Tugas Besar 1 IF3270 Pembelajaran Mesin 2024/2025

Convolutional Neural Network dan Recurrent Neural Network

Kelompok 4:

13522031 Zaki Yudhistira

13522039 Edbert Eddyson Gunawan

13522049 Vanson Kurnialim

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 40132

2025

Daftar Isi

Deskripsi Permasalahan

1. Klasifikasi Citra dengan Convolutional Neural Network (CNN)

Permasalahan pertama adalah membangun model CNN untuk klasifikasi gambar menggunakan dataset CIFAR-10. Model CNN yang dirancang harus memenuhi arsitektur dasar CNN, menggunakan `Conv2D`, `Pooling`, `Flatten`, dan `Dense Layer`, serta menggunakan fungsi loss `Sparse Categorical Crossentropy` dan optimizer `Adam`.

Model akan dilatih dengan pembagian data menjadi **training**, **validation**, dan **testing** set. Selain itu, dilakukan **eksperimen variasi hyperparameter** sebagai berikut:

- Jumlah layer konvolusi
- Banyaknya filter per layer konvolusi
- Ukuran filter
- Jenis pooling layer (Max Pooling vs Average Pooling)

Hasil model akan dievaluasi menggunakan **macro F1-score** serta visualisasi **training loss dan validation loss** pada setiap epoch untuk menganalisis dampak perubahan arsitektur terhadap performa model.

Terakhir, peserta diminta membuat **implementasi forward propagation from scratch** yang mampu membaca bobot hasil pelatihan dengan Keras dan melakukan perbandingan hasil inferensi dengan model Keras.

2. Klasifikasi Teks dengan Simple Recurrent Neural Network (RNN)

Permasalahan kedua adalah membangun model RNN untuk klasifikasi sentimen bahasa Indonesia menggunakan dataset **NusaX-Sentiment**. Proses dimulai dari preprocessing teks, yakni tokenisasi dan embedding menggunakan layer yang disediakan oleh Keras.

Model RNN harus memiliki minimal:

- Embedding layer
- Bidirectional/Unidirectional RNN layer
- Dropout layer
- Dense layer

Eksperimen dilakukan terhadap:

- Jumlah layer RNN
- Jumlah cell RNN per layer
- Arah layer RNN (bidirectional vs unidirectional)

Evaluasi model dilakukan menggunakan **macro F1-score** serta grafik ***training*** **dan** ***validation loss*** untuk tiap epoch. Forward propagation dari model RNN juga harus diimplementasikan dari nol dan dibandingkan hasilnya dengan model Keras.

3. Klasifikasi Teks dengan Long Short-Term Memory (LSTM)

Permasalahan terakhir adalah membangun model LSTM untuk klasifikasi sentimen dengan pendekatan yang mirip dengan RNN. Perbedaan utama adalah penggunaan LSTM sebagai arsitektur utama dalam menangani sequence data yang lebih kompleks.

Model harus mencakup:

- Embedding layer
- Bidirectional/Unidirectional LSTM layer
- Dropout layer
- Dense layer

Eksperimen dilakukan untuk mengamati pengaruh:

- Jumlah layer LSTM
- Jumlah cell LSTM per layer
- Arah LSTM layer (bidirectional vs unidirectional)

Evaluasi juga dilakukan menggunakan **macro F1-score**, serta visualisasi perkembangan loss selama training. Seperti pada CNN dan RNN, implementasi **forward propagation from scratch** wajib dibuat untuk menguji kesesuaian logika model terhadap hasil dari Keras.

Pembahasan Implementasi *Machine Learning Model*

A. Daftar Kelas

Bagian berikut memuat penjelasan mengenai kelas-kelas yang digunakan dalam sistem, beserta deskripsi fungsi dan atribut dari masing-masing kelas. Kelas-kelas tersebut dikategorikan ke dalam empat bagian utama, yaitu: *forward propagation* untuk CNN, RNN, LSTM, serta kelas dan fungsi utilitas lainnya.

a. Kelas terkait CNN

Kelas	Fungsi / Atribut	Deskripsi
NumpyCNNConfigurable	`Configs, weights, class_names`	Variabel konfigurasi model, nilai bobot model, dan daftar kolom dataset yang disimpan ketika kelas dibentuk untuk
	`_load_weights(self, weights_path)`	Fungsi untuk memuat bobot model dari file yang telah disimpan.
	`predict_batch(self, image_batch)`	Fungsi yang bertugas untuk menjalankan *training* model sesuai dengan jumlah batch yang diinput.
	`evaluate(self, x_test_normalized, y_test_true, batch_size=32)`	Fungsi untuk mengatur keberjalanan keseluruhan proses *training* hingga hasil prediksi model.
Fungsi pendukung lainnya	`relu_manual(x)`	Fungsi aktivasi ReLu
	`softmax_manual(x)`	Fungsi Aktivasi softMax

	conv2d_scipy(input_batch, W, b, stride=1, padding_mode='same')	Fungsi implementasi proses konvolusi secara manual.
	pool2d_reshaping(input_batch, pool_size=(2, 2), stride_val=None, mode='max')	Fungsi untuk melakukan *pooling*.
	flatten_manual(input_batch)	Fungsi untuk melakukan *flatten* atau mengubah data 2D menjadi 1D.
	dense_manual(A_prev_flattened, W, b)	Fungsi *dense* manual dengan memanfaatkan np.dot.

b. Kelas terkait LSTM

Kelas	Fungsi / Atribut	Deskripsi
ScratchLSTM	h, c	Nilai *hidden state* dan *cell state*.
	W_i, W_f, W_c, W_o U_i, U_f, U_c, U_o, b_i, b_f, b_c, b_o	Bobot *logic gate* LSTM terkait.
	forward(x: input)	Fungsi untuk melakukan *forward propagation*.
	manytomany	Parameter bertipe boolean yang menentukan bahwa keluaran dari lapisan LSTM memiliki format *many-to-many* dan dapat digunakan sebagai input untuk lapisan RNN berikutnya.
ScratchBiRNN	Fwd_lstm, bwd_lstm	Atribut berupa LSTM yang digunakan sebagai *forward* dan *backward layer* dari LSTM *bidirectional*.

c. Kelas terkait RNN

Kelas	Fungsi / Atribut	Deskripsi
ScratchRNN	Wx, Wh, b	Bobot atau *weight* dari RNN terkait.
	hidden_size	Dimensi *hidden layer*.
	activation	Fungsi aktivasi berupa lambda yang digunakan saat perhitungan antara *input* dan matriks bobot sudah selesai.
	manytomany	Parameter bertipe boolean yang menentukan bahwa keluaran dari lapisan RNN memiliki format *many-to-many* dan dapat digunakan sebagai input untuk lapisan RNN berikutnya.
	forward(x: input)	Fungsi untuk melakukan *forward propagation*.
ScratchBiRNN	Fwd_rnn, bwd_rnn	Atribut berupa simple RNN yang digunakan sebagai *forward* dan *backward layer* dari RNN *bidirectional*.
	manytomany	-

d. Kelas umum / utilitas

Kelas	Fungsi / Atribut	Deskripsi
ScratchDense	dense_weight	Bobot untuk setiap simpul *layer dense* terkait.
	activation	Fungsi aktivasi untuk *layer dense* terkait.
	forward()	Fungsi untuk melakukan *forward propagation*.
ScratchEmbedding	embedding_weight	*Map* atau pemetaan hasil tokenisasi kepada suatu vektor tertentu.
	forward()	Fungsi untuk melakukan *forward propagation*.
Model	convertModel()	Fungsi ini memiliki peran krusial dalam memuat bobot hasil pelatihan dari file berformat .keras ke dalam model yang diimplementasikan secara *from scratch*. Fungsi ini

		menghasilkan model siap pakai yang sepenuhnya menggunakan implementasi mandiri.
	predict(x: input)	Fungsi untuk melakukan prediksi terhadap suatu masukan nilai yang diberikan. Mengeluarkan vektor hasil perhitungan.

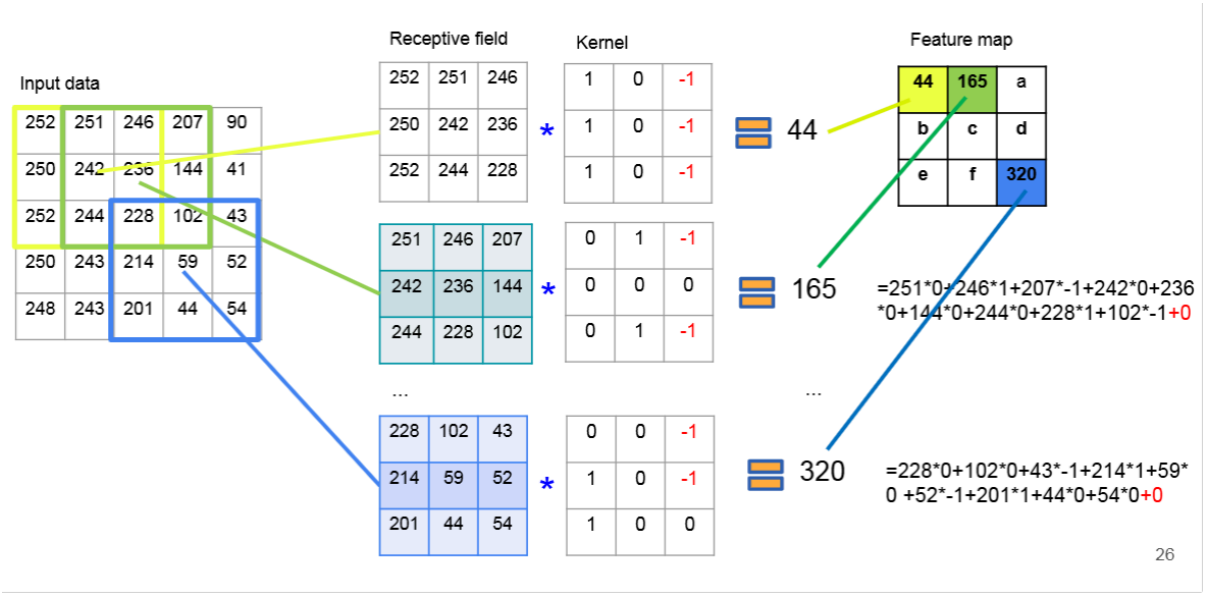
B. Forward Propagation

Bagian ini akan memaparkan logika di belakang proses *forward propagation* untuk ketiga implementasi *machine learning model*.

a. Convolutional Neural Network

Forward propagation untuk CNN yang dibuat terdiri dari beberapa bagian sebagai berikut :

1. Tahap Convolution Stage



Gambar 1. Ilustrasi proses konvolusi, pembentukan Feature Map

Bagian konvolusi ini diimplementasikan oleh fungsi conv2d_scipy yang menghitung output dari sebuah layer konvolusional untuk satu batch data input dengan melakukan iterasi melalui setiap item dalam batch dan setiap filter yang ada. Untuk setiap filter, bobotnya yang relevan diekstrak. Kemudian, untuk setiap channel dari item input saat ini, dilakukan operasi korelasi silang 2D antara slice channel input tersebut dengan slice kernel yang bersesuaian dari filter menggunakan fungsi scipy.signal.correlate dengan mode='same' untuk menjaga dimensi spasial dan method='auto' untuk efisiensi.

Hasil korelasi dari semua channel input dijumlahkan untuk membentuk satu *feature map*. Akhirnya, bias yang sesuai ditambahkan ke *feature map*

hasil akumulasi ini, dan hasilnya disimpan sebagai satu channel output untuk item batch tersebut sebelum fungsi mengembalikan keseluruhan tensor keluaran Z.

Banyaknya *layer* konvolusi yang digunakan adalah 3, dengan perubahan ketika melakukan percobaan variasi jumlah *layer* konvolusi. Penggunaan *library scipy* pada proses konvolusi dilakukan untuk mempercepat proses matematis konvolusi matriks agar tidak terlalu lama.

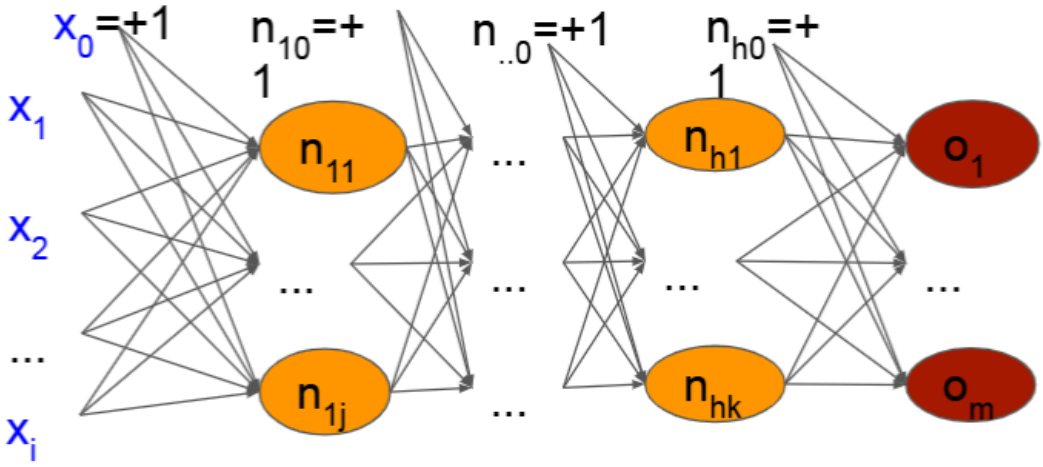
2. Tahap Detector Stage

Dilakukan penggunaan fungsi aktivasi ReLu atau softMax setelah selesai melakukan *layer* konvolusi dan *dense*. Fungsi aktivasi yang digunakan akan diterapkan pada seluruh data pada layer yang berkaitan.

3. Tahap Pooling Stage

Setelah fitur-fitur diekstraksi oleh layer-layer konvolusi dan diterapkan fungsi aktivasi, tahapan pooling diterapkan untuk mengurangi dimensi spasial (height dan width) dari feature map. Proses ini tidak hanya membantu mengurangi jumlah parameter dan beban komputasi pada layer-layer berikutnya, tetapi juga memberikan tingkat invariansi terhadap translasi kecil fitur dan dapat membantu mencegah overfitting. Jenis pooling digunakan pada implementasi ini adalah max pooling, yang mempertahankan fitur paling menonjol dengan mengambil nilai maksimum dari setiap patch pada feature map, dan average pooling, yang mengambil nilai rata-rata untuk representasi fitur yang lebih halus. Implementasi yang efisien untuk pooling non-tumpang tindih dapat dicapai dengan teknik reshaping array NumPy, yang jauh lebih cepat daripada iterasi manual.

4. Dense Layer



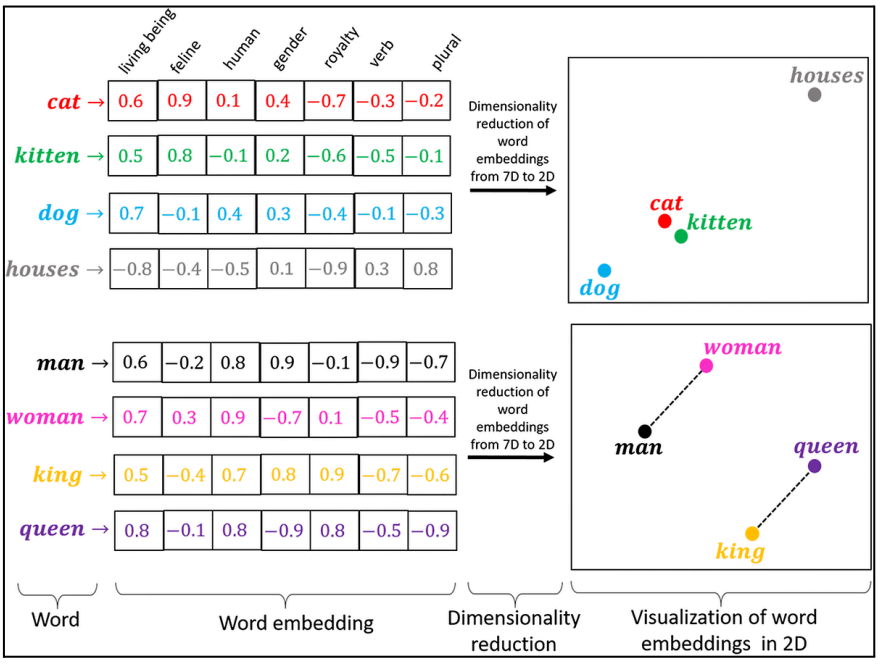
Gambar 2. Ilustrasi proses dense layer

Hasil dari *feature map* setelah *pooling* yang telah di*flatten* akan menjadi input untuk *layer* ini. *Dense layer* berfungsi untuk mempelajari kombinasi non-linear dari fitur-fitur yang telah diekstraksi dan melakukan tugas akhir seperti klasifikasi. Setiap neuron pada dense layer terhubung ke semua neuron pada layer sebelumnya, dan output-nya dihitung melalui perkalian matriks antara input vektor dengan matriks bobot layer, diikuti dengan penambahan vektor bias. Hasilnya kemudian dilewatkan melalui fungsi aktivasi, seperti yang telah dijelaskan di atas.

b. Simple Recurrent Neural Network

Proses *forward propagation* untuk RNN terdiri beberapa tahap utama:

1. Tahap *vectorization* dan *embedding*



Gambar 3. Ilustrasi embedding pada kata

Pada Tahapan ini, *input* berupa rangkaian kata **ditokenisasi** terlebih dahulu, kemudian dipetakan menjadi sebuah vektor berdasarkan *dictionary* atau *encoding* yang telah ditentukan. *Encoding* dilakukan berdasarkan besarnya kosa kata pada suatu cakupan *training*. Setelah itu, vektor kata akan diubah menjadi bentuk vektor-n dimensi, sesuai dengan konfigurasi yang ditentukan, proses ini dinamakan ***embedding***. Kemudian, dilakukan ***padding*** pada vektor agar setiap vektor memiliki dimensi atau panjang yang seragam.

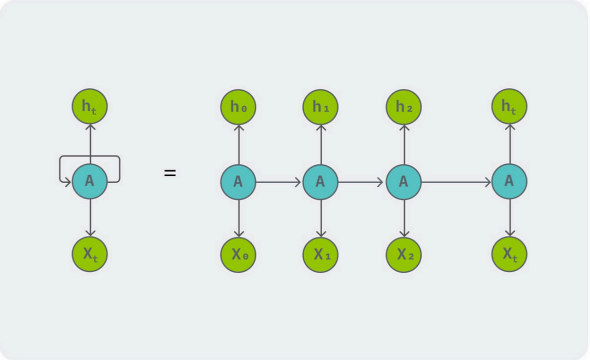
2. Tahap Perhitungan pada *layer* RNN

Vektor yang telah diterima oleh sistem akan dimasukkan ke jaringan *recurrent neural network* berbentuk kumpulan matriks: wi dan wh, wi merupakan matriks yang menjembatani antara *input* dan *hidden layer*, sedangkan matriks wh adalah matriks yang digunakan untuk

menyimpan *hidden weight*. *Input* pertama akan dikalikan dengan wi, diproses dengan fungsi aktivasi, dan menjadi h1, yang akan digunakan untuk iterasi *time step* berikutnya. Setelah melewati *hidden layer*, *input* siap diproses oleh *layer* lainnya.

3. Tahap Perhitungan pada *dense* layer

   Vektor hasil perhitungan dari lapisan RNN, akan diproses pada *dense layer* dan dikirimkan keluar, dapat menjadi *output* ataupun digunakan untuk *dense layer* atau layer RNN berikutnya.



Gambar 4. Ilustrasi rangkaian RNN

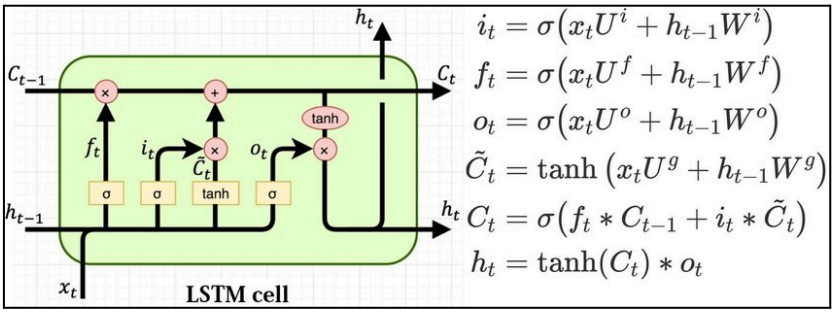
c. Long Short Term Memory (LSTM)

Proses *forward propagation* pada LSTM mengikuti proses yang sama seperti halnya pada RNN, hanya saja, perbedaan krusial terletak pada pola perhitungan *neural network* utamanya. Berikut adalah penjelasan lebih lanjut terkait LSTM.

LSTM sendiri dibagi menjadi 4 fase perhitungan:

1. Perhitungan *input* (i_t)
2. Perhitungan *candidate* (C_c)
3. Perhitungan *forget* (F_t)
4. Perhitungan *output* (O_t)

Setiap fase akan mengalikan masukan kepada sebuah vektor bobot, dan melakukan perhitungan sesuai dengan sistematika kerja LSTM.



Gambar 5. Ilustrasi logika perhitungan LSTM

Akan disimpan sebuah nilai h dan c, nilai h akan diganti dengan nilai yang baru di setiap perhitungan, sedangkan nilai c akan atau *cell state* akan diperbarui pada setiap iterasi, berdasarkan perhitungan yang dilakukan.

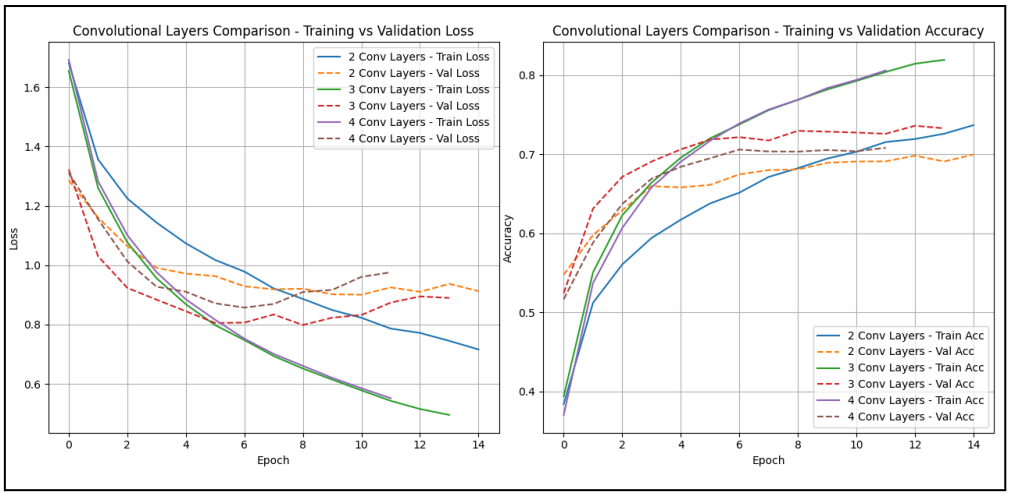
Logic perhitungan *output, embedding,* dan *tokenization,* kurang lebih mengikuti proses yang sama pada bagian RNN. Lebih tepat dinyatakan bahwa ketiga proses tersebut agnostik, alias bukan merupakan kepemilikan baik RNN maupun LSTM.

Hasil Pengujian

Analisis akan dilakukan terhadap sejumlah aspek arsitektur model, meliputi pengaruh jumlah lapisan (*layer*), konfigurasi *hidden layer* atau *cell*, variasi jumlah *filter* pada CNN, serta perbandingan antara penggunaan RNN atau LSTM bersifat *bidirectional* dengan versi regulernya. Pemaparan lebih lanjut mengenai masing-masing aspek akan dipaparkan pada bagian berikut.

A. Pengujian *Convolutional Neural Network*

1. Pengaruh jumlah layer konvolusi



Gambar 6. Plot grafik perbedaan skor validasi dan training jumlah layer

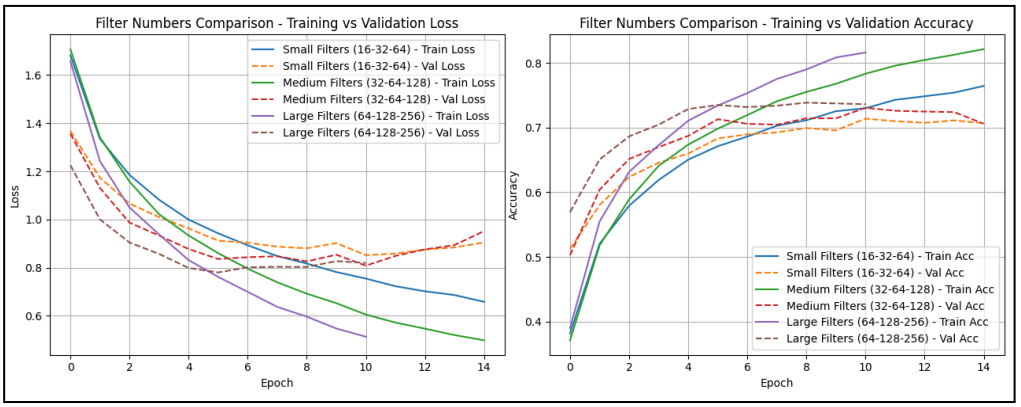
Pada grafik *loss* dan *accuracy* proses *training* dan validasi untuk tiap variasi terlihat bahwa model dengan 3 *layer* konvolusi memiliki nilai *accuracy* dan *loss* yang paling baik dibanding dengan variasi lainnya. Namun hasil tersebut baru berasal dari fase *training* dan validasi, mari kita lihat hasil prediksi untuk tiap variasi pada tabel di bawah ini :

Convolution Layer	Accuracy	F1-Score
2	0.6902	0.6897

3	0.7250	0.7246
4	0.7019	0.7035

Berdasarkan hasil prediksi, model dengan 3 *layer* konvolusi memiliki nilai akurasi dan F1-score yang paling baik, sehingga sesuai dengan analisa pada tahap *training* dan validasi. Maka dari itu, pada dataset CIFAR-10 yang digunakan, 3 jumlah *layer* konvolusi lah yang paling sesuai untuk mendapatkan hasil yang optimal.

2. Pengaruh banyak filter per layer konvolusi



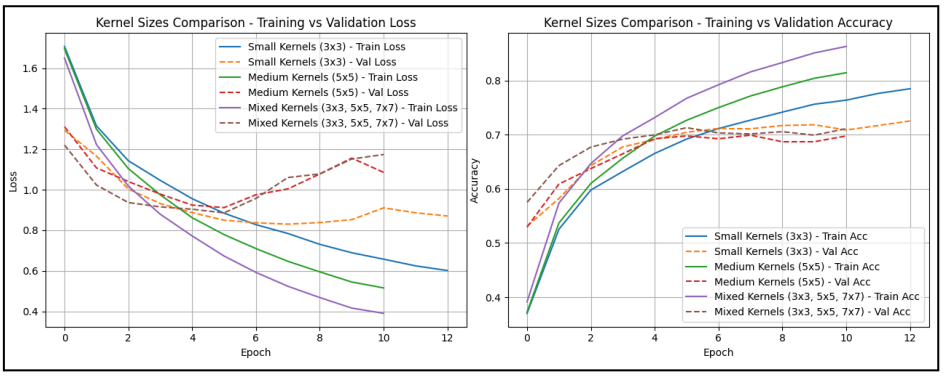
Gambar 7. Plot grafik perbedaan skor validasi dan training banyak filter

Pada grafik *loss* dan *accuracy* proses *training* dan validasi untuk tiap variasi terlihat bahwa model *medium filters* dan *large filters* memiliki nilai *loss* dan *accuracy* yang bisa dibilang sama. Namun karena program menggunakan fungsi *callbacks.EarlyStopping*, terlihat bahwa model *large filters* mencapai tingkat optimal lebih cepat beberapa *epochs* sebelum model *medium filters*.

Number of Filter	Accuracy	F1-Score
16 - 32 - 64	0.7075	0.7082
32 - 64 - 128	0.7299	0.7286
64 - 128 - 256	0.7288	0.7294

Hasil prediksi pada tabel menyatakan bahwa kedua model kurang lebih memiliki kapabilitas yang sama, dengan *medium filters* sedikit unggul pada nilai *accuracy* sedangkan *large filters* unggul pada F1-score. Namun melihat analisis pada tahap *training* dan validasi, model *large filters* lebih baik karena lebih cepat mencapai titik optimal. Jika dilihat berdasarkan *trend* hasil yang lebih baik didapatkan pada jumlah filter yang tergolong cukup banyak.

3. Pengaruh ukuran filter per layer konvolusi



Gambar 8. Plot grafik perbedaan skor validasi dan training ukuran filter

Untuk tahap *training*, model *mixed kernels* merupakan model dengan *loss* terendah dan *accuracy* tertinggi. Berbanding terbalik ketika tahap validasi, dimana model *small kernels* memiliki nilai yang paling baik dan model *mixed kernels* menjadi model dengan performa terburuk.

Size of Filter	Accuracy	F1-Score
3 x 3	0.7171	0.7168
5 x 5	0.6920	0.6909
Mixed (3x3, 5x5, 7x7)	0.7058	0.7082

Ketika melihat hasil tes prediksi, model *small kernels* merupakan model dengan akurasi dan F1-score terbaik walaupun hanya unggul sedikit dibanding dengan model *mixed kernels*.

4. Pengaruh jenis pooling layer

Sama seperti 2 model yang dibahas sebelumnya pada bagian pengaruh ukuran filter per layer konvolusi, kedua model dengan variasi *pooling layer* ini memiliki hasil yang berbanding terbalik pada tahap *training* dan validasi. Model *max pooling* unggul pada tahap *training* sedangkan model *average pooling* unggul pada tahap validasi. Walaupun begitu, perbedaan nilai kedua model terbilang kecil.

Pooling Type	Accuracy	F1-Score
Max Pooling	0.7078	0.7088
Average Pooling	0.7188	0.7197

Secara keseluruhan, model *average pooling* lebih unggul sedikit dibanding model *max pooling*. Namun jika melihat dari efisiensi pencapaian nilai optimal, model *max pooling* lebih unggul karena sudah lebih dulu mencapai kondisi optimal beberapa *epochs* sebelum model *average pooling*.

5. Perbandingan Model Keras dengan Forward from Scratch

Perbandingan kedua model menggunakan spesifikasi konfigurasi sebagai berikut :

- Convolution Layer(filters = 32, filters_size = (3,3), activation = ReLu, Stride = 1, pool_size = (2,2), stride = (2,2), pool_type = max)
- Convolution Layer(filters = 64, filters_size = (3,3), activation = ReLu, Stride = 1, pool_size = (2,2), stride = (2,2), pool_type = max)
- Flatten
- Dense Layer(units = 128, activation = ReLu)
- Dense Layer(units = 128, activation = softMax)

Perbandingan kedua model menggunakan spesifikasi konfigurasi sebagai berikut :

Model	Accuracy	F1-Score
Keras	0.6902	0.6897
Scratch	0.6902	0.6897

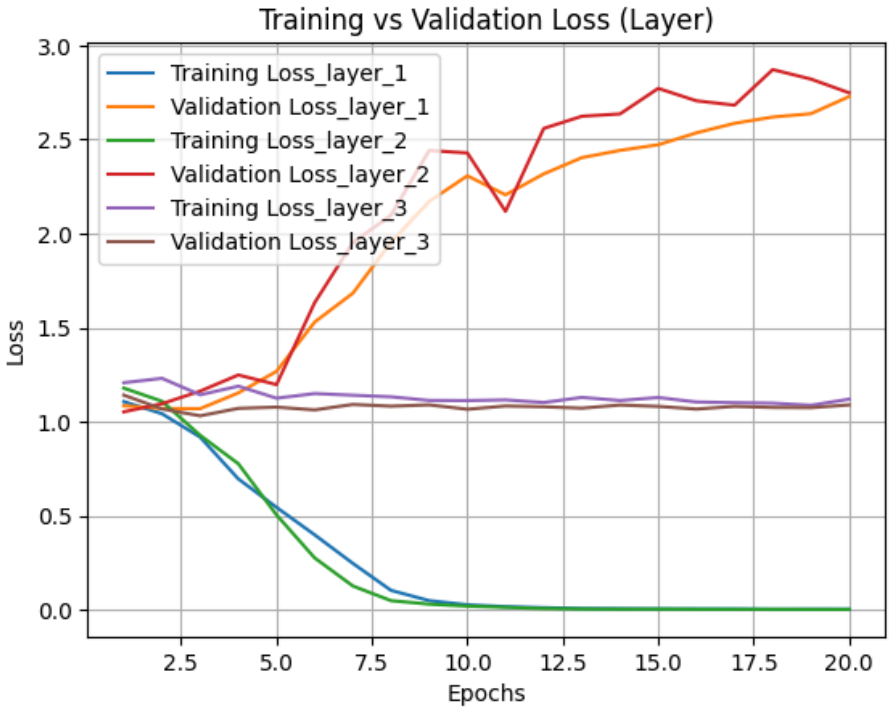
Hasil dari kedua model memberikan nilai *accuracy* dan F1-score yang persis. Hal ini menandakan bahwa model yang diimplementasikan secara *scratch* memiliki performa yang baik atau bisa dibilang persis dengan model dari keras. Walaupun begitu, prediksi model *scratch* membutuhkan waktu 4146,86 detik atau setara dengan 1 jam 9 menit 7 detik yang sangatlah lama dibanding dengan waktu prediksi model keras yang tidak sampai 5 menit. Hal ini terjadi karena tahap-tahap yang dilalui model tidak *optimized* atau bisa dibilang tidak efisien.

B. Pengujian *Simple Recurrent Neural Network*

1. Pengaruh jumlah layer RNN

Pada pengujian bagian ini, akan dilakukan variasi pada jumlah *layer* RNN yang terdapat pada model mesin. Berikut adalah konfigurasi dari ketiga RNN yang akan diujikan.

Nama LSTM	Konfigurasi	F1 Score	Akurasi
RRN_1	SimpleRNN(64)	0.4066	0.4050
RRN_2	SimpleRNN(64, return_sequences=True) ×2 + SimpleRNN(64)	0.3841	0.4025
RNN_3	SimpleRNN(64, return_sequences=True) ×4 + SimpleRNN(64)	0.2504	0.3800



Gambar 9. Plot grafik perbedaan skor validasi dan training jumlah layer

Berdasarkan hasil tes didapatkan bahwa Hasil yang diperoleh menunjukkan model dengan satu layer RNN (RNN_1) mengalami overfitting cukup parah. Hal ini terlihat dari nilai training loss yang menurun drastis sementara validation loss justru meningkat signifikan setelah beberapa epoch. Ini menandakan bahwa model terlalu sederhana dan tidak mampu melakukan generalisasi dengan baik terhadap data validasi. Sementara itu, model dengan tiga layer RNN (RNN_2) menunjukkan performa yang lebih stabil dibandingkan RNN_1. Validation loss-nya masih

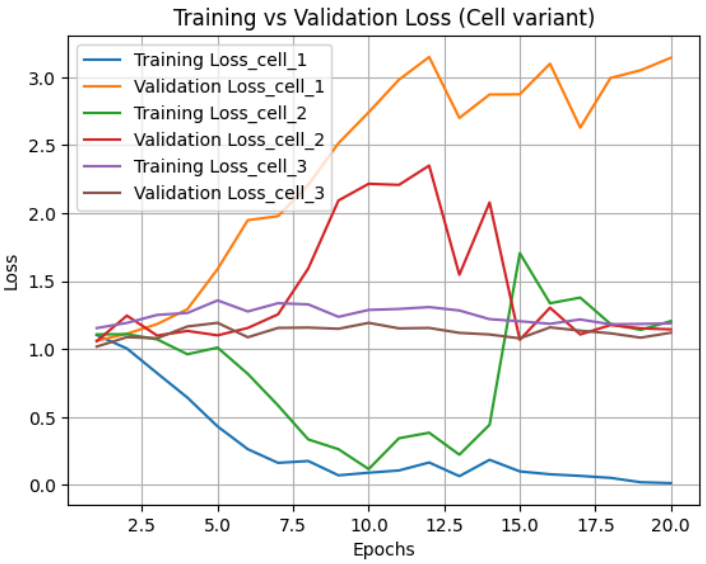
cenderung meningkat, namun tidak secepat dan sebesar model pertama, menunjukkan bahwa penambahan layer membantu model menangkap pola data yang lebih kompleks. Adapun model dengan lima layer RNN (RNN_3) menunjukkan hasil paling stabil di antara ketiganya. Baik training loss maupun validation loss-nya cenderung mendatar dan cukup dekat satu sama lain, yang mengindikasikan bahwa model ini mampu menghindari overfitting. Namun, terdapat kemungkinan bahwa model ini mengalami underfitting ringan karena arsitekturnya yang terlalu kompleks untuk jumlah data yang ada.

Secara keseluruhan, dapat disimpulkan bahwa penambahan jumlah layer RNN dapat meningkatkan kemampuan model dalam memahami pola data dan mengurangi overfitting, tetapi juga berisiko menyebabkan underfitting jika tidak diimbangi dengan jumlah dan kompleksitas data yang memadai. Oleh karena itu, pemilihan jumlah layer perlu disesuaikan dengan karakteristik dataset yang digunakan.

2. Pengaruh banyak *cell* RNN per layer

Pada pengujian bagian ini, akan dilakukan variasi pada jumlah *cell* RNN yang terdapat pada model mesin. Berikut adalah konfigurasi dari ketiga RNN yang akan diujikan.

Nama LSTM	Konfigurasi	F1 Score	Akurasi
RRN_1	64 Cell	0.3951	0.3950
RRN_2	128 Cell	0.2806	0.3350
RNN_3	256 Cell	0.2563	0.3300



Gambar 10. Plot grafik perbedaan skor validasi dan training banyak cell

Pada pengujian ini, model diuji dengan variasi jumlah unit (cell) pada layer SimpleRNN untuk melihat bagaimana kapasitas sel mempengaruhi performa. Model pertama (Cell_1) dengan 64 cell menunjukkan penurunan training loss yang sangat tajam, namun disertai dengan peningkatan drastis pada validation loss, mengindikasikan overfitting yang cukup parah. Model ini memiliki kapasitas terbatas, sehingga hanya mampu menghafal data latih tanpa bisa menggeneralisasi.

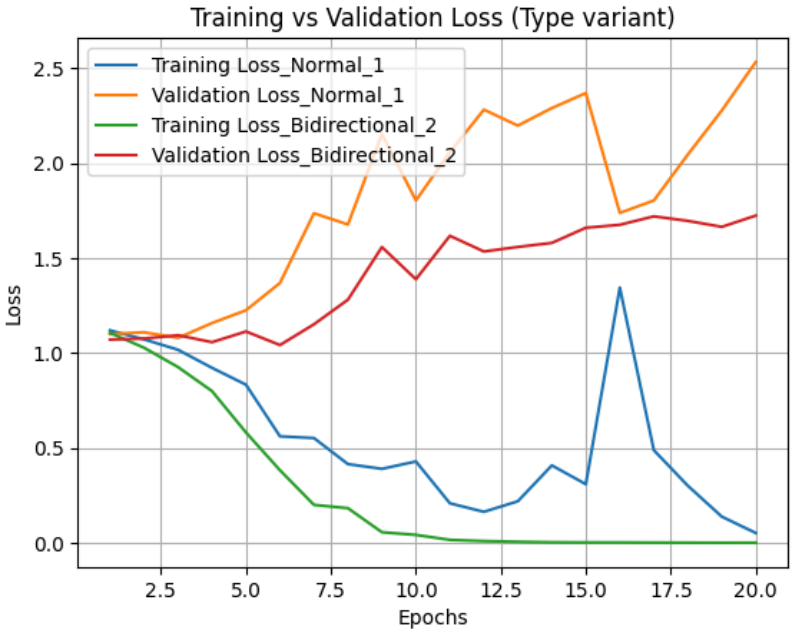
Model kedua (Cell_2) dengan 128 cell memiliki kapasitas lebih besar, memungkinkan model memahami relasi dalam data dengan lebih baik. Training loss juga menurun cepat, tetapi validation loss masih fluktuatif dan cenderung meningkat, meskipun tidak seburuk Cell_1. Ini menunjukkan bahwa model mulai lebih baik dalam mengenali pola, namun masih belum optimal dalam generalisasi.

Model ketiga (Cell_3) dengan 256 cell memperlihatkan tren yang paling stabil. Baik training loss maupun validation loss menurun perlahan dan tetap berada pada nilai yang relatif dekat satu sama lain. Ini mengindikasikan keseimbangan antara kemampuan model dalam belajar dan generalisasi. Walaupun terdapat sedikit underfitting, model ini menunjukkan performa yang paling stabil dan menjanjikan, terutama jika jumlah data ditambah.

3. Pengaruh jenis *layer* RNN berdasarkan arah

Pada pengujian bagian ini, akan dilakukan variasi pada jenis *layer* RNN yang terdapat pada model mesin. Berikut adalah konfigurasi dari ketiga RNN yang akan diujikan.]

Nama LSTM	**Konfigurasi**	**F1 Score**	**Akurasi**
RRN_1	SimpleRNN (1 arah)	0.4080	0.4050
RRN_2	Bidirectional(SimpleRNN)	0.4487	0.4525



Gambar 11. Plot grafik perbedaan skor validasi dan training jenis layer

Eksperimen ini membandingkan dua jenis RNN berdasarkan arah proses pembacaan data: satu arah (unidirectional) dan dua arah (bidirectional). Model Normal_1 menggunakan SimpleRNN biasa yang memproses input hanya dari masa lalu ke masa depan. Hasilnya, meskipun training loss menurun, validation loss justru meningkat tajam. Ini mengindikasikan bahwa model terlalu fokus pada data latih dan gagal menggeneralisasi, sehingga mengalami overfitting.

Sebaliknya, model Bidirectional_2 menggunakan Bidirectional RNN yang memproses data dalam dua arah (maju dan mundur). Dengan arsitektur ini, model dapat memahami konteks secara lebih menyeluruh. Hasilnya, training loss menurun dengan cepat dan validation loss cenderung stabil. Ini menunjukkan bahwa penggunaan layer bidirectional membantu model memahami struktur data secara lebih lengkap, sehingga mampu mengurangi overfitting dan meningkatkan generalisasi.

Secara keseluruhan, penggunaan Bidirectional RNN memberikan dampak positif terhadap performa model, terutama dalam menjaga keseimbangan antara kemampuan belajar dan kemampuan generalisasi.

4. Pengujian RNN *from scratch*

Model	**F1 Score**
From Scratch	0.5596044118211898
RNN_2 (Bidirectional)	0.5596044118211898

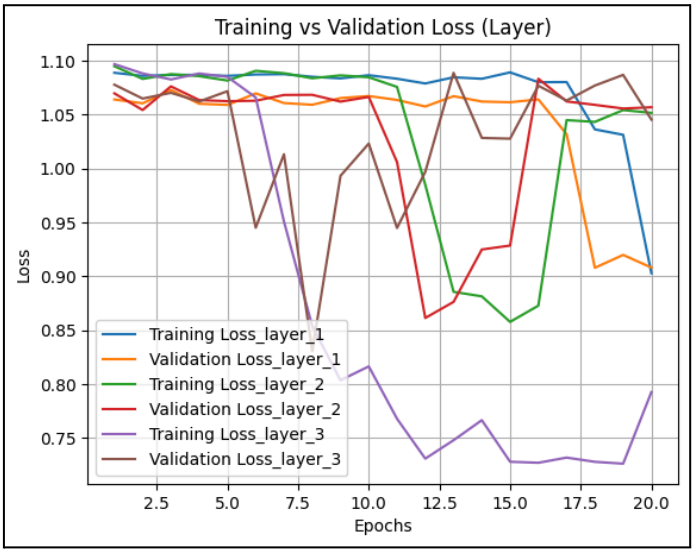
Hasil dari kedua model memberikan nilai accuracy dan F1-score yang persis. Hal ini menandakan bahwa model yang diimplementasikan secara scratch memiliki performa yang baik atau bisa dibilang persis dengan model dari keras.

C. Pengujian *Long Short Term Memory*

1. Pengaruh jumlah layer LSTM

Pada pengujian bagian ini, akan dilakukan variasi pada jumlah *layer* LSTM yang terdapat pada model mesin. Berikut adalah konfigurasi dari ketiga RNN yang akan diujikan.

Nama LSTM	Konfigurasi	F1 Score	Akurasi
LSTM_Layer_1	LSTM(64) 1 *layer* LSTM	0.4848	0.5550
LSTM_Layer_2	LSTM(64) LSTM(64) 2 *layer* LSTM	0.2073	0.3775
LSTM_Layer_3	LSTM(64) LSTM(64) LSTM(64) LSTM(64) 4 *layer* LSTM	0.3219	0.4475



Gambar 12. Plot grafik perbedaan skor validasi dan training

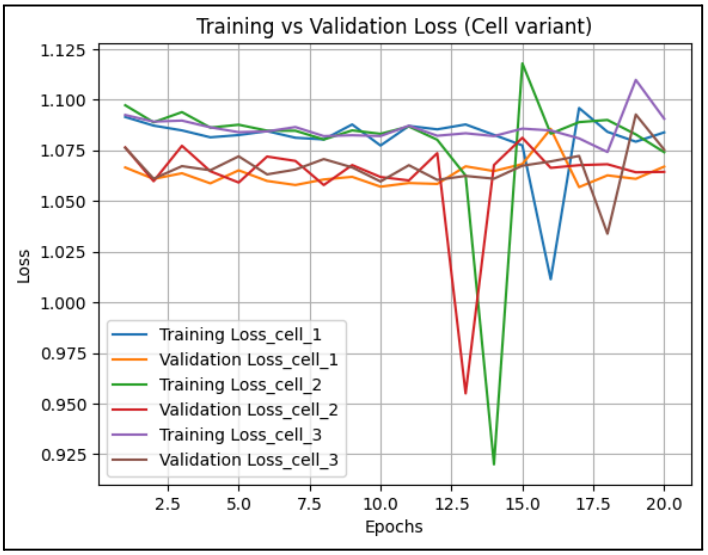
Dari Grafik, terlihat bahwa LSTM dengan *layer* yang lebih banyak menghasilkan skor *training loss* yang lebih kecil, sedangkan untuk LSTM_1 dan LSTM_2 tidak memiliki begitu banyak perbedaan. Disimpulkan bahwa LSTM_Layer_3 cenderung melakukan *overfitting* pada *dataset*, memiliki performa latihan yang baik tapi generalisasi yang buruk. LSTM_Layer_1 memiliki kapabilitas

di terbaik di antara LSTM_Layer_2 dan 3, berdasarkan performa grafik dan skor F-1 dan Akurasi.

2. Pengaruh banyak cell LSTM per layer

Pada pengujian bagian ini, akan dilakukan variasi pada konfigurasi *cell* LSTM yang terdapat pada model mesin. Berikut adalah konfigurasi dari ketiga RNN yang akan diujikan.

Nama LSTM	Konfigurasi	F1 Score	Akurasi
LSTM_Cell_1	LSTM(64) 64 *cell* LSTM	0.2059	0.3750
LSTM_Cell_2	LSTM(128) 128 *cell* LSTM	0.2069	0.3775
LSTM_Cell_3	LSTM(256) 256 *cell* LSTM	0.2217	0.3825



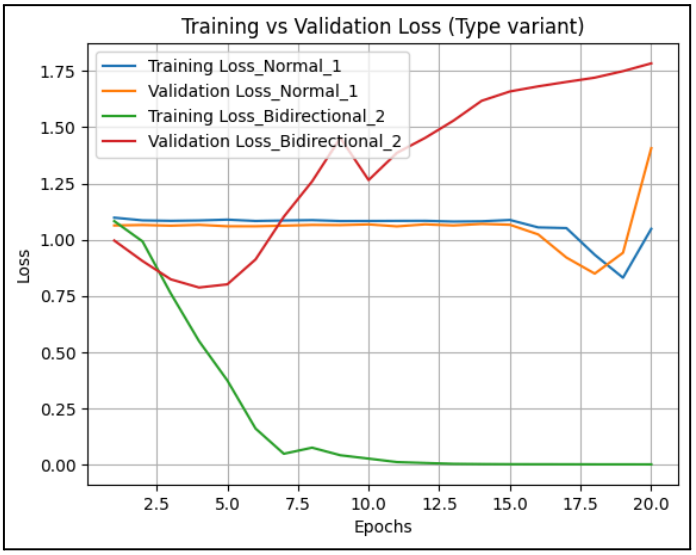
Gambar 13. Plot grafik perbedaan skor validasi dan training

Berdasarkan grafik *loss* dan evaluasi, LSTM Cell 3 menunjukkan performa terbaik dibandingkan Cell 1 dan 2, dengan akurasi tertinggi (38.25%) dan F1 Score terbesar (0.2217), meskipun terdapat sedikit fluktuasi pada loss. *Cell* 1 cenderung *underfitting* karena loss stabil tinggi tanpa perbaikan signifikan, sementara *Cell* 2 sempat mengalami penurunan loss drastis namun pelatihannya tidak stabil. Secara keseluruhan, *Cell* 3 merupakan model dengan performa terbaik dalam kasus ini.

3. Pengaruh jenis layer LSTM berdasarkan arah

Pada pengujian bagian ini, akan dilakukan variasi pada konfigurasi arah LSTM yang terdapat pada model mesin. Berikut adalah konfigurasi dari ketiga RNN yang akan diujikan.

Nama LSTM	Konfigurasi	F1 Score	Akurasi
LSTM_Type_1	LSTM(128) 128 *cell* LSTM	0.2234	0.3775
LSTM_Type_2	Bidirectional(LSTM(128)) 128 *cell* LSTM *bidirectional*	0.6743	0.6975



Gambar 14. Plot grafik perbedaan skor validasi dan training

Berdasarkan grafik, model *bidirectional* (model_type_2) menunjukkan performa jauh lebih baik dibandingkan model Normal (model_type_1), dengan akurasi mencapai 69.75% dan F1 Score sebesar 0.6743, jauh di atas model Normal yang hanya mencapai 37.75% akurasi dan F1 Score 0.2234. Meskipun *training loss* pada model Bidirectional sangat rendah, validation loss-nya meningkat tajam setelah beberapa epoch, mengindikasikan *overfitting*. Sebaliknya, model reguler menunjukkan *loss* yang stabil namun tidak terlalu rendah, mencerminkan *underfitting*. Dengan demikian, meskipun perlu penanganan *overfitting*, model *bidirectional* tetap merupakan pilihan yang lebih unggul karena kemampuannya belajar lebih efektif dan menghasilkan generalisasi yang lebih baik berdasarkan metrik kinerja.

4. Pengujian LSTM *from scratch*

Model	F1 Score
From Scratch LSTM 2 Bidirectional	0.711322303552166

LSTM_2 (Bidirectional)	0.7187758024196006

Kesimpulan & Saran

Berdasarkan hasil pengujian variasi *hyperparameter* pada model CNN, ditemukan bahwa untuk dataset CIFOR-10 yang digunakan, konfigurasi model yang paling baik untuk tiap variasinya berdasarkan F1_score adalah 3 *layer* konvolusi, jumlah filter sebanyak 64, 128, dan 256, besar filter sebesar 3x3, dan tipe *pooling average*. Lalu implementasi *Forward Propagation* dari *scratch* mencapai hasil *accuracy* dan F1-score yang persis sama dengan model Keras namun membutuhkan waktu yang jauh lebih lama walaupun wajar. Hal ini menandakan bahwa program *Forward* yang dibuat sudah benar dan tidak memiliki kesalahan model apapun.

Berdasarkan eksperimen terhadap tiga jenis hyperparameter—jumlah layer RNN, jumlah cell per layer, dan jenis layer RNN—dapat disimpulkan bahwa pemilihan dan kombinasi yang tepat sangat mempengaruhi performa model. Terlalu banyak layer atau cell dapat menyebabkan overfitting, seperti yang terlihat pada model dengan banyak cell atau layer bertumpuk, sementara penggunaan Bidirectional RNN terbukti efektif dalam meningkatkan generalisasi dan stabilitas performa. Oleh karena itu, keseimbangan antara kompleksitas arsitektur dan kemampuan generalisasi sangat penting untuk menghasilkan model RNN yang optimal.

Berdasarkan percobaan kombinasi dan parameter LSTM, disimpulkan bahwa pengubahan *layer* menjadi bidireksional memberikan efek paling berpengaruh terhadap akurasi dan skor model.

Juga, kami berhasil dalam mengimplementasikan model *forward propagation from scratch*, hal ini diindikasikan dengan berbedaan rentang skor F-1 yang tidak begitu jauh.

Saran yang dapat kami berikan untuk tugas besar kali ini adalah perbanyak eksplor dan *fine-tune* konfigurasi ataupun arsitektur yang paling optimal untuk mendapatkan model dengan performa terbaik. Pada pengerjaan tugas besar ini, dikarenakan batasan waktu dan beban kerja lain yang harus kami kerjakan, kami tidak sempat mendalami dan mencari tahu lebih terkait hal ini. Maka dari itu, walaupun pembelajaran ini sudah kami anggap baik, kami yakin dapat dikembangkan menjadi lebih baik lagi.

Pembagian Tugas

Deskripsi Tugas	Penanggung Jawab
CNN image classification with keras	13522049
CNN from scratch	13522049
RNN text classification with keras	13522031, 13522039
RNN from scratch	13522031, 13522039
LSTM text classification with keras	13522031
LSTM from scratch	13522031
Laporan & Pengujian setiap model	13522031, 13522039, 13522049

Lampiran

Pranala *notebook* RNN :
https://colab.research.google.com/drive/1NEaQ6b0P2204eRttePWvvF85LqQImUTs#scrollTo=GMUA-YsPCmeu

Pranala *notebook* LSTM :
https://colab.research.google.com/drive/1ZuXxFDh7M-DDToSqYOtr-5zTAIRZIvU5#scrollTo=rX3y6bikUwrw