Laporan Tugas Besar 2

IF3270 Pembelajaran Mesin

Convolutional Neural Network dan Recurrent Neural Network

Dipersiapkan oleh:

1. Panji Sri Kuncara Wisma 13522028
2. Diero Arga Purnama 13522056
3. Muhammad Rifki Virziadeili Harisman 13522120

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung,

Jl. Ganesha 10, Banding 40132

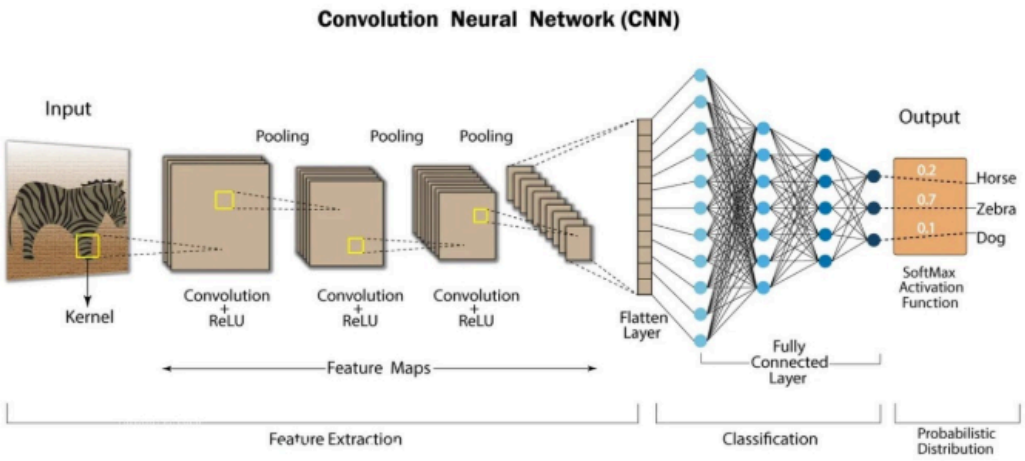
DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
DESKRIPSI PERSOALAN

1.1. CNN



Gambar 1.1 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) merupakan salah satu arsitektur pembelajaran mesin yang banyak digunakan untuk pemrosesan data visual, seperti pengenalan gambar dan klasifikasi objek. CNN dirancang untuk mempelajari fitur-fitur spasial dari data masukan dua dimensi melalui layer-layer khusus.

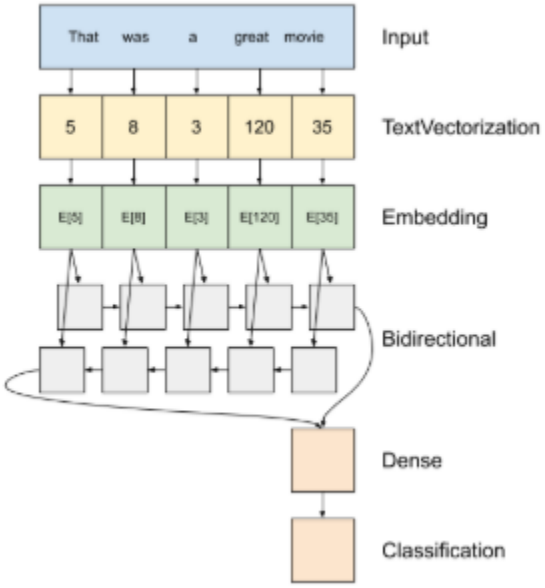
CNN terdiri dari beberapa jenis layer utama, yaitu layer konvolusi, layer pooling, layer flatten, dan layer dense (fully connected). Lapisan konvolusi menggunakan filter atau kernel yang bergerak melintasi input untuk mengekstraksi fitur penting. Operasi ini akan memberikan hasil dalam bentuk feature map. Setelah itu, feature map akan diproses lebih lanjut menggunakan fungsi aktivasi, seperti ReLU.

Kemudian, data melewati layer pooling, yang bertugas untuk mereduksi dimensi dari fitur sambil tetap mempertahankan informasi yang penting. Proses ini membantu menyederhanakan model dan mempercepat perhitungan. Hasil dari layer pooling akan di-flatten menjadi vektor satu dimensi pada layer flatten. Hal ini dilakukan agar data dapat dimasukkan ke dalam layer dense, yang menerima vektor sebagai masukan. Layer dense akan menggabungkan semua fitur yang telah diproses dan menghasilkan output akhir berupa klasifikasi atau prediksi.

Pada tugas besar ini, kami diminta untuk membuat model CNN menggunakan library Keras dan melakukan pelatihan menggunakan dataset CIFAR-10. Bobot yang

didapatkan melalui pelatihan akan digunakan untuk melakukan forward propagation yang akan dibuat dari awal (*from scratch*). Hasil dari forward propagation yang kami buat kemudian akan dibandingkan dengan hasil forward propagation dari library Keras.

1.2. Simple RNN



Gambar 1.2 Simple RNN (Recurrent Neural Network)

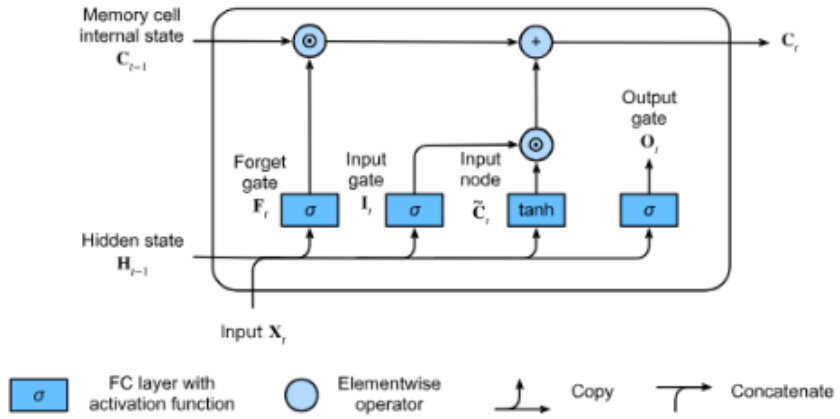
Simple Recurrent Neural Network (RNN) merupakan salah satu arsitektur pembelajaran mesin yang digunakan secara luas dalam pemrosesan data *sequential* (urutan), seperti teks. RNN dirancang untuk dapat mempertimbangkan konteks temporal atau urutan dari data dengan menyimpan informasi dari langkah sebelumnya dalam memori internal.

Arsitektur RNN umumnya terdiri dari beberapa jenis layer utama, yaitu embedding layer, recurrent layer, dropout layer, dan dense layer. Embedding layer mengubah token teks menjadi representasi vektor, dan recurrent layer memproses urutan tersebut dengan mempertimbangkan konteks dan urutan data. Dropout layer digunakan untuk mencegah overfitting, dan dense layer menghasilkan output akhir berupa klasifikasi.

Pada tugas besar ini, kami diminta untuk membuat model RNN menggunakan library Keras dan melakukan pelatihan menggunakan dataset NusaX-Sentiment (Bahasa Indonesia). Bobot yang diperoleh dari pelatihan ini kemudian digunakan untuk melakukan forward propagation yang akan diimplementasikan dari awal. Hasil dari

forward propagation yang kami buat kemudian akan dibandingkan dengan hasil forward propagation dari library Keras.

1.3. LSTM



Gambar 1.3 LSTM (Long Short-Term Memory)

Long Short-Term Memory (LSTM) merupakan salah satu jenis arsitektur RNN yang dirancang untuk mengatasi masalah utama RNN, yaitu *Vanishing Gradient Problem*. LSTM memiliki mekanisme khusus berupa cell state (Ct) yang mampu menyimpan informasi penting dalam jangka waktu panjang selama proses pelatihan.

Arsitektur LSTM pada umumnya terdiri dari beberapa layer utama, yaitu embedding layer, LSTM layer, dropout layer, dan dense layer. Struktur ini menyerupai RNN, namun perbedaan utamanya terletak pada LSTM layer, yang berfungsi untuk memproses urutan data panjang dengan mempertahankan konteks penting dari setiap langkah input.

Pada tugas besar ini, kami diminta untuk membangun dan melatih model LSTM menggunakan library Keras dengan dataset teks Bahasa Indonesia. Setelah pelatihan, bobot model digunakan untuk membuat modul forward propagation yang diimplementasikan dari awal. Hasil dari forward propagation yang kami buat kemudian akan dibandingkan dengan hasil forward propagation dari library Keras.

BAB II

PEMBAHASAN

2.1. Penjelasan Implementasi

2.1.1. Deskripsi Kelas

a. CNN

Kelas CNN merupakan representasi dari sebuah model Convolutional Neural Network dengan kemampuan untuk:

- Membangun model CNN menggunakan Keras.
- Melatih model menggunakan dataset CIFAR-10.
- Menyimpan bobot hasil latihan untuk melakukan forward propagation.

Tabel 2.1.1.1. Atribut Kelas CNN

Atribut	Tipe	Deskripsi
n_conv_layer	int	Jumlah layer konvolusi yang dimiliki oleh model.
filters_per_layer	list[int]	Jumlah filter pada setiap layar konvolusi.
kernel_sizes	list[int]	Ukuran kernel yang digunakan pada setiap layer konvolusi.
pooling_type	string	Tipe pooling yang digunakan, dapat berupa “max” atau “average”
n_epoch	int	Jumlah epoch yang digunakan ketika training.
weights_dir	string	Folder tempat penyimpanan dan pemuatan data.
model	tf.keras. Model	Model CNN yang dibuat menggunakan Keras.
history	History	Objek history hasil training.
f1_score	float	Nilai Macro F1-Score yang didapatkan model saat evaluasi.

Tabel 2.1.1.2. Metode Kelas CNN

Metode	Deskripsi
from_config(config)	Membuat objek CNN berdasarkan konfigurasi yang diberikan.
build()	Membuat model CNN menggunakan Keras.
train(x_train, y_train, x_val, y_val)	Melatih model menggunakan data pelatihan dan validasi.
evaluate(y_pred_probs, y_test)	Menghitung nilai Macro F1-Score menggunakan hasil forward propagation.
save()	Menyimpan bobot model ke file .h5.
load(filename)	Memuat bobot model dari file .h5.
forward_scratch(input)	Melakukan forward propagation terhadap input yang diberikan.

Tabel 2.1.1.3. Fungsi Forward Propagation CNN

Metode	Deskripsi
calc_output_size(size, filter_size, padding, stride)	Menghitung ukuran output feature map menggunakan rumus $V = \frac{W - F + 2P}{S} + 1$
pad_x(x, pad)	Melakukan padding terhadap x.
relu(x)	Mengembalikan hasil ReLU dari x.
softmax(x)	Mengembalikan hasil softmax dari x.
conv2d(x, W, b, stride, padding)	Forward propagation layer Conv2D.
pooling(x, pool_size, stride, type)	Forward propagation layer Pooling.
flatten(x)	Forward propagation layer Flatten.
dense(x, W, b, activation)	Forward propagation layer Dense.

b. SimpleRNNModel

Kelas SimpleRNNModel merupakan representasi dari sebuah model Recurrent Neural Network sederhana dengan kemampuan untuk:

- Membangun model SimpleRNN menggunakan Keras dengan opsi untuk lapisan tunggal atau *stacked*, serta unidirectional atau bidirectional.
- Melatih model menggunakan dataset teks
- Menyimpan dan memuat bobot hasil latihan untuk melakukan *forward propagation*.
- Melakukan forward propagation secara manual (*scratch*) menggunakan bobot yang telah dilatih.

Tabel 2.1.1.4. Atribut Kelas SimpleRNNModel

Atribut	Tipe	Deskripsi
vocab_size	int	Ukuran vocabulary (jumlah kata unik yang dipertimbangkan oleh layer embedding).
embedding_dim	int	Dimensi output embedding.
rnn_unit	list[int]	Daftar jumlah cell pada setiap layer RNN.
bidirectional	list[bool]	Daftar boolean yang menunjukkan apakah setiap layer RNN bersifat bidirectional.
num_classes	int	Jumlah kelas output untuk tugas klasifikasi.
sequence_length	int	Panjang sekuens input.
model	tf.keras.Model or None	Instance model Keras
history	History or None	Objek history hasil pelatihan.
f1_macro	float or None	Nilai Macro F1-Score yang dihitung saat evaluasi pada data uji.
w_dir	pathlib.Path	Direktori (path) untuk menyimpan dan memuat bobot model.
_dropout_rate	float	Tingkat dropout yang digunakan pada layer dropout.

_learning_rate	float	Learning rate yang digunakan oleh optimizer.

Tabel 2.1.1.5. Method Kelas SimpleRNNModel

Metode	Deskripsi singkat
__init__	Inisialisasi parameter dasar dan setup direktori bobot.
build_model	Membangun dan meng‑compile model Keras sesuai konfigurasi.
train	Melatih model dengan data training dan validasi.
evaluate	Mengevaluasi model di data uji dan hitung Macro F1-score.
save_weights	Menyimpan bobot model ke file di direktori w_dir.
load_weights	Memuat bobot model dari file di direktori w_dir.
load_layer_weights	Mengambil bobot layer ke-layer_idx sebagai array float32.
forward_scratch	Menjalankan forward pass manual (scratch) untuk tiap input.

Tabel 2.1.1.6. Fungsi Forward Propagation RNN

Fungsi	Deskripsi
tanh(x)	Fungsi Aktivasi
softmax(x)	Fungsi Aktivasi
linear(x)	Fungsi Aktivasi
embedding_forward(sekuens, mat)	Lookup embedding per token.
rnn_cell_forward(x_t, h_prev, …)	Forward cell RNN satu step.
rnn_layer_forward(...,	Forward pass penuh RNN.

return_sequences=False)	
dense_forward(x, W, b, fn)	Layer dense.
dropout_forward(x)	No-op dropout.
bidirectional_rnn_layer_forward(..., merge_mode, return_sequences=False)	Forward bidirectional RNN

c. LSTM

Kelas LTSMModel merupakan representasi dari sebuah model Long Short Term Memory dengan kemampuan untuk:

i. Membangun model LTSM menggunakan Keras dengan opsi untuk lapisan tunggal atau stacked, serta unidirectional atau bidirectional.
ii. Melatih model menggunakan dataset teks
iii. Menyimpan dan memuat bobot hasil latihan untuk melakukan forward propagation.
iv. Melakukan forward propagation secara manual (scratch) menggunakan bobot yang telah dilatih

Tabel 2.1.1.7. Tabel Atribut LSTM

Attribute	Tipe	Deskripsi
vocab_size	int	Ukuran vocab
embbeding_dim	int	Dimensi embedding layer
units	list	List integer yang merepresentasikan jumlah unit tiap layer
num_classes	int	Jumlah kelas output
sequence_length	int	Panjang input sequences
weights_dir	pathlib.Path	Direktori untuk menyimpan weight
bidirectional	List or bool	List boolean yang mengindikasikan layer

		tersebut bidirectional atau tidak
_dropout_rate	float	Dropout rate
_learning_rate	float	Learning rate
history	Objek tensorflow	Objek untuk menyimpan history
f1_score	float	F1 score hasil evaluasi
model	Objek tensorflow	Model keras saat ini
scratch_predictor	LSTMScratch	Objek LSTM yang digunakan untuk prediksi menggunakan forward propagation (from scratch)

Tabel 2.1.1.8. Tabel Method LSTM

Fungsi	Deskripsi
__init__	Inisiasi model
from_config	Membuat model LSTMModel dari konfigurasi
build	Membangun arsitektur LSTM menggunakan keras
train	Melatih model
evaluate	Mengevaluasi model LSTM pada data tes megngunakan metrik F1
save	Menyimpan bobot model
load	Memuat bobot model lstm dari file
predict_with_scratch	Melakukan prediksi menggunakan implementasi forward propagation LSTM (from scratch)
evaluate_from_scratch	Mengevaluasi model LSTM menggunakan implementasi forward

	propagation LSTM (from scratch)
compare_predictions	Membandingkan hasil prediksi antara implementasi Keras dan from scratch

2.1.2. Forward Propagation

a. CNN

Metode forward_scratch(input) melakukan proses forward propagation secara manual tanpa menggunakan Keras, melainkan menggunakan fungsi-fungsi yang didefinisikan oleh modul Models.CNN.forward_layers. Pada forward propagation, semua data pada dataset yang dimasukkan akan melewati semua layer pada CNN secara satu per satu.

Proses akan dimulai dengan memasukkan data ke layer konvolusi (Conv2D). Di sini, data akan dibagi menjadi beberapa "patch" yang sesuai dengan ukuran kernel (filter). Bagian-bagian tersebut kemudian akan dikalikan dengan kernel lalu ditambahkan bias. Setelah itu, fungsi aktivasi ReLU diterapkan untuk menghilangkan nilai negatif. Pada implementasi ini, padding digunakan untuk memastikan bahwa ukuran feature map yang dihasilkan akan sama dengan ukuran data yang dimasukkan. Proses konvolusi ini akan dilakukan berulang sebanyak jumlah layer konvolusi yang dimiliki oleh model.

Setelah melewati layer konvolusi, data akan masuk ke dalam layer pooling, yang berfungsi untuk mereduksi dimensi data. Data akan dibagi-bagi lagi menjadi beberapa "patch" berukuran dua kali dua. Setelah dibagi menjadi beberapa bagian, jika tipe pooling yang digunakan adalah max pooling, nilai yang akan diambil adalah nilai maksimum, sedangkan jika tipe pooling adalah average pooling, nilai yang akan diambil adalah nilai rata-rata dari patch. Tujuan dari layer ini adalah untuk mengurangi kompleksitas komputasi dan mengekstraksi fitur yang penting.

Kemudian, hasil dari pooling akan diubah menjadi vektor satu dimensi di layer flatten. Hal ini dilakukan karena layer dense membutuhkan input dalam bentuk vektor.

Data yang telah di-flatten akan dimasukkan ke dalam dense layer pertama, di mana vektor masukan akan dikalikan dengan matriks bobot dan ditambahkan bias, lalu diproses dengan fungsi aktivasi ReLU.

Terakhir, hasil dari dense layer pertama akan dimasukkan ke dalam dense layer output, yang akan memiliki 10 neuron, sesuai dengan jumlah kelas yang ingin diprediksi.

Fungsi yang digunakan pada layer ini adalah softmax, yang akan mengubah hasil akhir menjadi distribusi probabilitas.

Berikut merupakan perbandingan antara hasil dari forward propagation from scratch yang telah dibuat dengan forward propagation dari library Keras.

Tabel 2.1.2.1 Macro F1-Score | Keras vs Scratch

Model	Macro F1-Score
Keras	0.7210142229221305
From Scratch	0.7210142229221305

Berdasarkan hasil yang didapatkan, dapat disimpulkan bahwa forward propagation from scratch yang dibuat memberikan hasil yang sama persis dengan forward propagation yang disediakan oleh library Keras.

b. Simple RNN

Proses forward propagation pada RNN bekerja dengan melakukan pemrosesan sekuens input secara bertahap, di mana informasi dari langkah waktu sebelumnya dibawa ke langkah waktu berikutnya, memungkinkan model untuk menangkap konteks dalam data. Sebagai catatan penting, pelatihan menggunakan Keras menggunakan ukuran batch 128 karena dataset relatif sedikit.

Setiap sampel input, yang berupa sekuens token, pertama akan melalui layer embedding. Di sini, token-token numerik dari sekuens input dikonversi menjadi representasi vektor menggunakan matriks bobot yang telah dilatih dan dimuat dari layer embedding Keras.

Selanjutnya, sekuens vektor embedding ini dialirkan ke satu atau lebih layer RNN. Jika model dikonfigurasi dengan beberapa layer RNN, output dari satu layer RNN akan menjadi input bagi layer berikutnya. Jika layer tersebut bersifat bidirectional, maka pemrosesan akan dilakukan dalam dua arah yaitu maju (dari awal hingga akhir sekuens) dan mundur (dari akhir hingga awal sekuens). *Hidden state* awal untuk kedua arah ini diinisialisasi sebagai vektor nol. Fungsi *bidirectional_rnn_layer_forward* kemudian akan mengkalkulasi output dengan menggabungkan hasil dari kedua arah tersebut. Sebaliknya, jika layer bersifat unidirectional, pemrosesan hanya dilakukan dari awal hingga akhir sekuens, dengan *hidden state* awal yang juga diinisialisasi sebagai vektor nol dan fungsi *rnn_layer_forward* yang akan menanganinya.

Setelah melewati semua layer RNN, output kemudian diproses oleh layer *dropout*. Dalam konteks implementasi dari *scratch* untuk fase inferensi atau evaluasi, fungsi *dropout_forward* ini hanya bertindak untuk meneruskan inputnya tanpa perubahan.

Tahap akhir dari *forward propagation* adalah layer *dense*. Vektor representasi yang dihasilkan dari layer RNN terakhir menjadi input bagi layer ini. Bobot dan bias dari layer dense Keras yang telah dilatih dimuat, dan fungsi *dense_forward* dipanggil. Fungsi aktivasi "softmax" diterapkan pada output dari layer dense ini untuk menghasilkan distribusi probabilitas atas semua kelas target. Dengan demikian, untuk setiap sekuens input, model akan menghasilkan serangkaian probabilitas yang menunjukkan kemungkinan sekuens tersebut termasuk dalam masing-masing kelas. Proses ini diulang untuk setiap sekuens dalam batch input.

Berikut adalah perbandingan hasil antara *forward propagation* dari *scratch* dibandingkan dengan keras

Tabel 2.1.2.2 Macro F1-Score | Keras vs Scratch

Model	Macro F1-Score (Keras)	Macro F1-Score (Scratch)
SimpleRNN Unidirectional	0.4411	0.4411
SimpleRNN Bidirectional	0.4732	0.4732

Berdasarkan hasil percobaan dapat disimpulkan bahwa *forward propagation* dari *scratch* yang dibuat memberikan hasil yang sama persis dengan *forward propagation* yang disediakan oleh library Keras untuk model Unidirectional maupun Bidirectional SimpleRNN.

c. LSTM

Pada kelas LSTMModel, model yang dibangun dengan Keras, proses forward propagation berjalan secara otomatis di balik layar saat kami meminta model untuk memprediksi atau mengevaluasi data. Ini seperti memberikan tugas ke asisten yang sangat cerdas; kami hanya perlu memberi perintah, dan dia akan mengurus semua detailnya.

Pertama, data input yang berupa urutan kata (dalam bentuk angka/ID) akan melewati lapisan Embedding. Di sini, setiap kata diubah menjadi representasi numerik yang lebih bermakna, sebuah "vektor" padat. Bayangkan seperti kamus canggih yang mengubah setiap kata menjadi deskripsi karakteristiknya. Setelah itu, data yang sudah menjadi vektor ini masuk ke dalam serangkaian lapisan LSTM.

Lapisan LSTM ini adalah inti dari model. Mereka dirancang untuk memahami urutan data dan mengingat informasi penting dari langkah-langkah sebelumnya dalam urutan tersebut. Jika kami mengaturnya sebagai Bidirectional LSTM, model akan memproses urutan tersebut dua kali: sekali dari awal hingga akhir, dan sekali lagi dari akhir hingga awal. Ini membantu model menangkap konteks dari kedua arah. Penting untuk dicatat bahwa hampir semua lapisan LSTM di tengah akan mengembalikan "urutan" output (artinya, mereka memberikan hasil untuk setiap langkah waktu dalam urutan), kecuali lapisan LSTM terakhir. Lapisan LSTM terakhir ini hanya akan mengembalikan output dari "langkah waktu terakhir", yang merupakan ringkasan dari seluruh urutan.

Setelah setiap lapisan LSTM, ada lapisan Dropout yang berfungsi sebagai "pencegah overfitting". Selama proses prediksi, lapisan ini pada dasarnya "nonaktif" dan tidak mengubah data, memastikan bahwa model membuat prediksi berdasarkan semua informasi yang dipelajari.

Terakhir, output dari lapisan LSTM terakhir akan masuk ke lapisan Dense atau lapisan klasifikasi. Lapisan ini mengambil ringkasan informasi dari seluruh urutan dan mengubahnya menjadi probabilitas untuk setiap kategori atau kelas yang mungkin. Misalnya, jika kami mengklasifikasikan sentimen, lapisan ini akan memberi tahu kami seberapa besar kemungkinan input tersebut positif, negatif, atau netral. Probabilitas tertinggi inilah yang kemudian kami ambil sebagai prediksi akhir model.

Kelas LSTMScratch adalah kelas yang digunakan untuk forward propagation dengan algoritma yang dikembangkan berdasarkan pembelajaran di kelas, tanpa bergantung pada Keras.

Langkah pertama yang kami lakukan adalah mengambil semua 'berat' dan 'bias' (yaitu parameter yang dipelajari oleh model selama pelatihan) dari setiap lapisan model Keras yang sudah terlatih. Untuk lapisan LSTM, kami bahkan memecah weights ini menjadi bagian-bagian yang lebih kecil, yang sesuai dengan *gate* internal LSTM (input, forget, cell, dan output). Jika ada lapisan bidirectional, kami juga mengambil weights terpisah untuk arah maju dan mundur.

Setelah itu, kami memulai prosesnya dengan lapisan Embedding secara manual. Kami akan mencari vektor representasi untuk setiap kata input dari matriks embedding yang telah kami ambil. Kemudian, data yang sudah berbentuk vektor ini diatur ulang agar siap diproses secara berurutan.

Untuk setiap lapisan LSTM, kami akan memulai dengan hidden state dan cell state awal yang kosong. Kemudian, kami akan mengulanginya untuk setiap langkah

waktu dalam urutan input. Pada setiap langkah waktu, kami secara manual menghitung empat gate (input, forget, cell, dan output) menggunakan rumus-rumus matematika dasar LSTM. *Gates* ini menentukan informasi mana yang harus diingat, dilupakan, atau disampaikan ke hidden state berikutnya. Jika lapisan tersebut bidirectional, kami akan melakukan perhitungan ini dua kali—sekali untuk maju dan sekali untuk mundur—lalu menggabungkan hasilnya. Sama seperti pada Keras, lapisan LSTM terakhir hanya akan mengembalikan hidden state dari langkah waktu terakhir.

Akhirnya, output dari lapisan LSTM terakhir ini akan masuk ke lapisan Dense yang juga kami bangun sendiri. Kami akan mengalikan hidden state dengan weights klasifikasi dan menambahkan bias-nya. Hasilnya kemudian dilewatkan melalui fungsi softmax buatan kami, yang mengubahnya menjadi distribusi probabilitas di antara kelas-kelas yang mungkin.

Berikut adalah perbandingan hasil antara *forward propagation* dari *scratch* dibandingkan dengan keras

Tabel 2.1.2.3 Macro F1-Score | Keras vs Scratch

Model	Macro F1-score (keras)	Macro F1-score (scratch)
Undirectional	0.2117	0.3167
Bidirectional	0.6538	0.5577

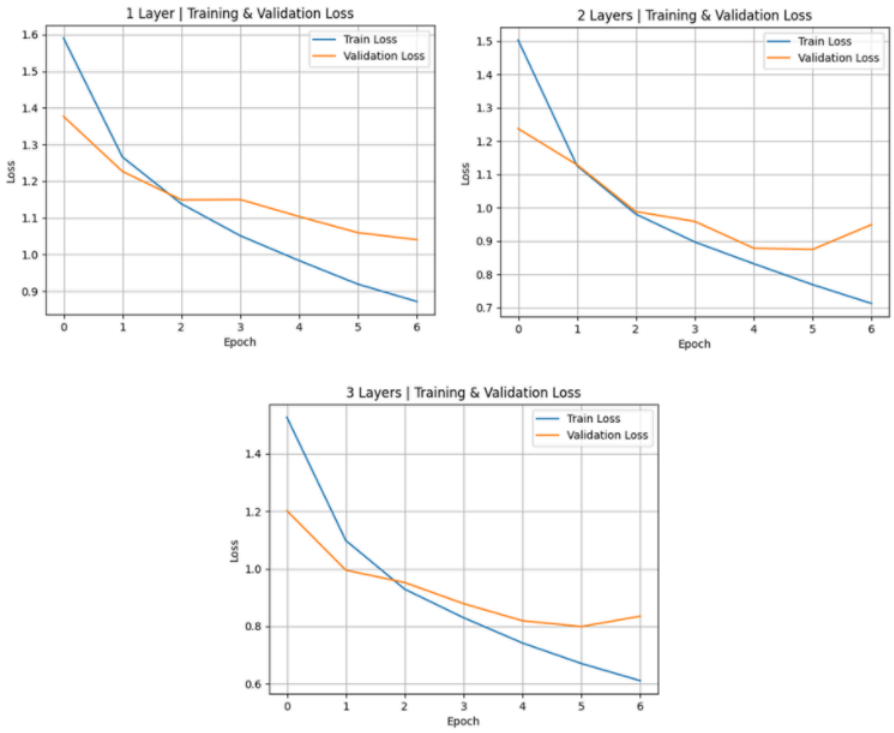
2.2. Hasil Pengujian

2.2.1. CNN

Seluruh pengujian menggunakan CNN dilakukan dengan jumlah epoch sama dengan 7. Hal ini dilakukan karena berdasarkan pengujian yang telah dilakukan, validation loss dan training loss memiliki nilai terbaik dalam rentang 5-10 epoch.

a. Pengaruh jumlah layer konvolusi

Berikut merupakan grafik yang didapatkan dari latihan model menggunakan satu, dua, dan tiga layer konvolusi.



Gambar 2.2.1.1. Grafik Loss | Jumlah Layer Konvolusi

Berikut merupakan Macro F1-score yang didapatkan untuk setiap model yang telah dibuat.

Tabel 2.2.1.1 Macro F1-Score | Jumlah Layer Konvolusi

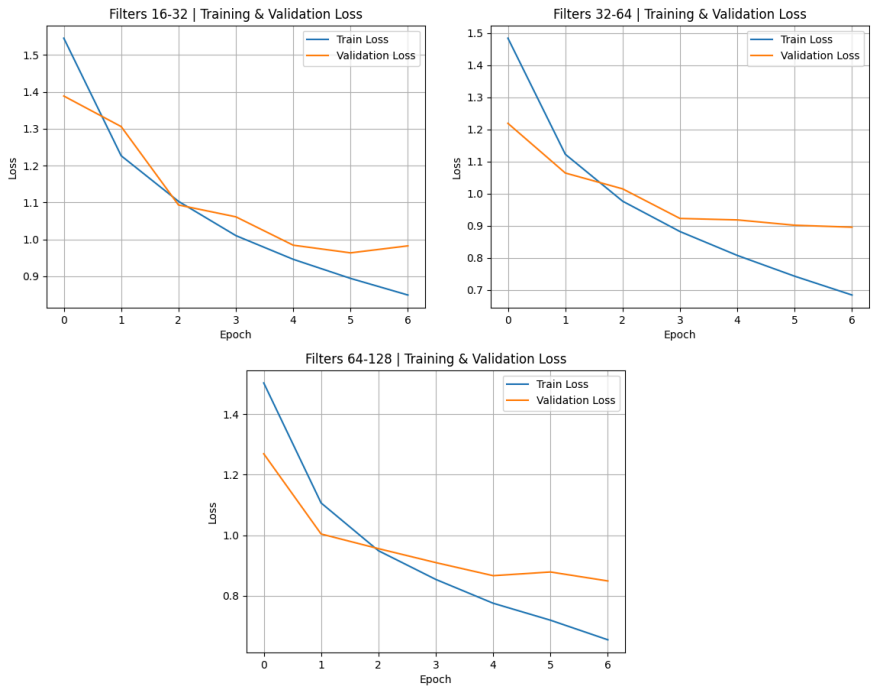
Model	Macro F1-Score
1 Layer Konvolusi	0.6316
2 Layer Konvolusi	0.6803
3 Layer Konvolusi	0.7210

Berdasarkan hasil yang telah didapatkan, dapat dilihat bahwa model dengan 3 layer konvolusi memiliki nilai training loss dan validation loss yang paling rendah serta nilai Macro F1-Score tertinggi. Sehingga dapat disimpulkan bahwa penambahan layer konvolusi meningkatkan kemampuan model untuk mengekstrak fitur-fitur kompleks dari gambar sehingga memberikan hasil akhir yang lebih baik.

Namun, perlu dipertimbangkan juga bahwa penambahan layer konvolusi meningkatkan waktu pelatihan. Jumlah layer konvolusi yang terlalu banyak juga dapat meningkatkan risiko terjadinya overfitting.

b. Pengaruh banyak filter per layer konvolusi

Berikut merupakan grafik yang didapatkan dari latihan model menggunakan 2 layer konvolusi dengan jumlah filter 16-32, 32-64, dan 64-128.



Gambar 2.2.1.2. Grafik Loss | Jumlah Filter

Berikut merupakan Macro F1-score yang didapatkan untuk setiap model yang telah dibuat.

Tabel 2.2.1.2 Macro F1-Score | Jumlah Filter

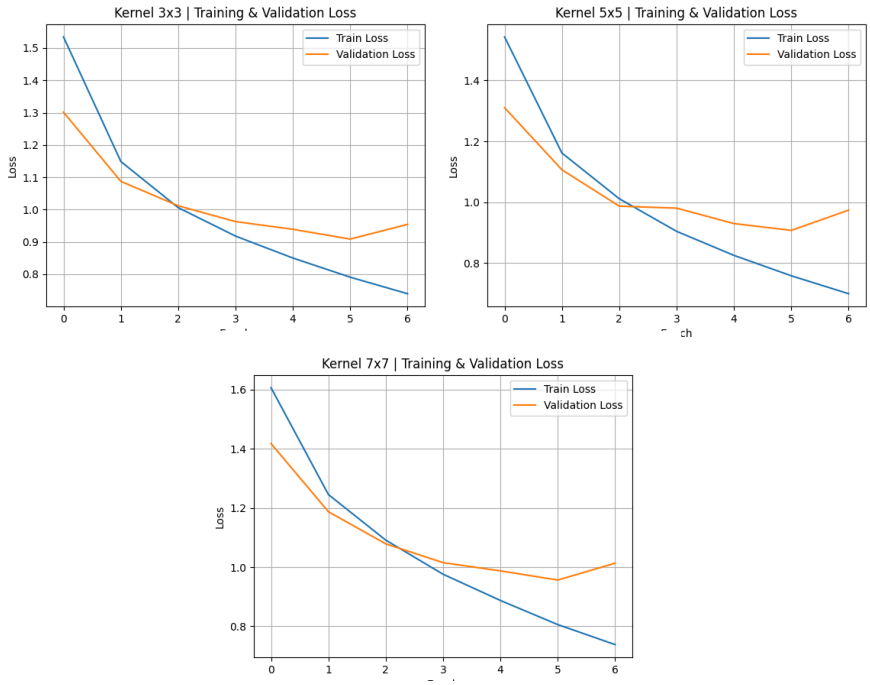
Model	Macro F1-Score
16-32 Filter	0.6536
32-64 Filter	0.6936
64-128 Filter	0.7081

Berdasarkan hasil yang telah didapatkan, dapat dilihat bahwa model yang memberikan hasil terbaik adalah model dengan jumlah filter terbesar, yaitu 64 filter pada layer konvolusi pertama dan 128 filter pada layer konvolusi kedua. Sehingga dapat disimpulkan bahwa, untuk dataset CIFAR-10, semakin banyak filter yang digunakan, maka hasil yang diberikan akan makin baik juga.

Namun, penambahan jumlah filter yang digunakan akan meningkatkan waktu pelatihan, dengan model dengan jumlah filter terbesar membutuhkan waktu pelatihan (24s) yang 3 kali lebih besar jika dibandingkan dengan model dengan jumlah filter terkecil (8s).

c. Pengaruh ukuran filter per layer konvolusi

Berikut merupakan grafik yang didapatkan dari latihan model menggunakan 2 layer konvolusi dengan ukuran filter 3x3, 5x5, dan 7x7.



Gambar 2.2.1.3. Grafik Loss | Ukuran Filter

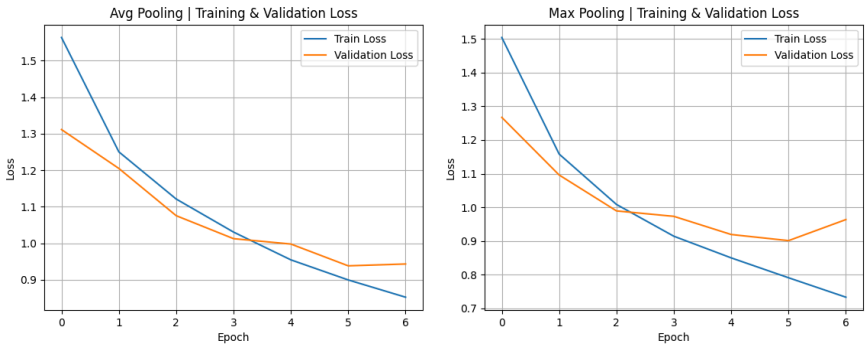
Berikut merupakan Macro F1-score yang didapatkan untuk setiap model yang telah dibuat.

Tabel 2.2.1.3 Macro F1-Score | Ukuran Filter

Model	Macro F1-Score
Filter 3x3	0.6713
Filter 5x5	0.6754
Filter 7x7	0.6474

Berdasarkan hasil yang telah didapatkan, dapat dilihat bahwa filter 7x7 memberikan hasil terburuk, dengan Macro F1-Score terkecil, serta validation loss terbesar. Sehingga dapat disimpulkan bahwa kernel dengan ukuran yang kecil lebih baik untuk digunakan untuk dataset CIFAR-10, dengan model dengan ukuran filter 3x3 dan 5x5 memberikan hasil yang hampir sama.

d. Pengaruh jenis pooling layer



Gambar 2.2.1.4. Grafik Loss | Jenis Pooling

Berikut merupakan Macro F1-score yang didapatkan untuk setiap model yang telah dibuat.

Tabel 2.2.1.4 Macro F1-Score | Jenis Pooling

Model	Macro F1-Score
Average Pooling	0.6681
Max Pooling	0.6751

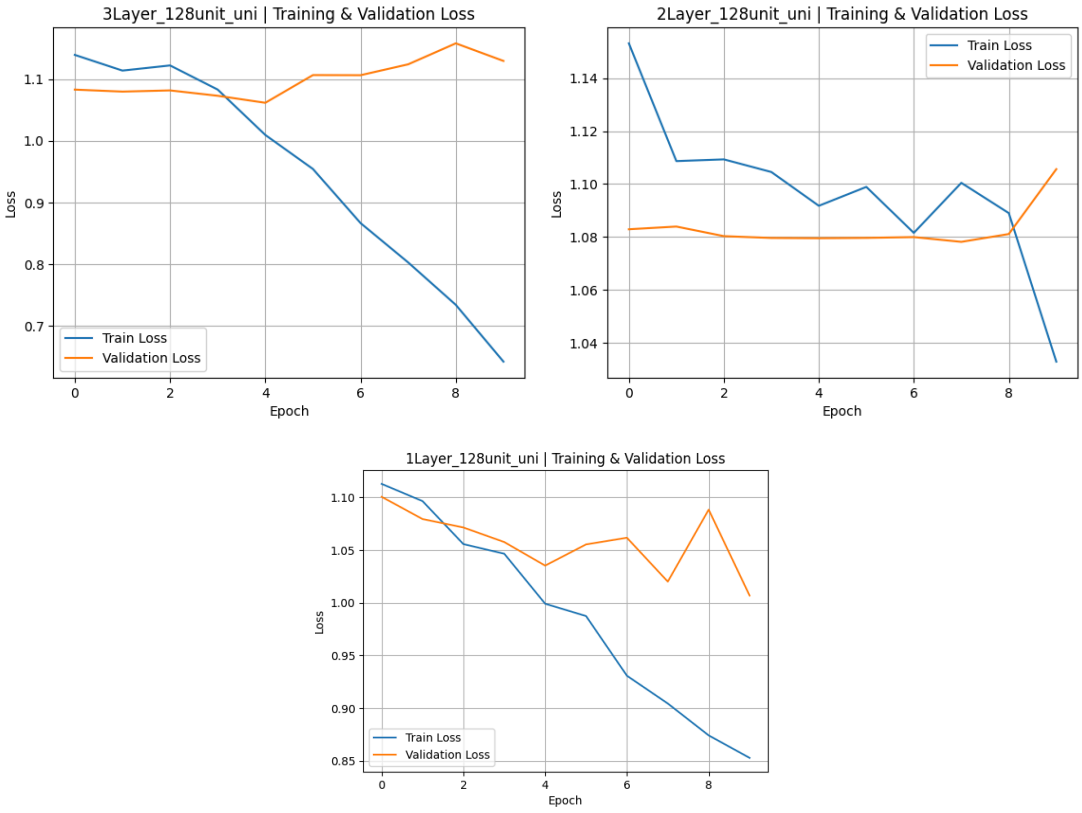
Berdasarkan hasil yang telah didapatkan, dapat dilihat bahwa model yang menggunakan max pooling memberikan hasil yang sedikit lebih baik jika dibandingkan dengan model yang menggunakan average pooling. Hal ini karena max pooling dapat lebih baik mempertahankan fitur yang penting, sedangkan average pooling menyebarkan informasi lebih merata namun beresiko kehilangan fitur yang menonjol.

2.2.2. Simple RNN

Seluruh pengujian menggunakan model SimpleRNN dilakukan dengan jumlah epoch 10 dan dimensi embedding yang digunakan adalah 128.

a. Pengaruh jumlah layer RNN

Berikut merupakan grafik yang didapatkan dari pelatihan model menggunakan satu, dua, dan tiga layer :



Gambar 2.2.2.1. Grafik Loss | Jumlah Layer RNN

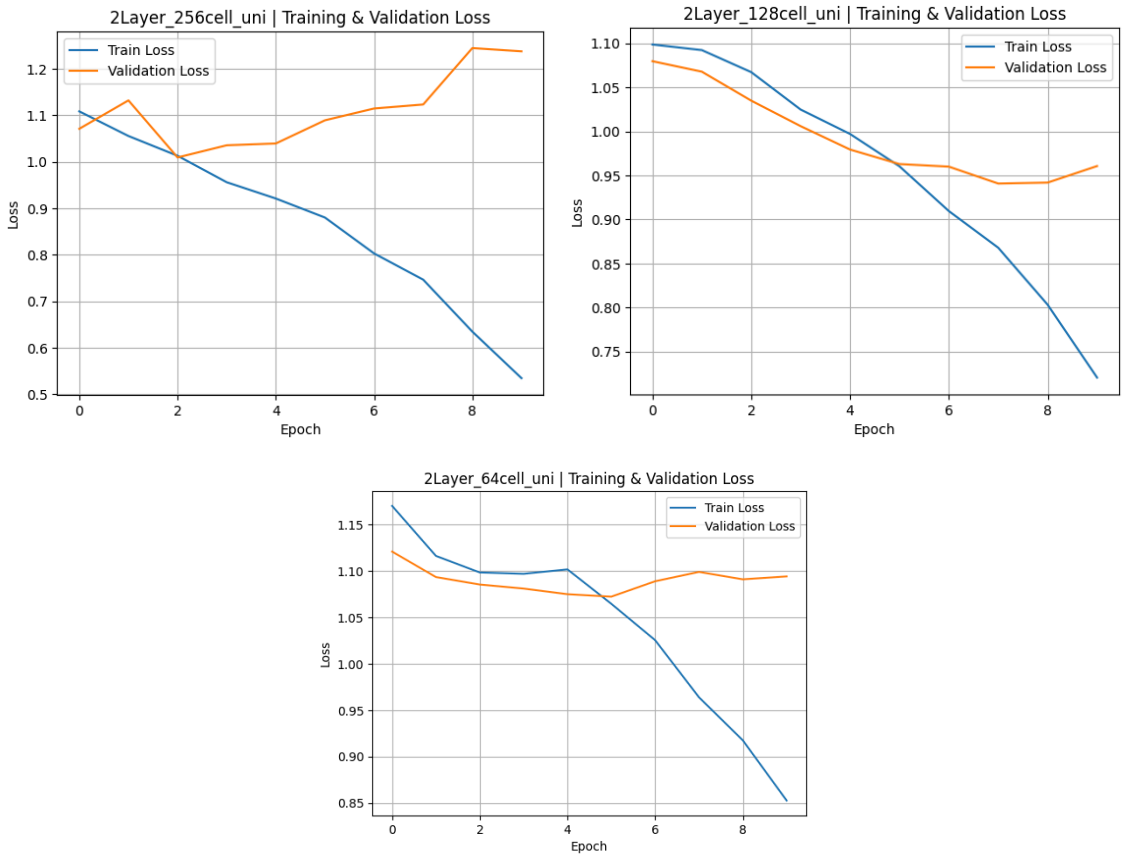
Tabel 2.2.2.1 Macro F1-Score | Jumlah Layer RNN

Model	Macro F1-Score
1 Layer	0.4247
2 Layer	0.3147
3 Layer	0.4388

Berdasarkan hasil yang telah didapatkan, model dengan 3 layer RNN unidirectional menunjukkan nilai Macro F1-Score tertinggi (0.4388), diikuti oleh model 1 layer (0.4247). Model dengan 2 layer menunjukkan performa yang lebih rendah dalam konfigurasi ini. Grafik loss untuk model 3 layer juga menunjukkan penurunan training loss yang lebih signifikan dibandingkan model lainnya meskipun *validation loss* cenderung fluktuatif dan menunjukkan adanya potensi *overfitting* seiring bertambahnya

epoch. Hal ini mengindikasikan bahwa penambahan kedalaman pada arsitektur RNN hingga 3 layer dapat membantu model menangkap pola yang lebih kompleks dalam data teks namun juga berpotensi untuk terjadi *overfitting* khususnya dalam kasus ini yang jumlah datasetnya sedikit.

b. Pengaruh banyak *cell*



Gambar 2.2.2.2. Grafik Loss | Banyak Cell

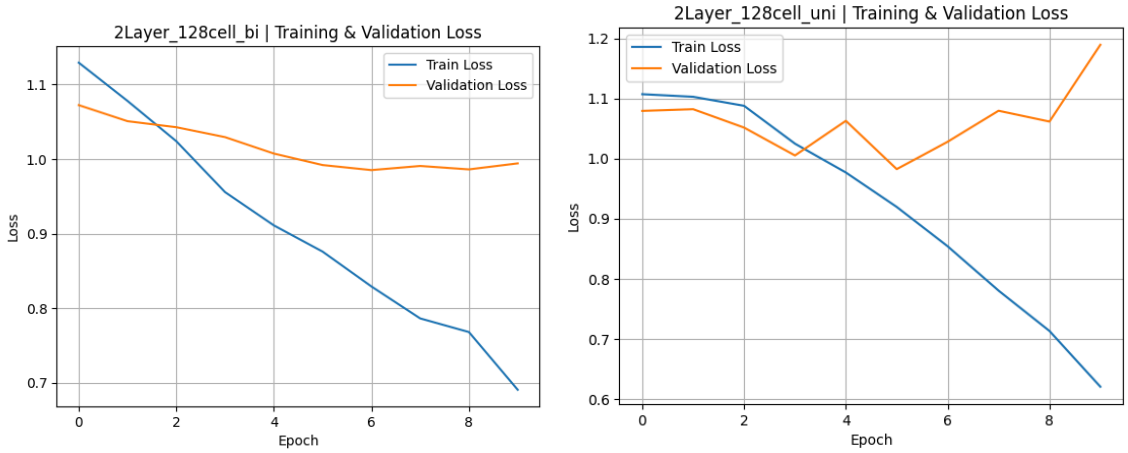
Tabel 2.2.2.2 Macro F1-Score | Banyak Cell

Model	Macro F1-Score
64	0.2816 |
128	0.5284
256	0.4772

Dari tabel di atas, model dengan 128 unit per layer pada arsitektur 2 layer unidirectional memberikan hasil Macro F1-Score terbaik (0.5284). Model dengan 64 unit

menunjukkan performa paling rendah mengindikasikan kapasitas model yang mungkin kurang untuk menangkap fitur penting. Model dengan 256 unit, meskipun memiliki kapasitas lebih besar, tidak menghasilkan skor yang lebih baik dari 128 unit dan dari grafiknya terlihat *validation loss* yang lebih tinggi. Hal tersebut menunjukkan kecenderungan *overfitting* yang lebih kuat. Ini menunjukkan bahwa pemilihan jumlah unit yang tepat penting untuk keseimbangan antara kapasitas model dan generalisasi.

c. Pengaruh arah



Gambar 2.2.2.3 Grafik Loss | Arah

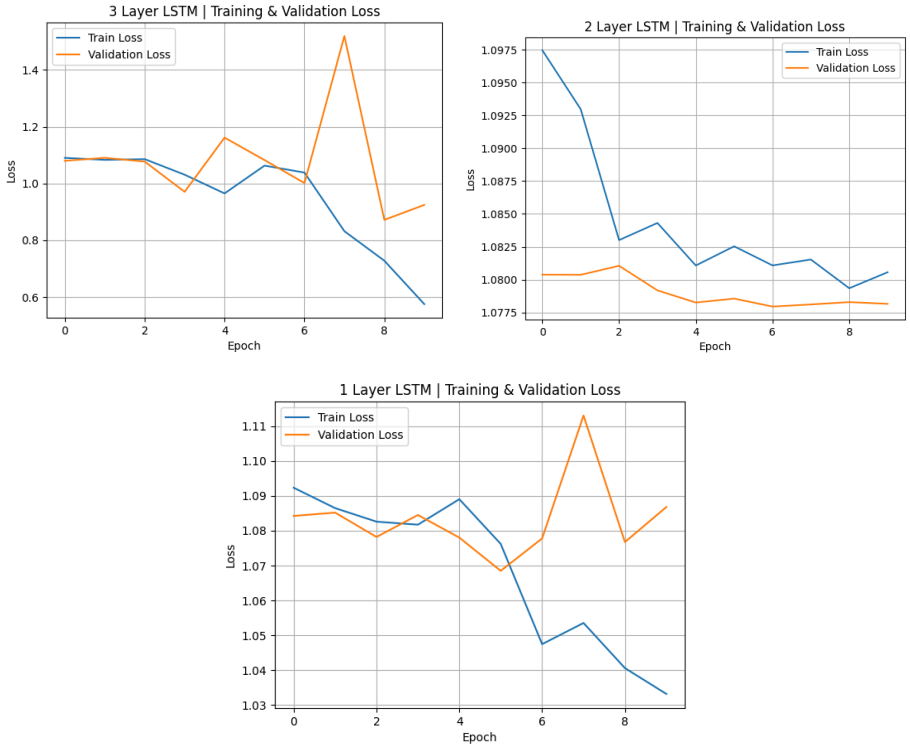
Tabel 2.2.2.3 Macro F1-Score | Arah

Model	Macro F1-Score
Bidirectional	0.4732
Unidirectional	0.4411

Berdasarkan hasil pengujian, model SimpleRNN bidirectional dengan 2 layer dan 128 unit per layer menghasilkan Macro F1-Score yang lebih tinggi (0.4732) dibandingkan dengan model unidirectional dengan konfigurasi serupa (0.4411). Hal ini menunjukkan bahwa kemampuan model *bidirectional* untuk memproses sekuens dari dua arah (maju dan mundur) membantu dalam menangkap konteks yang lebih kaya dari data teks yang pada akhirnya meningkatkan performa klasifikasi sentimen. Grafik loss juga cenderung menunjukkan bahwa model bidirectional memiliki *validation loss* yang tidak terlalu fluktuatif dan cenderung stabil.

2.2.3. LSTM

a. Pengaruh jumlah layer LSTM



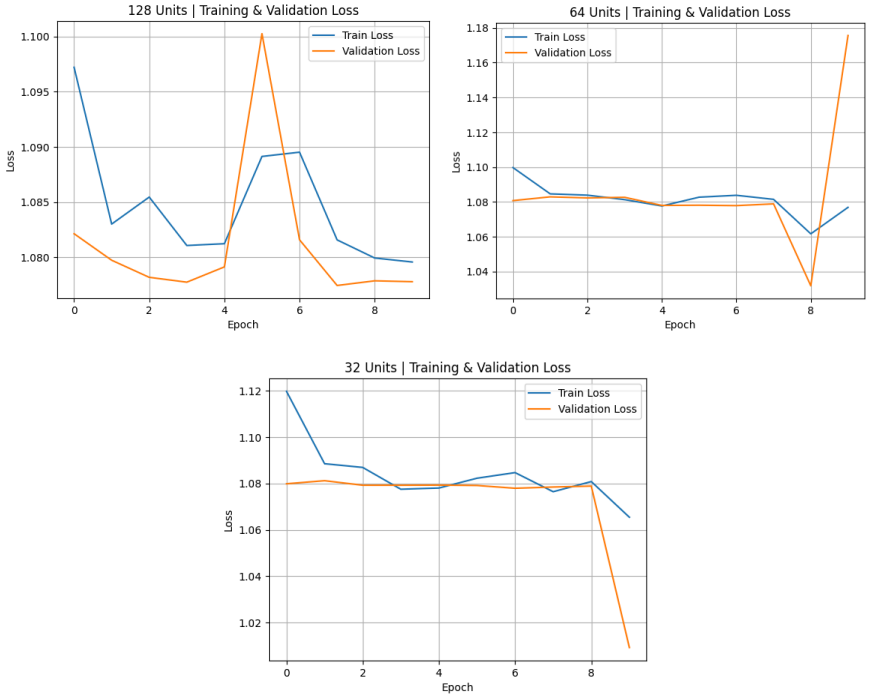
Gambar 2.2.3.1. Grafik Loss | Jumlah Layer LSTM

Tabel 2.2.3.1 Macro F1-Score | Jumlah Layer LSTM

Model	Macro F1-Score
1 Layer	0.2632
2 Layer	0.2069
3 Layer	0.5756

Berdasarkan hasil yang diperoleh, model LSTM dengan 3 layer unidirectional menunjukkan nilai Macro F1-Score tertinggi (0.5756) secara signifikan dibandingkan model dengan 1 atau 2 layer. Grafik loss untuk model 3 layer juga menunjukkan kemampuan untuk mencapai training loss yang lebih rendah. Meskipun *validation loss* pada model 3 layer menunjukkan beberapa fluktuasi, khususnya lonjakan di sekitar epoch 7-8 sebelum kembali turun, F1-score akhirnya lebih baik. Ini mengindikasikan bahwa penambahan kedalaman pada arsitektur LSTM hingga 3 layer dapat membantu model dalam menangkap fitur yang lebih kompleks dari data sekuensial. Model 1 dan 2 layer menunjukkan F1-score yang relatif rendah, menandakan kapasitas model yang mungkin belum memadai untuk tugas ini.

b. Pengaruh banyak *cell*



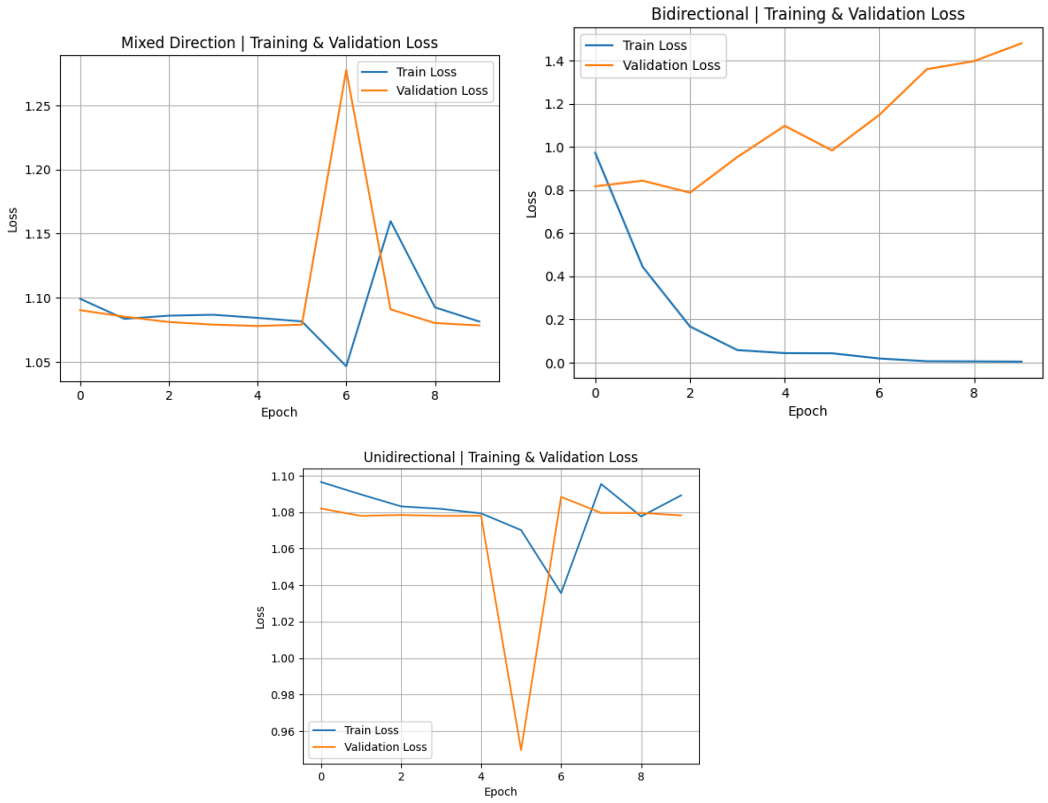
Gambar 2.2.3.2. Grafik Loss | Banyak Cell

Tabel 2.2.3.2 Macro F1-Score | Banyak Cell

Model	Macro F1-Score
32	0.4610
64	0.2170
128	0.2160

Dari tabel di atas, model dengan 32 unit per layer pada arsitektur 2 layer LSTM unidirectional memberikan hasil Macro F1-Score terbaik (0.4610). Model dengan jumlah unit yang lebih besar (64 dan 128 unit) justru menunjukkan performa yang jauh lebih rendah. Grafik loss untuk konfigurasi 64 dan 128 unit menunjukkan *validation loss* yang tinggi dan tidak stabil mengindikasikan *overfitting* atau kesulitan dalam konvergensi. Ini menunjukkan bahwa untuk arsitektur 2 layer LSTM pada dataset ini, jumlah unit yang lebih sedikit (32 unit) lebih efektif dan mampu generalisasi lebih baik.

c. Pengaruh arah



Gambar 2.2.3.3 Grafik Loss | Arah

Tabel 2.2.3.3 Macro F1-Score | Arah

Model	Macro F1-Score
Bidirectional	0.6538
Unidirectional	0.2117
Campuran	0.2117

Berdasarkan hasil pengujian, model LSTM bidirectional (kedua layer bidirectional) dengan konfigurasi 64 unit pada layer pertama dan 32 unit pada layer kedua menghasilkan Macro F1-Score yang paling tinggi (0.6538) secara signifikan. Model unidirectional dan mixed direction menunjukkan performa yang jauh lebih rendah dengan F1-score yang identik (0.2117). Ini menandakan bahwa kemampuan model LSTM bidirectional untuk memproses sekuens dari dua arah sangat membantu dalam menangkap konteks yang lebih kaya dari data teks yang pada akhirnya meningkatkan performa klasifikasi sentimen. Grafik loss untuk model bidirectional juga menunjukkan training loss yang turun sangat cepat dan mencapai nilai yang sangat rendah, meskipun validation lossnya agak menunjukkan *overfitting*.

BAB III
KESIMPULAN DAN SARAN

3.1. Kesimpulan

Berdasarkan hasil eksperimen yang telah dilakukan, peningkatan jumlah filter, layer, dan penggunaan arsitektur bidirectional secara umum memberikan peningkatan performa model baik untuk CNN, Simple RNN, maupun LSTM. Pada CNN, model dengan jumlah filter yang lebih banyak menghasilkan Macro F1-Score tertinggi meskipun membutuhkan waktu pelatihan yang lebih lama. Ukuran filter yang kecil (3x3 dan 5x5) juga terbukti lebih optimal untuk dataset CIFAR-10 dibandingkan ukuran besar (7x7). Selain itu, penggunaan max pooling menunjukkan kinerja yang sedikit lebih baik dibandingkan average pooling.

Pada klasifikasi sentimen teks NusaX-Sentiment, dengan konfigurasi 2 layer bidirectional (64-32 unit), LSTM menghasilkan F1-score paling tinggi yang menunjukan LSTM mengungguli SimpleRNN dalam menangkap konteks. Meskipun demikian, SimpleRNN juga menunjukkan peningkatan yang baik pada kedalaman layer tertentu (misalnya, 3 layer dengan 128 unit), arsitektur bidirectional, dan pemilihan jumlah unit yang optimal (seperti 128 unit untuk 2 layer). Jadi bukan hanya tentang menambahkan jumlah layer, *cell*, ataupun menentukan arah, tapi konfigurasi yang cocok dengan dataset menjadi hal yang sangat penting untuk menghasilkan model yang bagus.

Hasil-hasil ini menekankan pentingnya keseimbangan antara kompleksitas model dan kapasitas generalisasi, serta menunjukkan keunggulan arsitektur bidirectional dalam memahami konteks sekuensial dari data teks.

3.2. Saran

Untuk pengembangan mendatang disarankan untuk menggunakan regularisasi untuk mencegah *overfit*. Pengujian pada dataset yang lebih besar dan beragam juga dapat dipertimbangkan untuk penyempurnaan model lebih lanjut.

DAFTAR PUSTAKA

Tabel 4.1. Pembagian Tugas

NIM	Nama	Pembagian Tugas
13522028	Panji Sri Kuncara Wisma	SimpleRNN

13522056	Diero Arga Purnama	CNN, File ReadME
13522120	Muhammad Rifki Virziadeili Harisman	LSTM

Repository GitHub: https://github.com/DieroA/Tubes2_ML

REFERENSI

Slides Materi IF3270 Pembelajaran Mesin | CNN
Slides Materi IF3270 Pembelajaran Mesin | RNN part 1
Slides Materi IF3270 Pembelajaran Mesin | RNN part 2