LAPORAN TUGAS BESAR 1

IF3270 Pembelajaran Mesin

Kelompok 30

“Convolutional Neural Network dan Recurrent Neural Network”

Dosen:

Dr. Fariska Zakhralativa Ruskanda, S.T., M.T.

Oleh:

Owen Tobias Sinurat (13522131)

Ahmad Thoriq Saputra (13522141)

Muhammad Fatihul Irhab (13522143)

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER II TAHUN 2023/2024

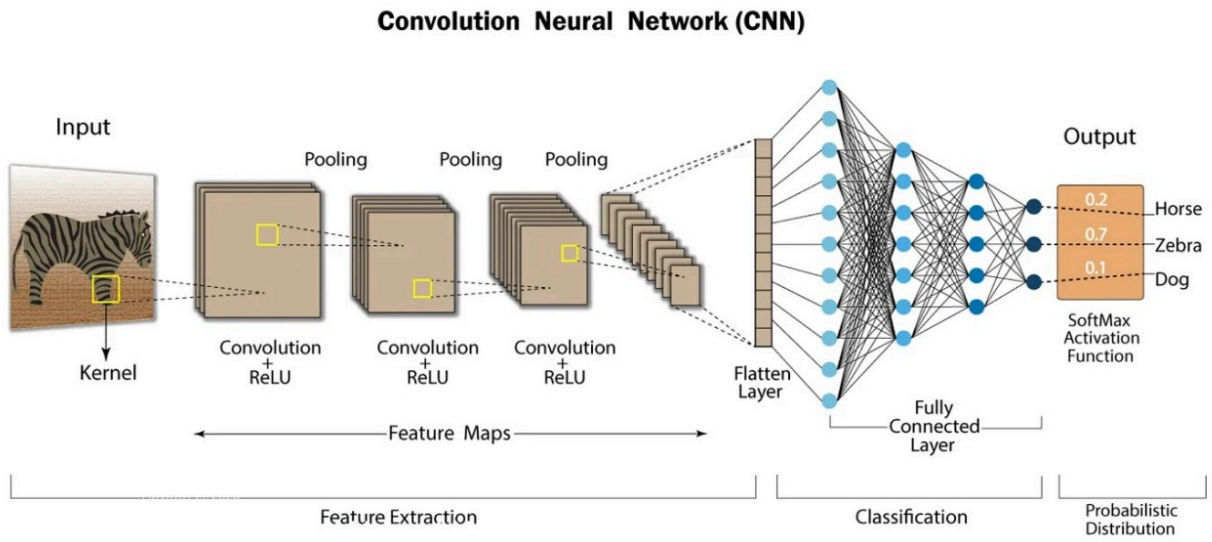
DESKRIPSI PERSOALAN

Spesifikasi

Pada tugas ini, Anda akan diminta untuk mengimplementasikan fungsi *forward propagation* untuk beberapa arsitektur berikut:

- CNN
- Simple RNN
- LSTM

Convolutional Neural Network



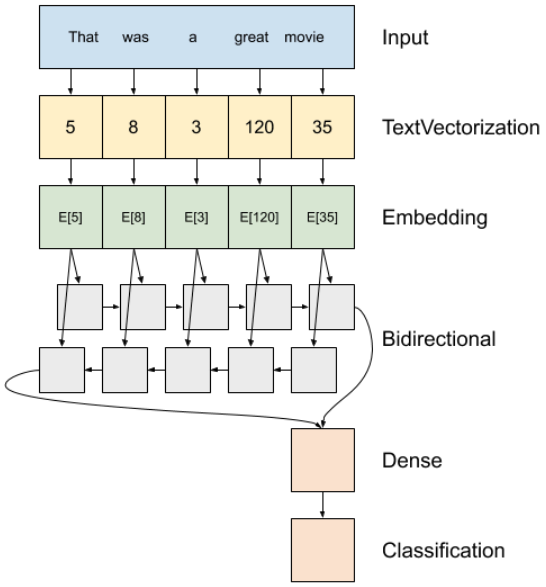
Untuk bagian CNN, Anda diminta untuk melakukan beberapa hal berikut:

- Lakukan pelatihan suatu model CNN untuk *image classification* dengan library Keras dan dengan dataset CIFAR-10 yang memenuhi ketentuan berikut ini.
    - Model minimal harus memiliki jenis layer berikut didalamnya (urutan dan jumlah layer silakan disesuaikan sendiri):
        - Conv2D layer
        - Pooling layers
        - Flatten/Global Pooling layer
        - Dense layer
    - Loss function yang digunakan adalah Sparse Categorical Crossentropy (untuk menangani kasus klasifikasi multikelas)
    - Optimizer yang digunakan adalah Adam

    - Dataset CIFAR-10 yang disediakan hanya terdiri dari dua split data saja, yaitu *train* dan *test*. Tambahkan split ke-3 (*validation set*) dengan cara membagi training set yang sudah ada dengan menjadi training set yang lebih kecil dan validation set dengan rasio 4:1 (jumlah data akhir adalah 40k *train* data, 10k validation data, dan 10k test data)
- Lakukan variasi pelatihan sebagai berikut untuk analisis pengaruh beberapa hyperparameter dalam CNN:
    - Pengaruh **jumlah layer konvolusi**
        - Pilih **3 variasi** jumlah layer konvolusi
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana jumlah layer konvolusi mempengaruhi kinerja model
    - Pengaruh **banyak filter per layer konvolusi**
        - Pilih **3 variasi** kombinasi banyak filter per layer konvolusi
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana banyak filter per layer konvolusi mempengaruhi kinerja model
    - Pengaruh **ukuran filter per layer konvolusi**
        - Pilih **3 variasi** kombinasi banyak filter per layer konvolusi
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana ukuran filter per layer konvolusi mempengaruhi kinerja model
    - Pengaruh **jenis pooling layer** yang digunakan
        - Pilih **2 variasi** pooling layer (antara max pooling atau average pooling)
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana jenis pooling layer mempengaruhi kinerja model
    - Catatan: Gunakan macro f1-score sebagai metrik ketika membandingkan hasil akhir prediksi.
- Simpan hasil bobot dari pelatihan.
- Buatlah modul *forward propagation from scratch* dari model yang telah dibuat dengan ketentuan sebagai berikut:
    - Dapat membaca model hasil pelatihan dengan Keras (bobotnya sama dengan bobot hasil pelatihan dengan Keras).

- Direkomendasikan untuk mengimplementasikan *forward propagation* secara modular, yaitu dengan cara mengimplementasikan method *forward propagation* untuk setiap layer.
- Lakukan pengujian dengan membandingkan hasil forward propagation *from scratch* dengan hasil forward propagation menggunakan Keras.
- Gunakan split data test untuk menguji implementasi forward propagation. Metrik yang digunakan adalah macro f1-score.
- Catatan: Khusus untuk **Dense layer**, Anda boleh menggunakan implementasi *forward propagation* FFNN dari Tubes 1.

Simple Recurrent Neural Network



Untuk bagian ini, Anda diminta untuk melakukan beberapa hal berikut:

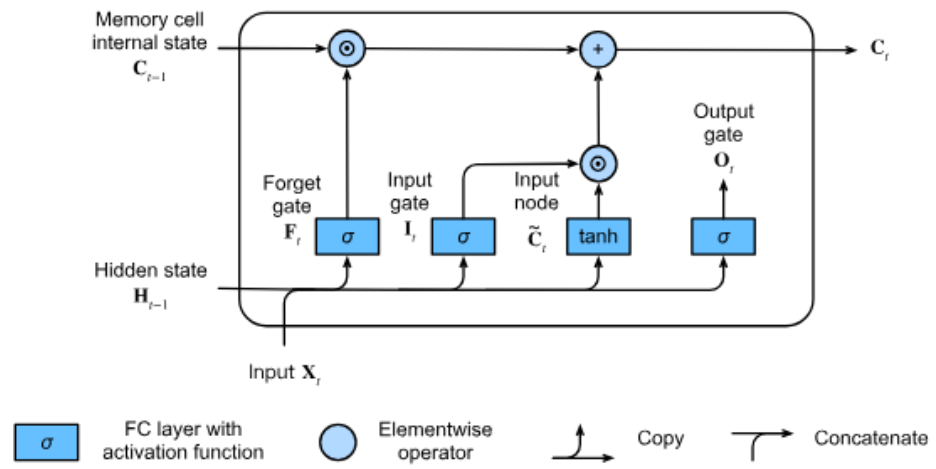
- Preprocessing data teks menjadi representasi numerik yang bisa diterima oleh model dengan tahap sebagai berikut:
  - Tokenization
    Tahap ini akan mengubah data teks menjadi bentuk list of tokens (integer), dimana teks akan dipecah-pecah menjadi bentuk satuannya yaitu *token* yang direpresentasikan sebagai suatu *integer*. Untuk tugas ini, Anda cukup memanfaatkan TextVectorization layer untuk memetakan input sequence menjadi list of tokens.
  - Embedding function
    Embedding function merupakan fungsi yang memetakan tiap token ke dalam suatu ruang vektor berdimensi-n, sehingga setiap token (yang sudah berbentuk vektor) dapat dioperasikan satu sama lain. Untuk tugas ini, Anda

cukup memanfaatkan embedding layer yang disediakan oleh Keras untuk mengonversi token ke dalam bentuk vektor.

- Lakukan pelatihan untuk suatu model RNN untuk *text classification* dengan dataset NusaX-Sentiment (Bahasa Indonesia) dan dengan menggunakan Keras yang memenuhi ketentuan berikut ini.
    - Model minimal harus memiliki jenis layer-layer berikut didalamnya (urutan dan jumlah layer silakan disesuaikan sendiri):
        - Embedding layer
        - Bidirectional RNN layer dan/atau Unidirectional RNN layer
        - Dropout layer
        - Dense layer
    - Loss function yang digunakan adalah Sparse Categorical Crossentropy (untuk menangani kasus klasifikasi multikelas)
    - Optimizer yang digunakan adalah Adam
- Lakukan variasi pelatihan sebagai berikut untuk analisis pengaruh beberapa hyperparameter dalam RNN:
    - Pengaruh **jumlah layer RNN**
        - Pilih **3 variasi** jumlah layer RNN
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana jumlah layer RNN mempengaruhi kinerja model
    - Pengaruh **banyak cell RNN per layer**
        - Pilih **3 variasi** kombinasi banyak cell RNN per layer
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana banyak cell RNN per layer mempengaruhi kinerja model
    - Pengaruh **jenis layer RNN berdasarkan arah**
        - Pilih **2 variasi** jenis layer RNN berdasarkan arah (bidirectional atau unidirectional)
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana jenis layer RNN berdasarkan arah mempengaruhi kinerja model
    - Catatan: Gunakan macro f1-score sebagai metrik ketika membandingkan hasil akhir prediksi.
- Simpan weight hasil pelatihan dengan Keras.
- Buatlah modul *forward propagation from scratch* dari model yang telah dibuat dengan ketentuan sebagai berikut:

- Dapat membaca model hasil pelatihan dengan Keras (bobotnya sama dengan bobot hasil pelatihan dengan Keras).
- Direkomendasikan untuk mengimplementasikan *forward propagation* secara modular, yaitu dengan cara mengimplementasikan method *forward propagation* untuk setiap layer.
- Bandingkan hasil forward propagation *from scratch* dengan hasil forward propagation menggunakan Keras. Gunakan split data test untuk menguji implementasi forward propagation. Metrik yang digunakan adalah macro f1-score.
- Catatan: Khusus untuk **Dense layer**, Anda boleh menggunakan implementasi *forward propagation* FFNN dari Tubes 1.

Long-Short Term Memory Network



Untuk bagian ini, Anda diminta untuk melakukan beberapa hal berikut:

- Preprocessing data teks menjadi representasi numerik yang bisa diterima oleh model dengan tahap sebagai berikut:
  - Tokenization
    Tahap ini akan mengubah data teks menjadi bentuk list of tokens (integer), dimana teks akan dipecah-pecah menjadi bentuk satuannya yaitu *token* yang direpresentasikan sebagai suatu *integer*. Untuk tugas ini, Anda cukup memanfaatkan TextVectorization layer untuk memetakan input sequence menjadi list of tokens.
  - Embedding function
    Embedding function merupakan fungsi yang memetakan tiap token ke dalam suatu ruang vektor berdimensi-n, sehingga setiap token (yang sudah berbentuk vektor) dapat dioperasikan satu sama lain. Untuk tugas ini, Anda

cukup memanfaatkan embedding layer yang disediakan oleh Keras untuk mengonversi token ke dalam bentuk vektor.

- Lakukan pelatihan untuk suatu model LSTM untuk *text classification* dengan dataset NusaX-Sentiment (Bahasa Indonesia) dan dengan menggunakan Keras yang memenuhi ketentuan berikut ini.
    - Model minimal harus memiliki jenis layer-layer berikut didalamnya (urutan dan jumlah layer silakan disesuaikan sendiri):
        - Embedding layer
        - Bidirectional LSTM layer dan/atau Unidirectional LSTM layer
        - Dropout layer
        - Dense layer
    - Loss function yang digunakan adalah Sparse Categorical Crossentropy (untuk menangani kasus klasifikasi multikelas)
    - Optimizer yang digunakan adalah Adam
- Lakukan variasi pelatihan sebagai berikut untuk analisis pengaruh beberapa hyperparameter dalam LSTM:
    - Pengaruh **jumlah layer LSTM**
        - Pilih **3 variasi** jumlah layer LSTM
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana jumlah layer LSTM mempengaruhi kinerja model
    - Pengaruh **banyak cell LSTM per layer**
        - Pilih **3 variasi** kombinasi banyak cell LSTM per layer
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana banyak cell LSTM per layer mempengaruhi kinerja model
    - Pengaruh **jenis layer LSTM berdasarkan arah**
        - Pilih **2 variasi** jenis layer RNN berdasarkan arah (bidirectional atau unidirectional)
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana jenis layer LSTM berdasarkan arah mempengaruhi kinerja model
    - Catatan: Gunakan macro f1-score sebagai metrik ketika membandingkan hasil akhir prediksi.
- Simpan weight hasil pelatihan dengan Keras.
- Buatlah modul *forward propagation from scratch* dari model yang telah dibuat dengan ketentuan sebagai berikut:

    - Dapat membaca model hasil pelatihan dengan Keras (bobotnya sama dengan bobot hasil pelatihan dengan Keras).
    - Direkomendasikan untuk mengimplementasikan *forward propagation* secara modular, yaitu dengan cara mengimplementasikan method *forward propagation* untuk setiap layer.
    - Bandingkan hasil forward propagation *from scratch* dengan hasil forward propagation menggunakan Keras. Gunakan split data test untuk menguji implementasi forward propagation. Metrik yang digunakan adalah macro f1-score.
    - Catatan: Khusus untuk **Dense layer**, Anda boleh menggunakan implementasi *forward propagation* FFNN dari Tubes 1.

Spesifikasi Bonus

Berikut merupakan beberapa spesifikasi bonus yang dapat Anda kerjakan:

- Implementasikan fungsi *backward propagation from scratch* untuk seluruh *layer* yang digunakan
- Seluruh implementasi *forward propagation* harus bisa menangani kasus batch inference, dimana model dapat menerima lebih dari satu input untuk satu kali forward propagation:
    - Jumlah instance dalam satu batch bisa diatur dengan menggunakan suatu hyperparameter *batch_size*

Aturan

Terdapat beberapa hal yang harus diperhatikan dalam pengerjaan tugas ini, yakni:

1. Jika terdapat hal yang tidak dimengerti, silahkan ajukan pertanyaan kepada asisten melalui **link QnA** yang telah diberikan di atas. Pertanyaan yang diajukan secara personal ke asisten **tidak akan dijawab** untuk menghindari perbedaan informasi yang didapatkan oleh peserta kuliah.
2. Dilarang melakukan **plagiarisme, menggunakan AI dalam bentuk apapun secara tidak bertanggungjawab, dan melakukan kerjasama antar kelompok**. Pelanggaran pada poin ini akan menyebabkan pemberian **nilai E** pada setiap anggota kelompok.
3. Tugas diimplementasikan dalam bahasa **Python.**
4. Implementasi *forward propagation* dari CNN dan RNN *from scratch* hanya boleh menggunakan library untuk perhitungan matematika (Contoh: NumPy, dll).
5. Tidak ada batasan penggunaan library untuk segala sesuatu diluar implementasi *forward propagation* dan *backward propagation*.

Deliverables

- Tugas dikumpulkan dalam bentuk link ke *repository* GitHub yang **minimal** berisi beberapa hal berikut (boleh ditambahkan jika dirasa perlu):
    - Folder **src**, digunakan untuk menyimpan source code beserta dengan notebook pengujian.
    - Folder **doc,** digunakan untuk menyimpan laporan dalam bentuk **.pdf** yang terdiri atas komponen berikut:
        - Cover
        - Deskripsi Persoalan
        - Pembahasan
            - Penjelasan implementasi
                - Deskripsi kelas beserta deskripsi atribut dan methodnya
                - Penjelasan forward propagation
                    - CNN
                    - Simple RNN
                    - LSTM
            - Hasil pengujian
                - CNN
                    - Pengaruh jumlah layer konvolusi
                    - Pengaruh banyak filter per layer konvolusi
                    - Pengaruh ukuran filter per layer konvolusi
                    - Pengaruh jenis pooling layer
                - Simple RNN
                    - Pengaruh jumlah layer RNN
                    - Pengaruh banyak cell RNN per layer
                    - Pengaruh jenis layer RNN berdasarkan arah
                - LSTM
                    - Pengaruh jumlah layer LSTM
                    - Pengaruh banyak cell LSTM per layer
                    - Pengaruh jenis layer LSTM berdasarkan arah
        - Kesimpulan dan Saran
        - Pembagian tugas tiap anggota kelompok
        - Referensi
    - **README.md**, yang berisi deskripsi singkat repository, cara setup dan run program, dan pembagian tugas tiap anggota kelompok.
- Pengumpulan dilakukan melalui form dengan tautan sebagai berikut: Form Pengumpulan IF3270 Pembelajaran Mesin
- Batas akhir pengumpulan adalah hari **Jum'at, 30 Mei 2025 23.59 WIB.** Tugas yang terlambat dikumpulkan tidak akan diterima.

- Pengumpulan dilakukan oleh NIM terkecil.

Referensi

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

PEMBAHASAN

Penjelasan Implementasi

CNN

a. main.py

File ini untuk memvalidasi sebuah model CNN yang dibuat from scratch. Validasi dilakukan dengan membandingkan kinerjanya terhadap model CNN referensi yang dibuat menggunakan keras, dengan memastikan kedua model memiliki arsitektur dan bobot yang sama saat diuji pada dataset CIFAR-10.

Hasil prediksi dari kedua model dievaluasi menggunakan metrik F1-score, persentase kesamaan prediksi, dan laporan klasifikasi detail. Tujuannya adalah untuk membuktikan bahwa implementasi CNN from scratch dapat menghasilkan output yang sangat mirip atau sama dengan implementasi standar keras.

b. train.py

File ini berisi kerangka kerja untuk melatih dan mengevaluasi berbagai arsitektur model CNN menggunakan keras pada dataset CIFAR-10. Tujuannya adalah untuk melakukan serangkaian eksperimen dengan memvariasikan hyperparameter CNN seperti jumlah layer konvolusi, jumlah filter, ukuran kernel, dan jenis pooling. File ini akan melatih setiap konfigurasi model, mencatat metrik kinerjanya, menyimpan plot riwayat pelatihan, dan menyimpan bobot model dengan F1-score terbaik.

Atribut/Method	Deskripsi
`load_and_preprocess_cifar10()`	Fungsi ini bertanggung jawab untuk load dataset CIFAR-10.
`create_keras_cnn_model(num_conv_layers, filters_list, kernel_sizes_list, pooling_type, use_global_pooling, model_name_suffix)`	Fungsi ini secara dinamis membuat model CNN keras berdasarkan parameter konfigurasi yang diberikan. Atribut: • num_conv_layers (int): Jumlah layer konvolusi. • filters_list (list): Daftar jumlah filter

	untuk setiap layer konvolusi. • kernel_sizes_list (list): Daftar ukuran kernel untuk setiap layer konvolusi. • pooling_type (str): Jenis layer *pooling* yang akan digunakan ('max' atau 'average'). • use_global_pooling (bool): Apakah akan menggunakan GlobalAveragePooling2D sebelum layer *dense*. Jika False, maka Flatten akan digunakan. • model_name_suffix (str): Nama model, berguna untuk identifikasi.
`plot_training_history(history, model_name, experiment_name_prefix)`	Fungsi ini mengambil objek history yang dihasilkan oleh metode fit() keras dan membuat plot untuk loss serta akurasi pelatihan dan validasi dari per epoch. Plot kemudian disimpan sebagai file gambar. Atribut: • history (keras.callbacks.History): Objek riwayat pelatihan. • model_name (str): Nama model untuk judul plot. • experiment_name_prefix (str): Nama eksperimen untuk nama file plot.
`evaluate_and_get_f1(model, x_test_data, y_test_data, model_name, experiment_name_prefix)`	Fungsi ini mengevaluasi model yang telah dilatih pada data uji. Ini menghitung loss, akurasi, dan F1-score pada data uji. Atribut: • model (keras.Model): Model keras yang akan dievaluasi. • x_test_data (numpy.ndarray): Data fitur uji. • y_test_data (numpy.ndarray): Data label uji. • model_name (str): Nama model untuk pencatatan. • experiment_name_prefix (str): Nama eksperimen untuk pencatatan.
`run_experiment(config,`	Fungsi utama yang mengatur jalannya satu

`x_train_data, y_train_data, x_val_data, y_val_data, x_test_data, y_test_data, experiment_name_prefix)`	eksperimen. Fungsi ini membuat model berdasarkan konfigurasi yang diberikan, melatihnya, mencatat waktu pelatihan, membuat plot riwayat pelatihan, dan mengevaluasi model. Atribut: • config (dict): Kamus yang berisi konfigurasi untuk create_keras_cnn_model. • x_train_data, y_train_data: Data pelatihan. • x_val_data, y_val_data: Data validasi. • x_test_data, y_test_data: Data uji. • experiment_name_prefix (str): Nama untuk identifikasi eksperimen.

c. cnn.py

File ini berisi implementasi CNN from scratch untuk berbagai komponen inti dari sebuah CNN menggunakan pustaka NumPy. Ini mencakup definisi kelas-kelas untuk berbagai jenis Layer seperti Conv2D, ReLU, Pooling (Max dan Average), Flatten, Dense, dan Softmax.

Atribut/Method	Deskripsi
`relu(x)`	Mengimplementasikan fungsi aktivasi ReLU.
`softmax(x, axis=-1)`	Mengimplementasikan fungsi aktivasi Softmax, yang mengubah skor mentah menjadi probabilitas.
`Layer`	Kelas dasar abstrak untuk semua jenis layer. Menyediakan fungsionalitas dasar seperti penyimpanan cache input/output dan load parameter. Atribut: • input_data_cache (NumPy array): Menyimpan data input terakhir yang diterima oleh layer. • output_data_cache (NumPy array): Menyimpan data output terakhir yang dihasilkan oleh layer.

	• weights (NumPy array): Bobot dari layer. • bias (NumPy array): Bias dari layer. • name (str): Nama layer, defaultnya adalah nama kelas itu sendiri.
`Conv2D(Layer)`	Mengimplementasikan layer konvolusi 2D. Atribut: • num_filters (int): Jumlah filter (kernel) konvolusi. • filter_height (int), filter_width (int): Dimensi filter konvolusi. • stride (int): Langkah konvolusi. • padding_mode (str): Mode padding. • input_depth (int): Kedalaman dari input.
`ReLU(Layer)`	Mengimplementasikan layer aktivasi ReLU. Mewarisi dari layer.
`PoolingBase(Layer)`	Kelas dasar untuk layer pooling (MaxPooling2D dan AveragePooling2D). Atribut: • pool_height (int), pool_width (int): Ukuran pooling. • stride_val (int): Langkah untuk pooling. • padding_mode (str): Mode padding.
`MaxPooling2D(PoolingBase)`	Mengimplementasikan layer Max Pooling 2D. Mewarisi dari poolingbase.
`AveragePooling2D(PoolingBase)`	Mengimplementasikan layer Average Pooling 2D. Mewarisi dari poolingbase.
`Flatten(Layer)`	Mengubah input multi-dimensi menjadi vektor satu dimensi per sampel batch. Atribut: • original_input_shape_cache (tuple): Menyimpan bentuk asli dari input sebelum di-flatten.
`Dense(Layer)`	Mengimplementasikan layer fully connected. Atribut: • output_size (int): Jumlah neuron dalam

	layer ini. • input_size (int): Jumlah fitur input.
`SoftmaxActivation(Layer)`	Mengimplementasikan layer aktivasi Softmax. Mewarisi dari layer.
`CNNModel`	Kelas yang merepresentasikan model CNN secara keseluruhan, terdiri dari urutan layer-layer. Atribut: • layers (list): Daftar objek layer yang membentuk model. • keras_layer_name_map (dict): Pemetaan antara layer dari model from scratch dengan layer dari model keras.
`get_keras_layer_output_shape(model, layer_name_to_find)`	Fungsi utilitas untuk mendapatkan bentuk output dari sebuah layer spesifik dalam model keras yang diberikan.
`create_scratch_cnn_model(config, input_shape_channels_first, keras_model_reference)`	Fungsi ini membangun dan mengembalikan instance CNN model from scratch berdasarkan konfigurasi arsitektur yang diberikan dan model keras sebagai referensi.

d. Forward Propagation

1. Input:
    a. Data input (input_data) pertama kali diterima oleh metode CNNModel.forward(). Data ini biasanya berupa array NumPy 4D dengan format (batch_size, depth, height, width), di mana depth adalah jumlah channel.
2. Operasi per Layer:
    a. Untuk setiap layer, metode forward() dari layer tersebut dipanggil dengan current_output (yang merupakan output dari layer sebelumnya, atau data input awal untuk layer pertama) sebagai argumennya.
    b. Conv2D.forward(input_data):
        i. Menerima feature map dari layer sebelumnya.
        ii. Operasi konvolusi dilakukan
        iii. Proses ini diulang untuk semua filter, menghasilkan sejumlah feature map baru.

iv. Bias ditambahkan ke setiap feature map hasil konvolusi.
v. Outputnya adalah kumpulan feature map baru yang telah diekstraksi fiturnya.

c. ReLU.forward(input_data):
   i. Menerima output dari layer sebelumnya.
   ii. Menerapkan fungsi aktivasi ReLU ke setiap elemen input secara individual.
   iii. Nilai negatif diubah menjadi 0, sementara nilai positif tetap.
d. MaxPooling2D.forward(input_data)/AveragePooling2D.forward(input_data):
   i. Menerima feature map dari ReLU.
   ii. Mengurangi dimensi spasial dari feature map.
   iii. Untuk MaxPooling2D: Jendela pooling digeser melintasi input. Dari setiap area yang dicakup jendela, nilai maksimum dipilih.
   iv. Untuk AveragePooling2D: Sama seperti max pooling, tetapi nilai rata-rata yang diambil dari setiap jendela.
e. Flatten.forward(input_data):
   i. Setelah serangkaian layer konvolusi dan pooling.
   ii. Menerima feature map 4D (batch_size, depth, height, width).
   iii. Mengubahnya menjadi array 2D (batch_size, num_features), di mana num_features = depth * height * width.
   iv. Urutan elemen diatur (dengan transpose internal) agar sesuai dengan output flatten dari keras.
   v. Output data untuk layer Dense.
f. Dense.forward(input_data):
   i. Menerima input satu dimensi (setelah flatten) atau dua dimensi (jika sudah dalam batch).
   ii. Melakukan transformasi linear
   iii. Setiap neuron output terhubung ke semua neuron input dari layer sebelumnya.
   iv. Output layer fully connected.
g. SoftmaxActivation.forward(input_data):
   i. layer terakhir untuk tugas klasifikasi.
   ii. Menerima skor mentah dari layer dense sebelumnya.
   iii. Mengubah skor ini menjadi distribusi probabilitas atas semua kelas.
   iv. Jumlah semua probabilitas output akan menjadi 1.
   v. Setiap nilai output merepresentasikan probabilitas input termasuk dalam kelas tertentu.

3. Output Akhir Model:
   Setelah data melewati semua layer, current_output terakhir yang dihasilkan adalah output final dari CNNModel.forward(). Hasilnya berupa probabilitas untuk setiap kelas.

RNN

a. main.py

Atribut/Method	Deskripsi
`load_keras_model_and_vectorizer(model_config_name: str)`	Memuat model Keras yang tersimpan dan lapisan TextVectorization di dalamnya
`run_test_scratch(...)`	Inti dari mode --test-scratch. Memuat model Keras, membangun model dari nol dengan arsitektur dan bobot yang sesuai, melakukan inferensi (forward pass) dengan dukungan batch inference, dan membandingkan hasilnya dengan model Keras
`main()`	Mem-parsing argumen command-line dan memanggil fungsi yang relevan untuk menjalankan pipeline pelatihan atau pengujian perbandingan model
`scratch_model.forward(vectorized_test_data_np, training=False)`	Menjalankan forward pass melalui lapisan-lapisan untuk mendapatkan prediksi dari model yang dibangun dari nol

b. train.py

Atribut/Method	Deskripsi
`load_data(file_path: str)`	Memuat dan melakukan pra-pemrosesan data dari file CSV

`build_model(...)`	Membangun model tf.keras.Sequential dengan arsitektur RNN tertentu, termasuk TextVectorization, Embedding, RNN (SimpleRNN, Bidirectional), Dropout, dan Dense
`train_and_evaluate_model(...)`	Melatih model Keras, melakukan validasi, evaluasi pada data tes, dan menyimpan model terbaik untuk konfigurasi tersebut
`run_training_pipeline()`	Mengorkestrasi seluruh proses pelatihan, termasuk hyperparameter sweep, memanggil build_model dan train_and_evaluate_model, serta menyimpan model terbaik keseluruhan
`model.fit()`	Secara internal melakukan forward propagation untuk menghitung prediksi, diikuti kalkulasi loss dan backward propagation untuk memperbarui bobot
`model.predict()`	Melakukan forward propagation untuk evaluasi model

c. rnn.py

Fungsi Aktivasi dan Turunannya

Atribut/Method	Deskripsi
`tanh_np()`	Implementasi fungsi aktivasi hyperbolic tangent menggunakan NumPy
`softmax_np()`	Implementasi fungsi aktivasi softmax dengan stabilitas numerik (mengurangi nilai maksimum)
`relu_np()`	Implementasi fungsi aktivasi ReLU (Rectified Linear Unit)
`sigmoid_np()`	Implementasi fungsi aktivasi sigmoid
`dtanh_np()`	Turunan dari fungsi tanh untuk backpropagation
`drelu_np()`	Turunan dari fungsi ReLU untuk backpropagation
`dsigmoid_np()`	Turunan dari fungsi sigmoid untuk backpropagation

Fungsi Loss dan Turunannya

Atribut/Method	Deskripsi
`cross_entropy_loss_np()`	Menghitung cross-entropy loss untuk klasifikasi multi-kelas
`d_cross_entropy_softmax_np()`	Turunan gabungan cross-entropy dan softmax untuk efisiensi komputasi

Class `EmbeddingLayerNP`

Atribut/Method	Deskripsi
`__init__()`	Inisialisasi layer embedding dengan matriks weight
`forward()`	Mengkonversi token indeks menjadi vektor embedding
`backward()`	Menghitung gradien untuk weight embedding menggunakan np.add.at()
`update_weights()`	Memperbarui weight embedding berdasarkan gradien

Class `SimpleRNNLayerNP`

Atribut/Method	Deskripsi
`__init__()`	Inisialisasi RNN layer dengan parameter units, aktivasi, return_sequences, dan go_backwards
`load_weights()`	Memuat weight (kernel, recurrent_kernel, bias) ke dalam layer
`forward()`	Implementasi forward pass RNN dengan dukungan arah maju/mundur dan return sequences
`backward()`	Implementasi backpropagation through time (BPTT) untuk RNN
`update_weights()`	Memperbarui semua weight RNN berdasarkan gradien yang dihitung

Class `BidirectionalWrapperNP`

Atribut/Method	Deskripsi
`__init__()`	Menggabungkan dua RNN layer (forward dan backward) untuk membuat bidirectional RNN
`forward()`	Menjalankan forward pass pada kedua arah dan menggabungkan output
`backward()`	Membagi gradien untuk kedua arah dan menjalankan backward pass
`update_weights()`	Memperbarui weight untuk kedua RNN layer

Class `DropoutLayerNP`

Atribut/Method	Deskripsi
`__init__()`	Inisialisasi dropout layer dengan tingkat dropout tertentu
`forward()`	Menerapkan dropout saat training atau passthrough saat inference
`backward()`	Menerapkan mask yang sama pada gradien saat training
`update_weights()`	Method kosong karena dropout tidak memiliki parameter yang dapat dilatih

Class `DenseLayerNP`

Atribut/Method	Deskripsi
`__init__()`	Inisialisasi fully connected layer dengan jumlah unit dan fungsi aktivasi
`load_weights()`	Memuat weight (kernel dan bias) ke dalam layer
`forward()`	Implementasi forward pass dengan transformasi linear dan aktivasi

`backward()`	Menghitung gradien untuk weight dan bias, serta gradien untuk layer sebelumnya
`update_weights()`	Memperbarui weight dan bias berdasarkan gradien

Class `SequentialFromScratch`

Atribut/Method	Deskripsi
`__init__()`	Inisialisasi model sequential dengan daftar layer
`forward()`	Menjalankan forward pass melalui semua layer secara berurutan
`backward()`	Menjalankan backward pass melalui semua layer dalam urutan terbalik
`update_weights()`	Memperbarui weight untuk semua layer yang memiliki parameter

d. Forward Propagation

Forward propagation pada SimpleRNNLayerNP melibatkan:

$$ht = activation((Xt \cdot Wx) + (ht - 1 \cdot Wh) + bh)$$

Alur Data:

1. Teks input diubah menjadi token integer (oleh Keras TextVectorization di main.py atau train.py).
2. EmbeddingLayerNP mengubah token menjadi vektor.
3. SimpleRNNLayerNP memproses sekuens vektor, memperbarui hidden state di setiap time step.
4. Output dari RNN diteruskan ke lapisan berikutnya (misalnya, DenseLayerNP).

LSTM

a. main.py

Atribut/Method	Deskripsi

`main()`	Fungsi utama yang mengatur seluruh alur program: memuat model Keras, mengekstrak konfigurasi, membuat model from-scratch, mentransfer weights, dan membandingkan performa kedua model

b. lstm.py

Class `Embedding`

Atribut/Method	Deskripsi
`weights`	Matrix embedding dengan shape (input_dim, output_dim) yang menyimpan vektor representasi untuk setiap token dalam vocabulary
`__init__(self, input_dim, output_dim, weights=None)`	Konstruktor untuk inisialisasi layer embedding dengan dimensi input/output dan optional pre-trained weights
`forward(self, x)`	Forward pass untuk mengkonversi indeks token menjadi vektor embedding dengan clipping untuk mencegah index out of bounds

Class `LSTM`

Atribut/Method	Deskripsi
`units`	Jumlah unit/neuron dalam LSTM cell
`return_sequences`	Flag untuk menentukan apakah mengembalikan seluruh sequence (`True`) atau hanya output terakhir (`False`)
`input_dim`	Dimensi input untuk layer LSTM
`weights_initialized`	Flag untuk menandai apakah weights sudah diinisialisasi

`W_i, W_f, W_c, W_o`	Weight matrices untuk transformasi input pada input gate, forget gate, candidate values, dan output gate
`U_i, U_f, U_c, U_o`	Weight matrices untuk transformasi hidden state pada input gate, forget gate, candidate values, dan output gate
`b_i, b_f, b_c, b_o`	Bias vectors untuk input gate, forget gate, candidate values, dan output gate
`__init__(self, units, return_sequences=False, weights=None, input_dim=None)`	Konstruktor LSTM dengan parameter jumlah unit, mode return sequences, optional weights, dan dimensi input
`_initialize_weights(self, input_dim)`	Inisialisasi weights LSTM menggunakan Xavier/Glorot initialization untuk gate input, forget, cell, dan output
`sigmoid(self, x)`	Implementasi fungsi aktivasi sigmoid dengan numerical stability menggunakan clipping dan conditional computation
`_orthogonal_init(self, shape)`	Inisialisasi orthogonal untuk weights (method ini ada tapi tidak digunakan dalam kode)
`tanh(self, x)`	Implementasi fungsi aktivasi tanh dengan clipping untuk mencegah overflow
`forward(self, x)`	Forward pass LSTM yang mengimplementasikan gate mechanism lengkap (input, forget, cell, output) untuk setiap timestep

Class `BidirectionalLSTM`

Atribut/Method	**Deskripsi**

`units`	Jumlah unit untuk setiap LSTM (forward dan backward)
`return_sequences`	Flag untuk menentukan format output sequence
`forward_lstm`	Instance LSTM untuk memproses sequence dari awal ke akhir
`backward_lstm`	Instance LSTM untuk memproses sequence dari akhir ke awal
`__init__(self, units, return_sequences=False, weights=None, input_dim=None)`	Konstruktor untuk Bidirectional LSTM yang membuat dua LSTM terpisah untuk forward dan backward direction
`forward(self, x)`	Forward pass yang memproses input secara forward dan backward, kemudian menggabungkan output kedua arah

Class `Dropout`

Atribut/Method	Deskripsi
`rate`	Tingkat dropout (proporsi unit yang akan di-drop selama training)
`mask`	Mask untuk menentukan unit mana yang akan di-drop (saat ini tidak diimplementasikan)
`__init__(self, rate)`	Konstruktor untuk layer dropout dengan parameter dropout rate
`forward(self, x, training=False)`	Forward pass dropout yang mengembalikan input tanpa modifikasi (dropout tidak diimplementasi secara aktif)

Class `Dense`

Atribut/Method	Deskripsi
`units`	Jumlah neuron dalam layer dense
`activation`	Jenis fungsi aktivasi (`softmax`, `relu`, `tanh`, atau `None`)
`weights`	Weight matrix dengan shape `(input_dim, units)`
`bias`	Bias vector dengan shape `(units,)`
`weights_initialized`	Flag untuk menandai apakah weights sudah diinisialisasi
`_initialize_weights(self, input_dim)`	Inisialisasi weights dense layer menggunakan uniform distribution dengan Xavier initialization
`softmax(self, x)`	Implementasi fungsi softmax dengan numerical stability menggunakan shifting dan clipping
`forward(self, x)`	Forward pass dense layer dengan linear transformation dan aplikasi fungsi aktivasi (softmax, relu, tanh, atau linear)

Class `LSTMModel`

Atribut/Method	Deskripsi
`vocab_size`	Ukuran vocabulary untuk layer embedding
`embedding_dim`	Dimensi vektor embedding

`lstm_units`	Jumlah unit dalam setiap layer LSTM
`lstm_layers`	Jumlah layer LSTM dalam model
`bidirectional`	Flag untuk menggunakan Bidirectional LSTM
`dropout_rate`	Tingkat dropout yang diterapkan
`num_classes`	Jumlah kelas untuk klasifikasi
`layers`	List berisi tuple `(nama_layer, instance_layer)` untuk semua layer dalam model
`__init__(self, vocab_size, embedding_dim, lstm_units, lstm_layers, bidirectional, dropout_rate, num_classes)`	Konstruktor model LSTM lengkap dengan konfigurasi arsitektur neural network
`initialize_layers(self)`	Inisialisasi semua layer dalam model sesuai konfigurasi (embedding, LSTM/Bidirectional, dropout, dense)
`forward(self, x, training=False)`	Forward pass untuk seluruh model yang melewatkan input melalui semua layer secara berurutan
`load_weights_from_keras(self, keras_model)`	Transfer weights dari model Keras ke implementasi from-scratch dengan mapping layer yang sesuai

`_load_lstm_weights(self, keras_lstm, scratch_lstm)`	Helper method untuk mentransfer weights dari LSTM Keras ke implementasi LSTM from-scratch
`_load_bidirectional_weights(self, keras_bidirectional, scratch_bidirectional)`	Helper method untuk mentransfer weights dari Bidirectional LSTM Keras ke implementasi from-scratch

Fungsi

Atribut/Method	Deskripsi
`batch_predict(model, vectorizer, X, batch_size=32)`	Melakukan prediksi dalam batch untuk menangani dataset besar secara memory-efficient
`load_and_preprocess_test_data(test_csv_path)`	Memuat dan preprocessing data test dari CSV, termasuk encoding label menggunakan LabelEncoder

c. Forward Propagation

1. Input

   Data input (input_data) pertama kali diterima oleh metode `LSTMModel.forward()`. Data ini biasanya berupa array NumPy 2D dengan format (batch_size, sequence_length), di mana setiap elemen adalah indeks token dari vocabulary yang telah di-vectorize.

2. Operasi per Layer:

   Untuk setiap layer, metode `forward()` dari layer tersebut dipanggil dengan current_output (yang merupakan output dari layer sebelumnya, atau data input awal untuk layer pertama) sebagai argumennya.

   a. Embedding.forward(input_data):
      - Menerima indeks token dari input awalnya (batch_size, sequence_length).
      - Operasi lookup embedding dilakukan dengan mengambil vektor embedding untuk setiap indeks token.

- Indeks di-clip untuk mencegah index out of bounds menggunakan `np.clip()`.
- Menggunakan `np.take()` untuk mengambil vektor embedding dari weight matrix.
- Outputnya adalah tensor 3D (batch_size, sequence_length, embedding_dim) yang merepresentasikan representasi vektor dari setiap token.

b. LSTM.forward(input_data)/BidirectionalLSTM.forward(input_data):
   - Menerima output dari layer embedding dengan shape (batch_size, sequence_length, embedding_dim).
   - Untuk LSTM biasa:
     - Inisialisasi hidden state (h_t) dan cell state (c_t) dengan nilai nol.
     - Untuk setiap timestep dalam sequence:
       - **Input Gate**: `i_t = sigmoid(W_i * x_t + U_i * h_t-1 + b_i)` - menentukan informasi baru mana yang akan disimpan.
       - **Forget Gate**: `f_t = sigmoid(W_f * x_t + U_f * h_t-1 + b_f)` - menentukan informasi lama mana yang akan dilupakan.
       - **Candidate Values**: `c_tilde = tanh(W_c * x_t + U_c * h_t-1 + b_c)` - nilai kandidat baru untuk cell state.
       - **Cell State Update**: `c_t = f_t * c_t-1 + i_t * c_tilde` - memperbarui cell state.
       - **Output Gate**: `o_t = sigmoid(W_o * x_t + U_o * h_t-1 + b_o)` - menentukan bagian mana dari cell state yang akan dioutput.
       - **Hidden State**: `h_t = o_t * tanh(c_t)` - hidden state baru.
   - Untuk BidirectionalLSTM:
     - Memproses input secara forward dan backward secara terpisah.
     - Input dibalik untuk backward LSTM menggunakan slicing `[:, ::-1, :]`.
     - Output backward di-reverse kembali untuk menyelaraskan dengan forward.
     - Kedua output digabungkan menggunakan `np.concatenate()` di axis terakhir.

- Output untuk return_sequences=True: (batch_size, sequence_length, lstm_units) atau (batch_size, sequence_length, 2*lstm_units) untuk bidirectional.
- Output untuk return_sequences=False: (batch_size, lstm_units) atau (batch_size, 2*lstm_units) untuk bidirectional.

c. Dropout.forward(input_data):
   - Menerima output dari layer LSTM sebelumnya.
   - Dalam implementasi ini, dropout tidak aktif diimplementasikan (hanya mengembalikan input tanpa modifikasi).
   - Pada kondisi training=True seharusnya mengaplikasikan random masking, namun di sini hanya sebagai placeholder.
   - Output sama dengan input tanpa perubahan.

d. Dense.forward(input_data):
   - Menerima hidden state terakhir dari LSTM atau output dari dropout.
   - Melakukan transformasi linear: `output = input_data * weights + bias`.
   - Setiap neuron output terhubung ke semua neuron input dari layer sebelumnya.
   - Jika menggunakan aktivasi softmax:
     - Menerapkan shifting untuk numerical stability: `x_shifted = x - max(x)`.
     - Menghitung exponential dengan clipping: `e_x = exp(clip(x_shifted, -500, 500))`.
     - Normalisasi: `softmax = e_x / sum(e_x)` untuk menghasilkan distribusi probabilitas.
   - Jika menggunakan aktivasi lain (relu, tanh): menerapkan fungsi aktivasi yang sesuai.
   - Output layer fully connected dengan shape (batch_size, num_classes).

3. Output Akhir Model:

Setelah data melewati semua layer, current_output terakhir yang dihasilkan adalah output final dari `LSTMModel.forward()`. Hasilnya berupa:

- **Untuk klasifikasi dengan softmax**: Distribusi probabilitas atas semua kelas dengan shape (batch_size, num_classes), di mana jumlah probabilitas untuk setiap sample adalah 1.

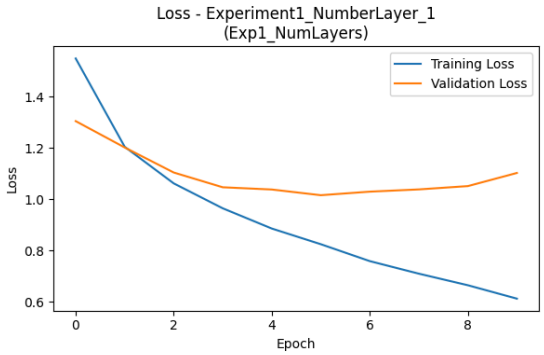
- **Tanpa softmax**: Raw scores/logits yang dapat digunakan untuk berbagai keperluan downstream.

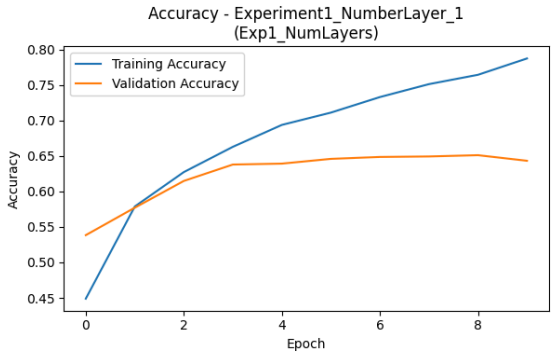
Setiap nilai output merepresentasikan probabilitas atau skor untuk input sequence termasuk dalam kelas tertentu, berdasarkan pemahaman kontekstual yang telah dipelajari oleh LSTM dari seluruh sequence input.

Hasil Pengujian

CNN

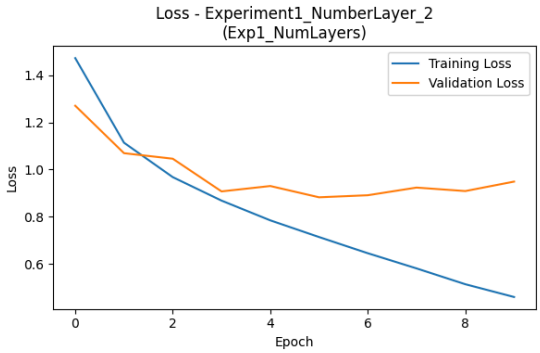
Pengaruh jumlah layer konvolusi

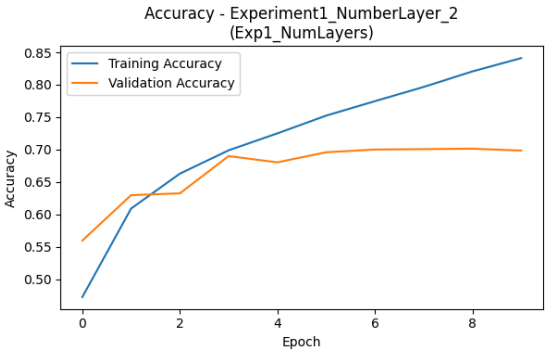




Jumlah Layer Konvolusi 1

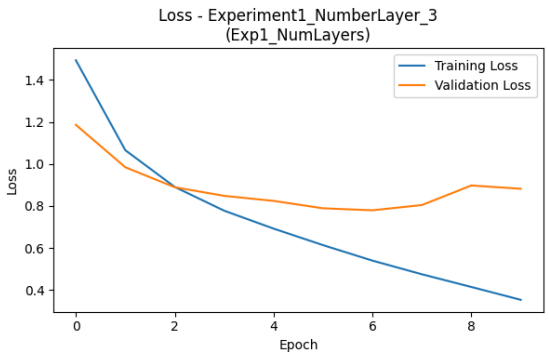
F1 Score : 0.6407
Loss : 1.0800
Accuracy : 0.6411

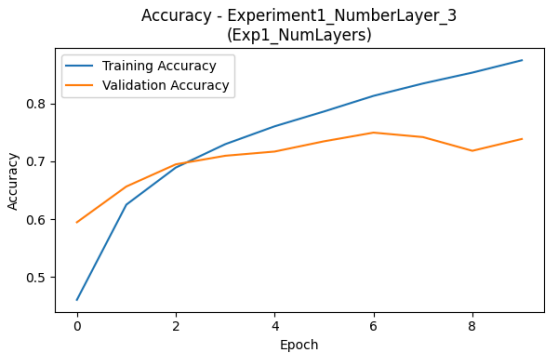




Jumlah Layer Konvolusi 2

F1 Score : 0.7067
Loss : 0.9731
Accuracy : 0.7068





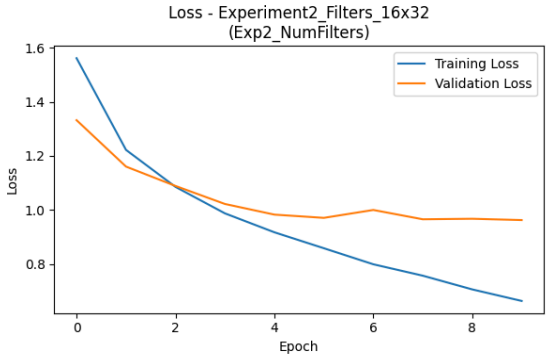
Jumlah Layer Konvolusi 3

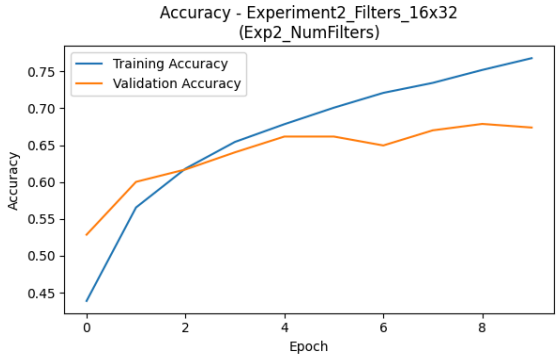
F1 Score : 0.7350
Loss : 0.8874
Accuracy : 0.7334

Kesimpulan

Penambahan jumlah layer konvolusi dari satu ke tiga layer menunjukkan peningkatan performa yang konsisten pada metrik F1 Score, Loss, dan Accuracy. Model dengan menggunakan tiga layer konvolusi memberikan hasil terbaik.

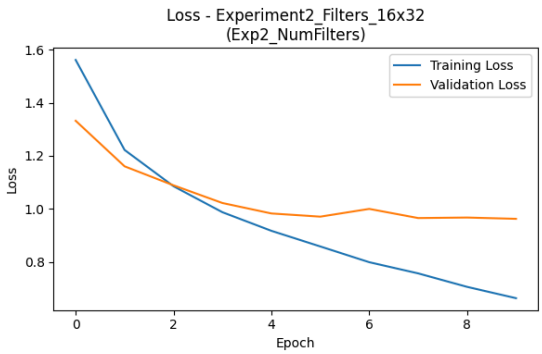
Pengaruh banyak filter per layer konvolusi

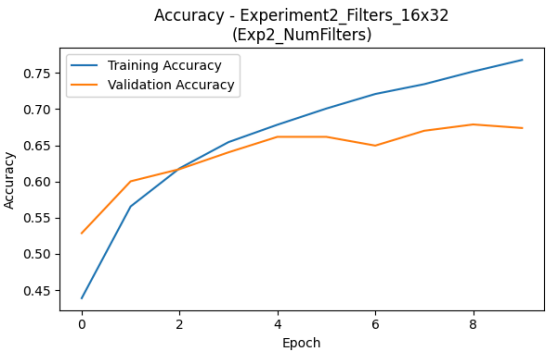




Banyak Filter per Layer Konvolusi [16,32]

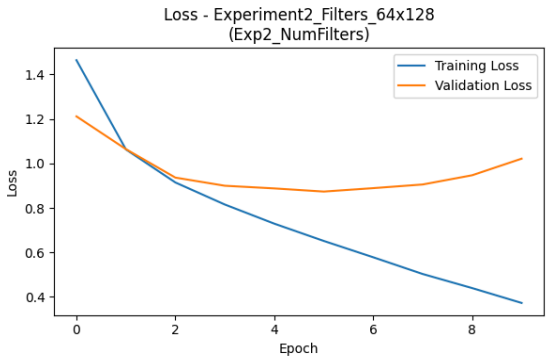
F1 Score : 0.6810
Loss : 0.9531
Accuracy : 0.6810

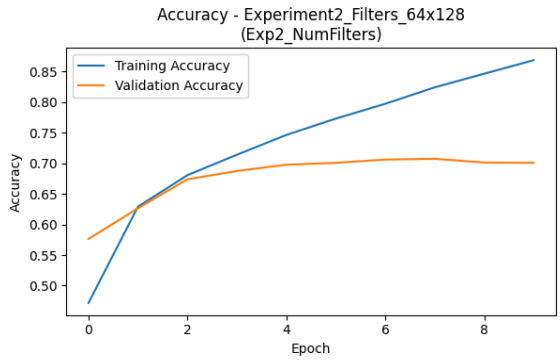




Banyak Filter per Layer Konvolusi [32,64]

F1 Score : 0.6983
Loss : 0.9714
Accuracy : 0.6984





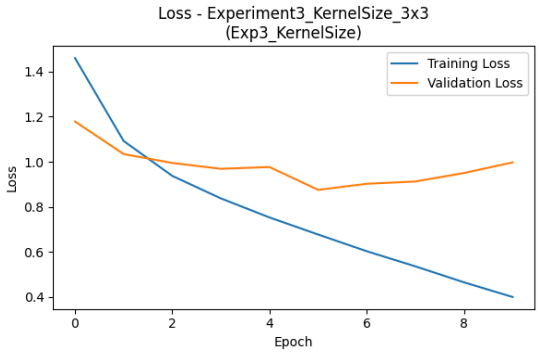
Banyak Filter per Layer Konvolusi [64,128]

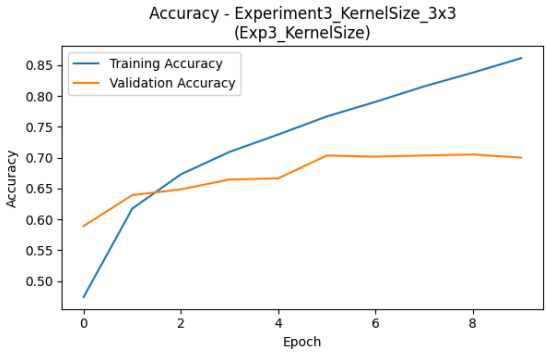
F1 Score : 0.7106
Loss : 1.1120
Accuracy : 0.7123

Kesimpulan

Penambahan jumlah filter per layer konvolusi menunjukkan peningkatan pada F1 Score dan Accuracy, namun terjadi juga peningkatan pada Loss, karena ini bisa menjadi indikasi awal bahwa model menjadi terlalu kompleks atau mulai overfit.

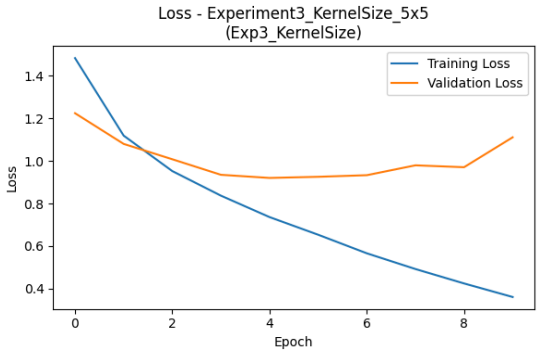
Pengaruh ukuran filter per layer konvolusi

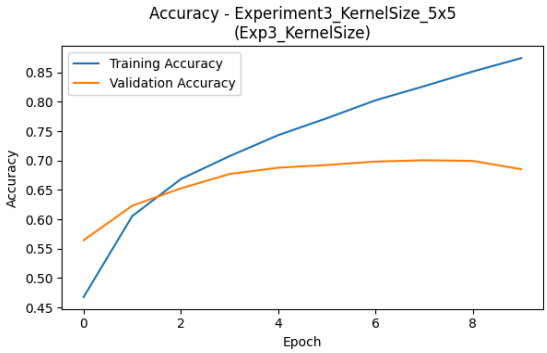




Ukuran Filter per Layer Konvolusi [3x3]

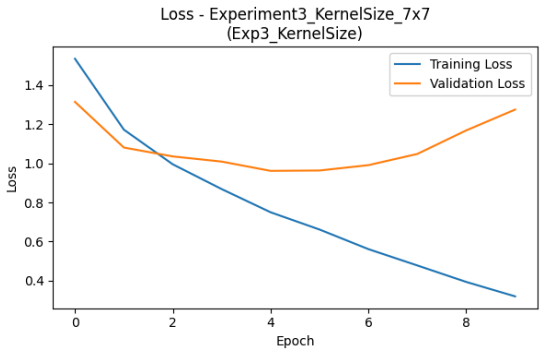
F1 Score : 0.6982
Loss : 0.9790
Accuracy : 0.6956

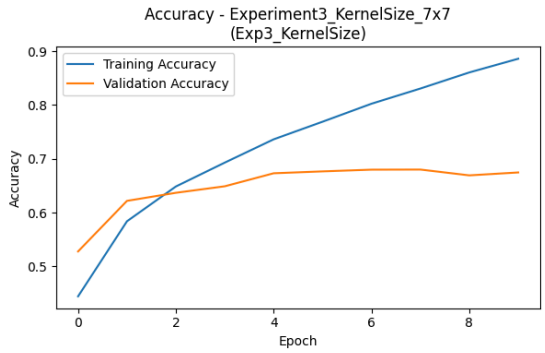




Ukuran Filter per Layer Konvolusi [5x5]

F1 Score : 0.7032
Loss : 1.0384
Accuracy : 0.7025





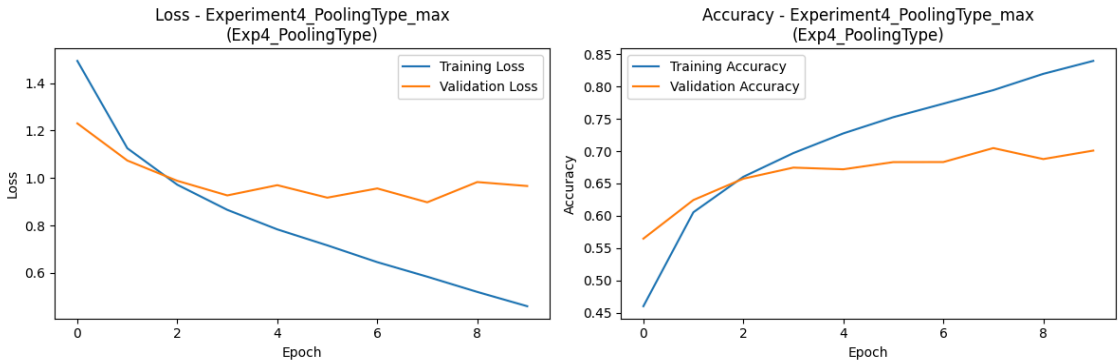
Ukuran Filter per Layer Konvolusi [7x7]

F1 Score : 0.6492
Loss : 1.2382
Accuracy : 0.6492

Kesimpulan

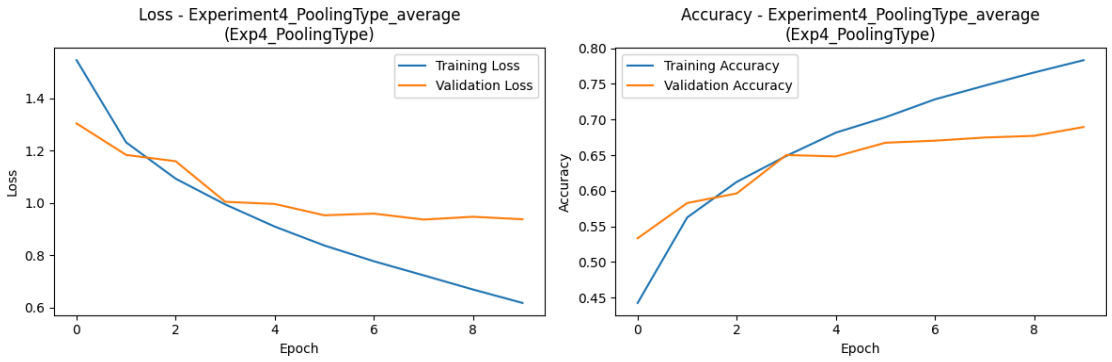
Pada kasus ini, filter berukuran 5x5 menjadi pilihan yang paling seimbang untuk memaksimalkan F1 Score dan Accuracy, meskipun adanya sedikit kenaikan loss. Filter 3x3 memberikan loss terendah, sementara filter 7x7 jelas tidak optimal.

Pengaruh jenis pooling layer yang digunakan



Jenis Pooling Layer yang Digunakan (Max Pooling)

F1 Score : 0.7061
Loss : 0.9442
Accuracy : 0.7082



Jenis Pooling Layer yang Digunakan (Avg Pooling)

F1 Score : 0.6804
Loss : 0.9552
Accuracy : 0.6831

Kesimpulan

Penggunaan Max Pooling menghasilkan performa model yang lebih baik dibandingkan dengan Average Pooling. Max Pooling cenderung lebih efektif dalam mengekstraksi fitur-fitur yang paling dari feature map, yang bermanfaat untuk dataset ini.

Perbandingan CNN Keras dengan LSTM *From Scratch*

Keras : 0.735046
Scratch : 0.735046
Jumlah prediksi kelas yang sama persis antara Keras dan Scratch: 10000/10000 (100.00%).

Laporan Klasifikasi Model Keras:

	precision	recall	f1-score	support
airplane	0.79	0.78	0.78	1000
automobile	0.84	0.86	0.85	1000
bird	0.63	0.67	0.65	1000
cat	0.54	0.56	0.55	1000
deer	0.70	0.70	0.70	1000
dog	0.59	0.67	0.63	1000
frog	0.84	0.74	0.79	1000
horse	0.85	0.71	0.77	1000
ship	0.84	0.82	0.83	1000
truck	0.77	0.81	0.79	1000
accuracy			0.73	10000
macro avg	0.74	0.73	0.74	10000
weighted avg	0.74	0.73	0.74	10000

Laporan Klasifikasi Model From Scratch:

	precision	recall	f1-score	support
airplane	0.79	0.78	0.78	1000
automobile	0.84	0.86	0.85	1000
bird	0.63	0.67	0.65	1000
cat	0.54	0.56	0.55	1000
deer	0.70	0.70	0.70	1000
dog	0.59	0.67	0.63	1000
frog	0.84	0.74	0.79	1000
horse	0.85	0.71	0.77	1000
ship	0.84	0.82	0.83	1000
truck	0.77	0.81	0.79	1000
accuracy			0.73	10000
macro avg	0.74	0.73	0.74	10000
weighted avg	0.74	0.73	0.74	10000

Kesimpulan

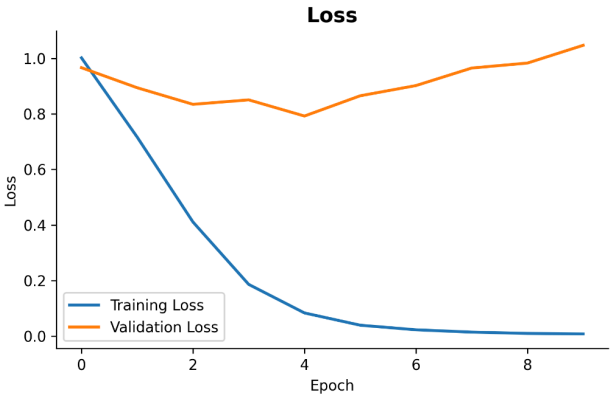
Implementasi model dari scratch berhasil bekerja dalam memprediksi sama seperti model yang dibangun menggunakan keras dengan sempurna (100%).

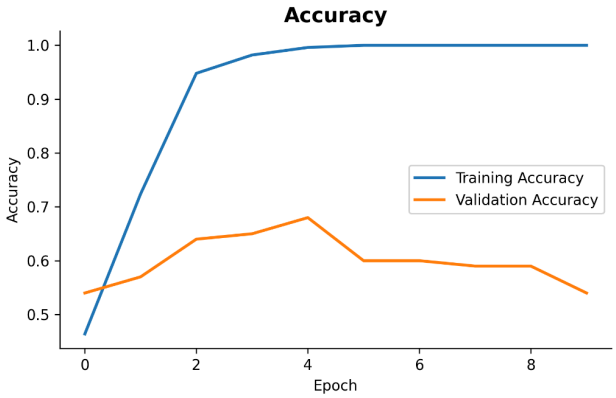
RNN

Pengaruh jumlah layer RNN

1. **1 Layer**

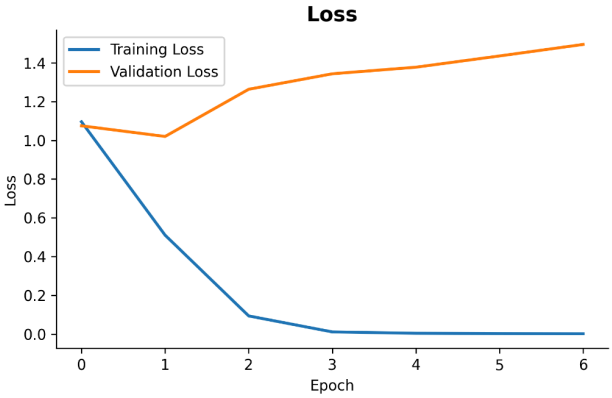
F1 Score : 0.5920
Loss : 1.0472
Accuracy : 0.5400

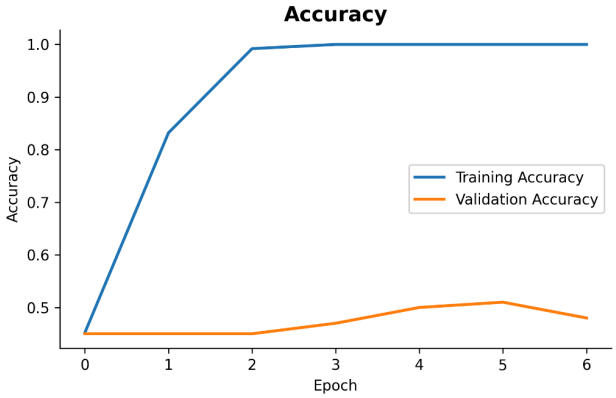




2. 2 Layer

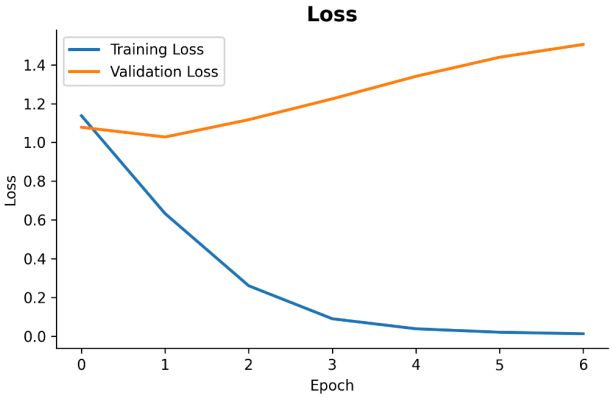
F1 Score : 0.5358
Loss : 1.4951
Accuracy : 0.4800

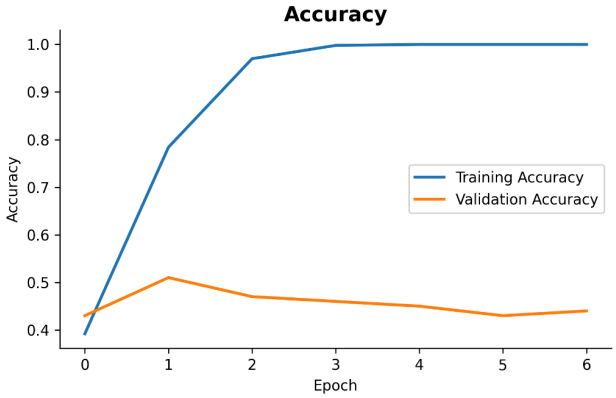




3. 3 Layer

F1 Score : 0.4234
Loss : 1.5048
Accuracy : 0.4400





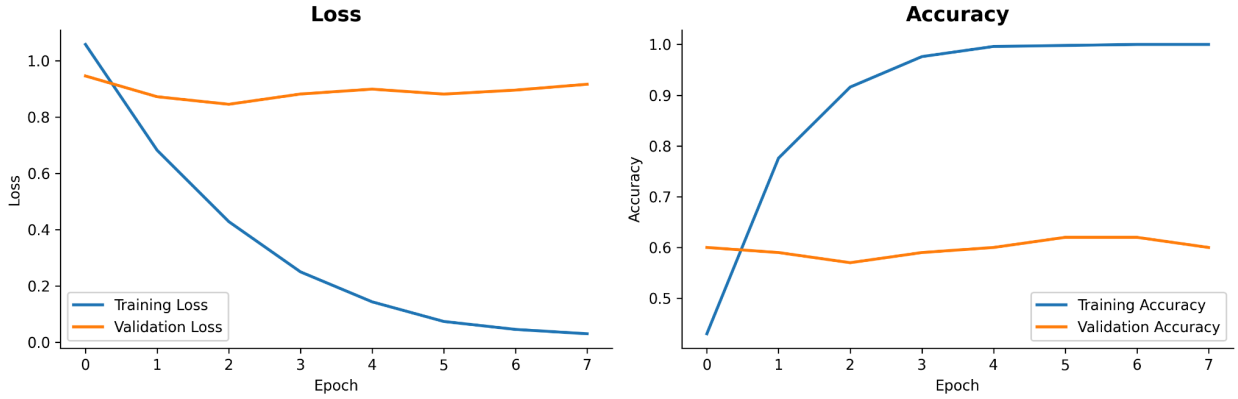
4. Kesimpulan

Berdasarkan data yang diberikan, penambahan jumlah lapisan (layer) pada RNN dalam kasus ini menunjukkan pengaruh negatif terhadap kinerja model. Model dengan 1 layer RNN mencapai F1 Score (0.5920), Loss (1.0472), dan Akurasi (0.5400) yang lebih baik dibandingkan model dengan 2 layer (F1 Score: 0.5358, Loss: 1.4951, Akurasi: 0.4800) dan 3 layer (F1 Score: 0.4234, Loss: 1.5048, Akurasi: 0.4400). Grafik loss dan akurasi juga mengindikasikan bahwa model dengan 1 layer memiliki kinerja yang lebih stabil dan generalisasi yang lebih baik, ditandai dengan perbedaan yang lebih kecil antara metrik training dan validasi serta pencapaian akurasi validasi yang lebih tinggi dibandingkan model dengan jumlah layer yang lebih banyak, yang cenderung mengalami overfitting lebih parah seiring bertambahnya epoch.

Pengaruh banyak cell RNN per layer

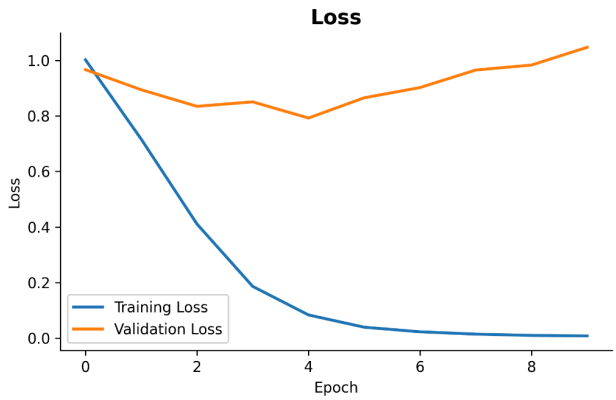
1. **32 Units**

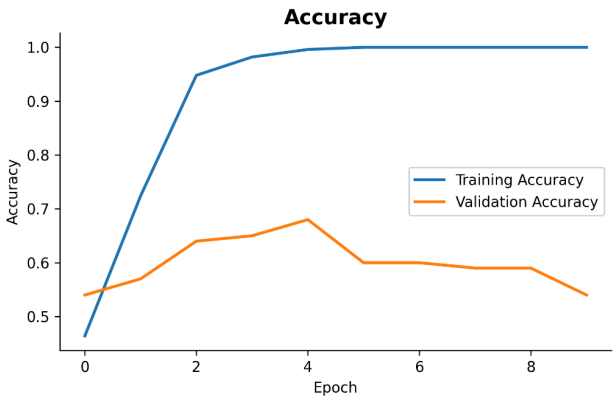
F1 Score : 0.5368
Loss : 0.9168
Accuracy : 0.6000



2. 64 Units

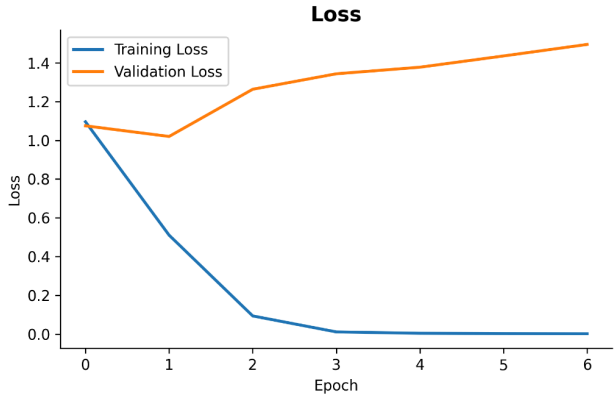
F1 Score : 0.5358
Loss : 1.4951
Accuracy : 0.4800

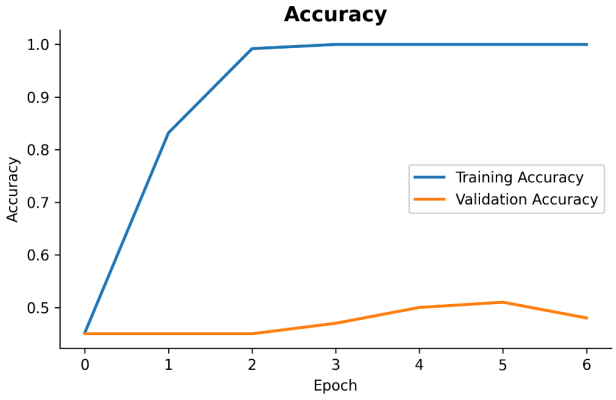




3. 128 Units

F1 Score : 0.4234
Loss : 2.1084
Accuracy : 0.2900





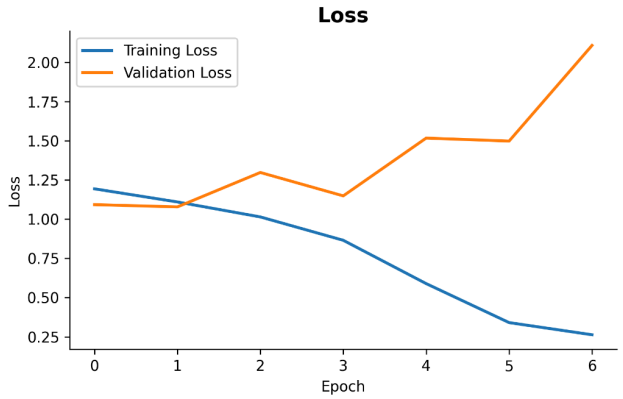
4. Kesimpulan

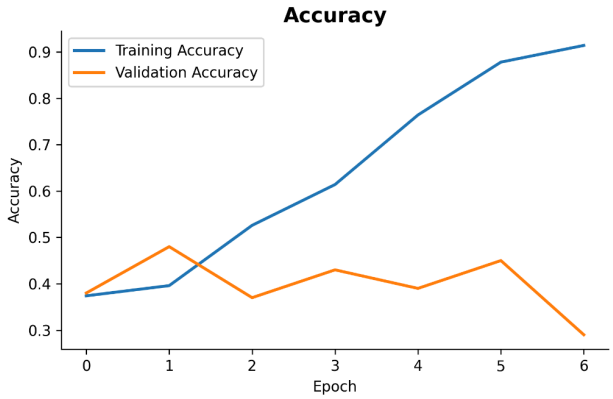
Berdasarkan data yang disajikan, peningkatan jumlah unit (cell) RNN per layer dalam kasus ini cenderung menurunkan kinerja model. Model dengan 32 unit menunjukkan F1 Score (0.5368), Loss (0.9168), dan Akurasi (0.6000) yang paling baik. Seiring dengan penambahan jumlah unit menjadi 64, terjadi penurunan kinerja (F1 Score: 0.5358, Loss: 1.4951, Akurasi: 0.4800), dan penurunan ini semakin signifikan pada model dengan 128 unit (F1 Score: 0.4234, Loss: 2.1084, Akurasi: 0.2900). Grafik loss dan akurasi juga mendukung kesimpulan ini, di mana model dengan 32 unit memiliki loss validasi yang lebih rendah dan akurasi validasi yang lebih tinggi dibandingkan model dengan jumlah unit yang lebih banyak, yang menunjukkan kecenderungan overfitting yang lebih besar dengan meningkatnya kompleksitas model per layer.

Pengaruh jenis layer RNN berdasarkan arah

1. Unidirectional

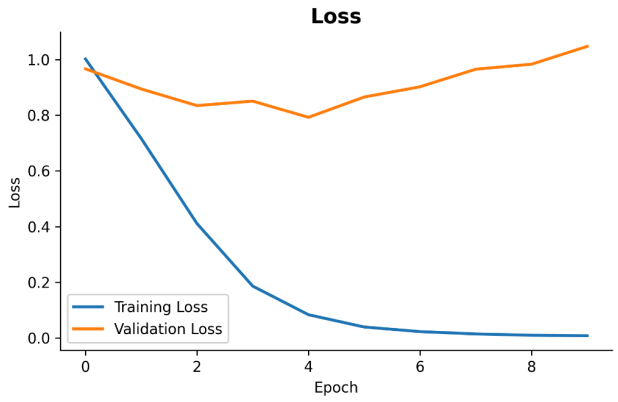
F1 Score : 0.3721
Loss : 2.1084
Accuracy : 0.2900

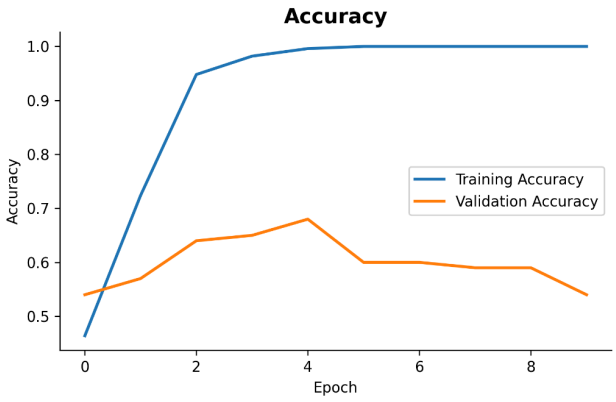




2. Bidirectional

F1 Score : 0.5920
Loss : 1.0472
Accuracy : 0.5400





3. Kesimpulan

Berdasarkan data yang ditampilkan, penggunaan layer RNN Bidirectional menunjukkan pengaruh yang positif signifikan terhadap kinerja model dibandingkan dengan layer Unidirectional. Model Bidirectional mencapai F1 Score (0.5920), Loss (1.0472), dan Akurasi (0.5400) yang jauh lebih baik daripada model Unidirectional (F1 Score: 0.3721, Loss: 2.1084, Akurasi: 0.2900). Grafik loss dan akurasi juga memperkuat kesimpulan ini, di mana model Bidirectional menunjukkan loss validasi yang lebih rendah dan konvergen lebih baik, serta akurasi validasi yang lebih tinggi dan lebih stabil seiring berjalannya epoch, mengindikasikan kemampuan generalisasi yang lebih superior dibandingkan model Unidirectional yang menunjukkan performa lebih buruk dan perbedaan yang lebih besar antara metrik training dan validasi.

Perbandingan RNN Keras dengan RNN *From Scratch*

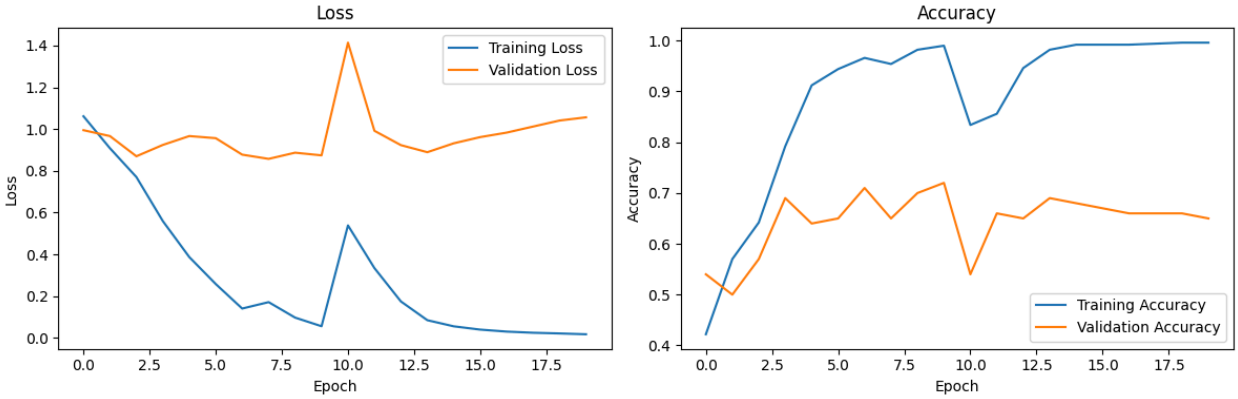
```
Loaded configuration for 'best_model' from JSON: Layers=1, Units=64, Direction=bidirectional
Getting from-scratch model predictions (for inference comparison)...
Running scratch model inference in batches of size 32...
Getting Keras model predictions (for comparison)...
13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 92ms/step
Shape of Keras predicted probabilities: (400, 3)
Shape of Scratch predicted probabilities: (400, 3)

SUCCESS: Keras and From-Scratch model probability outputs are very close!

Macro F1 Score (Keras): 0.5920
Macro F1 Score (From-Scratch): 0.5920
SUCCESS: Predicted labels from Keras and Scratch models are identical.
From-scratch test finished.
From-scratch test call for best_model finished. Check console output.
```

Berdasarkan output yang ditampilkan, perbandingan antara model RNN yang dibangun menggunakan Keras dengan model RNN yang dibangun menunjukkan hasil yang sangat identik dan berhasil. Kedua model menghasilkan output probabilitas yang sangat dekat, yang dibuktikan dengan nilai Macro F1 Score yang sama persis untuk kedua model, yaitu 0.5920. Ini mengindikasikan bahwa implementasi model RNN *from scratch* telah berhasil meniru kinerja model yang dibangun menggunakan library Keras dengan konfigurasi yang sama (Layers=1, Units=64, Direction=bidirectional).

LSTM

Dengan 20 epoch

Pengaruh jumlah layer LSTM

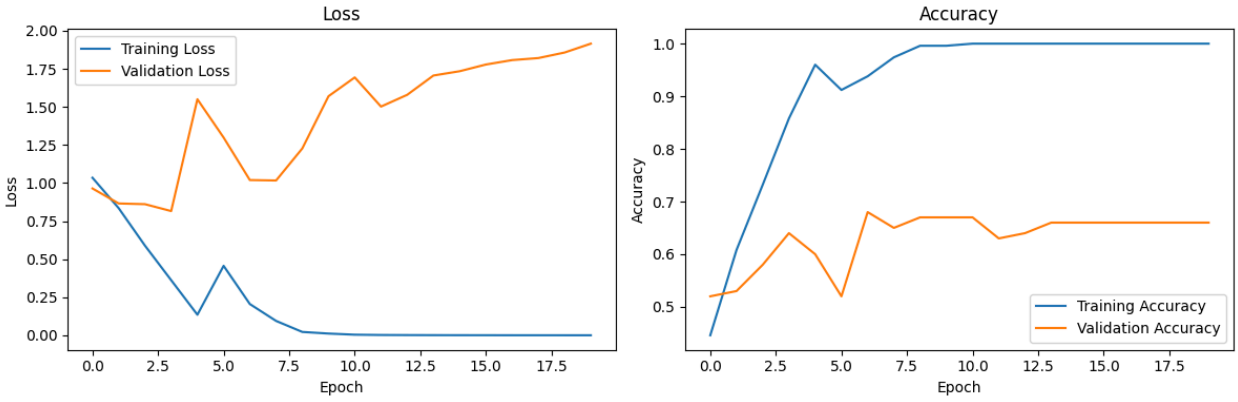


1 LSTM Layer

F1 Score : 0.6959732630789665
Loss : 1.0563
Accuracy : 0.6500

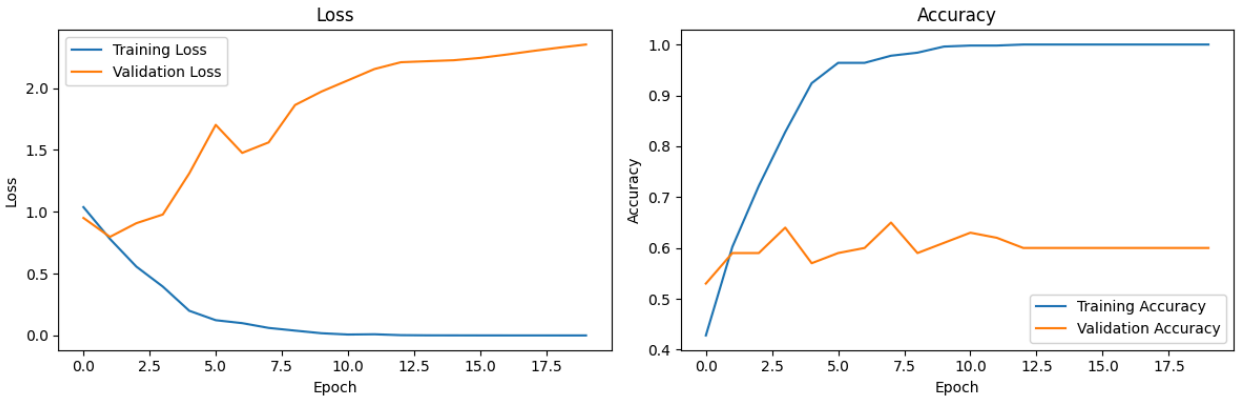
Kesimpulan

Penambahan jumlah layer LSTM menurunkan performa model. Model dengan 1 layer LSTM memberikan F1 score tertinggi (0.696), sementara penambahan layer ke-2 dan ke-3 justru menurunkan performa secara konsisten. Hal ini mengindikasikan terjadinya overfitting atau vanishing gradient problem pada arsitektur yang lebih dalam. Untuk dataset ini, arsitektur yang sederhana (1 layer) lebih optimal.



2 LSTM Layer

F1 Score : 0.672535299585543
Loss : 1.9156
Accuracy : 0.6600



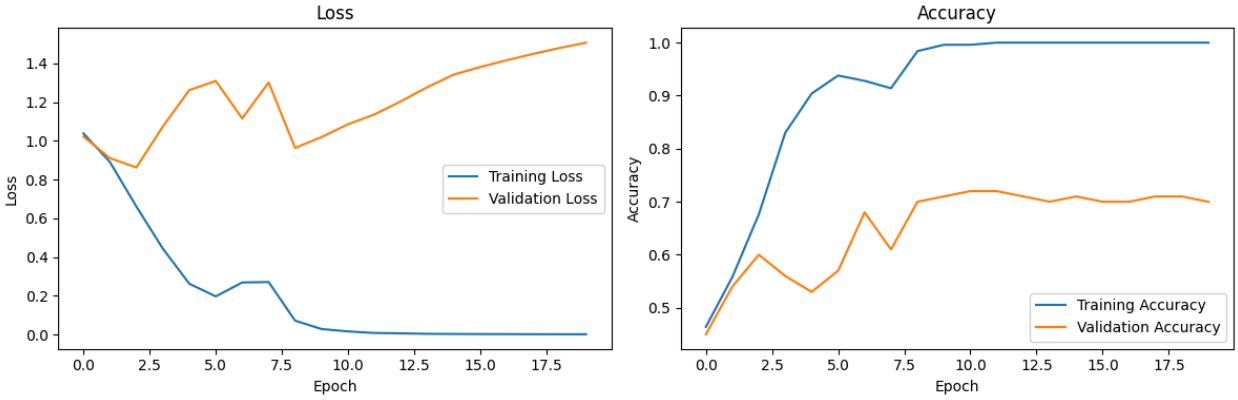
3 LSTM Layer

F1 Score : 0.6166030062785101
Loss : 2.3520
Accuracy : 0.6000

Kesimpulan

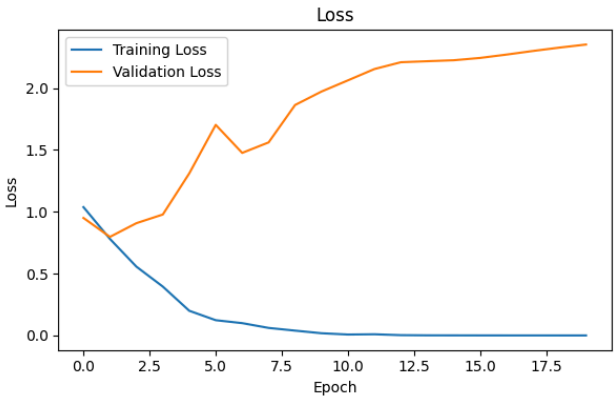
Penambahan jumlah layer LSTM menurunkan performa model. Model dengan 1 layer LSTM memberikan F1 score tertinggi (0.696), sementara penambahan layer ke-2 dan ke-3 justru menurunkan performa secara konsisten. Hal ini mengindikasikan terjadinya overfitting atau vanishing gradient problem pada arsitektur yang lebih dalam. Untuk dataset ini, arsitektur yang sederhana (1 layer) lebih optimal.

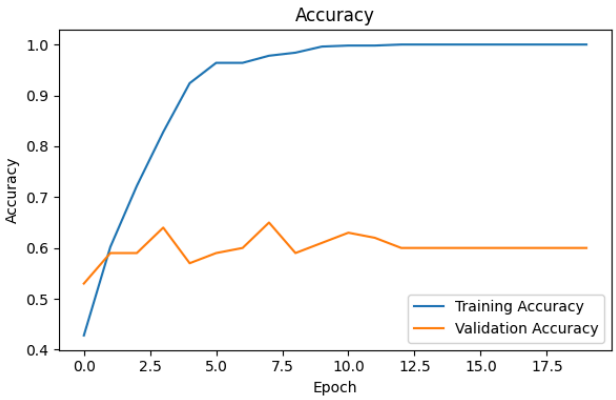
Pengaruh banyak cell LSTM per layer



64 LSTM Cell

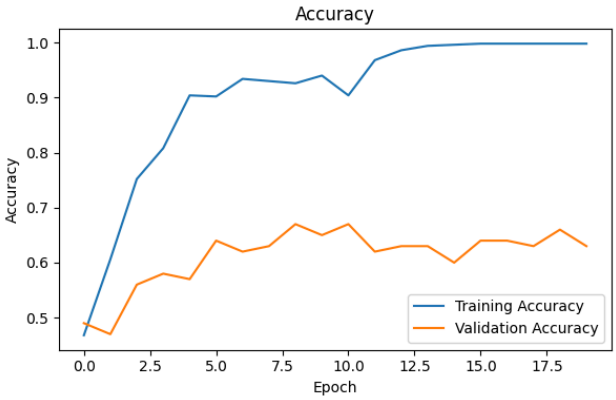
F1 Score : 0.7044453754131174
Loss : 1.5076
Accuracy : 0.7000





128 LSTM Cell

F1 Score : 0.6166030062785101
Loss : 2.3520
Accuracy : 0.6000



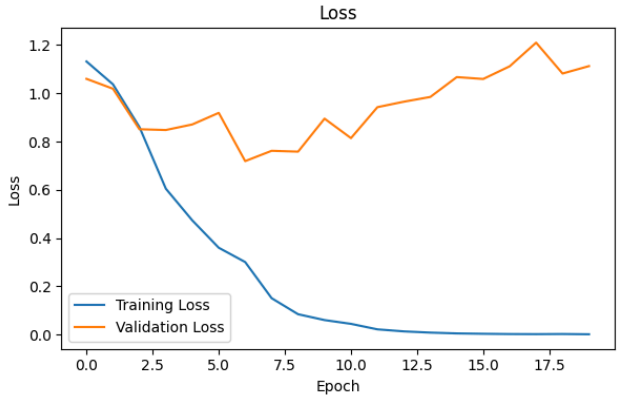
256 LSTM Cell

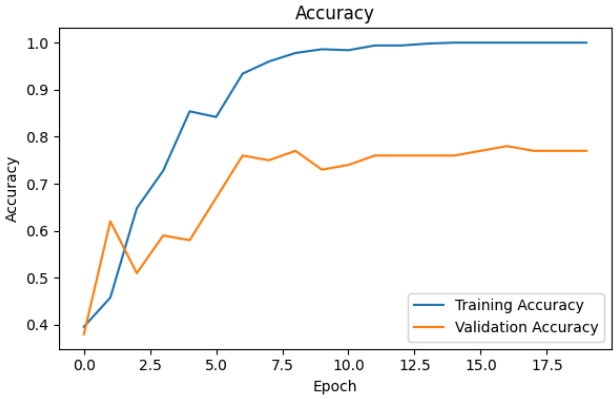
F1 Score : 0.6495954204378432
Loss : 2.2844
Accuracy : 0.6300

Kesimpulan

Jumlah *cell* yang lebih sedikit (64) memberikan performa terbaik dengan F1 score 0.704. Penambahan cell ke 128 justru menurunkan performa secara signifikan, kemungkinan karena overfitting atau kompleksitas model yang berlebihan untuk ukuran dataset. Model dengan 256 cell menunjukkan sedikit perbaikan dibanding 128, namun masih di bawah performa 64 cell. Ini menunjukkan bahwa untuk dataset ini, kapasitas model yang lebih kecil lebih sesuai.

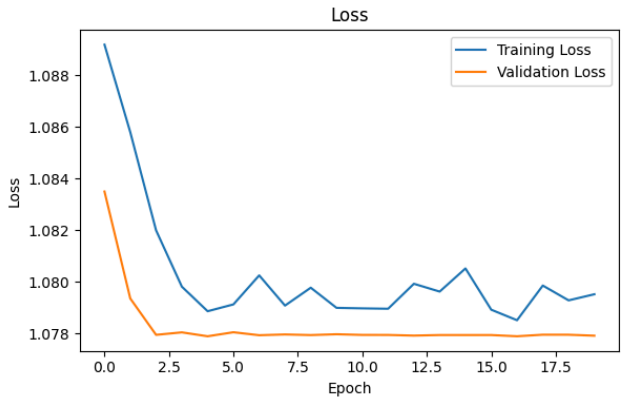
Pengaruh jenis layer LSTM berdasarkan arah

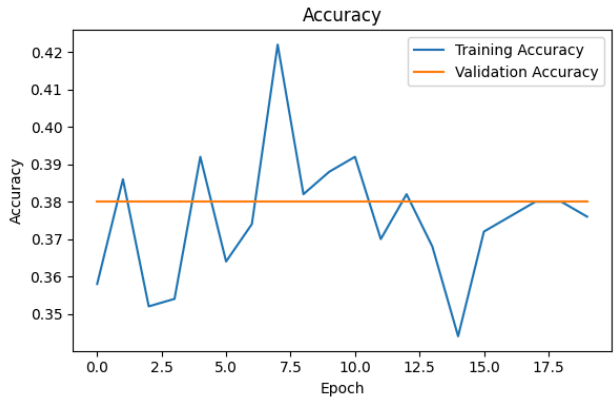




Bidirectional LSTM

F1 Score : 0.7209559470866004
Loss : 1.1129
Accuracy : 0.7700





Unidirectional LSTM

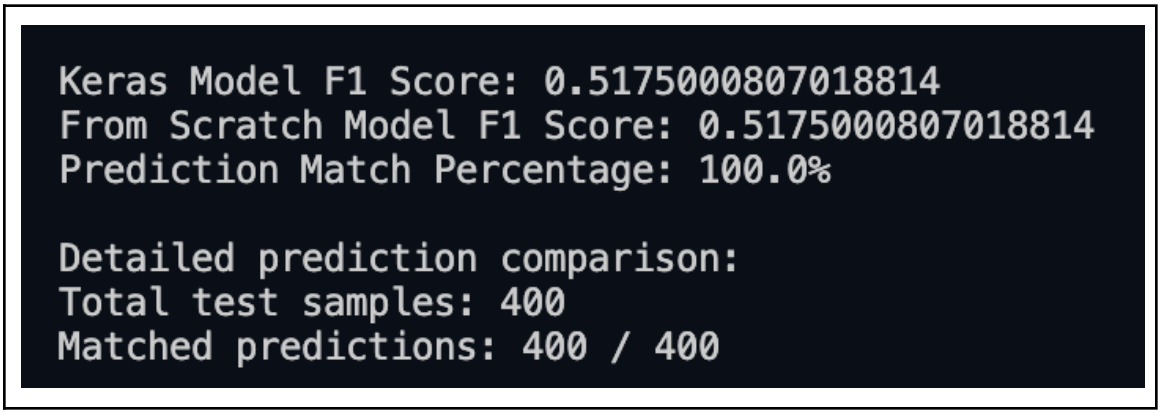
F1 Score : 0.1844484629294756
Loss : 1.0779
Accuracy : 0.3800

Kesimpulan

Bidirectional LSTM memberikan performa yang jauh lebih superior dibandingkan Unidirectional LSTM dengan selisih F1 score yang sangat signifikan. Hal ini menunjukkan bahwa konteks dari kedua arah (masa lalu dan masa depan) sangat penting untuk tugas klasifikasi teks ini. Bidirectional LSTM mampu menangkap pola dan dependensi yang lebih kompleks dalam *sequence*, menghasilkan representasi yang lebih kaya untuk proses klasifikasi.

Perbandingan LSTM Keras dengan LSTM *From Scratch*

Keras : 0.5175000807018814
Scratch : 0.5175000807018814
Jumlah prediksi kelas yang sama persis antara Keras dan *From Scratch*: 400/400 (100.00%)



BONUS

CNN

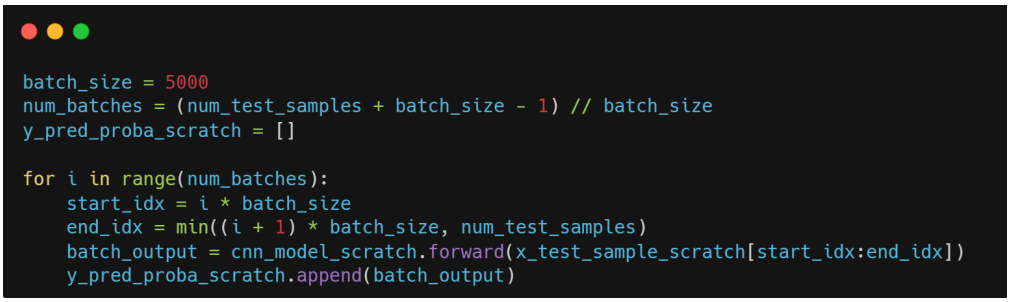
Mengimplementasikan forward propagation bisa menangani kasus batch inference, dimana model dapat menerima lebih dari satu input untuk satu kali forward propagation. Jumlah instance dalam satu batch bisa diatur dengan menggunakan suatu hyperparameter batch_size.

- Dimensi input:
  Input data memiliki shape (N, C, H, W) dimana N adalah batch size.

```
def _im2col(self, input_data, filter_h, filter_w,
stride):N, C, H, W = input_data.shape
        out_h = (H - filter_h) // stride + 1
        out_w = (W - filter_w) // stride + 1
```

Contoh code (Diimplementasikan pada setiap layer)

- Batch processing pada layer-layer utama:
  - Convolutional Layer: Menggunakan im2col untuk operasi konvolusi efisien pada batch
  - Pooling Layer: Menggunakan as_strided untuk operasi pooling pada batch
  - Flatten Layer: Mempertahankan dimensi batch saat melakukan reshape
  - Dense Layer: Melakukan operasi linear pada seluruh sampel dalam batch
- Contoh implementasi Inferensi dengan batching:



```
batch_size = 5000
num_batches = (num_test_samples + batch_size - 1) // batch_size
y_pred_proba_scratch = []

for i in range(num_batches):
    start_idx = i * batch_size
    end_idx = min((i + 1) * batch_size, num_test_samples)
    batch_output = cnn_model_scratch.forward(x_test_sample_scratch[start_idx:end_idx])
    y_pred_proba_scratch.append(batch_output)
```

Pada dataset CIFAR-10 dengan jumlah test data sebanyak 10.000, disarankan menggunakan batch dalam melakukan inference dengan model from scratch agar tidak terlalu membebani komputer yang digunakan, karena proses komputasinya cukup berat.

RNN

Pada implementasi *backward propagation from scratch*, setiap layer yang digunakan dalam model (Embedding, SimpleRNN, Bidirectional Wrapper, Dropout, dan Dense) kini memiliki fungsi backward. Fungsi ini bertanggung jawab untuk menghitung gradien error terhadap input, bobot, dan bias layer tersebut, berdasarkan gradien yang diterima dari layer berikutnya. Proses ini dimulai dari layer output, di mana gradien awal dihitung dari fungsi loss (dalam kasus ini, d_cross_entropy_softmax_np), dan kemudian disebarkan mundur melalui setiap layer dalam model SequentialFromScratch. Layer seperti Embedding dan RNN mengakumulasi gradien untuk bobotnya, yang nantinya dapat digunakan untuk proses pembaruan bobot, meskipun dalam skrip run_test_scratch pembaruan bobot tidak dilakukan dan hanya demonstrasi perhitungan gradien yang dijalankan jika flag --demo-backward aktif.

Untuk bonus *batch inference*, seluruh implementasi *forward propagation* pada layer-layer *from scratch* (EmbeddingLayerNP, SimpleRNNLayerNP, BidirectionalWrapperNP, DropoutLayerNP, DenseLayerNP) dirancang untuk dapat memproses input dalam bentuk batch. Ini berarti model dapat menerima dan mengolah beberapa instance data sekaligus dalam satu pemanggilan fungsi forward. Jumlah instance dalam satu batch dapat diatur melalui argumen --inference-batch-size saat menjalankan skrip main.py dengan perintah --test-scratch. Jika parameter ini disetel, fungsi run_test_scratch akan memproses data uji secara iteratif dalam batch-batch dengan ukuran yang ditentukan; jika tidak, seluruh data uji akan diproses sekaligus. Kemampuan ini memastikan bahwa baik model Keras maupun model *from scratch* dapat melakukan inferensi pada beberapa input secara efisien.

LSTM

Mengimplementasikan *forward propagation* pada LSTM dapat menangani kasus *batch inference*, yaitu model dapat memproses beberapa sequence input sekaligus dalam satu kali eksekusi. Hal ini penting untuk efisiensi komputasi terutama ketika menangani dataset besar atau proses inferensi real-time.

- Dimensi input:
  Input data memiliki shape (N, T, D)
  - N: *batch size* (jumlah *sequence* yang diproses bersamaan)
  - T: panjang *sequence* (*timesteps*)
  - D: dimensi fitur pada tiap *timestep*
- Contoh code (Diimplementasikan pada setiap bagian LSTM)
- Batch processing pada komponen-komponen utama:
  - *Input Gate*, *Forget Gate*, *Output Gate*: Operasi vektor dilakukan dalam bentuk batch menggunakan matrix multiplication (contohnya `np.dot(X, W)`)
  - *Cell State* dan *Hidden State*: Dimensi batch dipertahankan dan *update* dilakukan untuk seluruh batch secara paralel

- *Loop Time-step*: Perhitungan *gate* dilakukan per *timestep*, namun untuk seluruh batch sekaligus

KESIMPULAN & SARAN

CNN

Hasil dari eksperimen menunjukkan bahwa arsitektur CNN dengan tiga layer konvolusi, filter berukuran 5x5, dan penggunaan Max Pooling secara konsisten menghasilkan performa terbaik dengan F1-score 0.7350, sementara implementasi model from scratch berhasil divalidasi dengan kesesuaian 100% terhadap model Keras. Saran pengembangannya untuk mengembangkan model dengan mengkombinasikan konfigurasi optimal tersebut, sambil mengimplementasikan teknik regularisasi seperti Dropout atau L2, augmentasi data, dan mekanisme early stopping selama pelatihan untuk mengatasi potensi overfitting akibat peningkatan kompleksitas, hal ini memungkinkan akurasi yang lebih baik lagi.

RNN

Berdasarkan hasil analisis keseluruhan, dapat disimpulkan bahwa model RNN dengan konfigurasi yang lebih sederhana (1 layer, 32 unit) dan menggunakan arsitektur Bidirectional menunjukkan kinerja terbaik dalam kasus ini, di mana penambahan layer maupun unit cenderung menurunkan performa dan menyebabkan overfitting, sementara implementasi "from scratch" berhasil mereplikasi kinerja model Keras. Oleh karena itu, disarankan untuk fokus pada optimalisasi model dengan jumlah layer dan unit yang lebih sedikit, serta selalu mempertimbangkan penggunaan arsitektur Bidirectional untuk menangkap konteks data secara lebih baik, dan jika membangun model dari awal, pastikan validasi ketat terhadap implementasi library standar untuk memastikan kesesuaian kinerja.

LSTM

Berdasarkan hasil pengujian model LSTM selama 20 epoch, dapat disimpulkan bahwa konfigurasi arsitektur memiliki pengaruh signifikan terhadap performa model. Pada variasi jumlah layer LSTM, model dengan satu lapisan LSTM memberikan performa terbaik dengan F1 Score sebesar 0.696, loss terendah (1.0563), dan akurasi 65%. Penambahan jumlah layer justru menurunkan performa secara keseluruhan, di mana dua dan tiga layer menghasilkan F1 Score yang lebih rendah serta peningkatan nilai loss, yang mengindikasikan kemungkinan overfitting atau kesulitan optimasi pada arsitektur yang lebih dalam.

Pada variasi jumlah cell per layer, performa terbaik dicapai saat menggunakan 64 LSTM cell, dengan F1 Score sebesar 0.704 dan akurasi 70%. Menambah jumlah cell menjadi

128 dan 256 justru menyebabkan penurunan performa secara signifikan, yang kemungkinan disebabkan oleh kompleksitas model yang meningkat tanpa diimbangi dengan kapasitas data atau regularisasi yang memadai.

Sementara itu, pada pengujian jenis arah LSTM, model Bidirectional LSTM secara konsisten menunjukkan performa tertinggi, dengan F1 Score mencapai 0.721 dan akurasi 77%. Sebaliknya, model Unidirectional hanya mampu menghasilkan F1 Score sebesar 0.184 dan akurasi 38%, yang mengindikasikan bahwa pemrosesan konteks dari kedua arah sangat penting dalam tugas klasifikasi teks seperti ini.

Secara keseluruhan, konfigurasi optimal untuk model LSTM pada tugas ini adalah menggunakan 1 *layer* Bidirectional LSTM dengan 256 *cell*, yang memberikan keseimbangan terbaik antara kompleksitas model dan performa. Disarankan untuk menghindari penggunaan arsitektur yang terlalu dalam atau terlalu kompleks tanpa justifikasi yang kuat atau penyesuaian lain seperti regularisasi dan peningkatan data.

PEMBAGIAN TUGAS

NIM	Nama	Tugas
13522131	Owen Tobias Sinurat	Implementasi dan analisis LSTM
13522141	Ahmad Thoriq Saputra	Implementasi dan analisis RNN
13522143	Muhammad Fatihul Irhab	Implementasi dan analisis CNN

REFERENSI

- 10.1. Long Short-Term Memory (LSTM) — Dive into Deep Learning 1.0.3 documentation
- 10.3. Deep Recurrent Neural Networks
- 10.4. Bidirectional Recurrent Neural Networks — Dive into Deep Learning 1.0.3 documentation
- 9. Recurrent Neural Networks
- 7. Convolutional Neural Networks
- numpy.einsum — NumPy v2.1 Manual