LAPORAN TUGAS BESAR 02

IF3270 Pembelajaran Mesin

CNN, RNN, dan LSTM

Disusun Oleh:

Juan Alfred Widjaya	13522073
Albert	13522081
Ivan Hendrawan Tan	13522111

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
DESKRIPSI PERMASALAHAN

Tugas Besar 2 ini membahas mengenai klasifikasi data menggunakan tiga pendekatan model pembelajaran mesin, yaitu Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), dan Long Short-Term Memory (LSTM). Model CNN digunakan untuk menyelesaikan permasalahan klasifikasi citra pada dataset CIFAR-10 yang terdiri dari gambar berwarna berukuran 32x32 piksel dalam 10 kelas objek yang berbeda. Model ini dieksplorasi melalui berbagai konfigurasi jumlah layer konvolusi, banyaknya filter, ukuran kernel, dan jenis pooling yang digunakan (max/average), diuji dengan metrik evaluasi macro F1-score. Selain implementasi menggunakan library Keras, forward propagation juga dibuat dari scratch untuk dibandingkan dengan forward propagation milik Keras.

Sementara itu, model Simple RNN dan LSTM digunakan untuk menyelesaikan permasalahan klasifikasi sentimen pada data teks bahasa Indonesia menggunakan dataset NusaX-Sentiment. Kedua model dirancang untuk menerima input berupa sekuens kata yang diproses melalui tahap tokenisasi dan embedding, lalu dilewatkan ke dalam arsitektur RNN atau LSTM, dan diakhiri dengan layer klasifikasi. Eksperimen dilakukan dengan memvariasikan jumlah layer, jumlah unit neuron per layer, serta penggunaan arsitektur bidirectional. LSTM digunakan untuk mengatasi keterbatasan RNN dalam menangani dependensi jangka panjang antar token. Setiap model dievaluasi dengan macro F1-score dan forward propagation juga dibuat dari scratch.

BAB II
PEMBAHASAN

A. Penjelasan Implementasi

a. Deskripsi Kelas

i. Deskripsi Kelas CNN beserta Deskripsi Atribut dan Methodnya

Kelas atau Fungsi	Deskripsi
build_cnn_scratch	Membangun model CNN untuk tes *forward propagation* dari *scratch*
run_cnn_scratch	Menjalankan model CNN untuk tes *forward propagation* dari *scratch*
Conv2D_scr	Layer *convolution* yang dibuat secara *scratch* dengan NumPy
ReLU_scr	Mengimplementasikan ReLU
MaxPool2D_scr	Layer *max pooling* yang dibuat secara *scratch* dengan NumPy
AvgPool2D_scr	Layer *average pooling* yang dibuat secara *scratch*
Flatten_scr	Layer *flatten* yang dibuat secara *scratch*
Dense_scr	Layer *dense* yang dibuat secara *scratch*
Softmax_scr	Mengimplementasikan fungsi aktivasi softmax
scratch_predict	Menjalankan *forward propagation* untuk layer per layer yang dibuat dari *scratch*
build_layer	Membangun ulang layer dari model Keras yang telah dibuat sebelumnya secara *scratch*
build_cnn_model	Membuat model CNN untuk tes eksperimen beberapa *hyper parameter*
plot_loss_curves	Membuat *plotting* untuk *training*

	loss dan *validation loss*
show_f1_table	Membuat tabel untuk menampilkan hasil F1-score dari eksperimen
run_experiments	Menjalankan eksperimen yang diinginkan

ii. Deskripsi Kelas RNN beserta Deskripsi Atribut dan Methodnya

Kelas atau Fungsi	Deskripsi
build_rnn_model(vocab_size : int, num_layers : int, units, bidirectional : bool)	Fungsi pembangun model RNN di Keras. Membuat arsitektur Embedding → (Bi)SimpleRNN (multi-layer, units bisa custom per layer) → Dropout → Dense. Mendukung dropout, regularisasi, learning rate, dan input sequence length yang bisa diatur.
softmax	Fungsi yang menghitung softmax secara manual, dipakai pada output layer Dense.
tanh	Fungsi yang menerapkan aktivasi tanh secara manual.
EmbeddingLayer	Kelas manual layer embedding. Menyimpan matriks bobot embedding. Method forward(tokens) melakukan lookup ke vektor embedding dan men-nolkan embedding untuk token padding (token 0)
SimpleRNNLayer	Kelas manual untuk layer SimpleRNN satu arah. Menyimpan bobot input (Wx), bobot recurrent (Wh), bias (b), dan atribut return_sequences. Method utama: forward(x) (iterasi seluruh timestep, update hidden state manual, masking PAD).
BidirectionalSimpleRNN	Kelas pembungkus dua SimpleRNNLayer (maju & mundur) secara manual. Method utama: forward(x), menjalankan dua SimpleRNN (normal & reversed),

	lalu menggabungkan hasil dengan concatenation.
DenseLayer	Kelas untuk fully connected (Dense) output. Menyimpan bobot dan bias Dense. Method forward(h) mengalikan bobot, menambah bias, dan softmax.
NumpyRNNModel	NumpyRNNModel menerima bobot dan konfigurasi dari model Keras yang sudah dilatih (file .h5 dan vectorizer .keras), lalu memetakan semua layer ke kelas-kelas NumPy (EmbeddingLayer, SimpleRNNLayer, BidirectionalSimpleRNN, DenseLayer). Setiap prediksi (baik batch maupun satuan) dilakukan secara manual. Kelas ini juga mendukung batching untuk inference yang efisien.
set_global_seed(seed : int)	Fungsi utilitas, memastikan reproducibility dengan mengatur seed random Python, NumPy, dan TensorFlowdataset
load_raw	Fungsi load data dari tiga file CSV (train/valid/test), mapping label string ke integer, dan mengubah ke array NumPydataset.
prepare_datasets(seed : int)	Fungsi utama preprocessing: load data, adaptasi vocab ke TextVectorization, batching, dan shuffle dataset training. Output: tuple (train_ds, val_ds, (vectorizer, x_test, y_test)) untuk keperluan pipeline model.

iii. Deskripsi Kelas LSTM beserta Deskripsi Atribut dan Methodnya

Kelas atau Fungsi	Deskripsi
load_data_from_github()	Fungsi ini bertanggung jawab untuk mengambil dataset analisis sentimen

	(set pelatihan, validasi, dan pengujian) dari file CSV yang berlokasi di GitHub. Fungsi ini mengembalikan tiga DataFrame pandas (df_train, df_val, df_test). Jika terjadi kesalahan selama pemuatan, fungsi ini mencetak pesan kesalahan dan mengembalikan None untuk ketiga DataFrame tersebut.
clean_text(text: str)	Fungsi ini melakukan pra-pemrosesan terhadap string masukan text agar sesuai untuk tugas pemrosesan bahasa alami. Fungsi ini pertama-tama memeriksa apakah masukannya adalah string, mengembalikan string kosong jika tidak. Jika ya, fungsi ini menghapus URL, mention gaya Twitter (misalnya, @username), dan karakter apa pun yang bukan alfanumerik atau spasi, kemudian mengubah teks menjadi huruf kecil dan menghilangkan spasi di awal atau akhir sebelum mengembalikan string yang sudah dibersihkan.
prepare_data(max_vocab_size=10000, max_sequence_length=256, batch_size=32)	Fungsi ini mengatur seluruh alur persiapan data dengan pertama-tama memanggil load_data_from_github() untuk mendapatkan DataFrame mentah, dan jika berhasil, menerapkan clean_text ke kolom 'text' dari setiap DataFrame sehingga membuat kolom 'cleaned_text'. Selanjutnya, fungsi ini menggunakan LabelEncoder untuk mengubah kolom 'label' kategorikal menjadi representasi numerik untuk set pelatihan, validasi, dan pengujian, sekaligus menentukan num_classes. Terakhir, fungsi ini menggunakan TextVectorization dari Keras untuk beradaptasi dengan teks pelatihan dan mengubah data teks yang sudah dibersihkan (X_train, X_val, X_test) menjadi urutan bilangan bulat dengan panjang tetap max_sequence_length, yang kemudian dikelompokkan ke dalam

	objek Dataset (train_ds, val_ds, test_ds). Fungsi ini mengembalikan dataset-dataset tersebut, vectorize_layer yang telah dikonfigurasi, num_classes, dan sebuah tuple test_data_for_scratch yang berisi fitur tes yang telah divektorisasi dan label yang telah dikodekan sebagai array NumPy.
build_keras_lstm_model(num_classes, vocab_size, emb_dim, seq_len, lstm_layers_config, dense_layers_config=None, dropout_rate=0.5)	Fungsi ini membangun model Keras Sequential yang dirancang untuk klasifikasi sekuens menggunakan lapisan LSTM. Fungsi ini dimulai dengan lapisan input dan lapisan Embedding yang dikonfigurasi dengan vocab_size, emb_dim, dan seq_len. Kemudian, secara dinamis menambahkan satu atau lebih lapisan LSTM (baik 'unidirectional' maupun 'bidirectional') berdasarkan daftar lstm_layers_config, memastikan return_sequences bernilai true untuk semua kecuali lapisan LSTM terakhir jika ada beberapa yang ditumpuk. Sebuah lapisan Dropout dengan dropout_rate yang ditentukan ditambahkan untuk regularisasi. Secara opsional, lapisan Dense perantara dengan aktivasi 'relu' dapat ditambahkan berdasarkan dense_layers_config sebelum lapisan output Dense akhir dengan aktivasi 'softmax' dan num_classes unit. Model dikompilasi dengan optimizer 'adam', loss 'sparse_categorical_crossentropy', dan metrik 'accuracy', lalu dikembalikan.
train_model(model, train_ds, val_ds, epochs=10, model_name=None)	Fungsi ini melatih model Keras yang diberikan menggunakan dataset pelatihan (train_ds) dan validasi (val_ds) yang disediakan selama jumlah epoch tertentu. Setelah pelatihan, fungsi ini membuat direktori 'results' dan 'models' jika belum ada. Riwayat pelatihan (berisi loss pada train dan val per epoch)

	disimpan sebagai file JSON di direktori 'results', dan bobot model disimpan dalam format H5 di direktori 'models', keduanya dinamai menggunakan model_name. Fungsi ini mengembalikan history pelatihan dan model yang telah dilatih.
evaluate_model(model, test_ds)	Fungsi ini mengevaluasi kinerja model Keras yang telah dilatih pada dataset tes (test_ds) tertentu. Fungsi ini pertama-tama menggabungkan semua label sebenarnya dari set tes, kemudian menggunakan model untuk memprediksi probabilitas untuk data tes, dan akhirnya mengubah probabilitas ini menjadi prediksi kelas dengan mengambil argmax. Fungsi ini menghitung dan mengembalikan skor F1 makro menggunakan fungsi f1_score dari scikit-learn.
softmax(x)	Fungsi ini mengimplementasikan fungsi aktivasi Softmax menggunakan NumPy.
sigmoid(x)	Fungsi ini mengimplementasikan fungsi aktivasi Sigmoid menggunakan NumPy.
tanh(x)	Fungsi ini mengimplementasikan fungsi aktivasi Tanh menggunakan NumPy.
relu(x)	Fungsi ini mengimplementasikan fungsi aktivasi ReLU menggunakan NumPy.
EmbeddingScratch	Kelas ini menyimpan matriks embedding dan metode forward-nya mengubah urutan angka (indeks kata) menjadi urutan vektor embedding berdasarkan matriks tersebut.
EmbeddingScratch::forward(self, x)	Metode ini melakukan pencarian embedding. Diberikan masukan x (sekumpulan sekuens bilangan bulat), metode ini menggunakan

	bilangan bulat ini sebagai indeks untuk mengambil vektor embedding yang sesuai dari self.embedding_matrix, lalu mengembalikan kumpulan sekuens yang telah di-embed.
DropoutScratch	Kelas ini adalah lapisan dropout. Konstruktornya (__init__) menyimpan rate dropout.
DropoutScratch::forward(self, x, training=False)	Metode ini mensimulasikan operasi dropout selama forward pass. Metode ini hanya mengembalikan masukan x tanpa perubahan, karena dropout umumnya hanya diterapkan selama pelatihan untuk mencegah overfitting.
DenseScratch	Kelas ini mengimplementasikan lapisan fully connected (dense) menggunakan NumPy. Konstruktornya (__init__) menerima daftar weights (berisi matriks bobot W dan vektor bias b) dan nama fungsi activation (standarnya 'relu'), lalu menyimpan ini sebagai atribut instance.
DenseScratch::forward(self, x)	Metode ini melakukan forward pass untuk lapisan dense. Metode ini menghitung perkalian dari masukan x dan matriks bobot self.W, menambahkan bias self.b, dan kemudian menerapkan fungsi aktivasi yang ditentukan (softmax atau relu) pada hasilnya sebelum mengembalikannya.
LSTMSingleScratch	Kelas ini mengimplementasikan sel LSTM tunggal searah (unidirectional) dan logika lapisannya menggunakan NumPy, dirancang untuk pemrosesan batch. Konstruktornya (__init__) menerima weights LSTM gaya Keras (daftar yang berisi kernel W, kernel rekuren U, dan bias b) dan sebuah flag boolean return_sequences. Konstruktor ini membagi weights

	tersebut menjadi matriks dan vektor individual untuk gerbang input, forget, sel, dan output (misalnya, W_i, U_i, b_i) dan menyimpannya bersama dengan jumlah unit LSTM (n_units) serta flag return_sequences.
LSTMSingleScratch::forward(self, x)	Metode ini menjalankan forward pass untuk lapisan LSTM searah. Metode ini memproses sekuens masukan x (batch_size, sequence_length, embedding_dim) langkah demi langkah seiring waktu. Pada setiap langkah waktu t, metode ini menghitung gerbang input, forget, dan output menggunakan aktivasi sigmoid, dan kandidat cell state menggunakan aktivasi tanh, kemudian memperbarui cell state c_t dan hidden state h_t. Jika self.return_sequences adalah True, metode ini menyimpan dan mengembalikan semua hidden state untuk setiap sekuens dalam batch; jika tidak, metode ini hanya mengembalikan hidden state terakhir h_t untuk setiap sekuens.
BidirectionalLSTMScratch	Kelas ini mengimplementasikan lapisan LSTM dua arah (bidirectional) menggunakan NumPy dengan memanfaatkan dua instance LSTMSingleScratch. Konstruktornya (__init__) menerima weights LSTM Bidirectional gaya Keras (yang secara internal berisi bobot untuk LSTM maju dan mundur) dan sebuah flag boolean return_sequences. Konstruktor ini menginisialisasi sebuah LSTMSingleScratch untuk forward pass dengan separuh pertama bobot dan satu lagi untuk backward pass dengan separuh kedua, keduanya dikonfigurasi dengan flag return_sequences yang diberikan.
BidirectionalLSTMScratch::forward (self, x)	Metode ini melakukan forward pass untuk LSTM dua arah. Metode ini

	memproses masukan x menggunakan instance LSTMSingleScratch maju dan versi x yang waktunya dibalik menggunakan instance LSTMSingleScratch mundur. Jika self.return_sequences adalah True, metode ini menggabungkan sekuens penuh hidden state maju dengan sekuens hidden state mundur yang waktunya telah dibalik (kemudian dibalik lagi ke urutan waktu asli) di sepanjang sumbu terakhir. Jika self.return_sequences adalah False, metode ini menggabungkan hidden state terakhir dari LSTM maju dengan hidden state terakhir dari LSTM mundur.
NumpyLSTM	Kelas ini berfungsi sebagai kelas untuk mereplikasi model sekuensial Keras menggunakan lapisan kustom berbasis NumPy. Konstruktornya (__init__) menerima keras_model yang telah dilatih sebagai masukan. Konstruktor ini melakukan iterasi melalui setiap lapisan model Keras ini, mengidentifikasi jenisnya (embedding, LSTM bidirectional, LSTM unidirectional, dropout, dense, atau output), mengekstrak bobotnya, dan membuat instance lapisan Scratch yang sesuai (EmbeddingScratch, BidirectionalLSTMScratch, dll.) dengan bobot ini. Lapisan-lapisan scratch yang dibuat instance-nya ini disimpan dalam daftar self.layers.
NumpyLSTM::predict(self, X_data, batch_size=64)	Metode ini melakukan inferensi menggunakan urutan lapisan berbasis NumPy yang dibangun di __init__. Metode ini menerima data masukan X_data dan memprosesnya dalam batch berukuran batch_size. Untuk setiap batch, metode ini secara sekuensial melewatkan data melalui setiap lapisan di self.layers (memanggil metode forward masing-masing). Keluaran dari

	lapisan terakhir yang berupa probabilitas softmax kemudian diubah menjadi prediksi kelas dengan mengambil argmax di sepanjang sumbu kelas. Semua prediksi batch digabungkan dan dikembalikan sebagai array NumPy tunggal berisi label kelas yang diprediksi.

b. Penjelasan Forward Propagation

i. CNN

Proses dimulai dengan input berupa batch gambar CIFAR-10 yang telah dinormalisasi. Input ini dilewatkan ke layer pertama, yaitu layer konvolusi (Conv2D_scr). Layer ini mengaplikasikan sejumlah filter dari input untuk mengekstraksi fitur spasial. Padding “same” digunakan agar ukuran output tetap sama dengan input. Setelah konvolusi, hasilnya dilewatkan ke fungsi aktivasi ReLU (ReLU_scr).

Setelah proses ekstraksi fitur, hasilnya masuk ke dalam layer pooling. Dalam percobaan ini, dengan max pooling (MaxPool2D_scr) yang akan mengambil nilai maksimum dari setiap patch berukuran 2x2. Jika model menggunakan average pooling, maka layer AvgPool2D_scr akan digunakan untuk menghitung rata-rata pada setiap patch. Proses ini diulang beberapa kali tergantung kedalaman model (jumlah blok Conv2D+Pooling) yang digunakan.

Setelah semua proses konvolusi dan pooling selesai, output tadi diubah menjadi vektor 2D menggunakan layer flatten (Flatten_scr). Hasil flatten ini kemudian dilewatkan ke beberapa layer fully connected (dense). Setiap layer dense (Dense_scr) mengalikan input dengan bobot (W) dan menambahkan bias (b), kemudian hasilnya dilewatkan ke fungsi aktivasi ReLU atau softmax tergantung model yang dibuat. Pada layer terakhir, fungsi aktivasi softmax (Softmax_scr) digunakan untuk mengubah hasil akhir menjadi probabilitas klasifikasi dari 10 kelas.

Seluruh proses tersebut diimplementasikan di dalam fungsi scratch_predict(x) yang akan menyalin input, lalu melewatkannya satu per satu ke setiap layer yang telah dikonversi dari model Keras ke model scratch. Setiap layer memiliki method forward() yang melakukan proses komputasinya

secara manual. Output akhir dari scratch_predict kemudian dibandingkan dengan output hasil prediksi model Keras dengan data uji. Di akhir, kedua model dibandingkan hasil F1-score-nya.

ii. RNN

A

iii. LSTM

Kelas LSTMSingleScratch mengimplementasikan proses propagasi maju untuk LSTM satu arah sebagai berikut,

1. Inisialisasi

   Input x memiliki dimensi (batch_size, sequence_length, embedding_dim). Hidden state (h_t) dan cell state (c_t) diinisialisasi sebagai matriks nol. Dimensi self.n_units merepresentasikan jumlah unit atau dimensi dari hidden state dan cell state. Pada awal sekuens (sebelum loop timestep pertama), h_t dan c_t ini secara efektif berperan sebagai h0 dan c0. Jika self.return_sequences adalah True, all_h_t diinisialisasi untuk mengakumulasi hidden state dari setiap timestep.

2. Iterasi per Timestep

   Kode melakukan iterasi dari t = 0 hingga seq_len - 1:

   1. x_t = x[:, t, :]: Mengambil irisan data input untuk timestep t saat ini, untuk semua sampel dalam batch.
   2. i_t = sigmoid(np.dot(x_t, self.W_i) + np.dot(h_t, self.U_i) + self.b_i): Mengimplementasikan input gate (it). h_t di sisi kanan persamaan ini adalah hidden state dari timestep sebelumnya (ht−1).
   3. f_t = sigmoid(np.dot(x_t, self.W_f) + np.dot(h_t, self.U_f) + self.b_f): Mengimplementasikan forget gate (ft).
   4. c_tilde = tanh(np.dot(x_t, self.W_c) + np.dot(h_t, self.U_c) + self.b_c): Menghitung kandidat cell state (c~t).
   5. c_t = f_t * c_t + i_t * c_tilde: Memperbarui cell state. c_t di sisi kanan adalah cell state dari timestep sebelumnya (ct-1), sedangkan c_t di sisi kiri adalah cell state yang baru dihitung (ct). Operasi * di sini adalah perkalian elemen-bijaksana.

6. o_t = sigmoid(np.dot(x_t, self.W_o) + np.dot(h_t, self.U_o) + self.b_o): Mengimplementasikan output gate (ot).
7. h_t = o_t * tanh(c_t): Menghitung hidden state baru (ht).Semua operasi np.dot dan penjumlahan dilakukan secara batch-wise, memanfaatkan kemampuan NumPy untuk operasi matriks yang efisien.
8. Jika self.return_sequences adalah True, maka h_t yang baru dihitung (output untuk timestep t) disimpan ke dalam all_h_t[:, t, :].

3. Output Final

   Jika self.return_sequences adalah True, metode mengembalikan all_h_t, yang merupakan tensor dengan dimensi (batch_size, sequence_length, n_units). Ini berisi hidden state untuk setiap timestep dalam sekuens. Output semacam ini berguna untuk model sequence-to-sequence atau ketika lapisan LSTM berikutnya memerlukan seluruh sekuens output. Jika self.return_sequences adalah False, metode hanya mengembalikan h_t terakhir, yang merupakan tensor dengan dimensi (batch_size, n_units). Ini merepresentasikan ringkasan atau enkoding dari seluruh sekuens input, sering digunakan untuk tugas klasifikasi sekuens.

Kelas BidirectionalLSTMScratch mengimplementasikan propagasi maju LSTM dua arah sebagai berikut,

1. Inisialisasi

   Saat objek BidirectionalLSTMSratch dibuat, ia menginisialisasi dua instance LSTMSingleScratch yang terpisah: self.lstm_forward: Untuk pemrosesan sekuens arah maju. self.lstm_backward: Untuk pemrosesan sekuens arah mundur. Bobot (weights) yang diberikan dibagi dua; separuh pertama (weights[:3]) digunakan untuk lstm_forward dan separuh sisanya (weights[3:]) untuk lstm_backward. Ini mencerminkan bagaimana Keras biasanya menyimpan bobot untuk lapisan Bidirectional. Parameter return_sequences dari BiLSTM juga diteruskan ke kedua instance LSTM tunggal ini.

2. Propagasi Maju (forward(self, x))

Input x memiliki dimensi (batch_size, sequence_length, embedding_dim). Kemudian input x dilakukan propagasi maju melewati pemrosesan maju, pemrosesan mundur, dan penggabungan output. Berikut adalah tahapan pemrosesannya,

1. h_forward_output = self.lstm_forward.forward(x): Input x diproses oleh lstm_forward dalam urutan aslinya. Jika self.return_sequences adalah True, h_forward_output akan memiliki dimensi (batch_size, sequence_length, n_units). Jika self.return_sequences adalah False, h_forward_output akan memiliki dimensi (batch_size, n_units), yang merupakan hidden state terakhir dari pemrosesan maju.
2. x_reversed = np.flip(x, axis=1): Sekuens input x dibalik sepanjang sumbu waktu (axis=1). Jadi, jika x adalah [timestep1, timestep2, timestep3], maka x_reversed menjadi [timestep3, timestep2, timestep1].
3. h_backward_output = self.lstm_backward.forward(x_reversed): Sekuens yang sudah dibalik (x_reversed) diproses oleh lstm_backward.
4. Jika self.return_sequences adalah True, h_backward_output akan memiliki dimensi (batch_size, sequence_length, n_units). Perlu diingat bahwa sequence_length di sini mengacu pada urutan waktu yang terbalik. Misalnya, h_backward_output[:, 0, :] adalah hidden state setelah memproses elemen terakhir dari sekuens asli. Jika self.return_sequences adalah False, h_backward_output akan memiliki dimensi (batch_size, n_units). Ini adalah hidden state terakhir dari pemrosesan mundur, yang berarti telah memproses seluruh sekuens terbalik (dari elemen terakhir hingga elemen pertama sekuens asli).

3. Output Final

Logika penggabungan bergantung pada nilai self.return_sequences. Jika self.return_sequences adalah True, berikut adalah prosesnya. h_backward_output_reversed_time = np.flip(h_backward_output, axis=1). Output dari lstm_backward (h_backward_output), yang merupakan sekuens hidden state dalam

urutan waktu terbalik, perlu dibalik kembali agar urutan waktunya sesuai dengan output dari lstm_forward. Setelah operasi flip ini, h_backward_output_reversed_time[:, t, :] akan berisi hidden state mundur yang sesuai dengan timestep t dari sekuens asli. Output maju dan output mundur (yang sudah disesuaikan urutan waktunya) digabungkan (konkatenasi) sepanjang sumbu fitur (axis=2) pada return np.concatenate((h_forward_output,
h_backward_output_reversed_time), axis=2). Hasilnya adalah tensor dengan dimensi (batch_size, sequence_length, 2 * n_units), di mana 2 * n_units adalah karena n_units dari LSTM maju dan n_units dari LSTM mundur. Jika self.return_sequences adalah False, return np.concatenate((h_forward_output, h_backward_output), axis=1). Dalam kasus ini, h_forward_output adalah hidden state akhir dari pemrosesan maju (setelah melihat seluruh sekuens dari awal hingga akhir). h_backward_output adalah hidden state akhir dari pemrosesan mundur (setelah melihat seluruh sekuens dari akhir hingga awal). Keduanya langsung digabungkan sepanjang sumbu fitur (axis=1). Hasilnya adalah tensor dengan dimensi (batch_size, 2 * n_units).

B. Hasil Pengujian

a. CNN

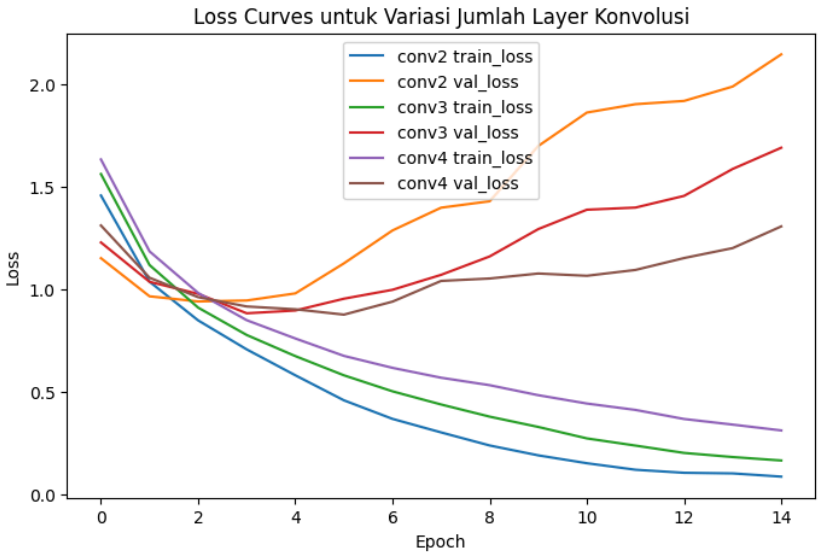
i. Pengaruh Jumlah Layer Konvolusi

Berdasarkan hasil pengujian terhadap variasi jumlah layer konvolusi, jumlah layer yang semakin banyak memperbagus hasil F1-score. Jumlah layer konvolusi sebanyak 2 memberikan F1-score sebesar 66,01%, meningkat dengan layer sebanyak 3 dengan F1-score menjadi 67,60%, lalu ketika 4 layer menjadi 67,87%.

Training loss yang dihasilkan cenderung menurun ketika layer yang digunakan lebih sedikit dan jumlah epoch ditambah, namun khusus validation loss cenderung menurun di awal epoch lalu meningkat seiring berjalannya epoch, jumlah layer juga mempengaruhi validation loss dengan menambah layer akan mengurangi loss tersebut.

Tabel 2.2.1.1.1. F1-score Variasi Jumlah Layer

Jumlah Layer	F1-Score
2	66,01%
3	67,60%
4	67,87%



Gambar 2.2.1.1.1. Kurva Loss Variasi Jumlah Layer

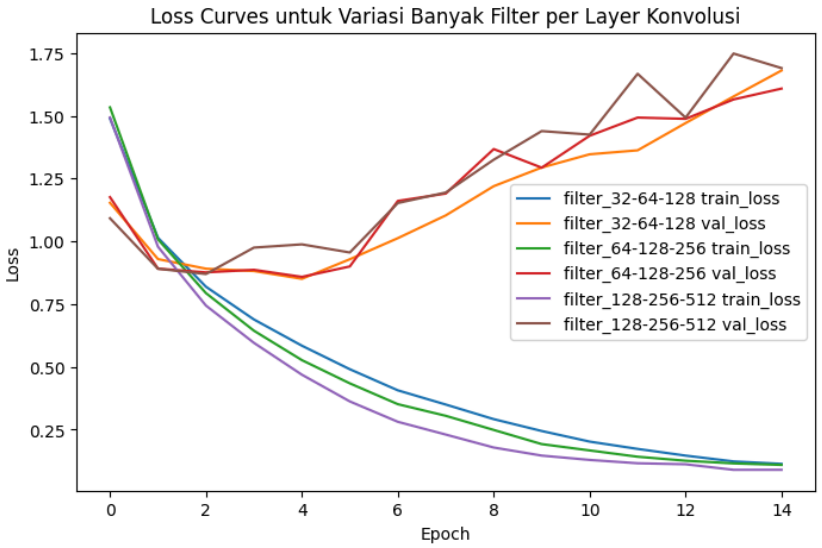
ii. Pengaruh Banyak Filter per Layer Konvolusi

Berdasarkan hasil pengujian terhadap variasi filter per layer konvolusi, penggunaan filter yang semakin banyak menghasilkan F1-score yang lebih baik, F1-score tersebut dapat dilihat di tabel bawah dengan filter sebanyak 32-64-128 (32 filter di layer konvolusi pertama, 64 filter di layer kedua, dan 128 filter di layer ketiga) menghasilkan F1-score sebesar 69,69%, 64-128-256 sebesar 71,38%, dan 128-256-512 sebesar 71,41%. Meskipun begitu, penggunaan filter yang semakin besar juga akan memperbesar penggunaan komputasi dan lama waktu yang digunakan sehingga perlu mencari titik optimalnya.

Untuk pola lossnya, filter yang semakin besar cenderung memperkecil nilai training lossnya, sedangkan validation loss yang dihasilkan cenderung semakin tidak stabil untuk jumlah filter yang lebih banyak.

Tabel 2.2.1.2.1. F1-score Variasi Banyak Filter

Banyak Filter	F1-Score
32-64-128	69,69%
64-128-256	71,38%
128-256-512	71,41%



Gambar 2.2.1.2.1. Kurva Loss Variasi Banyak Filter

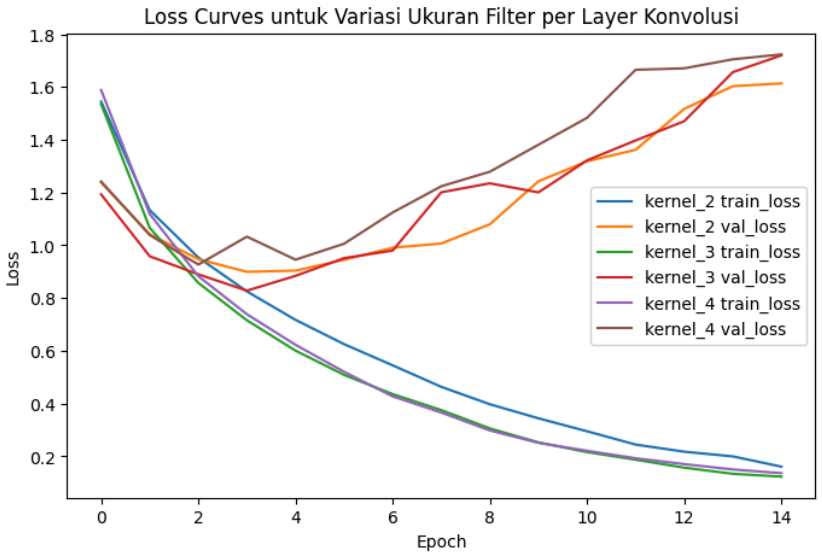
iii. Pengaruh Ukuran Filter per Layer Konvolusi

Berdasarkan hasil pengujian terhadap ukuran filter yang digunakan, ukuran yang terlalu besar atau terlalu kecil menyebabkan hasil F1-score yang memburuk seperti pada tabel di bawah ini. Filter berukuran 2x2 menghasilkan F1-score sebesar 68,17%, 3x3 sebesar 69,26%, dan 4x4 sebesar 67,6%. Filter yang terlalu kecil membuat model sulit untuk menangkap pola spasial, sedangkan filter yang terlalu besar membuat detail kecil menjadi kabur karena banyak piksel yang diekstrak.

Pola training loss yang dihasilkan semakin menurun seiring berjalannya epoch, dengan filter berukuran 3x3 memiliki training loss terkecil, lalu filter 4x4, dan filter 2x2. Untuk validation loss, loss yang dihasilkan cenderung meningkat setelah epoch 3, dengan hasil validation loss akhir yang terkecil yaitu filter 2x2, filter 3x3, dan filter 4x4.

Tabel 2.2.1.3.1. F1-score Variasi Ukuran Filter

Ukuran Filter	F1-Score
2 x 2	68,17%
3 x 3	69,26%
4 x 4	67,60%



Gambar 2.2.1.3.1. Kurva Loss Variasi Ukuran Filter

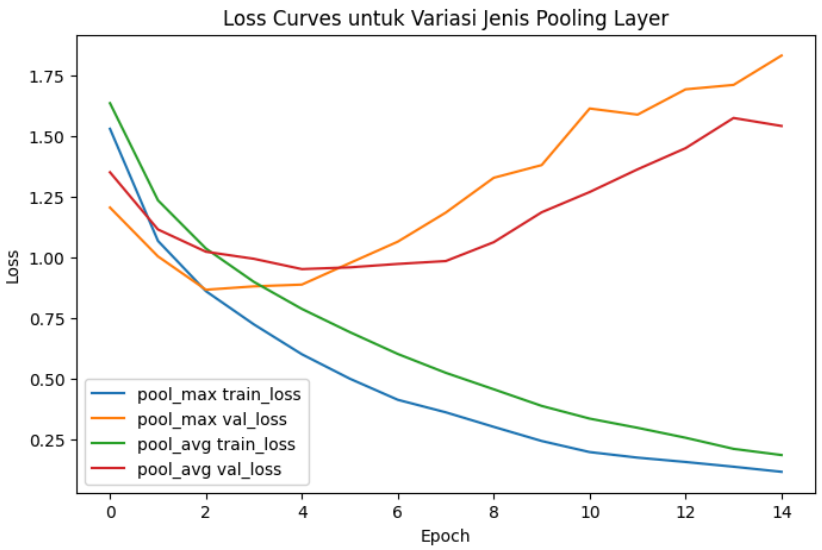
iv. Pengaruh Jenis Pooling Layer yang Digunakan

Berdasarkan hasil percobaan terhadap jenis pooling layer yang digunakan, max pooling menghasilkan F1-score yang lebih bagus daripada average pooling dengan max pooling sebesar 73,18% dan average pooling sebesar 71,87%.

Pola loss yang dihasilkan cenderung terbalik satu sama lain dengan max pooling yang menghasilkan train loss lebih kecil dan average pooling untuk validation loss.

Tabel 2.2.1.4.1. F1-score Variasi Jenis Pooling

Jenis Pooling	F1-Score
MaxPooling	69,27%
AveragePooling	68,63%



Gambar 2.2.1.4.1. Kurva Loss Variasi Jenis Pooling

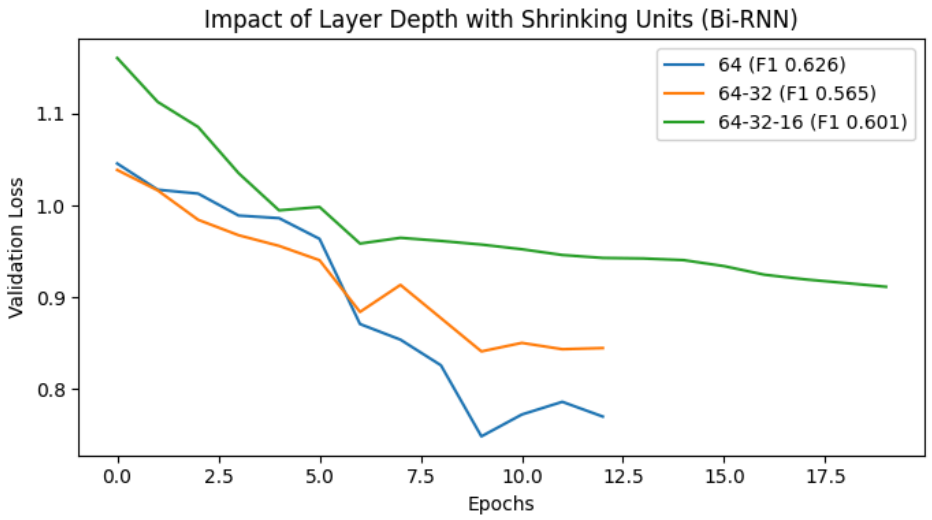
b. RNN

i. Pengaruh Jumlah Layer RNN

Pada pengujian ini, dilakukan pengujian terhadap arsitektur BiRNN (Bidirectional RNN) dengan jumlah layer yang bertambah, namun dengan jumlah unit yang "mengecil" di setiap layer. Tiga konfigurasi utama yang diuji adalah:

Tabel 2.2.2.1.1. Konfigurasi Layer RNN

Konfigurasi Layer RNN	**F1-Score**
Bi-64	62.6%
Bi-64, Bi-32	56.5%
Bi-64, Bi-32, Bi-16	60.1%



Gambar 2.2.2.1.1. Kurva Loss Variasi Layer dengan Jumlah Unit yang Berbeda

1. Model dengan 1 layer (64 Unit)

   Model dengan satu layer dan 64 unit menunjukkan penurunan validation loss yang paling cepat dan konsisten selama training, hingga akhirnya mencapai nilai F1 tertinggi (F1: 0.626). Hal ini mengindikasikan bahwa untuk dataset ini, kapasitas 64 unit dalam satu layer sudah cukup untuk menangkap pola utama dalam data sekuensial tanpa menjadi terlalu kompleks. Kurva loss-nya menurun tajam tanpa tanda-tanda overfitting yang berarti.

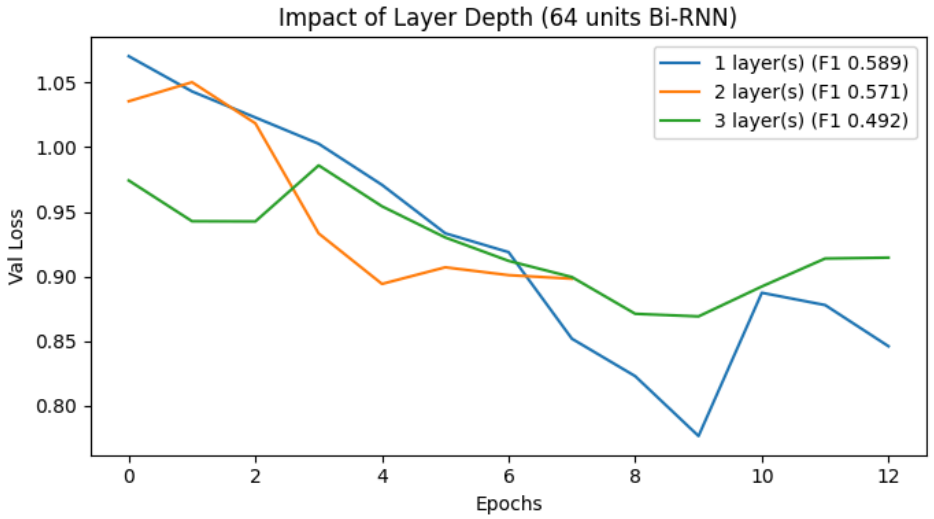
2. Model dengan 2 layer (64–32 Unit)

   Ketika model dibuat menjadi dua layer dengan unit yang mengecil (64 di layer pertama, 32 di layer kedua), terlihat bahwa validation loss menurun lebih lambat dan lebih terdapat fluktuasi, serta akhirnya loss yang dicapai tidak lebih baik dari model satu layer. F1 score-nya (F1: 0.565) juga justru menurun. Ini menandakan bahwa penambahan layer kedua, meskipun dengan unit lebih sedikit, belum tentu menambah kemampuan generalisasi model pada dataset ini. Bisa jadi model butuh regularisasi lebih kuat atau data lebih banyak.

3. Model dengan 3 layer (64–32–16 Unit)

   Pada model tiga layer dengan unit semakin mengecil di tiap layer (64, 32, 16), kurva validation loss menurun sangat lambat dan cenderung stagnan, meskipun akhirnya sedikit membaik di epoch terakhir. F1 score-nya (F1: 0.601) masih di bawah model satu layer,

meskipun lebih baik dari model dua layer. Ini menandakan adanya dampak bottleneck yaitu unit yang sangat kecil di layer terakhir membatasi kemampuan model membawa informasi, apalagi pada data sekuensial yang membutuhkan memori panjang. Model seperti ini seringkali underfitting pada dataset berukuran kecil hingga menengah.



Gambar 2.2.2.1.2. Kurva Loss Variasi Layer dengan 64 Unit

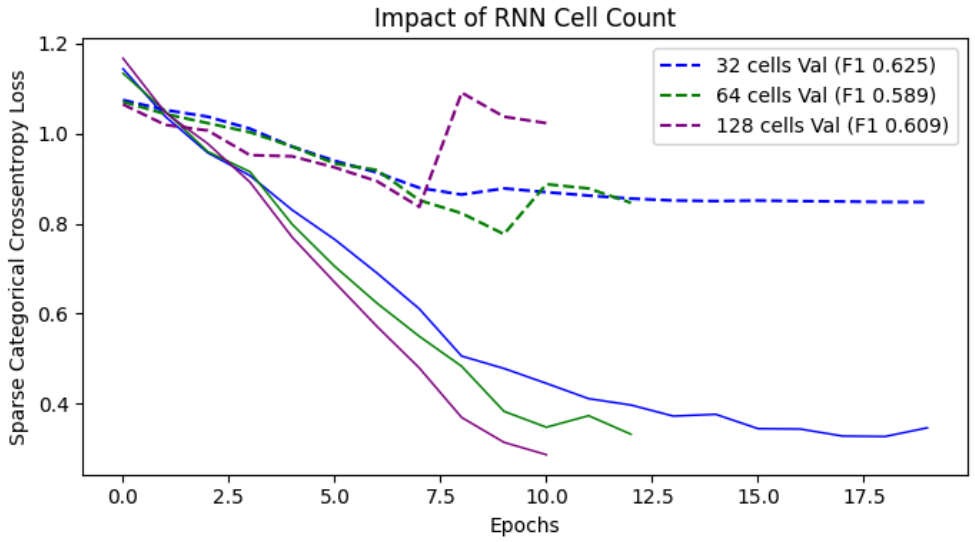
Selanjutnya pada eksperimen ini, model Bi-RNN diuji dengan variasi jumlah layer (1, 2, dan 3) di mana setiap layer selalu memiliki 64 unit. Grafik menunjukkan kurva validation loss selama training beserta skor F1 untuk masing-masing konfigurasi.

Hasil utama yang didapat:

1. Model 1 layer (64 unit): Menunjukkan penurunan val loss yang cukup stabil dengan skor F1 tertinggi (F1: 0.589) di antara tiga konfigurasi.
2. Model 2 layer (64-64): Val loss sedikit lebih rendah di beberapa epoch awal, namun akhirnya performanya tidak melebihi model 1 layer, dan skor F1-nya sedikit lebih rendah (F1: 0.571).
3. Model 3 layer (64-64-64): Meskipun di awal training val loss sempat menurun, kurva loss-nya cenderung stagnan dan berfluktuasi pada epoch berikutnya. Skor F1 juga yang terendah (F1: 0.492).

ii. Pengaruh Banyak Cell RNN per Layer

Variasi jumlah unit (cell) pada tiap layer RNN juga sangat memengaruhi performa model. Dengan menambah jumlah cell (misal, dari 32 menjadi 64 atau 128 unit), kapasitas model untuk merepresentasikan dan mengingat informasi dalam sekuens meningkat. Pada hasil pengujian, F1-score meningkat seiring bertambahnya unit hingga titik tertentu biasanya, penggunaan 64 atau 128 unit per layer menghasilkan F1-score lebih tinggi dibanding 32 unit. Namun, peningkatan kapasitas model juga membuat model lebih rentan mengalami overfitting, di mana train loss menurun sangat cepat tetapi validation loss stagnan atau bahkan meningkat. Oleh karena itu, pemilihan jumlah unit harus mempertimbangkan trade-off antara kapasitas belajar dan risiko overfitting. Dalam eksperimen ini, model dengan 64 dan 128 unit menunjukkan performa terbaik pada metrik F1-score, tetapi dengan regularisasi yang cukup agar tidak cepat overfit.

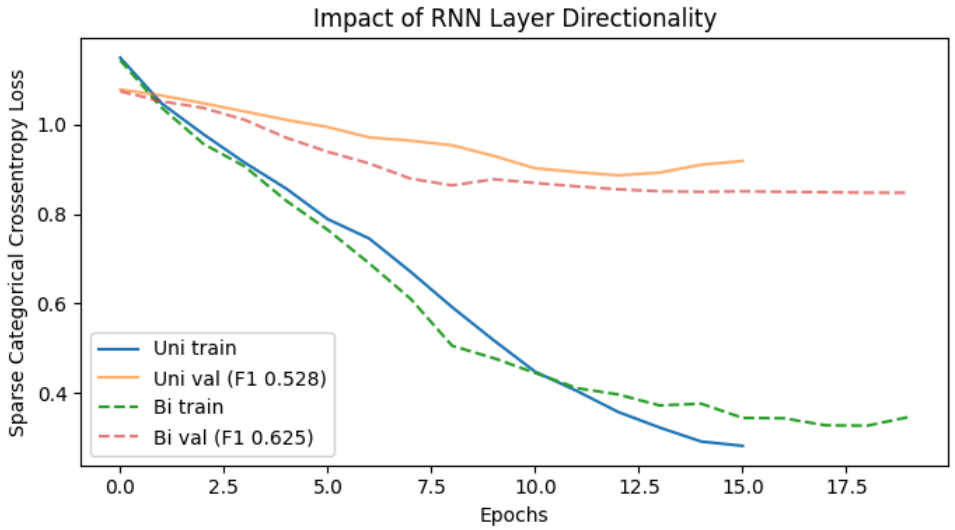


Gambar 2.2.2.2.1. Kurva Loss Variasi Units

iii. Pengaruh Jenis Layer RNN Berdasarkan Arah

Berdasarkan hasil pengujian, menunjukkan bahwa RNN bidirectional secara konsisten unggul dibandingkan arsitektur unidirectional. Model bidirectional mampu mengakses konteks dari kedua arah dalam sekuens data, sehingga informasi dari masa lalu maupun masa depan dapat dimanfaatkan secara bersamaan pada setiap timestep. Hal ini sangat bermanfaat untuk tugas-tugas seperti analisis sentimen, di mana makna kata bisa sangat

dipengaruhi oleh konteks sebelum dan sesudahnya. Pada eksperimen, model bidirectional tidak hanya menghasilkan F1-score yang lebih tinggi, tetapi juga memiliki kurva train loss dan validation loss yang lebih stabil, menandakan proses pembelajaran yang lebih efektif dan generalisasi yang lebih baik. Sebaliknya, model unidirectional cenderung lebih terbatas dalam menangkap pola kompleks dalam data sekuensial.



Gambar 2.2.2.3.1. Kurva Loss Variasi Direction

c. LSTM

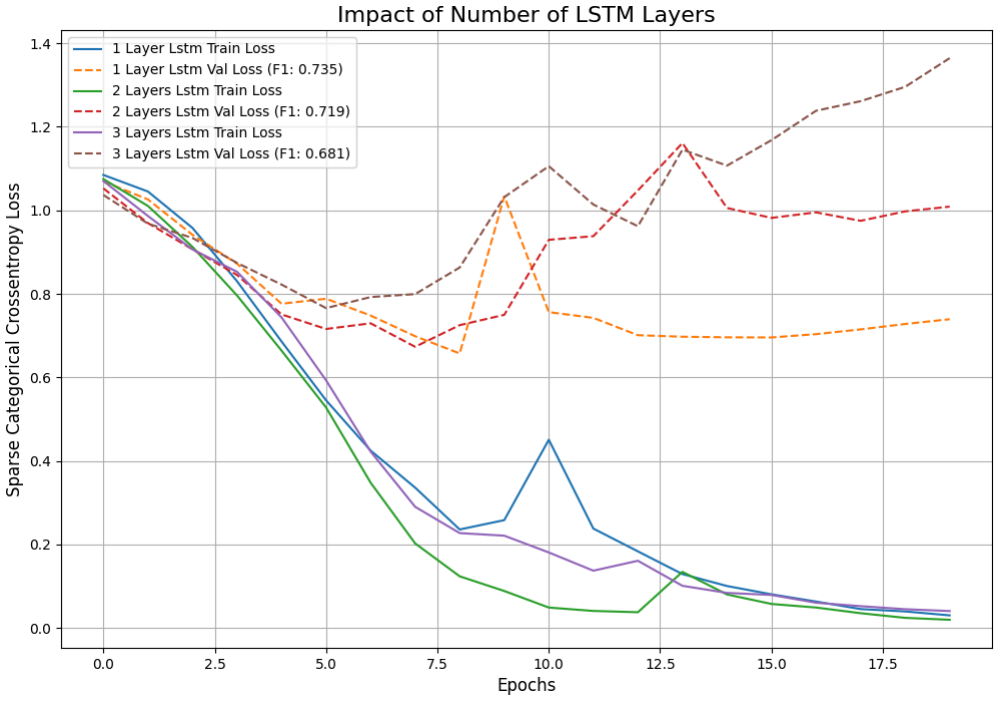
i. Pengaruh Jumlah Layer LSTM

Berdasarkan hasil pengujian, penambahan jumlah layer LSTM secara konsisten menurunkan F1-score dan mempercepat serta memperparah overfitting. Model dengan satu layer Bidirectional LSTM (Bi-64) mencapai F1-score tertinggi (73.5%) dengan tanda overfitting paling minim, sedangkan model tiga layer (Bi-64, Bi-32, Bi-16) memiliki F1-score terendah (68.1%) dan mengalami overfitting parah. Alasan utamanya adalah peningkatan kompleksitas model seiring bertambahnya layer; model yang lebih dalam memiliki kapasitas lebih besar untuk mempelajari pola, namun tanpa data yang sangat banyak atau teknik regularisasi yang kuat, ia cenderung menghafal noise dan detail spesifik dari data training daripada pola general yang berlaku untuk data baru. Akibatnya, meskipun training loss mungkin terus menurun, performa pada data validasi (yang tercermin pada validation loss dan F1-score) memburuk karena model gagal melakukan generalisasi

dengan baik. Ini mengindikasikan bahwa untuk konfigurasi dan dataset yang diuji, kapasitas model yang lebih besar justru, menyebabkan model terlalu sensitif terhadap data training dan kurang adaptif terhadap data yang belum pernah dilihat.

Tabel 2.2.3.1.1. F1 Score Konfigurasi Layer LSTM

Konfigurasi Layer LSTM	**F1-Score**
Bi-64	73.5%
Bi-64, Bi-32	71.9%
Bi-64, Bi-32, Bi-16	68.1%



Gambar 2.2.3.1.1. Kurva Loss Variasi Layer

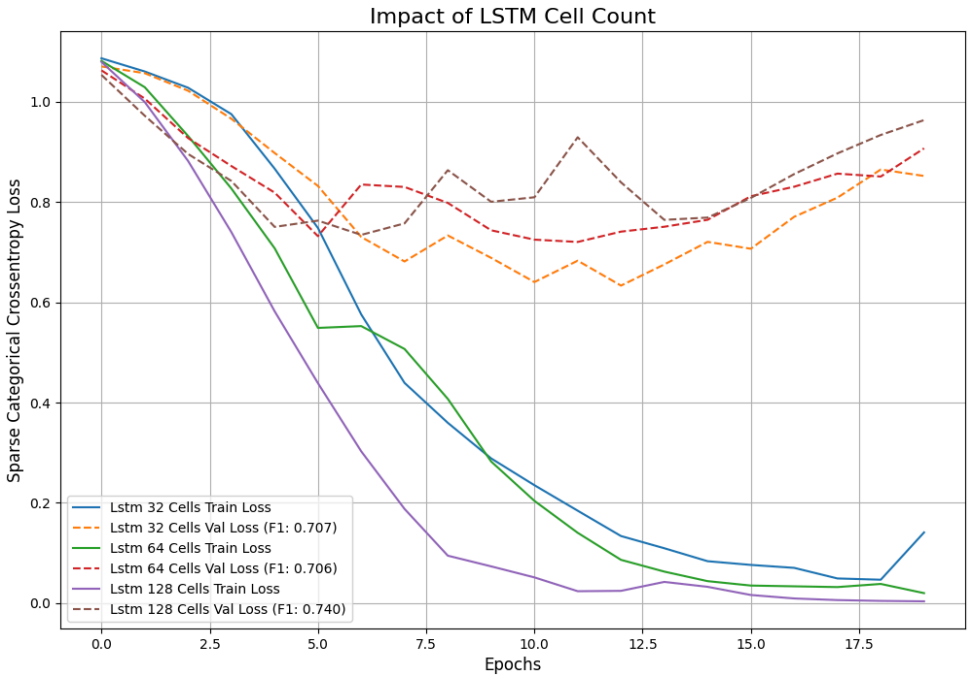
ii. Pengaruh Banyak Cell LSTM per Layer

Berdasarkan hasil pengujian, LSTM dengan jumlah sel terbanyak yakni 128 sel mencapai F1-score tertinggi (0.740), diikuti oleh 32 sel (0.707) dan 64 sel (0.706). Peningkatan jumlah sel meningkatkan kapasitas model, memungkinkan LSTM 128 sel untuk mempelajari data training paling cepat dan mencapai train loss terendah. Namun, kapasitas yang lebih besar ini juga menyebabkan overfitting yang paling parah dan paling awal, terlihat dari

kenaikan drastis pada validation loss-nya. Meskipun demikian, F1-score tertinggi pada 128 sel, meskipun overfitting pada kurva loss, mengindikasikan bahwa kemampuan model yang lebih besar ini berhasil menangkap fitur-fitur yang lebih esensial untuk klasifikasi yang benar, setidaknya hingga titik optimal sebelum overfitting terlalu merusak performa generalisasinya. Sementara itu, model 32 sel menunjukkan keseimbangan yang lebih baik antara pembelajaran dan kontrol overfitting pada kurva loss, sedangkan 64 sel tidak memberikan keunggulan jelas dan justru sedikit lebih buruk dari 32 sel dalam F1-score. Ini menyoroti adanya trade-off krusial antara kapasitas model untuk belajar dan risikonya untuk menghafal data training alih-alih melakukan generalisasi

Tabel 2.2.3.2.1. F1 Score Variasi Sel LSTM

Jumlah Sel LSTM	F1-Score
32	70.7%
64	70.6%
128	74.0%



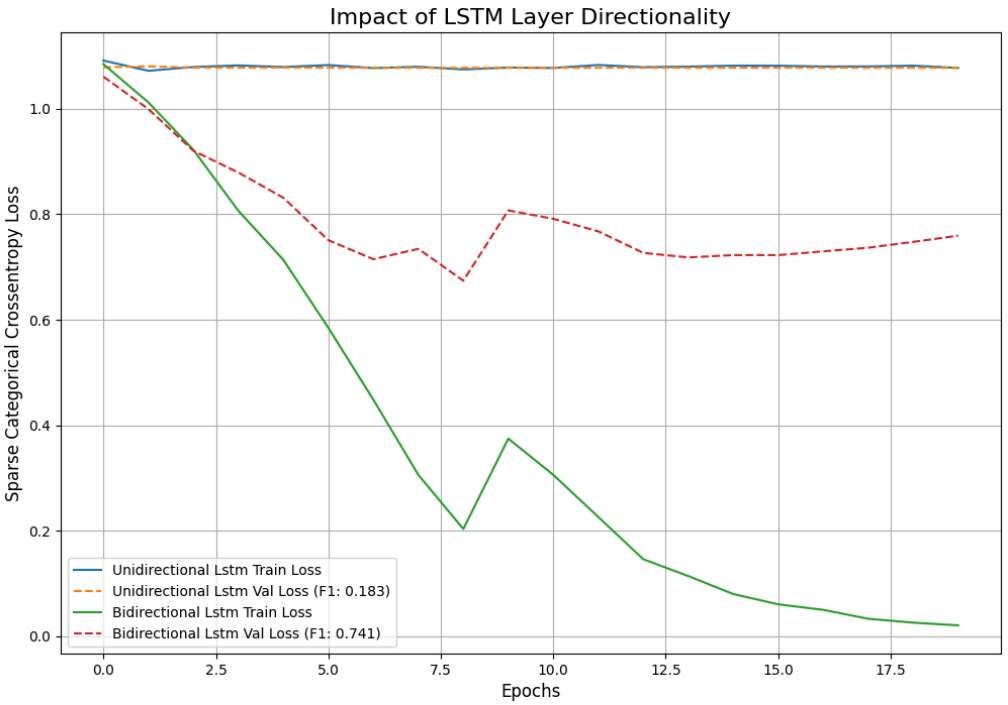
Gambar 2.2.3.2.1. Kurva Loss Variasi Units

iii. Pengaruh Jenis Layer LSTM Berdasarkan Arah

Berdasarkan hasil pengujian, LSTM Bidirectional menunjukkan performa lebih baik yang sangat signifikan dibandingkan LSTM Unidirectional. Model Unidirectional LSTM tampak gagal total dalam mempelajari pola dari data, yang ditunjukkan oleh kurva train loss dan validation loss yang stagnan pada nilai tinggi sepanjang proses pelatihan. Sebaliknya, model Bidirectional LSTM berhasil menurunkan train loss-nya secara drastis, menandakan kemampuan belajar yang baik, meskipun kurva validation loss-nya mulai sedikit meningkat setelah sekitar epoch ke-7, mengindikasikan potensi overfitting ringan. Alasan di balik perbedaan mencolok ini terletak pada bagaimana kedua arsitektur memproses informasi sekuensial. LSTM Unidirectional hanya menganalisis data secara kronologis dari awal hingga akhir, sehingga kehilangan konteks yang mungkin datang dari elemen-elemen berikutnya dalam sekuens. Sementara itu, LSTM Bidirectional memproses sekuens dari kedua arah (maju dan mundur) dan menggabungkan informasi tersebut, memungkinkan model untuk menangkap dependensi dan konteks yang lebih kaya pada setiap timestep. Dengan demikian, kemampuan untuk memahami konteks masa lalu dan masa depan secara bersamaan memberikan keunggulan besar bagi LSTM Bidirectional dalam mengekstraksi fitur yang relevan dan mencapai akurasi yang jauh lebih tinggi pada dataset ini.

Tabel 2.2.3.3.1. F1 Score Variasi Arah LSTM

Arah LSTM	**F1-Score**
Unidirectional	18.3%
Bidirectional	74.1%



Gambar 2.2.3.3.1. Kurva Loss Variasi Arah

BAB III
KESIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan rangkaian eksperimen yang dilakukan pada tugas besar ini, dapat disimpulkan bahwa seluruh model utama, baik CNN untuk gambar maupun RNN dan LSTM untuk teks berhasil diimplementasikan menggunakan dua pendekatan, yaitu Keras dan forward propagation “from scratch”. Evaluasi membuktikan bahwa kedua pendekatan tersebut mampu menghasilkan prediksi identik pada test set, asalkan bobot dan arsitektur yang digunakan benar-benar sama.

Pada eksperimen model CNN untuk klasifikasi citra CIFAR-10, penambahan jumlah layer konvolusi, jumlah filter, dan penggunaan max pooling secara umum mampu meningkatkan akurasi model. Namun, kompleksitas model yang meningkat juga membawa risiko overfitting dan kebutuhan komputasi yang lebih besar. Oleh karena itu, pemilihan parameter optimal menjadi kunci agar model yang dihasilkan tetap efisien sekaligus memiliki performa yang tinggi.

Untuk model RNN pada klasifikasi teks, hasil terbaik justru diperoleh pada arsitektur sederhana, yaitu satu layer BiRNN dengan 32 unit. Penambahan layer dengan jumlah unit yang makin sedikit pada layer bawah tidak selalu berdampak positif; bahkan, dapat menurunkan performa akibat efek bottleneck dan underfitting, terutama pada dataset berukuran kecil. Terlalu banyak unit juga dapat menyebabkan overfitting, sehingga keseimbangan antara kapasitas model dan kompleksitas data sangat penting. Penggunaan RNN bidirectional terbukti secara konsisten memberikan hasil yang lebih baik daripada unidirectional.

Pada LSTM, model ini konsisten mengungguli RNN dalam mengatasi dependensi jangka panjang pada data sekuensial. Satu layer BiLSTM dengan jumlah unit yang cukup besar (misal 64–128) menjadi konfigurasi optimal, sementara penambahan layer atau unit yang terlalu banyak justru meningkatkan risiko overfitting tanpa disertai peningkatan performa. LSTM bidirectional juga memperlihatkan keunggulan signifikan atas versi unidirectional, menandakan pentingnya pemanfaatan konteks dua arah pada data teks.

Secara keseluruhan, evaluasi dengan macro F1-score pada test set membuktikan bahwa model yang diimplementasikan sudah berjalan sesuai harapan.

Keseimbangan antara kompleksitas model, ukuran data, serta teknik regularisasi dan preprocessing membangun model pembelajaran mesin yang efektif dan efisien.

B. Saran

- Untuk meningkatkan generalisasi, disarankan melakukan augmentasi data (pada citra maupun teks) atau menambah data latih. Teknik regularisasi seperti dropout lebih lanjut, L2 regularization, dan early stopping dapat dioptimalkan untuk mencegah overfitting, khususnya pada model yang lebih kompleks.
- Untuk data sekuensial model transformer bisa menjadi alternatif yang lebih powerful dibandingkan dengan RNN dan LSTM.
- Lakukan grid search atau random search lebih sistematis terhadap kombinasi hyperparameter (learning rate, batch size, jumlah unit/layer, dropout) untuk menemukan konfigurasi yang paling optimal untuk masing-masing dataset.
- Perlu pembelajaran lebih lanjut mengenai model CNN untuk menangani validation loss yang besar

PEMBAGIAN TUGAS

Nama / NIM	Tugas
Juan Alfred Widjaya / 13522073	LSTM, laporan
Albert / 13522081	RNN, laporan
Ivan Hendrawan Tan / 13522111	CNN, laporan

REFERENSI

analyticsvidhya.com. (2025, 08 Januari). Fundamentals of RNN forward Propagation in Deep Learning. Diakses pada 26 Mei 2025, dari https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/

geeksforgeeks.org. (2025, 03 April). Introduction to Convolution Neural Network. Diakses pada 17 Mei 2025, dari https://www.geeksforgeeks.org/introduction-convolution-neural-network/

geeksforgeeks.org. (2025, 28 Mei). What is LSTM - Long Short Term Memory?. Diakses pada 26 Mei 2025, dari https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/

linkedin.com. (2025, 08 Januari). How Batch Size Impacts Model Predictions in CNNs. Diakses pada 17 Mei 2025, dari https://www.linkedin.com/pulse/how-batch-size-impacts-model-predictions-cnns-shashwat-dev-hans-oizgf/

medium.com. (2020, 02 September). LSTMs Explained: A Complete, Technically Accurate, Conceptual Guide with Keras. Diakses pada 26 Mei 2025, dari https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2

medium.com. (2023, 21 Januari). RNN (Forward and Backward Propagation). Diakses pada 26 Mei 2025, dari https://medium.com/@balajicena1995/rnn-26914e8d19b7