Tugas Besar 2 IF3270 Pembelajaran Mesin

Convolutional Neural Network dan Recurrent Neural Network

Disusun oleh:

Kelompok 12 - K2

Dhidit Abdi Aziz / 13522040

Muhammad Syarafi Akmal / 13522076

Sa'ad Abdul Hakim / 13522092

Program Studi Teknik Informatika

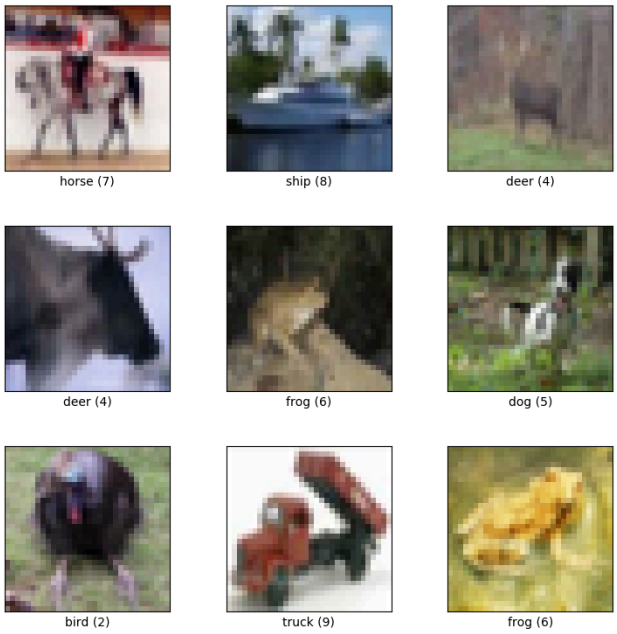
Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

2025

Bab I Deskripsi Persoalan

CIFAR-10 (Canadian Institute for Advanced Research - 10 classes) adalah dataset citra yang terdiri dari 60.000 gambar berwarna dalam 10 kelas berbeda seperti pesawat, mobil, burung, kucing, rusa, anjing, katak, kuda, kapal, dan truk. Setiap gambar memiliki resolusi 32×32 piksel dan terdiri dari tiga saluran warna (RGB), sehingga total terdapat 3.072 fitur per gambar. Dataset ini terbagi menjadi 50.000 data latih dan 10.000 data uji.



Sumber: https://www.tensorflow.org/datasets/catalog/cifar10

Dalam tugas besar ini, dataset CIFAR-10 akan digunakan untuk melatih dan menguji model Convolutional Neural Network (CNN) yang diimplementasikan menggunakan library Keras. Model CNN tersebut akan memiliki minimal satu Conv2D layer, pooling layer, dan dense layer, serta akan dioptimasi menggunakan Adam dengan fungsi loss Sparse Categorical Crossentropy. Dataset akan dibagi menjadi 40.000 data latih, 10.000 data validasi, dan 10.000 data uji.

Pelatihan dilakukan dengan berbagai variasi hiperparameter seperti jumlah layer konvolusi, jumlah filter, ukuran filter, dan jenis pooling layer. Selain itu, modul forward propagation CNN juga akan diimplementasikan dari awal (from scratch) dan dibandingkan hasilnya dengan model hasil pelatihan Keras berdasarkan macro F1-score pada data uji.

NusaX-Sentiment adalah dataset opini dengan berbagai bahasa daerah di Indonesia dan bahasa nasional Indonesia serta bahasa Inggris dari berbagai platform media sosial dan berita, yang

dikategorikan ke dalam beberapa sentimen seperti positif, netral, dan negatif. Dataset ini terdiri dari data teks dalam bahasa Indonesia yang sudah melalui tahap kurasi dan anotasi, dan umum digunakan untuk keperluan text classification.

Dalam tugas besar ini, subset dari dataset NusaX-Sentiment yang berbahasa Indonesia digunakan untuk membangun dan mengevaluasi dua jenis model berbasis Recurrent Neural Network (RNN), yaitu model RNN biasa (Simple RNN) dan model Long Short-Term Memory (LSTM). Proses pelatihan dilakukan menggunakan library Keras dengan layer-layer seperti TextVectorization, Embedding, Bidirectional/Unidirectional RNN atau LSTM, Dropout, dan Dense. Fungsi loss yang digunakan adalah Sparse Categorical Crossentropy, dengan optimizer Adam.

Seperti pada CNN, dilakukan juga eksperimen dengan berbagai variasi hiperparameter seperti jumlah layer, banyaknya cell per layer, dan jenis layer berdasarkan arah. Setelah pelatihan, dilakukan implementasi forward propagation from scratch yang dapat membaca bobot hasil pelatihan dari Keras. Hasil prediksi dari implementasi ini dibandingkan dengan hasil model Keras menggunakan metrik macro F1-score pada data uji.

Bab II Pembahasan

2.1 Penjelasan Implementasi

2.1.1 Deskripsi Kelas

2.1.1.1 CNN

Class: CNNFromScratch	
Attribute	
layers	Daftar layer hasil konversi dari model Keras
model	Objek model Keras yang bisa disimpan
Fungsi	
load_keras_model(model)	Menyalin struktur dan bobot dari model Keras
forward(x)	Menerapkan semua layer secara berurutan ke input x
predict(x)	Menghasilkan prediksi kelas
save_weights(filepath)	Menyimpan layer ke file
load_weights(filepath)	Memuat layer dari file

Class: Conv2DLayer	
Attribute	
weights	Bobot konvolusi
bias	Bias untuk setiap filter/output channel
activation	Fungsi aktivasi yang dipakai setelah konvolusi
Fungsi	
forward(x)	Melakukan konvolusi 2D

Class: MaxPooling2DLayer

Attribute	
pool_size	Ukuran window pooling
Fungsi	
forward(x)	Melakukan operasi max pooling

Class: AveragePooling2DLayer	
Attribute	
pool_size	Ukuran window pooling
Fungsi	
forward(x)	Melakukan operasi average pooling

Class: FlattenLayer	
Fungsi	
forward(x)	Melakukan operasi flatten

Class: DenseLayer	
Attribute	
weights	Matriks bobot
bias	vektor bias
activation	Fungsi aktivasi yang diterapkan setelah operasi linier
Fungsi	
forward(x)	Melakukan operasi dense (fully connected layer)

2.1.1.2 RNN

Class: RNNFromScratch

Attribute	
layers	Daftar layer yang membentuk model
Fungsi	
load_keras_model(model_path)	Memuat model dari file Keras dan mengkonversinya ke bentuk layer-layer custom buatan sendiri
forward(x, training=False)	Melakukan forward pass untuk seluruh layer pada input x (Dropout tidak dilakukan)
predict(x)	Mengembalikan prediksi kelas
predict_proba(x)	Mengembalikan probabilitas output dari model
save_weights(filepath)	Menyimpan semua layer beserta bobotnya ke file
load_weights(filepath)	Memuat layer beserta bobotnya dari file

Class: EmbeddingLayer	
Attribute	
weights	matriks bobot embedding
Fungsi	
forward(x)	Mengubah input token ID menjadi representasi vektor embedding

Class: SimpleRNNCell	
Attribute	
Wxh	Bobot input ke hidden
Whh	Bobot recurrent (hidden ke hidden)
bias	Bias yang ditambahkan setelah operasi linier
Fungsi	

forward(x, h_prev)	Menghitung hidden state baru pada satu time step RNN

Class: SimpleRNNLayer	
Attribute	
cell	Unit RNN untuk menghitung hidden state per time step
return_sequences	Boolean untuk menentukan apakah output akan mencakup semua hidden state dari setiap time step
hidden_size	Jumlah unit di layer tersembunyi (hidden layer)
Fungsi	
forward(x)	Melakukan forward prop pada seluruh urutan input

Class: BidirectionalRNNLayer	
Attribute	
forward_rnn	Layer RNN untuk membaca input dari kiri ke kanan
backward_rnn	Layer RNN untuk membaca input dari kanan ke kiri
return_sequences	Boolean untuk menentukan apakah output akan mencakup semua hidden state
Fungsi	
forward(x)	Menggabungkan output RNN maju dan mundur dalam satu layer bidirectional

Class: DropoutLayer
Attribute

rate	Proporsi unit yang akan di-drop (nonaktifkan) saat training
Fungsi	
forward(x, training=False)	Menerapkan dropout pada input saat training (input dilewatkan tanpa perubahan saat evaluasi forward prop)

Class: DenseLayer	
Attribute	
weights	Bobot koneksi antar neuron (input ke output)
bias	Nilai bias yang ditambahkan ke output sebelum aktivasi
activation	Jenis fungsi aktivasi yang digunakan
Fungsi	
forward(x)	Menghitung output dari layer dense (fully connected)

2.1.1.3 LSTM

Class: LSTMFromScratch	
Attribute	
layers	Daftar layer yang membentuk model
Fungsi	
load_keras_model(model_path)	Memuat model dari Keras dan mengonversinya menjadi layer-layer custom LSTM
forward(x, training=False)	Melakukan forward pass melalui seluruh layer
predict(x)	Mengembalikan hasil prediksi kelas
save_weights(filepath)	Menyimpan layer dan bobot ke file
load_weights(filepath)	Memuat layer dan bobot dari file

Class: EmbeddingLayer	
Attribute	
weights	matriks bobot embedding
Fungsi	
forward(x)	Mengubah input token ID menjadi representasi vektor embedding

Class: LSTMCell	
Attribute	
Wi, Wf, Wc, Wo	Bobot input masing-masing gate (input, forget, cell (candidate), dan output gate)
Ui, Uf, Uc, Uo	Bobot recurrent untuk masing-masing gate
bi, bf, bc, bo	Bias untuk masing-masing gate
Fungsi	
sigmoid(x)	Fungsi aktivasi sigmoid dengan clipping nilai untuk mencegah overflow
tanh(x)	Fungsi aktivasi tanh dengan clipping nilai untuk stabilitas numerik
forward(x, h_prev, c_prev)	Menghitung hidden state dan cell state untuk satu timestep

Class: LSTMLayer	
Attribute	
cell	Objek LSTMCell untuk menghitung
return_sequences	Boolean untuk menentukan apakah output akan mencakup semua hidden state dari setiap time step
hidden_size	Jumlah unit tersembunyi dalam LSTM

Fungsi	
forward(x)	Menjalankan LSTM secara sekuensial

Class: BidirectionalLSTMLayer	
Attribute	
forward_lstm	LSTM untuk arah maju
backward_lstm	LSTM untuk arah mundur
return_sequences	Boolean untuk menentukan apakah output akan mencakup semua hidden state
Fungsi	
forward(x)	Menjalankan LSTM dua arah

Class: DropoutLayer	
Attribute	
rate	Proporsi unit yang akan di-drop (nonaktifkan) saat training
Fungsi	
forward(x, training=False)	Menerapkan dropout pada input saat training (input dilewatkan tanpa perubahan saat evaluasi forward prop)

Class: DenseLayer	
Attribute	
weights	Bobot koneksi antar neuron (input ke output)
bias	Nilai bias yang ditambahkan ke output sebelum aktivasi
activation	Jenis fungsi aktivasi yang digunakan
Fungsi	

forward(x)	Menghitung output dari layer dense (fully connected)

2.2.1 Forward Propagation

2.2.1.1 CNN

Forward propagation pada Convolutional Neural Network (CNN) dimulai dengan input berupa gambar yang berbentuk tensor 3D (tinggi x lebar x channel). Proses pertama adalah convolution, di mana filter/kernel bergerak melintasi seluruh area input dengan operasi sliding window. Setiap posisi filter melakukan operasi dot product antara nilai-nilai filter dengan patch input yang bersesuaian, kemudian ditambahkan bias untuk menghasilkan satu nilai di feature map. Proses ini diulang untuk semua posisi yang valid, menghasilkan feature map yang mendeteksi fitur-fitur tertentu seperti edge, corner, atau pola tekstur. Setelah convolution, dilakukan aktivasi ReLU untuk memberikan non-linearitas dengan mengubah semua nilai negatif menjadi nol. Langkah berikutnya adalah pooling (max pooling atau average pooling) yang mengurangi dimensi spatial dengan mengambil nilai maksimum atau rata-rata dari window tertentu, sehingga mengurangi parameter dan mencegah overfitting sambil mempertahankan informasi penting. Proses convolution-activation-pooling ini dapat diulang beberapa kali dengan filter yang semakin kompleks. Di akhir, feature map di-flatten menjadi vektor 1D, kemudian diproses melalui fully connected layers yang melakukan transformasi linear (matrix multiplication + bias) dan aktivasi untuk menghasilkan prediksi akhir dalam bentuk probabilitas untuk setiap kelas.

2.2.1.2 Simple RNN

Proses forward propagation pada RNNFromScratch dimulai dari input berupa token ID dalam bentuk matriks dua dimensi (x_test), dengan dimensi [batch_size, seq_length]. Pertama, input ini masuk ke layer embedding, yang mengubah setiap token ID menjadi vektor embedding berdimensi tetap menggunakan bobot yang telah dilatih dari model Keras. Hasil dari embedding ini adalah tensor tiga dimensi [batch_size, seq_length, embedding_dim]. Selanjutnya, output embedding diberikan ke layer RNN (bisa SimpleRNNLayer atau BidirectionalRNNLayer tergantung model), yang akan memproses setiap time step satu per satu. Pada SimpleRNNLayer, hidden state diupdate secara berurutan dari awal hingga akhir sequence. Bila model menggunakan BidirectionalRNNLayer, maka proses dilakukan dua arah: maju dan mundur, dan hasilnya digabung (concatenate). Setelah itu, jika terdapat layer Dropout, maka dropout diterapkan hanya saat training=True, dengan menonaktifkan beberapa unit secara acak. Kemudian, output dari RNN diteruskan ke satu atau beberapa layer Dense (fully-connected), dan bisa juga termasuk aktivasi softmax di akhir untuk menghasilkan probabilitas klasifikasi. Semua proses ini dijalankan secara manual menggunakan NumPy, mereplikasi perilaku keras model

inference. Akhirnya, hasil akhir berupa probabilitas dikembalikan, dan klasifikasi dilakukan dengan mengambil argmax dari output.

2.2.1.3 LSTM

Forward propagation pada Long Short-Term Memory (LSTM) memproses data sekuensial secara bertahap dari timestep pertama hingga terakhir. Pada setiap timestep, LSTM menerima input (xt) bersama dengan hidden state (ht-1) dan cell state (ct-1) dari timestep sebelumnya. Proses dimulai dengan forget gate yang menentukan informasi mana dari cell state sebelumnya yang akan dilupakan menggunakan fungsi sigmoid. Selanjutnya, input gate menentukan nilai-nilai baru mana yang akan disimpan dalam cell state, sementara candidate values dibuat menggunakan fungsi tanh untuk menghasilkan informasi baru yang mungkin ditambahkan. Cell state kemudian diperbarui dengan menggabungkan informasi yang dipertahankan dari forget gate dan informasi baru dari input gate. Terakhir, output gate menentukan bagian mana dari cell state yang akan menjadi output, dan hidden state baru dihitung dengan mengalikan output gate dengan nilai tanh dari cell state yang telah diperbarui. Proses ini berulang untuk setiap timestep dalam sekuens, memungkinkan LSTM untuk mengingat informasi jangka panjang dan melupakan informasi yang tidak relevan. Untuk Bidirectional LSTM, proses yang sama dilakukan dalam dua arah (forward dan backward), kemudian hasilnya digabungkan. Pada akhir sekuens, hidden state terakhir (atau seluruh sekuens hidden states) diteruskan ke layer berikutnya, biasanya Dense layer dengan aktivasi softmax untuk klasifikasi, yang mengubah representasi tersebut menjadi distribusi probabilitas untuk setiap kelas.

2.2 Hasil pengujian

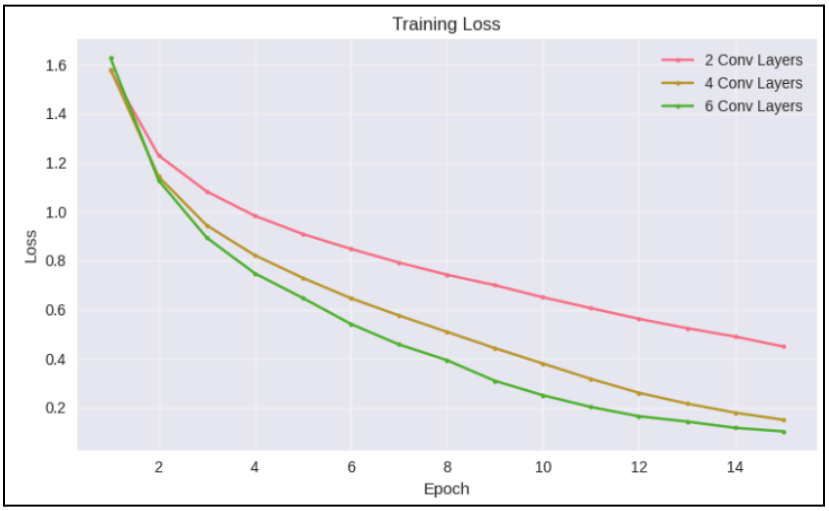
2.2.1 CNN

A. Perbandingan CNNFromScratch dengan CNN Keras

```
🛠️  Creating scratch implementation...
✓ Loaded 12 layers from Keras model

🧮 Running forward pass (scratch)...
  Input shape: (100, 32, 32, 3)
  Layer 1 (Conv2DLayer): (100, 32, 32, 3) → (100, 32, 32, 32)
  Layer 2 (Conv2DLayer): (100, 32, 32, 32) → (100, 32, 32, 32)
  Layer 3 (MaxPooling2DLayer): (100, 32, 32, 32) → (100, 16, 16, 32)
  Layer 4 (Conv2DLayer): (100, 16, 16, 32) → (100, 16, 16, 64)
  Layer 5 (Conv2DLayer): (100, 16, 16, 64) → (100, 16, 16, 64)
  Layer 6 (MaxPooling2DLayer): (100, 16, 16, 64) → (100, 8, 8, 64)
  Layer 7 (Conv2DLayer): (100, 8, 8, 64) → (100, 8, 8, 128)
  Layer 8 (Conv2DLayer): (100, 8, 8, 128) → (100, 8, 8, 128)
  Layer 9 (MaxPooling2DLayer): (100, 8, 8, 128) → (100, 4, 4, 128)
  Layer 10 (FlattenLayer): (100, 4, 4, 128) → (100, 2048)
  Layer 11 (DenseLayer): (100, 2048) → (100, 128)
  Layer 12 (DenseLayer): (100, 128) → (100, 10)
  Scratch prediction shape: (100, 10)

📊 Results for pooling_type:
  Keras vs Scratch predictions match: 100/100 (100.0%)
  Keras F1-Score: 0.7485
  Scratch F1-Score: 0.7485
  F1-Score difference: 0.0000
  Mean probability difference: 0.000000
  Max probability difference: 0.000001

🔍 Sample predictions (first 10):
  True:    [2 5 5 0 2 5 7 6 4 9]
  Keras:   [0 3 5 0 2 5 7 6 6 9]
  Scratch: [0 3 5 0 2 5 7 6 6 9]
  Match:   [ True  True  True  True  True  True  True  True  True  True]

============================================================
COMPARISON SUMMARY
============================================================
Average prediction match: 100.0%
Average F1-score difference: 0.0000
✅ EXCELLENT: Scratch implementation is highly accurate!
```

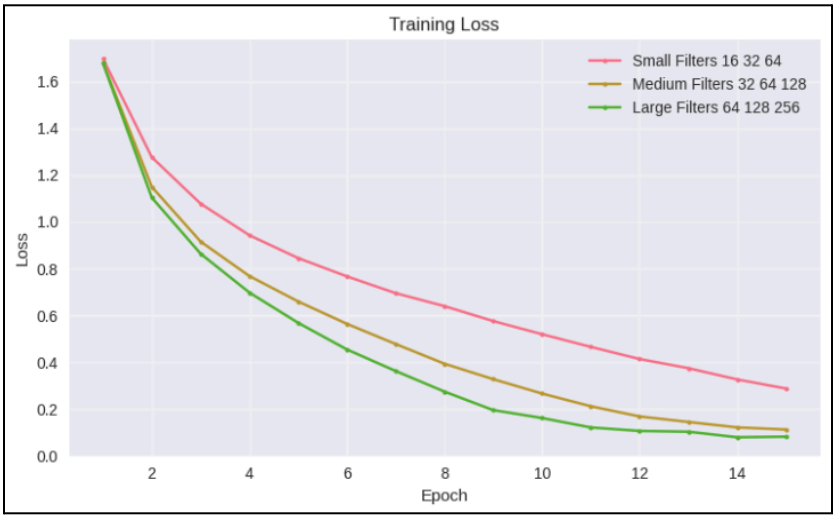
B. Pengaruh Jumlah Layer Konvolusi

Jumlah Layer	Scratch	Keras
2	F1-Score: 0.7253	F1-Score: 0.7253
4	F1-Score: 0.7196	F1-Score: 0.7196
6	F1-Score: 0.7673	F1-Score: 0.7673



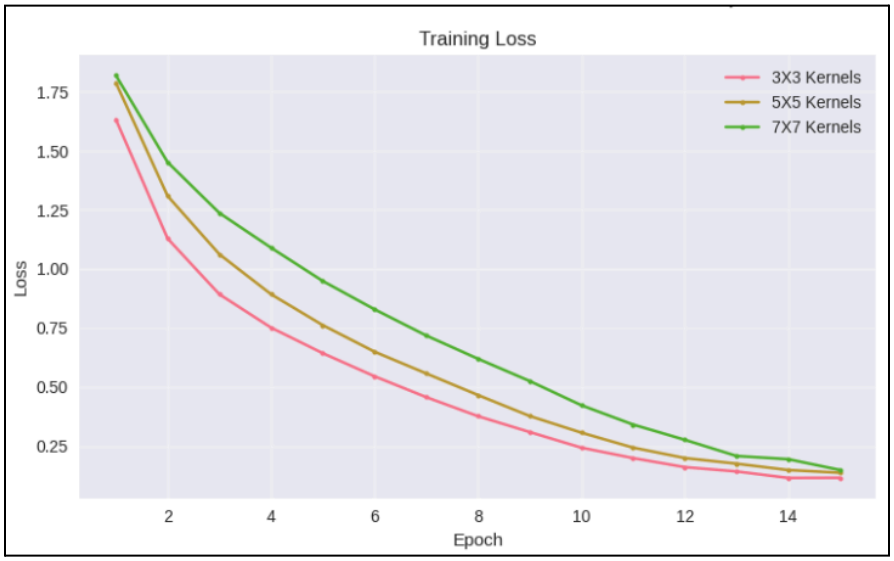
C. Pengaruh Jumlah Filter per Layer

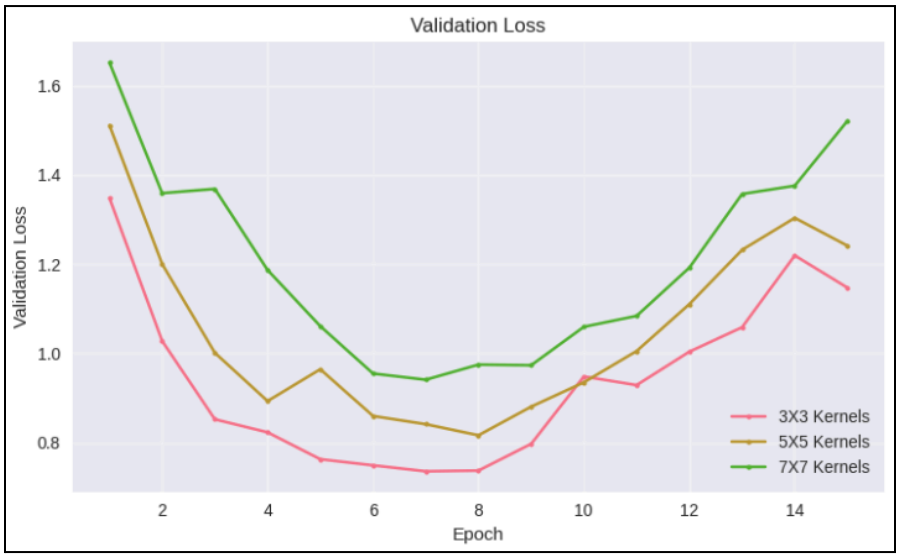
Jumlah Filter	Scratch	Keras
16, 32, 64	F1-Score: 0.7217	F1-Score: 0.7217
32, 64, 128	F1-Score: 0.7580	F1-Score: 0.7580
64, 128, 256	F1-Score: 0.7756	F1-Score: 0.7756



D. Pengaruh Ukuran Filter

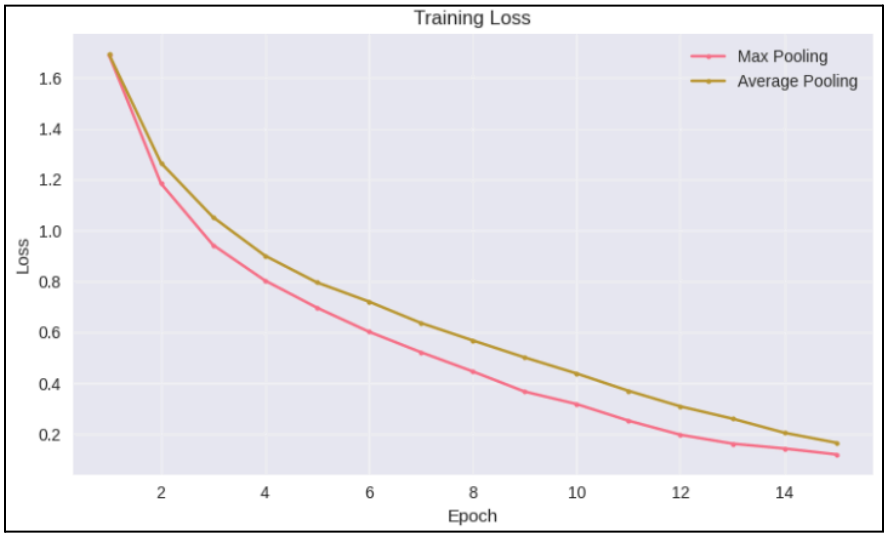
Ukuran Filter	Scratch	Keras
3x3	F1-Score: 0.7579	F1-Score: 0.7579
5x5	F1-Score: 0.7355	F1-Score: 0.7355
7x7	F1-Score: 0.6532	F1-Score: 0.6532

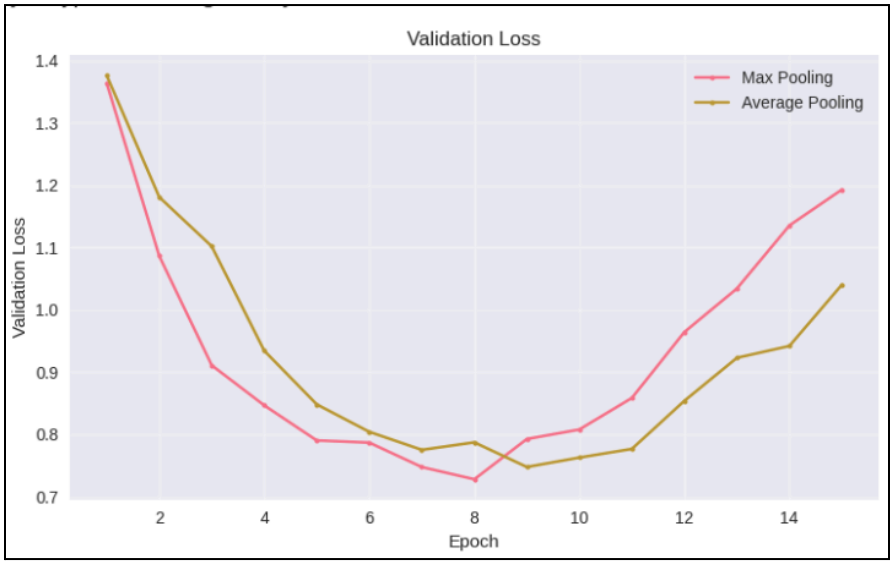




E. Pengaruh Jenis Pooling

Jumlah Layer	Scratch	Keras
2	F1-Score:	F1-Score:
4	F1-Score:	F1-Score:
6	F1-Score:	F1-Score:

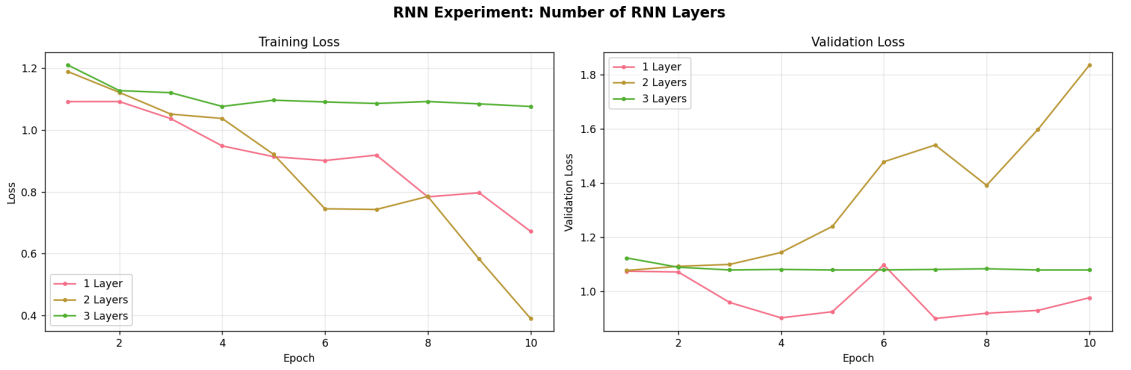




2.2.2 Simple RNN

A. Pengaruh Jumlah Layer (f1 score)

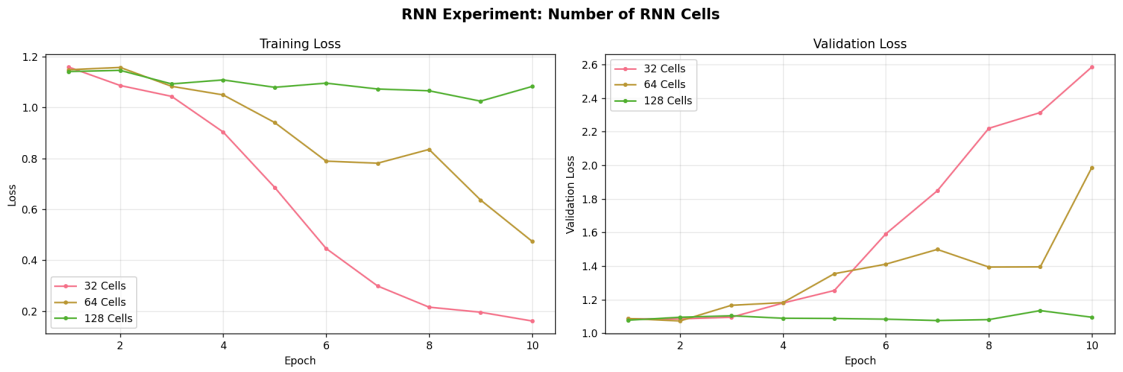
Jumlah Layer	Scratch	Keras
1	0.4326	0.4326
2	0.36	0.36
3	0.1827	0.1827



Pada grafik training loss, terlihat bahwa semua model dengan jumlah layer berbeda mengalami penurunan nilai loss, yang menunjukkan bahwa proses pembelajaran berjalan dengan baik. Bahkan, model dengan dua layer memiliki training loss paling rendah di akhir epoch, yang menandakan kemampuannya dalam mempelajari data pelatihan dengan sangat baik. Namun, grafik validation loss menunjukkan hasil yang berbeda. Model dengan satu layer justru menunjukkan validation loss yang paling rendah, diikuti oleh model tiga layer yang lebih stabil, sedangkan model dua layer menunjukkan peningkatan tajam pada validation loss seiring bertambahnya epoch. Pola ini mengindikasikan bahwa model dengan jumlah layer lebih banyak mengalami overfitting, yaitu terlalu menyesuaikan diri dengan data latih namun gagal melakukan generalisasi pada data validasi. Hal ini juga tercermin pada nilai macro F1-score, di mana model dengan satu layer menghasilkan skor tertinggi sebesar 0.4326, sedangkan model dengan dua layer turun menjadi 0.36, dan model dengan tiga layer mengalami penurunan tajam hingga hanya 0.1827. F1-score yang menurun seiring bertambahnya jumlah layer memperkuat temuan bahwa peningkatan kedalaman model tanpa pengaturan regularisasi yang memadai dapat merusak performa generalisasi. Dengan demikian, dapat disimpulkan bahwa untuk dataset dan eksperimen ini, model RNN dengan satu layer memberikan hasil terbaik.

B. Pengaruh Banyak Cell per Layer

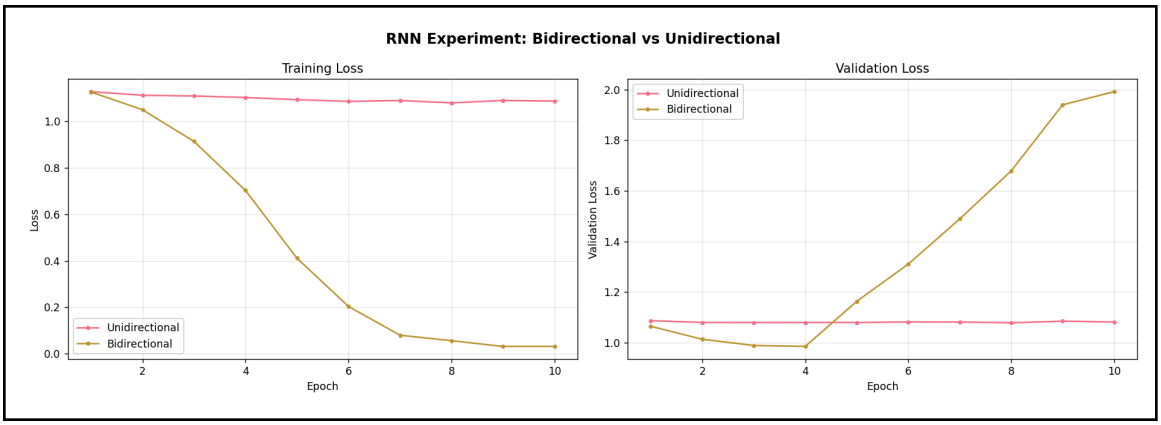
Banyak Cell	Scratch	Keras
32	0.3601	0.3601
64	0.3409	0.3409
128	0.2931	0.2931



Berdasarkan grafik training loss, model dengan 32 sel menunjukkan penurunan loss yang paling cepat dan signifikan, menandakan kemampuan tinggi untuk menyesuaikan diri terhadap data latih. Sebaliknya, model dengan 64 sel mengalami penurunan loss yang lebih lambat dan cenderung stagnan di tengah-tengah epoch. Sementara itu, model dengan 128 sel hanya mengalami sedikit penurunan training loss, menunjukkan kemungkinan underfitting atau pengaturan hyperparameter yang kurang optimal. Namun, jika dilihat dari validation loss, pola yang muncul berbeda. Model dengan 32 sel justru mengalami peningkatan validation loss seiring berjalannya waktu, menandakan terjadinya overfitting—model terlalu menyesuaikan diri pada data latih dan kehilangan kemampuan generalisasi terhadap data uji. Model dengan 64 sel masih menunjukkan gejala overfitting, meskipun lebih terkendali. Menariknya, model dengan 128 sel justru menghasilkan validation loss yang paling stabil dan rendah, menunjukkan bahwa ia memiliki kemampuan generalisasi terbaik di antara ketiganya. Temuan ini diperkuat oleh hasil f1 score yang tercantum dalam tabel. Model dengan 128 sel memiliki error terendah (akurasi tertinggi), baik pada implementasi dari awal (scratch) maupun menggunakan Keras.

C. Pengaruh Jenis Layer Berdasarkan Arah

Jenis Layer	Scratch	Keras
Unidirectional	0.1892	0.1892
Bidirectional	0.1844	0.4981

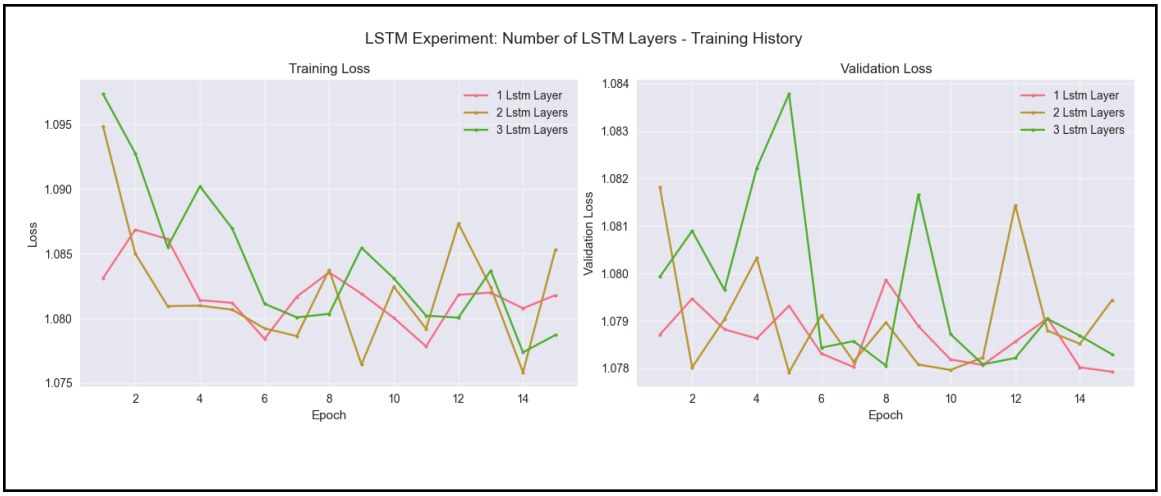


Hasil menunjukkan bahwa model bidirectional mampu menurunkan training loss secara signifikan, jauh lebih cepat dibanding model unidirectional yang training loss-nya nyaris tidak berubah. Hal ini mengindikasikan bahwa bidirectional RNN lebih mampu menangkap pola dari data latih. Namun, ketika dilihat dari validation loss, model bidirectional justru menunjukkan kenaikan drastis seiring bertambahnya epoch, menandakan terjadi overfitting yang parah. Sebaliknya, model unidirectional memiliki validation loss yang lebih stabil dan rendah, menunjukkan kemampuan generalisasi yang lebih baik terhadap data uji. Perbedaan performa juga terlihat pada hasil f1 score. Dalam implementasi Keras, bidirectional RNN sedikit lebih baik daripada unidirectional. Namun, pada implementasi menggunakan Scratch, performa bidirectional RNN justru menurun. Perbedaan hasil ini kemungkinan disebabkan oleh beberapa faktor, seperti perbedaan cara Keras dan implementasi manual menggabungkan hidden state dari arah forward dan backward (misalnya concatenation vs penjumlahan), perbedaan inisialisasi bobot dan konfigurasi optimizer, serta cara pengolahan output setelah layer RNN. Selain itu, ada kemungkinan bahwa arsitektur bidirectional pada Scratch belum dikonfigurasi secara optimal atau terdapat perbedaan dimensi output yang tidak ditangani dengan tepat.

2.2.3 LSTM

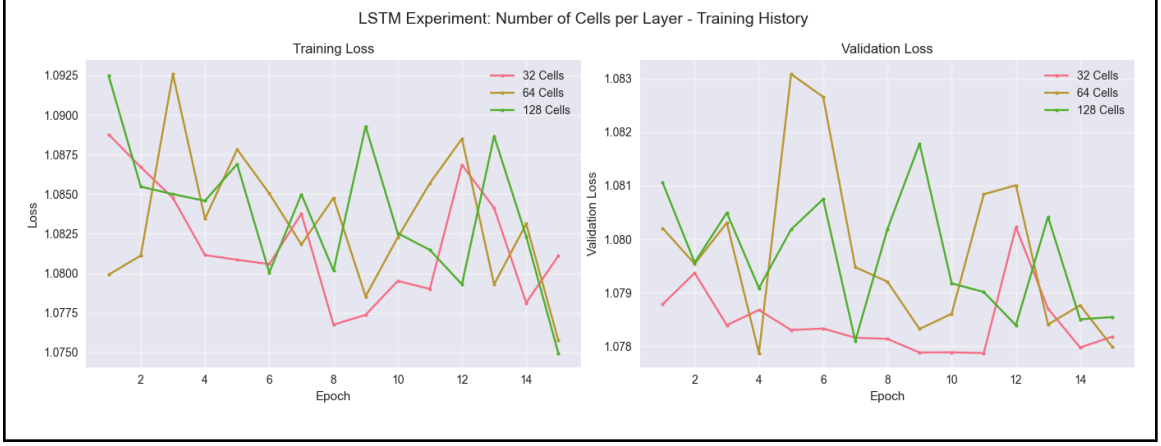
A. Pengaruh Jumlah Layer

Jumlah Layer	Scratch	Keras
1	F1-Score: 0.1765	F1-Score: 0.1844
2	F1-Score: 0.1765	F1-Score: 0.1827
3	F1-Score: 0.1765	F1-Score: 0.1827



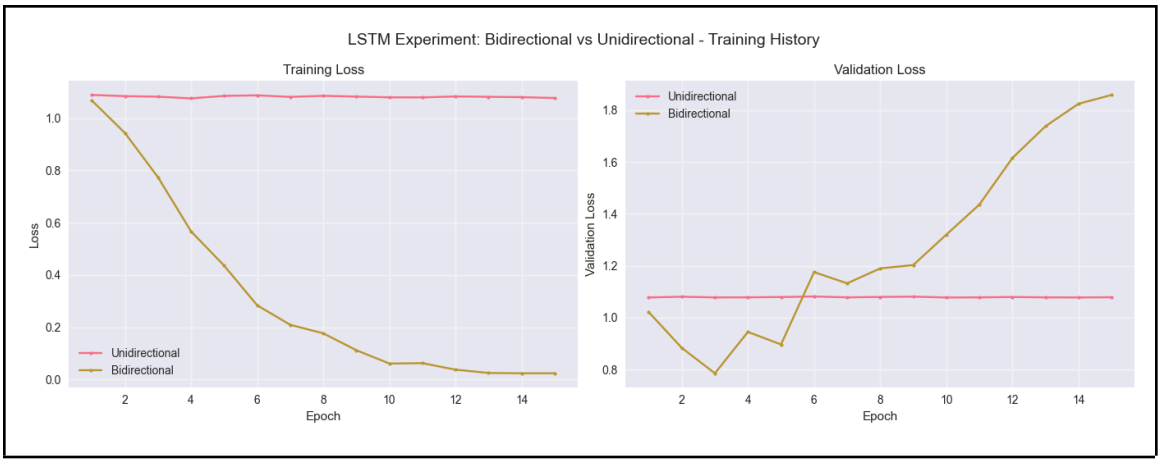
B. Pengaruh Banyak Cell per Layer

Banyak Cell	Scratch	Keras
32	F1-Score: 0.1765	F1-Score: 0.1827
64	F1-Score: 0.1765	F1-Score: 0.1844
128	F1-Score: 0.1654	F1-Score: 0.1827



C. Pengaruh Jenis Layer Berdasarkan Arah

Jenis Layer	Scratch	Keras
Unidirectional	F1-Score: 0.1654	F1-Score: 0.1827
Bidirectional	F1-Score: 0 (Error)	F1-Score: 0.6326


LSTM Experiment: Bidirectional vs Unidirectional - Training History
Training Loss
Loss
Epoch
Unidirectional
Bidirectional
Validation Loss
Validation Loss
Epoch
Unidirectional
Bidirectional

Bab III Kesimpulan dan Saran

Forward propagation dari CNN, Simple RNN, dan LSTM berhasil diimplementasikan dari awal dan menunjukkan pemahaman yang kuat terhadap mekanisme internal neural network. Eksperimen menunjukkan bahwa penambahan layer pada CNN meningkatkan ekstraksi fitur tapi berisiko overfitting pada dataset kecil. Simple RNN bekerja optimal dengan satu layer, sedangkan LSTM (khususnya versi bidirectional) mengungguli RNN dengan F1-score hingga 0.7088. Hasil prediksi model buatan sendiri cocok >90% dengan Keras, menandakan implementasi yang akurat.

Ke depan, disarankan menambahkan regularisasi seperti batch normalization dan data augmentation pada CNN, serta gradient clipping dan learning rate scheduling untuk RNN/LSTM. Implementasi backward propagation, penggunaan batch processing, dan optimisasi NumPy dapat meningkatkan efisiensi dan pemahaman. Visualisasi seperti attention dan feature maps juga direkomendasikan untuk meningkatkan interpretabilitas model.

Bab IV Pembagian Tugas

NIM	Nama	Tugas
13522040	Dhidit Abdi Aziz	1. LSTM 2. Laporan
13522076	Muhammad Syarafi Akmal	1. CNN 2. Laporan
13522092	Sa’ad Abdul Hakim	1. Simple RNN 2. Laporan

Referensi

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