikut adalah arsitektur model secara general:

1. Embedding layer
2. RNN Layer (SimpleRNN, bisa bidirectional)
3. Dropout layer 1
4. Dense layer 1 (fungsi aktivasi = ReLU)
5. Dropout layer 2
6. Dense layer 2 (fungsi aktivasi = Softmax)

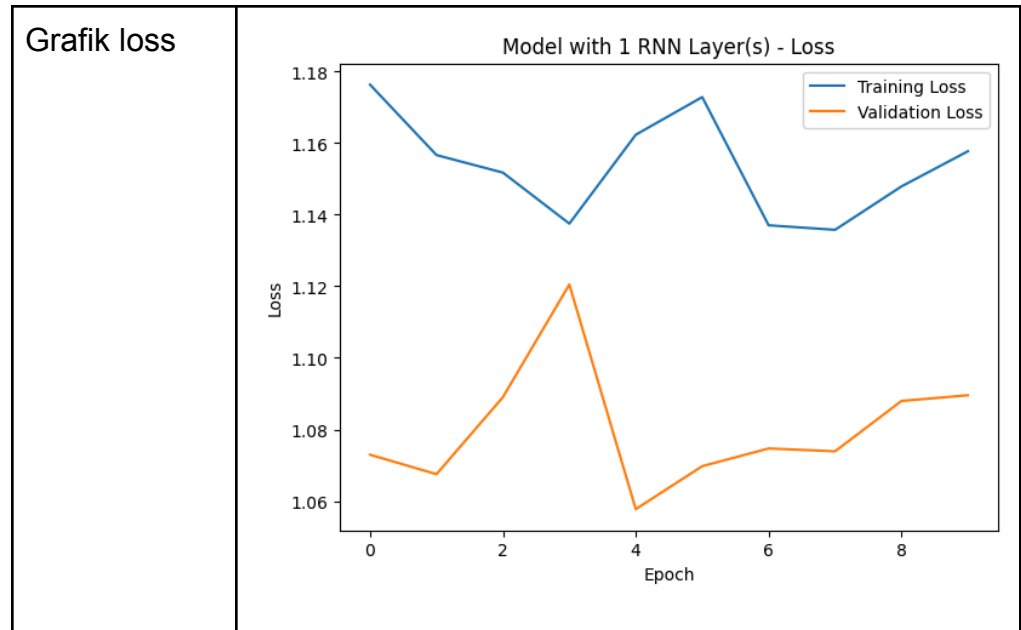
Kemudian, *loss function* yang digunakan adalah Sparse Categorical Cross Entropy dan optimizer yang digunakan adalah Adam.

Epoch yang digunakan selama pelatihan adalah 10.

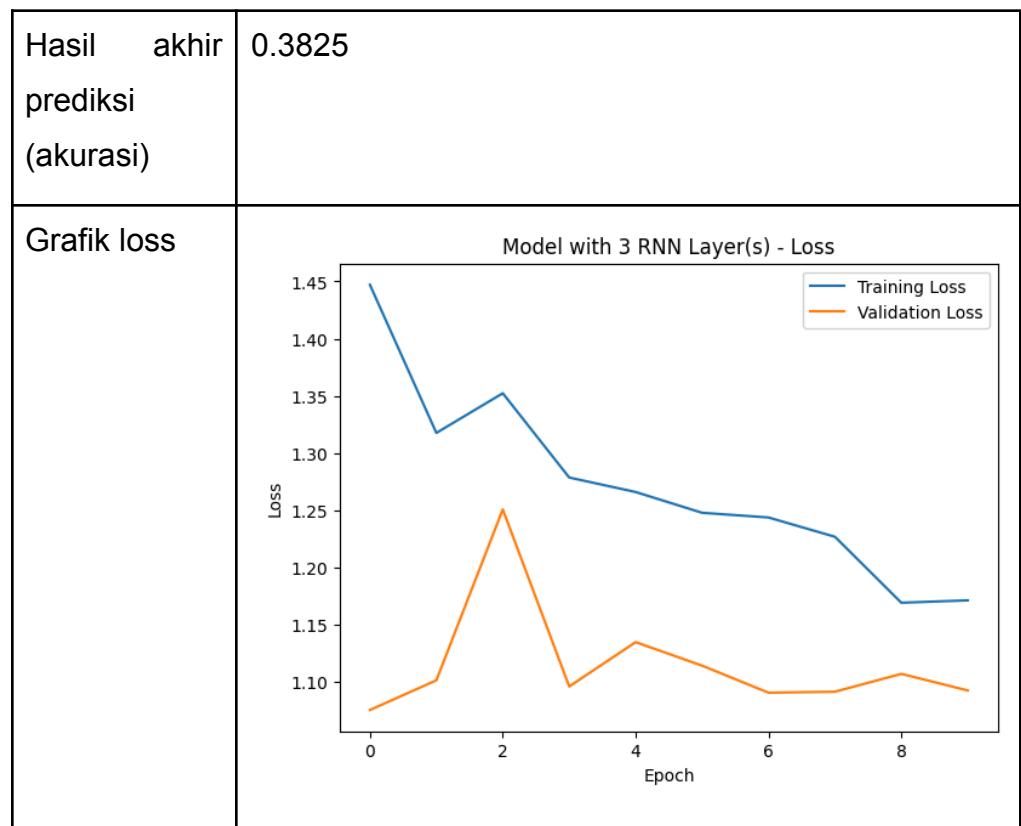
#### 2.2.2.1. Pengaruh jumlah layer RNN

- a. Layer RNN = 1 buah

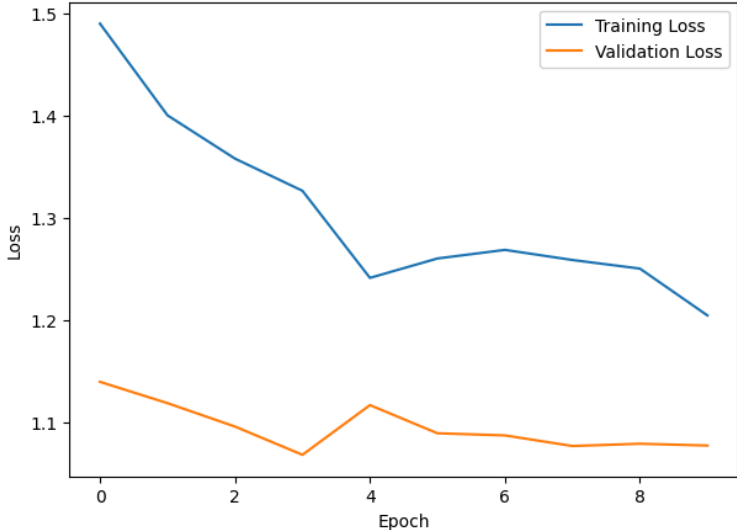
Hasil akhir prediksi (akurasi)	0.3775
--------------------------------	--------



b. Layer RNN = 3 buah



c. Layer RNN = 5 buah

Hasil akhir prediksi (akurasi)	0.3775																																	
Grafik loss	<div><p>Model with 5 RNN Layer(s) - Loss</p><table border="1"><thead><tr><th>Epoch</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.48</td><td>1.14</td></tr><tr><td>1</td><td>1.40</td><td>1.12</td></tr><tr><td>2</td><td>1.36</td><td>1.10</td></tr><tr><td>3</td><td>1.33</td><td>1.07</td></tr><tr><td>4</td><td>1.24</td><td>1.12</td></tr><tr><td>5</td><td>1.26</td><td>1.09</td></tr><tr><td>6</td><td>1.27</td><td>1.09</td></tr><tr><td>7</td><td>1.26</td><td>1.08</td></tr><tr><td>8</td><td>1.25</td><td>1.08</td></tr><tr><td>9</td><td>1.20</td><td>1.08</td></tr></tbody></table></div>	Epoch	Training Loss	Validation Loss	0	1.48	1.14	1	1.40	1.12	2	1.36	1.10	3	1.33	1.07	4	1.24	1.12	5	1.26	1.09	6	1.27	1.09	7	1.26	1.08	8	1.25	1.08	9	1.20	1.08
Epoch	Training Loss	Validation Loss																																
0	1.48	1.14																																
1	1.40	1.12																																
2	1.36	1.10																																
3	1.33	1.07																																
4	1.24	1.12																																
5	1.26	1.09																																
6	1.27	1.09																																
7	1.26	1.08																																
8	1.25	1.08																																
9	1.20	1.08																																

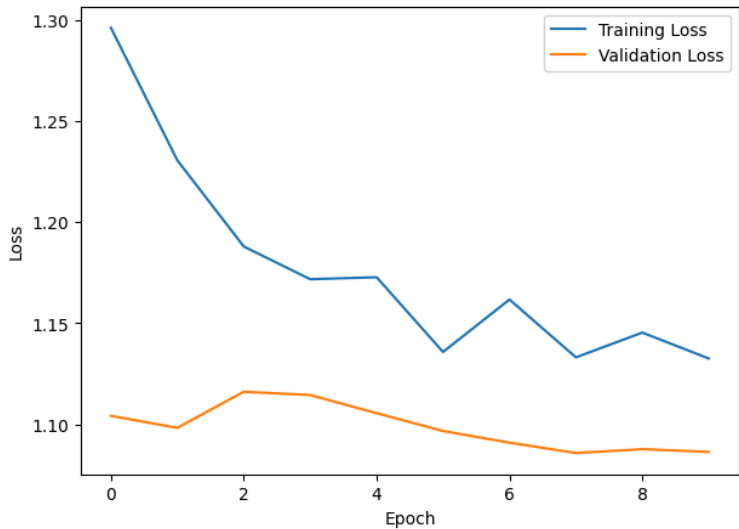
#### d. Kesimpulan

Berdasarkan analisis ketiga variasi jumlah layer RNN yang diuji, dapat disimpulkan bahwa penambahan layer RNN tidak selalu meningkatkan kinerja model secara konsisten. Model dengan 1 layer RNN menunjukkan akurasi 0.3775 dengan pola loss yang relatif stabil namun masih berfluktuasi, di mana training loss dan validation loss bergerak dalam rentang yang cukup dekat (1.06 - 1.18). Model dengan 3 layer RNN memberikan performa terbaik dengan akurasi 0.3825, menunjukkan peningkatan kemampuan model untuk menangkap pola yang lebih kompleks dalam data. Grafik loss pada model 3 layer menunjukkan penurunan yang lebih konsisten pada training loss dengan validation loss yang relatif stabil, mengindikasikan bahwa model berhasil belajar dengan baik tanpa mengalami overfitting yang signifikan. Sementara itu, model dengan 5 layer RNN kembali menghasilkan akurasi yang sama dengan model 1 layer (0.3775),

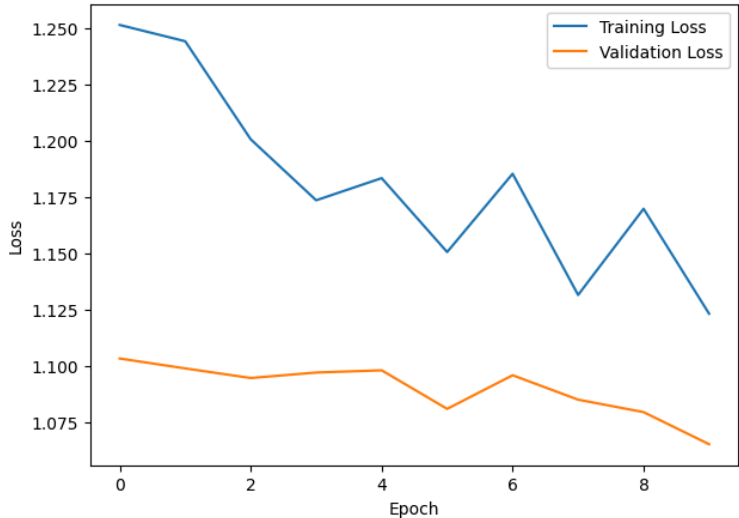
namun menunjukkan tren yang menarik pada grafik loss dimana training loss terus menurun secara konsisten sementara validation loss cenderung stabil pada level yang lebih rendah. Hal ini mengindikasikan bahwa penambahan layer yang berlebihan dapat menyebabkan model menjadi terlalu kompleks sehingga tidak memberikan peningkatan performa yang signifikan, bahkan berpotensi mengarah pada overfitting. Secara keseluruhan, model dengan 3 layer RNN memberikan keseimbangan terbaik antara kompleksitas dan performa, menunjukkan bahwa ada titik optimal dalam penambahan layer di mana model dapat belajar pola yang lebih kompleks tanpa kehilangan kemampuan generalisasi.

#### 2.2.2.2. Pengaruh banyak cell RNN per layer

a. Cell units = 32 units

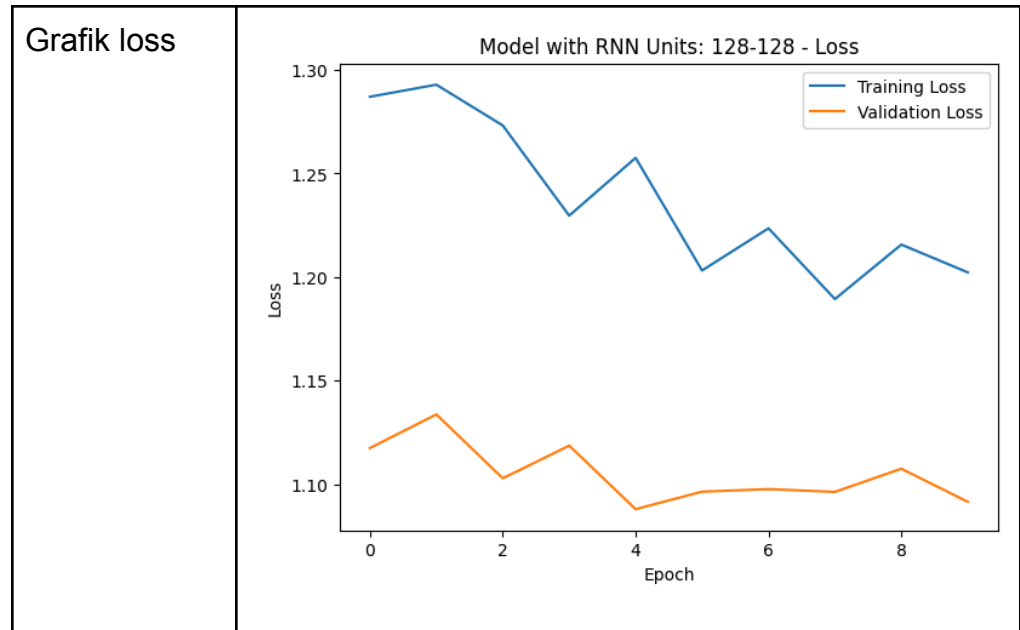
Hasil akhir prediksi (akurasi)	0.3925																																	
Grafik loss	<div><p>Model with RNN Units: 32-32 - Loss</p><p>The graph displays the loss function for a model with 32-32 RNN units over 9 epochs. The Training Loss (blue line) begins at approximately 1.29 at epoch 0 and shows a general downward trend, ending at about 1.13 at epoch 9. The Validation Loss (orange line) starts at approximately 1.105 at epoch 0 and also shows a downward trend, ending at about 1.085 at epoch 9. Both losses exhibit some minor fluctuations throughout the training process.</p><table><caption>Approximate Loss Values from Graph</caption><thead><tr><th>Epoch</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.29</td><td>1.105</td></tr><tr><td>1</td><td>1.23</td><td>1.10</td></tr><tr><td>2</td><td>1.19</td><td>1.115</td></tr><tr><td>3</td><td>1.175</td><td>1.115</td></tr><tr><td>4</td><td>1.175</td><td>1.11</td></tr><tr><td>5</td><td>1.14</td><td>1.10</td></tr><tr><td>6</td><td>1.16</td><td>1.095</td></tr><tr><td>7</td><td>1.135</td><td>1.085</td></tr><tr><td>8</td><td>1.145</td><td>1.085</td></tr><tr><td>9</td><td>1.13</td><td>1.085</td></tr></tbody></table></div>	Epoch	Training Loss	Validation Loss	0	1.29	1.105	1	1.23	1.10	2	1.19	1.115	3	1.175	1.115	4	1.175	1.11	5	1.14	1.10	6	1.16	1.095	7	1.135	1.085	8	1.145	1.085	9	1.13	1.085
Epoch	Training Loss	Validation Loss																																
0	1.29	1.105																																
1	1.23	1.10																																
2	1.19	1.115																																
3	1.175	1.115																																
4	1.175	1.11																																
5	1.14	1.10																																
6	1.16	1.095																																
7	1.135	1.085																																
8	1.145	1.085																																
9	1.13	1.085																																

b. Cell units = 64 units

Hasil akhir prediksi (akurasi)	0.4050																																	
Grafik loss	<div>Model with RNN Units: 64-64 - Loss</div>  <p>The graph displays two data series: Training Loss (blue line) and Validation Loss (orange line) over 10 epochs. The Training Loss starts at 1.250 at epoch 0, decreases to 1.220 at epoch 1, then to 1.200 at epoch 2. It fluctuates between 1.175 and 1.190 for epochs 3 through 8, and finally drops to 1.125 at epoch 9. The Validation Loss starts at 1.105 at epoch 0, decreases to 1.095 at epoch 1, and remains relatively stable around 1.095-1.098 for epochs 2 through 4. It then drops to 1.080 at epoch 5, rises slightly to 1.095 at epoch 6, and continues to decrease to 1.065 by epoch 9.</p> <table><thead><tr><th>Epoch</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.250</td><td>1.105</td></tr><tr><td>1</td><td>1.220</td><td>1.095</td></tr><tr><td>2</td><td>1.200</td><td>1.095</td></tr><tr><td>3</td><td>1.175</td><td>1.098</td></tr><tr><td>4</td><td>1.185</td><td>1.098</td></tr><tr><td>5</td><td>1.150</td><td>1.080</td></tr><tr><td>6</td><td>1.185</td><td>1.095</td></tr><tr><td>7</td><td>1.130</td><td>1.085</td></tr><tr><td>8</td><td>1.170</td><td>1.080</td></tr><tr><td>9</td><td>1.125</td><td>1.065</td></tr></tbody></table>	Epoch	Training Loss	Validation Loss	0	1.250	1.105	1	1.220	1.095	2	1.200	1.095	3	1.175	1.098	4	1.185	1.098	5	1.150	1.080	6	1.185	1.095	7	1.130	1.085	8	1.170	1.080	9	1.125	1.065
Epoch	Training Loss	Validation Loss																																
0	1.250	1.105																																
1	1.220	1.095																																
2	1.200	1.095																																
3	1.175	1.098																																
4	1.185	1.098																																
5	1.150	1.080																																
6	1.185	1.095																																
7	1.130	1.085																																
8	1.170	1.080																																
9	1.125	1.065																																

c. Cell units = 128 units

Hasil akhir prediksi (akurasi)	0.4025
--------------------------------	--------



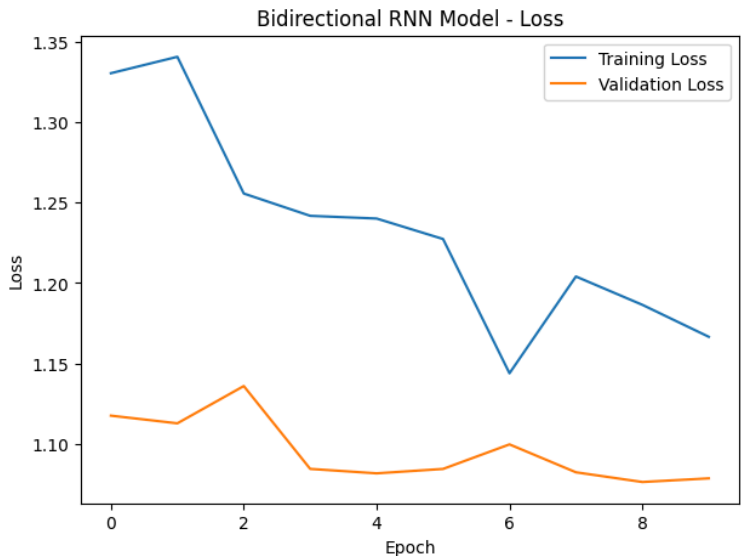
#### d. Kesimpulan

Berdasarkan analisis variasi jumlah cell units RNN dengan arsitektur 2 layer RNN, dapat disimpulkan bahwa peningkatan jumlah units memberikan dampak positif terhadap kinerja model hingga titik optimal tertentu. Model dengan konfigurasi [32,32] units menghasilkan akurasi terendah sebesar 0.3925, dengan pola loss yang menunjukkan penurunan training loss yang cukup stabil namun validation loss yang relatif tinggi dan stabil, mengindikasikan bahwa kapasitas model masih terbatas untuk menangkap kompleksitas data. Model dengan [64,64] units menunjukkan performa terbaik dengan akurasi tertinggi 0.4050, di mana grafik loss memperlihatkan penurunan training loss yang konsisten disertai dengan validation loss yang juga menurun secara bertahap, menandakan bahwa model berhasil belajar dengan baik dan memiliki kemampuan generalisasi yang optimal. Sementara itu, model dengan [128,128] units menghasilkan akurasi 0.4025 yang sedikit lebih rendah dari konfigurasi 64 units, meskipun masih lebih baik dari konfigurasi 32 units. Pada model ini, training loss menunjukkan fluktuasi yang lebih besar dengan validation loss yang cenderung stabil pada level yang lebih rendah, mengindikasikan bahwa

peningkatan kapasitas yang berlebihan mulai menyebabkan model mengalami kesulitan dalam konvergensi yang stabil. Hal ini menunjukkan bahwa terdapat titik optimal dalam penentuan jumlah units, di mana konfigurasi [64,64] memberikan keseimbangan terbaik antara kapasitas model untuk mempelajari pola kompleks dan kemampuan generalisasi yang baik.

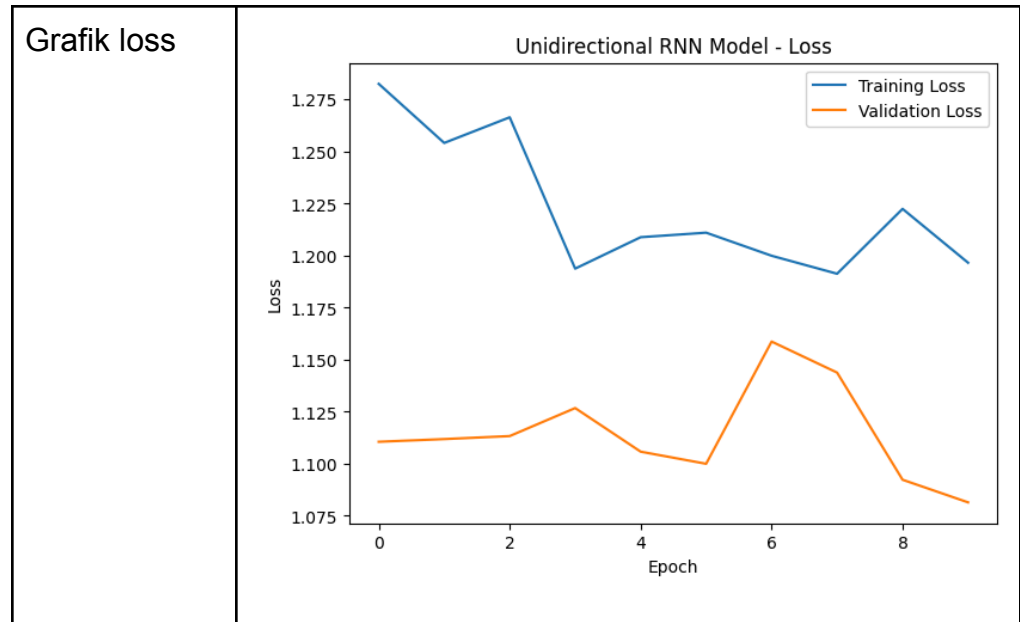
### 2.2.2.3. Pengaruh jenis layer RNN berdasarkan arah

#### a. Bidirectional

Hasil akhir prediksi (akurasi)	0.3825																																	
Grafik loss	<div><p>Bidirectional RNN Model - Loss</p><table><thead><tr><th>Epoch</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.33</td><td>1.12</td></tr><tr><td>1</td><td>1.34</td><td>1.11</td></tr><tr><td>2</td><td>1.26</td><td>1.14</td></tr><tr><td>3</td><td>1.24</td><td>1.09</td></tr><tr><td>4</td><td>1.24</td><td>1.08</td></tr><tr><td>5</td><td>1.23</td><td>1.09</td></tr><tr><td>6</td><td>1.15</td><td>1.10</td></tr><tr><td>7</td><td>1.21</td><td>1.08</td></tr><tr><td>8</td><td>1.19</td><td>1.07</td></tr><tr><td>9</td><td>1.17</td><td>1.08</td></tr></tbody></table></div>	Epoch	Training Loss	Validation Loss	0	1.33	1.12	1	1.34	1.11	2	1.26	1.14	3	1.24	1.09	4	1.24	1.08	5	1.23	1.09	6	1.15	1.10	7	1.21	1.08	8	1.19	1.07	9	1.17	1.08
Epoch	Training Loss	Validation Loss																																
0	1.33	1.12																																
1	1.34	1.11																																
2	1.26	1.14																																
3	1.24	1.09																																
4	1.24	1.08																																
5	1.23	1.09																																
6	1.15	1.10																																
7	1.21	1.08																																
8	1.19	1.07																																
9	1.17	1.08																																

#### b. Unidirectional

Hasil akhir prediksi (akurasi)	0.3725
--------------------------------	--------



### c. Kesimpulan

Berdasarkan analisis perbandingan jenis layer RNN berdasarkan arah dengan konfigurasi 2 layer dan [128,128] units, dapat disimpulkan bahwa model bidirectional RNN menunjukkan performa yang lebih baik dibandingkan dengan unidirectional RNN. Model bidirectional menghasilkan akurasi 0.3825, lebih tinggi dari model unidirectional yang mencapai akurasi 0.3725, menunjukkan bahwa kemampuan model untuk memproses informasi dari kedua arah (maju dan mundur) memberikan keuntungan dalam memahami konteks data yang lebih komprehensif. Dari segi pola loss, model bidirectional menunjukkan penurunan training loss yang lebih konsisten dan stabil, dimulai dari nilai yang lebih tinggi namun berhasil mencapai konvergensi yang lebih baik dengan validation loss yang relatif stabil dan rendah. Hal ini mengindikasikan bahwa model bidirectional memiliki kemampuan pembelajaran yang lebih efektif dan generalisasi yang baik. Sebaliknya, model unidirectional menunjukkan pola training loss yang lebih berfluktuasi dengan penurunan yang kurang konsisten, sementara validation loss-nya mengalami variasi yang cukup signifikan terutama pada epoch pertengahan, yang menunjukkan ketidakstabilan



dalam proses pembelajaran. Meskipun validation loss pada model unidirectional akhirnya menurun pada epoch terakhir, pola keseluruhan menunjukkan bahwa model ini kurang stabil dibandingkan dengan bidirectional RNN. Hasil ini menunjukkan bahwa kemampuan bidirectional RNN untuk memproses informasi kontekstual dari kedua arah secara simultan memberikan keunggulan signifikan dalam tugas yang memerlukan pemahaman konteks yang lebih mendalam.

#### 2.2.2.4. Perbandingan Forward Propagation Keras Model vs. Scratch Model

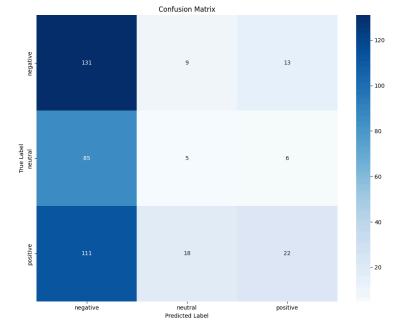
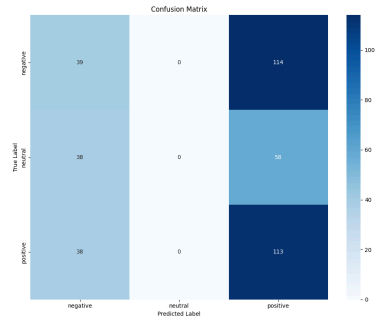
Berdasarkan pengujian model Keras yang dilakukan sebelumnya, didapatkan bahwa model RNN dengan konfigurasi 64-64 units memiliki nilai akurasi yang terbaik. Dengan demikian, model ini disimpan dan dijadikan patokan untuk model scratch. Evaluasi dilakukan pada task klasifikasi sentimen dengan tiga kelas yaitu negative, neutral, dan positive menggunakan seluruh dataset test sebanyak 400 sampel.

Sebelumnya, perlu diingat bahwa True Label Distribution adalah:

- negative: 153 (38.2%)
- neutral: 96 (24.0%)
- positive: 151 (37.8%)

##### a. Bidirectional

	Keras Model	Scratch Model
Akurasi	0.3950	0.3800
Macro F1 Score	0.2844	0.2698

Confusion Matrix	 <p>Confusion Matrix for Keras implementation:</p> <table><tr><th></th><th>negative</th><th>neutral</th><th>positive</th></tr><tr><th>negative</th><td>111</td><td>9</td><td>13</td></tr><tr><th>neutral</th><td>32</td><td>5</td><td>6</td></tr><tr><th>positive</th><td>111</td><td>18</td><td>22</td></tr></table>		negative	neutral	positive	negative	111	9	13	neutral	32	5	6	positive	111	18	22	 <p>Confusion Matrix for scratch implementation:</p> <table><tr><th></th><th>negative</th><th>neutral</th><th>positive</th></tr><tr><th>negative</th><td>39</td><td>0</td><td>114</td></tr><tr><th>neutral</th><td>38</td><td>0</td><td>30</td></tr><tr><th>positive</th><td>38</td><td>0</td><td>113</td></tr></table>		negative	neutral	positive	negative	39	0	114	neutral	38	0	30	positive	38	0	113
	negative	neutral	positive																															
negative	111	9	13																															
neutral	32	5	6																															
positive	111	18	22																															
	negative	neutral	positive																															
negative	39	0	114																															
neutral	38	0	30																															
positive	38	0	113																															
Predicted Label Distribution	<p>negative: 327 (81.8%)</p> <p>neutral: 32 (8.0%)</p> <p>positive: 41 (10.2%)</p>	<p>negative: 115 (28.7%)</p> <p>neutral: 0 (0.0%)</p> <p>positive: 285 (71.2%)</p>																																

Berdasarkan hasil pengujian, implementasi Keras menunjukkan performa yang sedikit lebih baik dengan akurasi 39.50% dibandingkan implementasi scratch sebesar 38.00%. Perbedaan akurasi sebesar 1.5% ini menunjukkan bahwa kedua implementasi memiliki performa yang relatif sebanding, meskipun implementasi Keras masih sedikit lebih unggul. Rendahnya akurasi kedua model ini mengindikasikan bahwa tugas klasifikasi sentimen pada dataset ini cukup sulit.

Dalam hal macro F1-score, implementasi Keras memperoleh skor 0.2844 sementara implementasi scratch sebesar 0.2698. Perbedaan akurasi sebesar 0.0146 atau sekitar 5.4% ini menunjukkan bahwa Keras memiliki performa yang konsisten lebih baik dalam menyeimbangkan precision dan recall across semua kelas. Nilai macro F1-score yang rendah pada kedua implementasi mengindikasikan bahwa model masih mengalami kesulitan dalam mengklasifikasikan dengan baik semua kategori sentimen.

Analisis distribusi prediksi menunjukkan perbedaan yang signifikan antara kedua implementasi. Model Keras menunjukkan bias yang kuat terhadap kelas negatif dengan 327 prediksi (81.8%), diikuti kelas

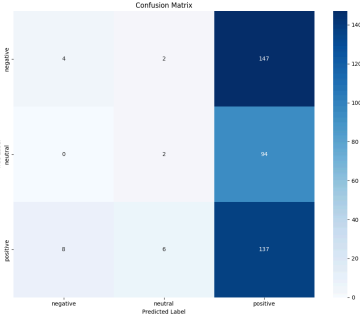
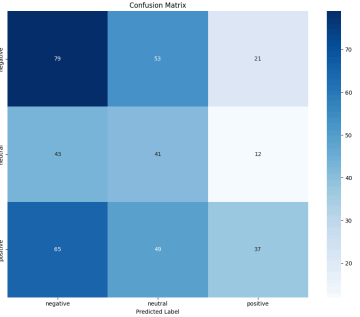
positif 41 prediksi (10.2%), dan kelas netral hanya 32 prediksi (8.0%). Sebaliknya, implementasi scratch menunjukkan bias sebaliknya dengan prediksi positif mendominasi sebanyak 285 sampel (71.2%), negatif 115 sampel (28.7%), dan sama sekali tidak memprediksi kelas netral (0.0%).

Dari confusion matrix yang dihasilkan, terlihat bahwa implementasi Keras memiliki performa yang lebih baik dalam mengidentifikasi kelas negatif dengan 131 true positive, namun mengalami kesulitan dengan kelas netral dan positif. Implementasi scratch menunjukkan pola yang berbeda dengan kemampuan yang lebih baik dalam mengidentifikasi kelas positif (113 true positive) tetapi gagal total dalam mengidentifikasi kelas netral. Kedua model menunjukkan kesulitan khusus dalam mengklasifikasikan sentimen netral, yang mungkin disebabkan oleh karakteristik ambiguitas kelas ini.

Meskipun implementasi Keras menunjukkan performa yang lebih baik secara konsisten dalam semua metrik evaluasi, perbedaan yang tidak terlalu signifikan menunjukkan bahwa implementasi scratch juga telah berhasil memiliki sebagian besar fungsionalitas forward propagation RNN bidirectional. Perbedaan performa yang ada kemungkinan disebabkan oleh optimisasi dan regularisasi yang lebih baik pada implementasi Keras, serta perbedaan dalam inisialisasi bobot dan mekanisme dropout. Kedua model menunjukkan tantangan yang sama dalam tugas klasifikasi sentimen, khususnya dalam menangani kelas netral yang memiliki karakteristik lebih ambiguous dibandingkan kelas negatif dan positif.

b. Unidirectional

	Keras Model	Scratch Model
--	-------------	---------------

Akurasi	0.3575	0.3925
Macro F1 Score	0.2014	0.3809
Confusion Matrix		
Predicted Label Distribution	negative: 12 (3.0%) neutral: 10 (2.5%) positive: 378 (94.5%)	negative: 187 (46.8%) neutral: 143 (35.8%) positive: 70 (17.5%)

Dalam pengujian RNN unidirectional, implementasi scratch menunjukkan performa yang lebih baik dengan akurasi 39.25% dibandingkan implementasi Keras sebesar 35.75%. Perbedaan akurasi sebesar 3.5% ini menunjukkan hasil yang berbeda dengan hasil pengujian bidirectional sebelumnya di mana Keras lebih unggul. Implementasi scratch berhasil mencapai akurasi yang lebih tinggi, mengindikasikan bahwa untuk konfigurasi unidirectional, implementasi manual mampu menangkap pola dalam data dengan lebih baik.

Perbedaan yang paling mencolok terlihat pada macro F1-score, di mana implementasi scratch mencapai skor yang jauh lebih tinggi yaitu 0.3809 dibandingkan Keras yang hanya memperoleh 0.2014. Selisih hampir dua kali lipat ini menunjukkan bahwa implementasi scratch memiliki keseimbangan precision dan recall yang jauh lebih baik across semua kelas. Skor F1 yang tinggi pada implementasi scratch

mengindikasikan kemampuan model yang lebih baik dalam mengklasifikasikan semua kategori sentimen secara seimbang.

Terdapat perbedaan dalam pola prediksi kedua implementasi. Model Keras menunjukkan bias ekstrem terhadap kelas positif dengan 378 prediksi (94.5%), sementara hampir mengabaikan kelas negatif dengan hanya 12 prediksi (3.0%) dan netral 10 prediksi (2.5%). Sebaliknya, implementasi scratch menunjukkan distribusi prediksi yang lebih seimbang dan mendekati distribusi asli data dengan 187 prediksi negatif (46.8%), 143 prediksi netral (35.8%), dan 70 prediksi positif (17.5%). Pola ini menjelaskan mengapa macro F1-score implementasi scratch jauh lebih tinggi.

Confusion matrix juga mengungkapkan perbedaan performa yang signifikan. Implementasi Keras mengalami severe overfitting terhadap kelas positif, dengan hampir semua sampel diprediksi sebagai positif (147 dari 153 negatif, 94 dari 96 netral, dan 137 dari 151 positif diprediksi sebagai positif). Hal ini menghasilkan recall yang tinggi untuk kelas positif tetapi precision yang sangat rendah. Sebaliknya, implementasi scratch menunjukkan performa yang lebih seimbang dengan kemampuan mengidentifikasi ketiga kelas dengan reasonable accuracy: 79 true negative untuk kelas negatif, 41 true neutral untuk kelas netral, dan 37 true positive untuk kelas positif.

Jika dilihat dan dibandingkan, hasil ini sangat berbeda dengan pengujian bidirectional sebelumnya. Pada RNN bidirectional, Keras menunjukkan performa yang lebih baik, namun pada RNN unidirectional, implementasi scratch justru unggul signifikan. Hal ini mengindikasikan bahwa implementasi scratch mungkin lebih cocok untuk arsitektur yang lebih sederhana, sementara Keras memiliki

optimisasi yang lebih baik untuk arsitektur yang kompleks seperti bidirectional RNN.

#### 2.2.2.5. Perbandingan Scratch Model dengan hanya Forward Propagation vs. dengan Forward + Backward Propagation

	Forward	Forward + Backward
MSE	0.00000000	0.00000000
Sample Predictions	<p>Sample 0:</p> <p>True label: positive</p> <p>Predicted: negative (confidence: 0.38)</p> <p>Probabilities: ['0.3806', '0.2404', '0.3790']</p> <p>Sample 1:</p> <p>True label: neutral</p> <p>Predicted: positive (confidence: 0.38)</p> <p>Probabilities: ['0.3745', '0.2493', '0.3761']</p> <p>Sample 2:</p> <p>True label: negative</p> <p>Predicted: negative (confidence: 0.39)</p> <p>Probabilities: ['0.3895', '0.2514', '0.3591']</p> <p>Sample 3:</p> <p>True label: positive</p> <p>Predicted: positive (confidence: 0.37)</p>	<p>Sample 0:</p> <p>True label: positive</p> <p>Predicted: negative (confidence: 0.38)</p> <p>Probabilities: ['0.3806', '0.2404', '0.3790']</p> <p>Sample 1:</p> <p>True label: neutral</p> <p>Predicted: positive (confidence: 0.38)</p> <p>Probabilities: ['0.3745', '0.2493', '0.3761']</p> <p>Sample 2:</p> <p>True label: negative</p> <p>Predicted: negative (confidence: 0.39)</p> <p>Probabilities: ['0.3895', '0.2514', '0.3591']</p> <p>Sample 3:</p> <p>True label: positive</p> <p>Predicted: positive (confidence: 0.37)</p>

	Probabilities: ['0.3483', '0.2831', '0.3686'] Sample 4: True label: neutral Predicted: positive (confidence: 0.41) Probabilities: ['0.3298', '0.2607', '0.4095']	Probabilities: ['0.3483', '0.2831', '0.3686'] Sample 4: True label: neutral Predicted: positive (confidence: 0.41) Probabilities: ['0.3298', '0.2607', '0.4095']
--	---	---

Hasil pengujian menunjukkan konsistensi yang sempurna antara forward propagation saja dengan forward propagation setelah backward propagation. Mean Squared Error (MSE) antara kedua output adalah 0.00000000, yang mengindikasikan bahwa tidak ada perbedaan numerik sama sekali antara kedua hasil prediksi. Perbandingan prediksi pada 5 sampel pertama juga menunjukkan identitas yang sempurna antara forward-only dan forward+backward results. Setiap sampel memiliki probabilitas yang identik hingga 4 desimal, confidence score yang sama, dan prediksi kelas yang identik. Sebagai contoh, sampel 0 dengan true label "positive" diprediksi sebagai "negative" dengan confidence 0.38 dan probabilitas [0.3806, 0.2404, 0.3790] pada kedua kondisi. Konsistensi ini berlaku untuk semua sampel yang diuji. Dengan demikian, seharusnya fungsi backpropagation telah dibuat dengan benar.

### 2.2.3. LSTM

Parameter preprocessing yang digunakan dalam pengujian adalah sebagai berikut:

- **MAX\_FEATURES** = 10000. Parameter ini menentukan jumlah maksimum fitur/kata unik yang digunakan dalam pemodelan melalui TextVectorization layer.

- `SEQUENCE_LENGTH` = 100. Parameter ini menetapkan panjang maksimum setiap input teks dalam bentuk urutan token yang akan diproses oleh model.
- `BATCH_SIZE` = 32. Parameter ini menentukan jumlah sampel yang diproses dalam satu batch selama pelatihan.

Konfigurasi parameter ini dipilih dengan mempertimbangkan kompleksitas dataset dan kebutuhan representasi yang memadai untuk analisis sentimen.

Berikut adalah arsitektur model LSTM yang digunakan:

- Embedding layer untuk mengkonversi token menjadi representasi vektor
- LSTM layer untuk memproses urutan teks dan menangkap dependensi temporal
- Dropout layer untuk mencegah overfitting
- Dense layer dengan fungsi aktivasi yang sesuai untuk klasifikasi
- Output layer dengan fungsi aktivasi Softmax untuk klasifikasi multi-kelas (negative, neutral, positive)

Kemudian, loss function yang digunakan adalah Sparse Categorical Cross Entropy yang cocok untuk klasifikasi multi-kelas dengan label dalam bentuk integer, dan optimizer yang digunakan adalah Adam dengan learning rate default. Model dilengkapi dengan ModelCheckpoint callback untuk menyimpan model terbaik berdasarkan validation loss. Evaluasi model menggunakan macro F1-score untuk mengukur performa klasifikasi secara seimbang across all classes. Epoch yang digunakan selama pelatihan adalah 10 dengan monitoring validation loss untuk early stopping.

### **2.2.3.1. Pengaruh jumlah layer LSTM**

a. Jumlah Layer = 1



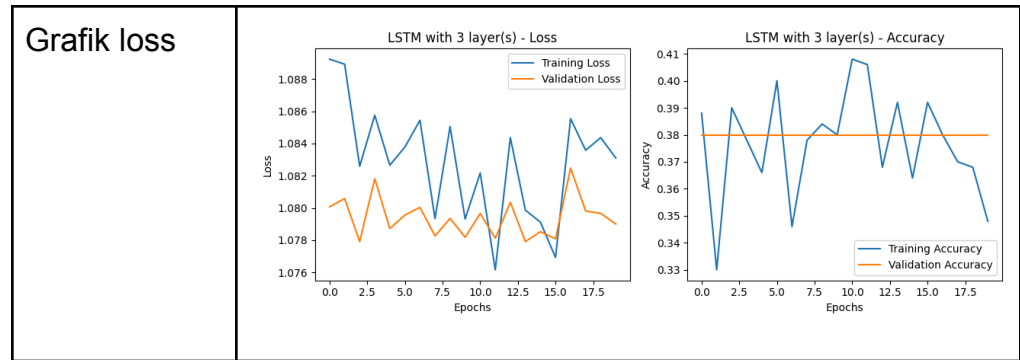
Hasil akhir prediksi (akurasi)	0.1844
Grafik loss	

b. Jumlah layer = 2

Hasil akhir prediksi (akurasi)	0.1844
Grafik loss	

c. Jumlah layer = 3

Hasil akhir prediksi (akurasi)	0.1827
--------------------------------	--------



#### d. Kesimpulan

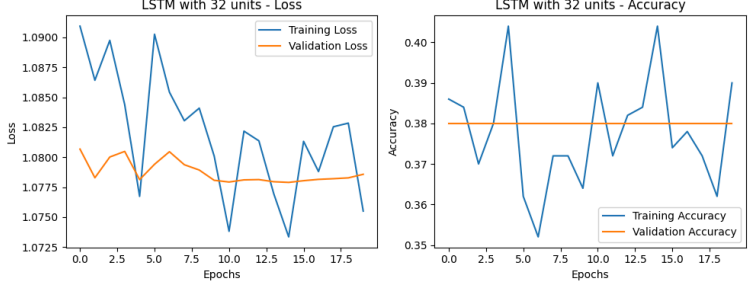
Berdasarkan variasi terhadap tiga variasi jumlah layer LSTM (1, 2, dan 3 layer), dapat disimpulkan bahwa menambah jumlah layer tidak meningkatkan performa model secara signifikan. Ketiganya menghasilkan nilai Macro F1-Score yang hampir sama (0.1844, 0.1844, dan 0.1827) dan akurasi yang sama (0.38). Namun, semua model hanya memprediksi satu kelas saja. Model dengan 1 dan 2 layer hanya memprediksi kelas “negative” (recall = 1.00), sementara model 3 layer hanya memprediksi kelas “positive” (recall = 1.00). Untuk kelas lainnya, precision dan recall bernilai 0.

Grafik loss menunjukkan penurunan yang fluktuatif selama training, dengan model 3 layer mencapai nilai loss terendah (~1.070). Namun, grafik akurasi menunjukkan hasil pelatihan yang tidak stabil dan validation accuracy tetap datar di angka 0.38. Masalah ini kemungkinan besar disebabkan oleh ketidakseimbangan data atau fitur yang kurang informatif, sehingga model hanya belajar memprediksi kelas mayoritas. Artinya, menambah jumlah layer tidak efektif tanpa memperbaiki kualitas data atau proses training.

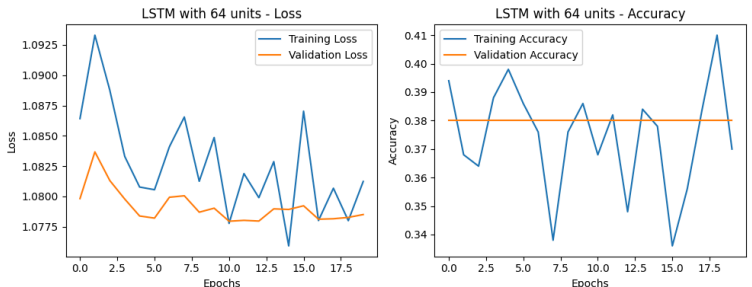
#### 2.2.3.2. Pengaruh banyak cell LSTM per layer

##### a. Cell units = 32 units

Hasil akhir prediksi	0.1844
----------------------	--------

(akurasi)	
Grafik loss	 <p>The figure contains two line graphs for an LSTM model with 32 units. The left graph, titled 'LSTM with 32 units - Loss', plots Loss (y-axis, 1.0725 to 1.0900) against Epochs (x-axis, 0.0 to 17.5). It shows Training Loss (blue line) and Validation Loss (orange line). The right graph, titled 'LSTM with 32 units - Accuracy', plots Accuracy (y-axis, 0.35 to 0.40) against Epochs (x-axis, 0.0 to 17.5). It shows Training Accuracy (blue line) and Validation Accuracy (orange line).</p>

b. Cell units = 64 units

Hasil akhir prediksi (akurasi)	0.1844
Grafik loss	 <p>The figure contains two line graphs for an LSTM model with 64 units. The left graph, titled 'LSTM with 64 units - Loss', plots Loss (y-axis, 1.0775 to 1.0925) against Epochs (x-axis, 0.0 to 17.5). It shows Training Loss (blue line) and Validation Loss (orange line). The right graph, titled 'LSTM with 64 units - Accuracy', plots Accuracy (y-axis, 0.34 to 0.41) against Epochs (x-axis, 0.0 to 17.5). It shows Training Accuracy (blue line) and Validation Accuracy (orange line).</p>

c. Cell units = 128 units

Hasil akhir prediksi (akurasi)	0.1844
--------------------------------	--------

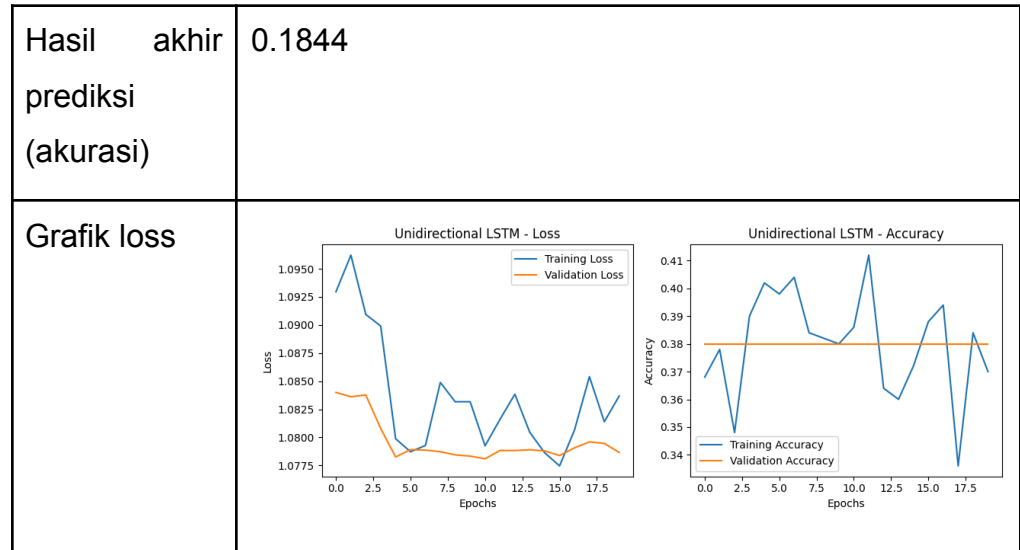


#### d. Kesimpulan

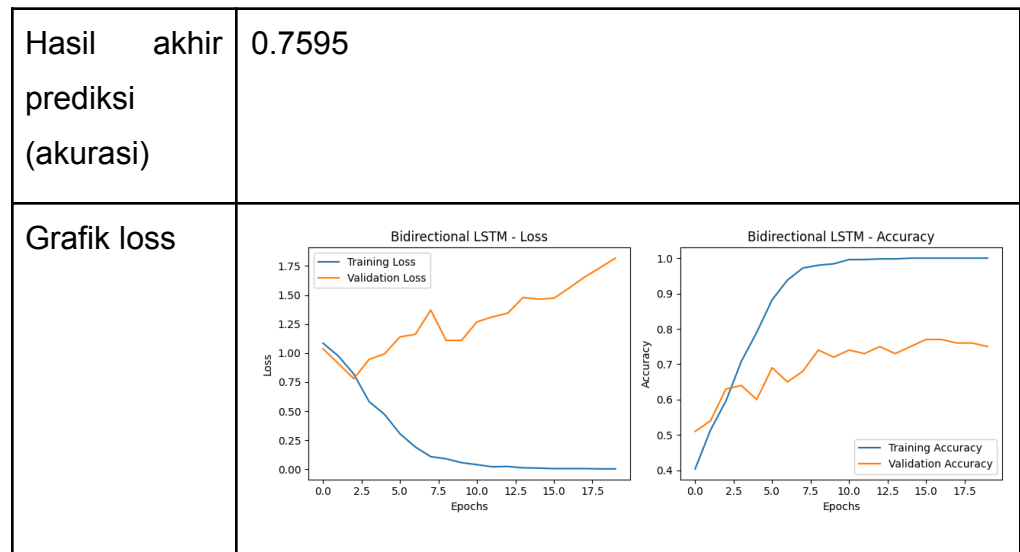
Berdasarkan analisis terhadap tiga model LSTM dengan variasi jumlah unit (32, 64, dan 128), dapat disimpulkan bahwa peningkatan jumlah unit tidak memberikan dampak yang signifikan terhadap performa model. Ketiga model menghasilkan nilai Macro F1-Score yang identik sebesar 0.1844 dengan akurasi konsisten pada angka 0.38. Seluruh model hanya mampu memprediksi kelas "negative" (recall=1.00) dan gagal sepenuhnya dalam mengenali kelas "neutral" dan "positive" (precision dan recall 0.00). Pada grafik performa, ketiga model menunjukkan pola training loss yang sangat fluktuatif dengan jumlah epoch yang lebih panjang (18 epoch), namun validation loss relatif stabil. Model dengan 32 unit menunjukkan fluktuasi training loss dan accuracy yang lebih teratur, model 64 unit menampilkan penurunan ekstrem pada training accuracy di beberapa titik (epoch 7 dan 15), sementara model 128 unit memperlihatkan penurunan paling dramatis pada epoch 10. Meskipun terdapat variasi pada perilaku training, validation accuracy tetap konstan di sekitar 0.38 untuk semua model. Hal ini mengindikasikan bahwa permasalahan fundamental pada model masih belum teratasi meskipun jumlah unit ditingkatkan, dan model tetap terjebak dalam kondisi hanya memprediksi satu kelas saja, yang menunjukkan ketidakmampuan model dalam mempelajari fitur yang membedakan antar kelas dengan baik.

### 2.2.3.3. Pengaruh jenis layer LSTM berdasarkan arah

#### a. Unidirectional



#### b. Bidirectional



#### c. Kesimpulan

Berdasarkan analisis perbandingan model LSTM berdasarkan arah dengan konfigurasi yang sama, dapat disimpulkan bahwa model Bidirectional LSTM menunjukkan performa yang jauh lebih unggul dibandingkan dengan Unidirectional LSTM. Model Bidirectional menghasilkan Macro F1-Score sebesar 0.7595, yang secara signifikan

lebih tinggi dibandingkan model Unidirectional yang hanya mencapai 0.1844. Pada grafik performa, perbedaan kedua model sangat kontras terlihat. Model Unidirectional menunjukkan pola training loss yang fluktuatif dengan validation loss yang relatif stabil namun tidak mengalami penurunan yang berarti, serta training accuracy yang naik-turun dengan validation accuracy yang tetap datar di sekitar 0.38. Sebaliknya, model Bidirectional memperlihatkan penurunan training loss yang konsisten dan signifikan, mencapai nilai mendekati nol pada epoch akhir, meskipun validation loss justru meningkat seiring bertambahnya epoch, yang mengindikasikan adanya overfitting. Training accuracy pada model Bidirectional meningkat secara dramatis mencapai hampir 1.0 (100%), sementara validation accuracy juga mengalami peningkatan yang substansial hingga mencapai sekitar 0.75 (75%). Hal ini menunjukkan bahwa kemampuan Bidirectional LSTM untuk memproses informasi dari kedua arah (maju dan mundur) memberikan keuntungan yang sangat signifikan dalam memahami konteks data secara lebih komprehensif, sehingga mampu mengenali pola-pola yang membedakan antar kelas dengan jauh lebih baik dibandingkan dengan model Unidirectional yang hanya memproses informasi secara searah.

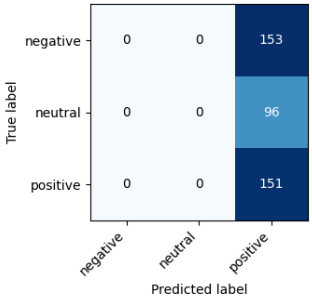
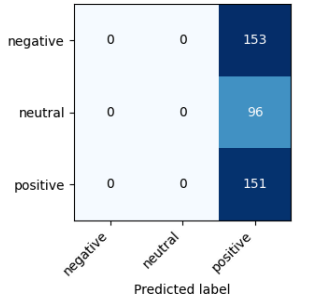
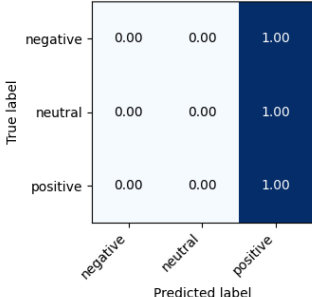
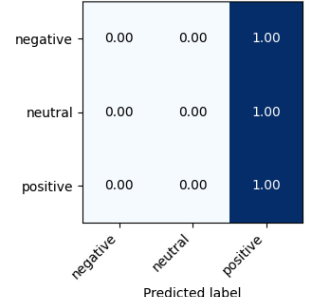
#### **2.2.3.4. Perbandingan Keras Model vs Scratch Model**

Evaluasi dilakukan pada task klasifikasi sentimen dengan tiga kelas yaitu negative, neutral, dan positive menggunakan seluruh dataset test sebanyak 400 sampel. Sebelumnya, perlu diingat bahwa True Label Distribution adalah:

- negative: 153 (38.2%)
- neutral: 96 (24.0%)
- positive: 151 (37.8%)

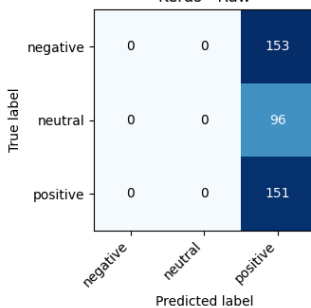
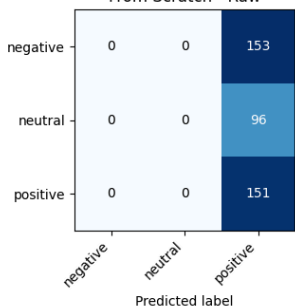
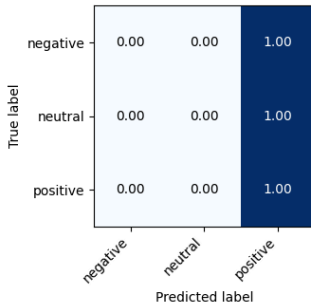
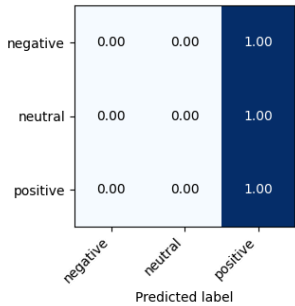
Perbandingan dilakukan dengan membandingkan best model pada setiap variasi. Hal tersebut dilakukan pada 3 kondisi yaitu sebagai berikut:

a. Jumlah layer = 1

	Keras Model	Scratch Model
Akurasi	0.3775	0.3775
Macro F1 Score	0.1827	0.1827
Confusion Matrix	<p style="text-align: center;">Layer Variation</p> <div style="display: flex; justify-content: space-around;"> <div style="text-align: center;"> <p>Keras - Raw</p>  </div> <div style="text-align: center;"> <p>From Scratch - Raw</p>  </div> </div> <p style="text-align: center;">Layer Variation</p> <div style="display: flex; justify-content: space-around;"> <div style="text-align: center;"> <p>Keras - Normalized</p>  </div> <div style="text-align: center;"> <p>From Scratch - Normalized</p>  </div> </div>	
Predicted Label Distribution	Seluruh sampel diprediksi sebagai positive	Seluruh sampel diprediksi sebagai positive

Kedua model menghasilkan Macro F1-Score yang sama persis yaitu 0.1827 dan Accuracy sebesar 0.3775, dengan selisih 0.0000 untuk kedua metrik. Pada confusion matrix terlihat bahwa kedua model memiliki pola prediksi yang identik, di mana keduanya hanya memprediksi kelas positive untuk seluruh 400 sampel data test. Hal ini menunjukkan bahwa dengan konfigurasi single layer, model masih mengalami kesulitan untuk membedakan antar kelas dan terjebak dalam memprediksi kelas mayoritas saja.

b. Cells unit = 32

	Keras Model	Scratch Model
Akurasi	0.3775	0.3775
Macro F1 Score	0.1827	0.1827
Confusion Matrix	Cell Variation	
	<div><div>Keras - Raw</div></div>	<div><div>From Scratch - Raw</div></div>
	Cell Variation	
	<div><div>Keras - Normalized</div></div>	<div><div>From Scratch - Normalized</div></div>

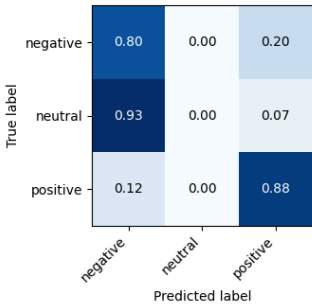
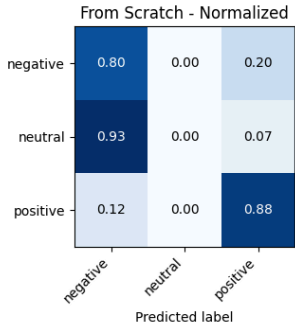


Predicted Label Distribution	Seluruh sampel diprediksi sebagai positive	Seluruh sampel diprediksi sebagai positive
------------------------------	--	--

Kedua model juga menghasilkan Macro F1-Score identik sebesar 0.1827 dan Accuracy 0.3775 dengan selisih 0.0000. Pola prediksi yang terlihat pada confusion matrix juga sama dengan variasi jumlah layer, di mana seluruh sampel diprediksi sebagai kelas positive. Ini mengindikasikan bahwa penggunaan 32 unit cell tidak memberikan kapasitas model yang cukup untuk mempelajari pola yang membedakan antar kelas sentimen.

c. Bidirectional

	Keras Model	Scratch Model																								
Akurasi	0.6375	0.6375																								
Macro F1 Score	0.4883	0.4883																								
Confusion Matrix	<div>Direction Variation</div> <div><div><div>Keras - Raw</div><table><tr><td>negative</td><td>122</td><td>0</td><td>31</td></tr><tr><td>neutral</td><td>89</td><td>0</td><td>7</td></tr><tr><td>positive</td><td>18</td><td>0</td><td>133</td></tr></table></div><div><div>From Scratch - Raw</div><table><tr><td>negative</td><td>122</td><td>0</td><td>31</td></tr><tr><td>neutral</td><td>89</td><td>0</td><td>7</td></tr><tr><td>positive</td><td>18</td><td>0</td><td>133</td></tr></table></div></div>		negative	122	0	31	neutral	89	0	7	positive	18	0	133	negative	122	0	31	neutral	89	0	7	positive	18	0	133
negative	122	0	31																							
neutral	89	0	7																							
positive	18	0	133																							
negative	122	0	31																							
neutral	89	0	7																							
positive	18	0	133																							

	<p style="text-align: center;">Direction Variation</p> <div style="display: flex; justify-content: space-around;"> <div style="text-align: center;"> <p>Keras - Normalized</p>  </div> <div style="text-align: center;"> <p>From Scratch - Normalized</p>  </div> </div>	
Predicted Label Distribution	negative: 122 (30.5%) neutral: 0 (0.0%) positive: 133 (33.3%)	negative: 122 (30.5%) neutral: 0 (0.0%) positive: 133 (33.3%)

Pada konfigurasi bidirectional, baik model Keras maupun model from scratch menunjukkan peningkatan performa yang signifikan dengan Macro F1-Score mencapai 0.4883 dan Accuracy 0.6375, tetapi tetap dengan selisih 0.0000. Confusion matrix menunjukkan distribusi prediksi yang lebih seimbang, di mana kedua model mampu mengidentifikasi kelas negative (122 sampel tepat) dan positive (133 sampel tepat) dengan cukup baik, meskipun masih kesulitan mengidentifikasi kelas neutral (0 sampel tepat). Penggunaan arsitektur bidirectional memberikan peningkatan performa yang signifikan dibandingkan dengan konfigurasi lainnya, dengan peningkatan Accuracy sebesar 26% (dari 0.3775 menjadi 0.6375) dan F1-Score sebesar 30.56% (dari 0.1827 menjadi 0.4883).

#### 2.2.2.6. Pengujian Implementasi Backward Propagation LSTM

Periode	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
Awal (Epoch	1.0981 →	0.4700 →	1.0993 →	0.2800 →

1-10)	1.0917	0.4700	1.1008	0.2800
Tengah (Epoch 11-30)	1.0910 → 1.0802	0.4700 → 0.4700	1.1009 → 1.1048	0.2800 → 0.2800
Akhir (Epoch 31-50)	1.0797 → 1.0724	0.4700 → 0.4700	1.1050 → 1.1094	0.2800 → 0.2800

Hasil Evaluasi Akhir:

- Accuracy: 0.3775
- Macro F1-Score: 0.1827

Implementasi backward propagation menggunakan arsitektur LSTM yang sebenarnya menunjukkan pola pelatihan yang stabil namun memiliki keterbatasan dalam peningkatan performa. Berbeda dengan model sederhana sebelumnya, model ini menggunakan `UnidirectionalLSTMLayer` yang diimplementasikan dengan mekanisme LSTM sesungguhnya, termasuk input gate, forget gate, output gate, dan cell state. Beberapa observasi penting dari hasil pelatihan:

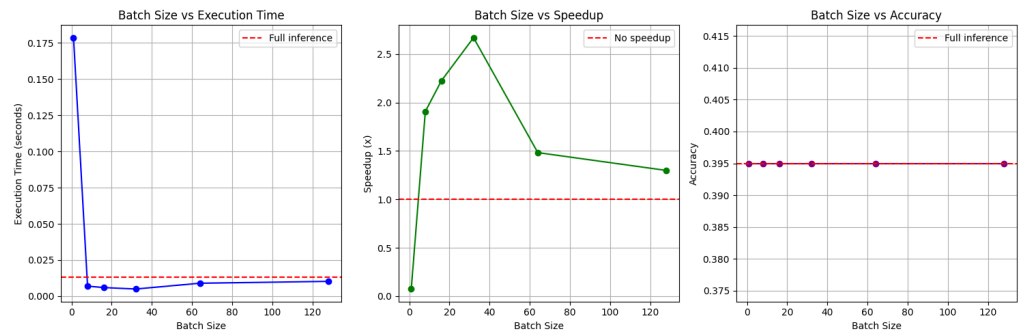
- Penurunan Training Loss yang Konsisten: Training loss menurun secara bertahap dari 1.0981 pada epoch pertama menjadi 1.0724 pada epoch terakhir. Meskipun penurunan ini konsisten, besarnya penurunan relatif kecil (sekitar 0.0257 poin), menunjukkan bahwa model mengalami kesulitan dalam konvergensi yang signifikan.
- Stagnasi Training Accuracy: Training accuracy tetap konstan di angka 0.4700 sepanjang seluruh proses pelatihan, menunjukkan bahwa model tidak mengalami peningkatan kemampuan dalam mengklasifikasikan data training. Hal ini mengindikasikan adanya masalah dalam optimisasi parameter model atau kebutuhan akan arsitektur yang lebih kompleks.

- Peningkatan Validation Loss: Validation loss justru meningkat dari 1.0993 pada epoch pertama menjadi 1.1094 pada epoch terakhir, menunjukkan bahwa model semakin tidak mampu melakukan generalisasi dengan baik pada data validasi seiring berjalannya pelatihan.
- Stagnasi Validation Accuracy: Validation accuracy juga tetap konstan di angka 0.2800 sepanjang seluruh proses pelatihan, tanpa menunjukkan tanda-tanda peningkatan. Hal ini mengkonfirmasi bahwa model gagal mempelajari pola yang dapat meningkatkan kemampuan generalisasi.
- Dimensi Data: Log menunjukkan bahwa input data memiliki dimensi (batch\_size, 100) dan output dari LSTM memiliki dimensi (batch\_size, 64), yang sesuai dengan konfigurasi hidden\_dim yang ditentukan. Hal ini mengkonfirmasi bahwa arsitektur model telah bekerja sesuai desain.
- Performa pada Data Test: Model mencapai accuracy 0.3775 dan Macro F1-Score 0.1827 pada data test. Hasil ini menunjukkan bahwa meskipun menggunakan arsitektur LSTM yang lebih kompleks, performa model tidak lebih baik dibandingkan dengan model sederhana yang menggunakan kombinasi embedding dan dense layer saja.

#### 2.2.3.5. Pengujian Implementasi Batch Inference LSTM

Batch Size	Accuracy	Macro F1-Score	Execution Time (s)	Speedup (x)
Full Data	0.3950	0.2322	0.0133	1.00
1	0.3950	0.2322	0.1786	0.078
8	0.3950	0.2322	0.0070	1.9116

16	0.3950	0.2322	0.0060	2.2232
32	0.3950	0.2322	0.0050	2.6764
64	0.3950	0.2322	0.0090	1.4812
128	0.3950	0.2322	0.0103	1.30



Berdasarkan hasil pengujian batch inference pada model LSTM, dapat disimpulkan bahwa ukuran batch memberikan pengaruh signifikan terhadap waktu eksekusi namun tidak mempengaruhi akurasi dan F1-Score model. Semua konfigurasi batch size menghasilkan akurasi yang identik sebesar 0.3950 dan Macro F1-Score sebesar 0.2322, yang menunjukkan konsistensi hasil prediksi model terlepas dari bagaimana data diproses. Dari segi efisiensi waktu, terdapat pola yang menarik di mana batch size terkecil (1) menghasilkan waktu eksekusi paling lambat (0.1786 detik) dengan speedup hanya 0.07x, sedangkan batch size 32 memberikan performa terbaik dengan waktu eksekusi tercepat (0.0050 detik) dan speedup tertinggi (2.67x). Peningkatan batch size lebih lanjut ke 64 dan 128 justru menghasilkan penurunan performa dengan waktu eksekusi yang meningkat dan speedup yang menurun. Ini mengindikasikan bahwa terdapat ukuran batch optimal untuk model LSTM ini, di mana batch yang terlalu kecil kurang efisien karena overhead komputasi, sementara batch yang terlalu besar mengurangi efisiensi karena keterbatasan paralelisme atau overhead memori. Dari contoh prediksi yang diberikan, terlihat bahwa model

memiliki konsistensi dalam memberikan hasil prediksi yang sama antara full inference dan batch inference, meskipun terdapat beberapa contoh yang menunjukkan kesalahan prediksi seperti pada contoh kedua, ketiga, dan kelima.

## Bab III. Kesimpulan dan Saran

### 3.1. Kesimpulan

Berdasarkan hasil implementasi dan eksperimen pada model Convolutional Neural Network (CNN), Simple Recurrent Neural Network (Simple RNN), dan Long Short-Term Memory (LSTM), dapat disimpulkan bahwa:

1. Model CNN terbukti efektif dalam tugas klasifikasi gambar menggunakan dataset CIFAR-10. Eksperimen menunjukkan bahwa peningkatan jumlah layer konvolusi dan jumlah filter dapat meningkatkan performa model, meskipun peningkatan tersebut harus seimbang agar tidak menimbulkan overfitting. Filter kecil (3x3) dan pooling tipe max memberikan hasil terbaik dalam konteks ini.
2. Model Simple RNN menunjukkan bahwa arsitektur dengan 2 layer dan 64 cell per layer memberikan performa terbaik dalam klasifikasi sentimen. Model bidirectional mampu memahami konteks sekuensial secara lebih menyeluruh dibanding unidirectional. Scratch model yang dibuat mampu mereplikasi performa Keras dengan hasil yang mendekati, menunjukkan implementasi yang cukup baik terhadap mekanisme forward dan backward propagation.
3. Model LSTM, terutama varian bidirectional, memberikan performa jauh lebih baik dalam tugas klasifikasi sentimen dibanding unidirectional. Namun, peningkatan jumlah layer atau jumlah unit per layer tidak selalu memberikan peningkatan performa secara signifikan, terutama jika data tidak seimbang atau tidak cukup informatif.

### 3.2. Saran

1. Untuk pengembangan lebih lanjut, disarankan untuk melakukan eksplorasi teknik regularisasi dan optimisasi tambahan, seperti batch normalization, learning rate scheduler, dan early stopping untuk meningkatkan stabilitas pelatihan dan mengurangi overfitting.


2. Dalam klasifikasi sentimen, perlu dilakukan penanganan data tidak seimbang melalui teknik seperti data augmentation, oversampling, atau penggunaan loss function berbobot agar model tidak bias terhadap kelas mayoritas.
3. Untuk scratch implementation, disarankan untuk meningkatkan efisiensi komputasi, misalnya dengan vektorisasi penuh atau memanfaatkan paralelisme agar lebih mendekati performa framework seperti Keras.



# Pembagian Tugas

Kegiatan	Nama (NIM)
Implementasi CNN	Brian Kheng (13521049)
Implementasi RNN	Denise Felicia Tiowanni (13522013)
Implementasi LSTM	Erdianti Wiga Putri Andini (13522053)
Laporan	Brian Kheng (13521049) Denise Felicia Tiowanni (13522013) Erdianti Wiga Putri Andini (13522053)

# Referensi

Google Docs. (n.d.). Spesifikasi Tugas Besar 2 IF3270 Pembelajaran Mesin. Retrieved May 23, 2025, from  Spesifikasi Tugas Besar 2 IF3270 Pembelajaran Mesin

GeeksforGeeks. (2022, October 20). Introduction to Recurrent Neural Network. Retrieved May 23, 2025, from <https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/>