LAPORAN TUGAS BESAR 1 IF3270

PEMBELAJARAN MESIN

Convolutional Neural Network dan Recurrent Neural Network

Kelompok 57

1. Eduardus Alvito Kristiadi 13522004
2. Rici Trisna Putra 13522026
3. Dimas Bagoes Hendrianto 13522112

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

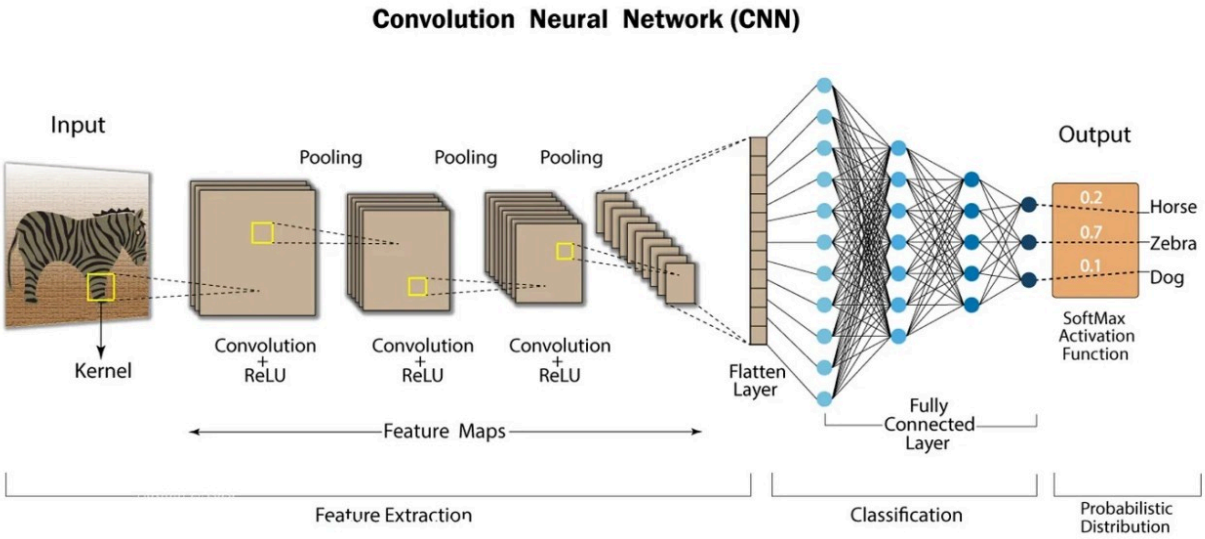
DAFTAR ISI 2
BAB 1 3
BAB 2 9
2.1. CNN 9
2.1.1. Fungsi init 9
2.1.2. Fungsi load_model_weights 9
2.1.3. Fungsi conv2d_forward 9
2.1.4. Fungsi max_pooling2d_forward 9
2.1.5. Fungsi average_pooling2d_forward 10
2.1.6. Fungsi global_average_pooling2d_forward 10
2.1.7. Fungsi dense_forward 10
2.1.8. Fungsi relu_activation 10
2.1.9. Fungsi softmax_activation 10
2.1.10. Fungsi forward 11
2.1.11. Fungsi predict 11
2.1.12. Fungsi get_model_summary 11
2.2. Simple RNN 11
2.2.1. Fungsi init 11
2.2.2. Fungsi load_model_weights 11
2.2.3. Fungsi embedding_forward 12
2.2.4. Fungsi simple_rnn_cell_forward 12
2.2.5. Fungsi simple_rnn_forward 12
2.2.6. Fungsi bidirectional_rnn_forward 12
2.2.7. Fungsi dense_forward 13
2.2.8. Fungsi softmax_activation 13
2.2.9. Fungsi forward 13
2.2.10. Fungsi predict 13
2.2.11. Fungsi get_model_summary 13
2.3. LSTM 14
2.3.1. Fungsi __init__ 14
2.3.2. Fungsi load_model_weights 14
2.3.3. Fungsi sigmoid 14
2.3.3. Fungsi tanh 14
2.3.4. Fungsi embedding_forward 15
2.3.5. Fungsi lstm_cell_forward 15
2.3.6. Fungsi lstm_forward 15
2.3.7. Fungsi bidirectional_lstm_forward 15
2.3.8. Fungsi dense_forward 16
2.3.9. Fungsi softmax_activation 16

2.3.10. Fungsi forward 16
2.3.11. Fungsi predict 16
2.3.12. Fungsi get_model_summary 17
BAB 3 17
3.1. CNN 17
3.1.1. Pengaruh jumlah layer konvolusi 18
3.1.2. Pengaruh banyak filter per layer konvolusi 18
3.1.3. Pengaruh ukuran filter per layer konvolusi 18
3.1.4. Pengaruh jenis pooling layer yang digunakan 20
3.1.5. Perbandingan Scratch dengan library Keras 20
3.2. Simple RNN 21
3.2.1. Pengaruh Jumlah Layer RNN 21
3.2.2. Pengaruh banyak cell RNN per layer 21
3.2.3. Pengaruh jenis layer RNN berdasarkan arah 22
3.2.4. Perbandingan Scratch dengan library Keras 22
3.3. LSTM 23
3.3.1. Pengaruh jumlah layer LSTM 23
3.3.2. Pengaruh banyak cell LSTM per layer 23
3.3.3. Pengaruh jenis layer LSTM berdasarkan arah 24
3.3.4. Perbandingan Scratch dengan library Keras 25
BAB 4 27
4.1 Kesimpulan 27
4.2 Saran 27
4.3 Refleksi 28
Lampiran 29

BAB 1

DESKRIPSI TUGAS

Convolutional Neural Network

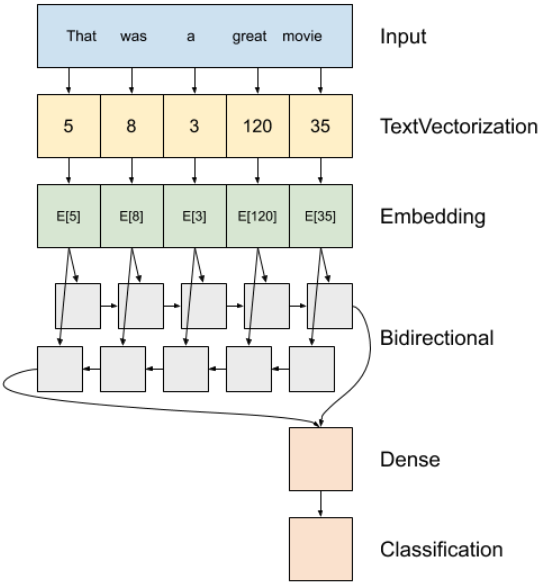


Untuk bagian CNN, Anda diminta untuk melakukan beberapa hal berikut:

- 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

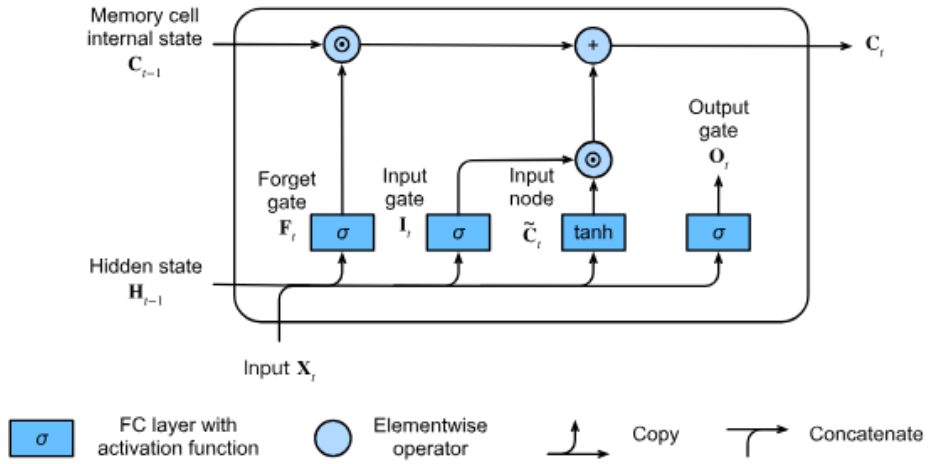


Untuk bagian ini, Anda diminta untuk melakukan beberapa hal berikut:

- 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.

BAB 2

IMPLEMENTASI

2.1. CNN

Implementasi forward propagation dari CNN kami buat dalam file cnn.py yang ada di dalam folder src/forward propagation custom. Di dalam file tersebut terdapat kelas CustomCNN yang memiliki berbagai fungsi dengan detail sebagai berikut:

2.1.1. Fungsi init

Fungsi **init** (wajib di python untuk konstruktor object) menginisialisasi objek dari kelas CustomCNN. Di dalamnya, diinisialisasikan berbagai atribut seperti layers, layer_configs, dan weights untuk menyimpan konfigurasi dan bobot model. Jika parameter model_path diberikan dan file model tersebut ada, maka fungsi ini akan memanggil load_model_weights untuk memuat bobot dari model Keras yang telah dilatih sebelumnya.

2.1.2. Fungsi load_model_weights

Fungsi ini memuat bobot dari model Keras yang telah disimpan. Setiap layer dari model Keras akan diekstrak, dan konfigurasi serta bobotnya disimpan dalam struktur internal seperti self.layer_configs dan self.weights. Layer yang didukung mencakup Conv2D dan Dense. Fungsi ini juga menangani pembagian bobot dengan indeks yang sesuai untuk setiap jenis layer, dimana Conv2D menggunakan indeks conv2d_0, conv2d_1, dan seterusnya, sedangkan Dense menggunakan indeks dense_0, dense_1, dan seterusnya.

2.1.3. Fungsi conv2d_forward

Fungsi ini mengimplementasikan operasi konvolusi 2D secara manual menggunakan NumPy. Fungsi ini menerima input data, kernel konvolusi, bias, stride, dan jenis padding sebagai parameter. Untuk padding 'same', fungsi menghitung padding yang diperlukan agar output memiliki dimensi yang sesuai. Operasi konvolusi dilakukan dengan sliding window approach dimana setiap posisi output dihitung dengan melakukan element-wise multiplication antara window input dengan kernel, kemudian dijumlahkan dan ditambahkan bias jika ada.

2.1.4. Fungsi max_pooling2d_forward

Fungsi ini mengimplementasikan operasi max pooling 2D yang mengurangi dimensi spasial dari feature map dengan mengambil nilai maksimum dari setiap region pooling. Fungsi menerima input data, ukuran pool, dan stride sebagai parameter. Untuk setiap window pooling, fungsi

mengambil nilai maksimum dari semua elemen dalam window tersebut. Operasi ini membantu mengurangi overfitting dan computational cost sambil mempertahankan informasi penting.

2.1.5. Fungsi average_pooling2d_forward

Fungsi ini mengimplementasikan operasi average pooling 2D yang serupa dengan max pooling, namun mengambil nilai rata-rata dari setiap region pooling alih-alih nilai maksimum. Fungsi ini berguna untuk mempertahankan informasi yang lebih halus dan mengurangi noise dalam feature map. Implementasinya menggunakan pendekatan yang sama dengan max pooling, namun menggunakan fungsi np.mean untuk menghitung rata-rata nilai dalam setiap window.

2.1.6. Fungsi global_average_pooling2d_forward

Fungsi ini mengimplementasikan global average pooling yang mengurangi setiap feature map menjadi satu nilai rata-rata. Operasi ini dilakukan dengan menghitung rata-rata dari semua pixel dalam setiap channel feature map, menghasilkan output dengan dimensi [batch_size, channels]. Fungsi ini sering digunakan sebagai alternatif flatten layer sebelum dense layer untuk mengurangi parameter dan mencegah overfitting.

2.1.7. Fungsi dense_forward

Fungsi ini mengimplementasikan layer Dense (fully-connected) yang melakukan operasi linear transformation. Input dikalikan dengan matriks kernel menggunakan operasi dot product, kemudian dijumlahkan dengan bias jika tersedia. Hasil akhirnya adalah output linier dari layer Dense sebelum fungsi aktivasi diaplikasikan. Fungsi ini menggunakan np.dot untuk melakukan matrix multiplication yang efisien.

2.1.8. Fungsi relu_activation

Fungsi ini mengimplementasikan fungsi aktivasi ReLU (Rectified Linear Unit) yang mengganti semua nilai negatif dengan nol sambil mempertahankan nilai positif. Implementasinya menggunakan np.maximum(0, input_data) yang secara element-wise membandingkan input dengan nol dan mengambil nilai yang lebih besar. ReLU banyak digunakan dalam CNN karena sederhana, efisien, dan membantu mengatasi masalah vanishing gradient.

2.1.9. Fungsi softmax_activation

Fungsi ini menghitung aktivasi softmax yang mengkonversi logits menjadi probabilitas untuk klasifikasi multi-class. Nilai input distabilkan dengan mengurangi nilai maksimum untuk menghindari overflow numerik, kemudian dihitung eksponensial dan dinormalisasi dengan membagi dengan jumlah total eksponensial. Hasilnya adalah distribusi probabilitas dimana semua nilai berada antara 0 dan 1 dan jumlahnya sama dengan 1.

2.1.10. Fungsi forward

Fungsi forward adalah fungsi utama untuk melakukan forward pass dari seluruh model CNN. Fungsi ini memproses input satu per satu sesuai urutan layer yang dimuat dalam self.layer_configs. Setiap jenis layer diproses dengan fungsi yang sesuai, mulai dari Conv2D dengan aktivasi ReLU, berbagai jenis pooling, flatten atau global average pooling, hingga Dense layer dengan aktivasi softmax untuk output. Layer Dropout diabaikan karena tidak relevan saat inference. Proses ini mencetak bentuk data (shape) di setiap layer untuk debugging dan monitoring.

2.1.11. Fungsi predict

Fungsi predict memproses input dalam batch untuk melakukan inference pada dataset yang lebih besar. Fungsi ini membagi input menjadi batch-batch kecil, menjalankan fungsi forward untuk setiap batch, kemudian menggabungkan hasil prediksi menjadi satu array besar menggunakan np.vstack. Pendekatan batch processing ini memungkinkan inference yang efisien pada dataset besar sambil mengelola penggunaan memori.

2.1.12. Fungsi get_model_summary

Fungsi ini mengembalikan ringkasan model yang telah dimuat dalam bentuk string yang mudah dibaca. Setiap layer ditampilkan bersama informasi pentingnya seperti jumlah filter, ukuran kernel, jenis aktivasi untuk Conv2D, ukuran pool untuk pooling layer, dan jumlah unit serta aktivasi untuk Dense layer.

2.2. Simple RNN

Implementasi forward propagation dari Simple RNN kami buat dalam file rnn.py yang ada di dalam folder src/forward propagation custom. Di dalam file tersebut terdapat kelas CustomRNN yang memiliki berbagai fungsi dengan detail sebagai berikut:

2.2.1. Fungsi init

Fungsi **init** (wajib di python untuk konstruktor object) menginisialisasi objek dari kelas CustomRNN. Di dalamnya, diinisialisasikan berbagai atribut seperti layer_configs, weights, dan konfigurasi default untuk vocab size, panjang maksimal input, serta dimensi embedding untuk pemrosesan data teks. Jika parameter model_path diberikan dan file model tersebut ada, maka fungsi ini akan memanggil load_model_weights untuk memuat bobot dari model Keras yang telah dilatih sebelumnya.

2.2.2. Fungsi load_model_weights

Fungsi ini memuat bobot dari model Keras yang telah disimpan untuk model RNN. Setiap layer dari model Keras akan diekstrak, dan konfigurasi serta bobotnya disimpan dalam struktur internal seperti self.layer_configs dan self.weights. Layer yang didukung mencakup Embedding, SimpleRNN, Bidirectional, dan Dense. Untuk layer Bidirectional, fungsi ini menangani pembagian bobot forward dan backward secara manual dengan membagi array weights menjadi dua bagian yang sama untuk masing-masing arah.

2.2.3. Fungsi embedding_forward

Fungsi ini mengimplementasikan layer embedding yang mengkonversi token ID menjadi representasi vektor dense. Untuk setiap token dalam setiap sequence pada batch, fungsi melakukan lookup pada matriks embedding untuk mendapatkan vektor yang sesuai. Hasilnya adalah tensor 3D dengan dimensi [batch_size, sequence_length, embedding_dim] yang siap diproses oleh layer RNN. Fungsi ini juga menangani token ID yang berada di luar range vocabulary dengan memastikan index berada dalam batas yang valid.

2.2.4. Fungsi simple_rnn_cell_forward

Fungsi ini mengimplementasikan perhitungan satu sel RNN pada satu langkah waktu (timestep). Fungsi menghitung transformasi input dan hidden state sebelumnya menggunakan kernel dan recurrent kernel, kemudian menambahkan bias jika tersedia. Formula yang diimplementasikan adalah h_t = tanh(W_ih * x_t + W_hh * h_{t-1} + b), dimana W_ih adalah kernel input-to-hidden, W_hh adalah recurrent kernel hidden-to-hidden, dan b adalah bias. Aktivasi tanh diterapkan untuk menghasilkan hidden state baru.

2.2.5. Fungsi simple_rnn_forward

Fungsi ini mengaplikasikan SimpleRNN secara sekuensial pada seluruh urutan input. Dengan iterasi untuk setiap timestep, fungsi memanggil simple_rnn_cell_forward dan memperbarui hidden state. Jika parameter return_sequences=True, semua hidden state dari setiap timestep disimpan dan dikembalikan. Jika False, hanya hidden state terakhir yang dikembalikan. Implementasi ini memungkinkan fleksibilitas dalam penggunaan output RNN untuk berbagai jenis tugas.

2.2.6. Fungsi bidirectional_rnn_forward

Fungsi ini mengimplementasikan RNN bidirectional dengan memproses sequence dalam dua arah: forward dan backward. Input diproses secara normal untuk arah forward menggunakan bobot forward. Untuk arah backward, input sequence dibalik terlebih dahulu sebelum diproses dengan bobot backward. Hasil dari kedua arah kemudian digabungkan (concatenate) untuk menghasilkan representasi yang lebih kaya yang menangkap konteks dari masa lalu dan masa depan dalam sequence.

2.2.7. Fungsi dense_forward

Fungsi ini mengimplementasikan layer Dense (fully-connected) yang identik dengan implementasi pada CNN. Input dikalikan dengan matriks kernel menggunakan operasi dot product dan dijumlahkan dengan bias jika ada. Hasil akhirnya adalah output linier dari layer Dense sebelum fungsi aktivasi diaplikasikan. Fungsi ini digunakan sebagai layer output untuk klasifikasi teks.

2.2.8. Fungsi softmax_activation

Fungsi ini menghitung aktivasi softmax untuk mengkonversi output Dense menjadi distribusi probabilitas untuk klasifikasi multi-class. Implementasinya identik dengan versi CNN, dimana nilai input distabilkan dengan mengurangi nilai maksimum untuk menghindari overflow, kemudian dihitung eksponensial dan dinormalisasi. Hasilnya adalah probabilitas untuk setiap kelas yang dapat digunakan untuk prediksi akhir.

2.2.9. Fungsi forward

Fungsi forward adalah fungsi utama untuk melakukan forward pass dari seluruh model RNN. Fungsi ini memproses input secara berurutan sesuai layer yang dimuat dalam self.layer_configs. Dimulai dari embedding layer untuk mengkonversi token menjadi vektor, kemudian SimpleRNN atau Bidirectional RNN untuk memproses sequence, dan diakhiri dengan Dense layer dengan aktivasi softmax untuk klasifikasi. Layer Dropout diabaikan karena tidak aktif saat inference. Fungsi ini mencetak bentuk data di setiap layer untuk monitoring dan debugging.

2.2.10. Fungsi predict

Fungsi predict memproses input sequences dalam batch untuk melakukan inference yang efisien. Fungsi membagi input menjadi batch-batch kecil sesuai parameter batch_size, menjalankan fungsi forward untuk setiap batch, kemudian menggabungkan hasil prediksi menggunakan np.vstack. Pendekatan ini memungkinkan pemrosesan dataset besar sambil mengelola penggunaan memori secara optimal.

2.2.11. Fungsi get_model_summary

Fungsi ini mengembalikan ringkasan model RNN yang telah dimuat dalam bentuk string yang terstruktur dan mudah dibaca. Setiap layer ditampilkan bersama informasi spesifiknya seperti ukuran vocabulary dan dimensi embedding untuk Embedding layer, jumlah unit dan return_sequences untuk RNN layer, serta jumlah unit dan aktivasi untuk Dense layer.

2.3. LSTM

Implementasi forward propagation dari LSTM kami buat dalam file lstm.py yang ada di dalam folder src/forward propagation custom. Di dalam file tersebut terdapat kelas CustomLSM yang memiliki berbagai fungsi dengan detail sebagai berikut:

2.3.1. Fungsi __init__

Fungsi __init__ (wajib di python untuk kontruktor object) menginisialisasi objek dari kelas CustomLSTM. Di dalamnya, diinisialisasikan berbagai atribut seperti layer_configs, weights, dan konfigurasi default untuk vocab size, panjang maksimal input, serta dimensi embedding. Jika parameter model_path diberikan dan file model tersebut ada, maka fungsi ini akan memanggil load_model_weights untuk memuat bobot dari model Keras yang telah dilatih sebelumnya.

2.3.2. Fungsi load_model_weights

Fungsi ini memuat bobot dari model Keras yang telah disimpan. Setiap layer dari model Keras akan diekstrak, dan konfigurasi serta bobotnya disimpan dalam struktur internal seperti self.layer_configs dan self.weights. Layer yang didukung mencakup Embedding, LSTM, Bidirectional, dan Dense. Fungsi ini juga menangani pembagian bobot forward dan backward secara manual untuk layer Bidirectional.

2.3.3. Fungsi sigmoid

Fungsi sigmoid merupakan implementasi fungsi aktivasi sigmoid, yang biasa digunakan dalam LSTM untuk mengontrol gate. Fungsi ini juga melakukan clipping terhadap nilai input agar menghindari overflow dalam komputasi eksponensial.

2.3.3. Fungsi tanh

Fungsi ini merupakan implementasi fungsi aktivasi tanh, yang digunakan dalam gate internal LSTM untuk mengatur nilai memori dan output. Sama seperti sigmoid, fungsi ini juga melakukan clipping untuk stabilitas numerik.

2.3.4. Fungsi embedding_forward

Fungsi ini mengambil input berupa ID token dan mengubahnya menjadi representasi vektor menggunakan matriks embedding. Untuk setiap token di dalam setiap sequence, vektor embedding-nya diambil dari matriks embedding dan disusun menjadi output berbentuk tensor 3D.

2.3.5. Fungsi lstm_cell_forward

Fungsi ini mengimplementasikan perhitungan satu sel LSTM pada satu langkah waktu (timestep). Fungsi ini menghitung transformasi input dan state sebelumnya dengan kernel dan recurrent kernel, serta membagi hasilnya menjadi gate input, forget, candidate cell, dan output. Setelah itu, fungsi ini memperbarui nilai sel dan hidden state sesuai rumus LSTM klasik dan mengembalikannya.

2.3.6. Fungsi lstm_forward

Fungsi ini mengaplikasikan LSTM secara sekuensial pada seluruh urutan input. Dengan mengulang untuk setiap timestep, fungsi ini memanggil lstm_cell_forward dan menyimpan hasil hidden state jika return_sequences=True. Jika tidak, hanya hidden state terakhir yang dikembalikan.

2.3.7. Fungsi bidirectional_lstm_forward

Fungsi ini mengimplementasikan LSTM bidirectional dengan dua arah: forward dan backward. Input diproses normal untuk arah forward, dan dibalik untuk arah backward. Hasil dari kedua arah kemudian digabung (concatenate) baik sepanjang fitur (jika return_sequences=False) maupun sepanjang waktu (jika True), sehingga menghasilkan representasi yang lebih kaya.

2.3.8. Fungsi dense_forward

Fungsi ini mengimplementasikan layer Dense (fully-connected). Input dikalikan dengan matriks kernel dan dijumlahkan dengan bias (jika ada). Hasil akhirnya adalah output linier dari layer Dense sebelum fungsi aktivasi diaplikasikan.

2.3.9. Fungsi softmax_activation

Fungsi ini menghitung aktivasi softmax, yang umum digunakan pada layer output untuk klasifikasi. Nilai input akan distabilkan dengan mengurangi nilai maksimum (untuk menghindari overflow), lalu eksponensial dan dibagi dengan jumlah total eksponensial untuk menghasilkan probabilitas.

2.3.10. Fungsi forward

Fungsi forward adalah fungsi utama untuk melakukan forward pass dari seluruh model. Fungsi ini memproses input satu per satu sesuai urutan layer yang dimuat dalam self.layer_configs. Setiap jenis layer diproses dengan fungsi yang sesuai, dan hasil dari layer sebelumnya menjadi input bagi layer berikutnya. Layer Dropout diabaikan karena tidak relevan saat inference. Proses ini mencetak bentuk data (shape) di setiap layer untuk debugging.

2.3.11. Fungsi predict

Fungsi predict memproses input dalam batch, menjalankan fungsi forward untuk setiap batch, lalu menggabungkan hasil prediksi menjadi satu array besar. Ini memungkinkan inference pada dataset besar secara efisien dengan menggunakan mini-batch.

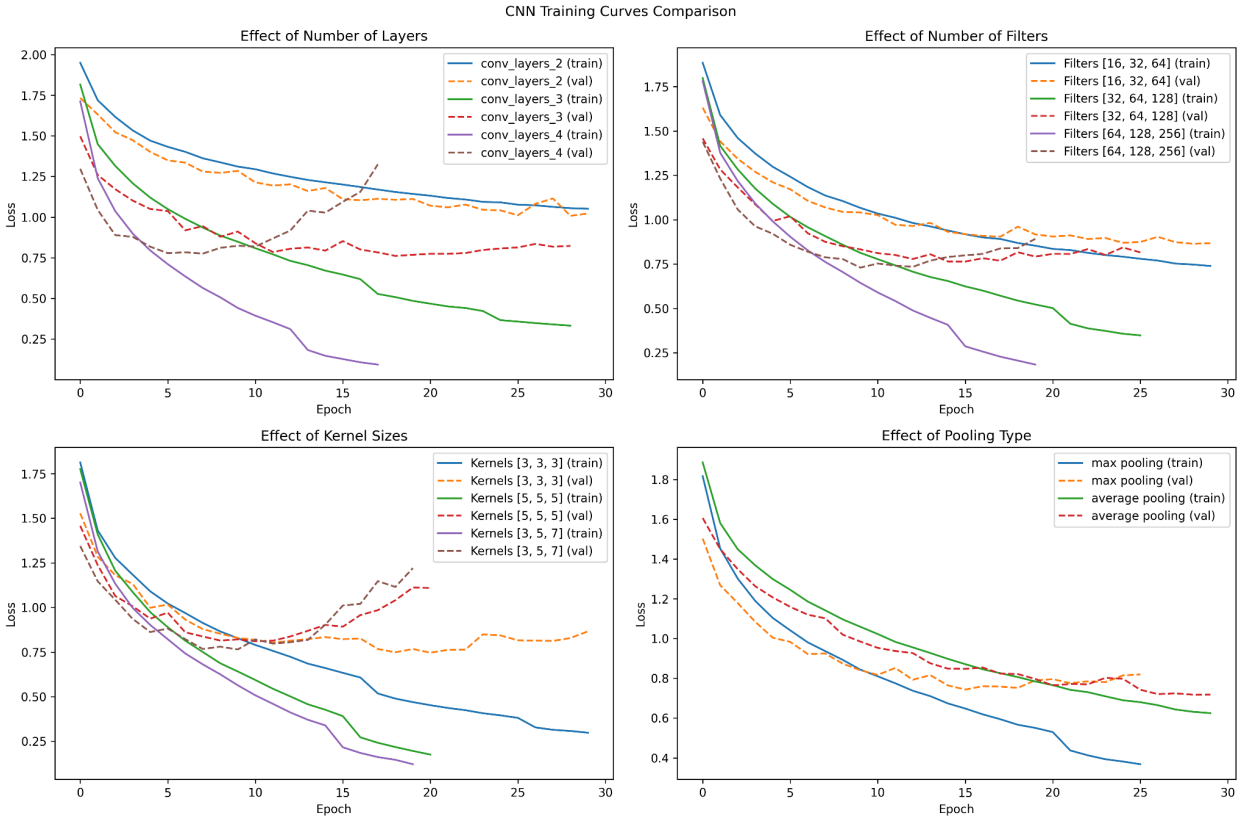
2.3.12. Fungsi get_model_summary

Fungsi ini mengembalikan ringkasan model yang telah dimuat dalam bentuk string. Setiap layer ditampilkan bersama informasi pentingnya seperti jumlah unit, jenis layer, dan apakah mengembalikan seluruh sequence. Ini sangat membantu dalam memahami arsitektur model sebelum digunakan.

BAB 3

PENGUJIAN

3.1. CNN



3.1.1. Pengaruh jumlah layer konvolusi

n_layer	f1_macro
2	0.6256903839817084
3	0.7397829836489478
4	0.741509692052773

Terlihat ada lompatan performa yang cukup besar ketika jumlah layer konvolusi ditingkatkan dari 2 menjadi 3. Nilai f1_macro naik sekitar 0.114. Ini mengindikasikan bahwa penambahan layer pada tahap ini memberikan kemampuan ekstraksi fitur yang jauh lebih baik bagi model

CNN. Meskipun masih ada peningkatan saat *layer* ditambah dari 3 menjadi 4, kenaikannya jauh lebih kecil, hanya sekitar 0.002.

3.1.2. Pengaruh banyak filter per layer konvolusi

n_filter_layer_1	n_filter_layer_2	n_filter_layer_3	f1_macro
16	32	64	0.696570298221488
32	64	128	0.7363116090010617
64	128	256	0.7430723949345223

Seiring dengan meningkatnya jumlah filter di setiap layer, skor f1_macro juga menunjukkan peningkatan. Dari Konfigurasi 1 ke Konfigurasi 2 (menggandakan jumlah filter), f1_macro meningkat dari sekitar 0.697 menjadi 0.736, sebuah kenaikan yang cukup noticeable (sekitar 0.039). Ini menunjukkan bahwa penambahan kapasitas representasi model melalui lebih banyak filter membantu model belajar fitur yang lebih baik. Dari Konfigurasi 2 ke Konfigurasi 3 (kembali menggandakan jumlah filter), f1_macro meningkat dari 0.736 menjadi 0.743, kenaikan yang lebih kecil (sekitar 0.007).

3.1.3. Pengaruh ukuran filter per layer konvolusi

size_filter_layer_1	size_filter_layer_2	size_filter_layer_3	f1_macro
3	3	3	0.696570298221488
5	5	5	0.7363116090010617
3	5	7	0.7430723949345223

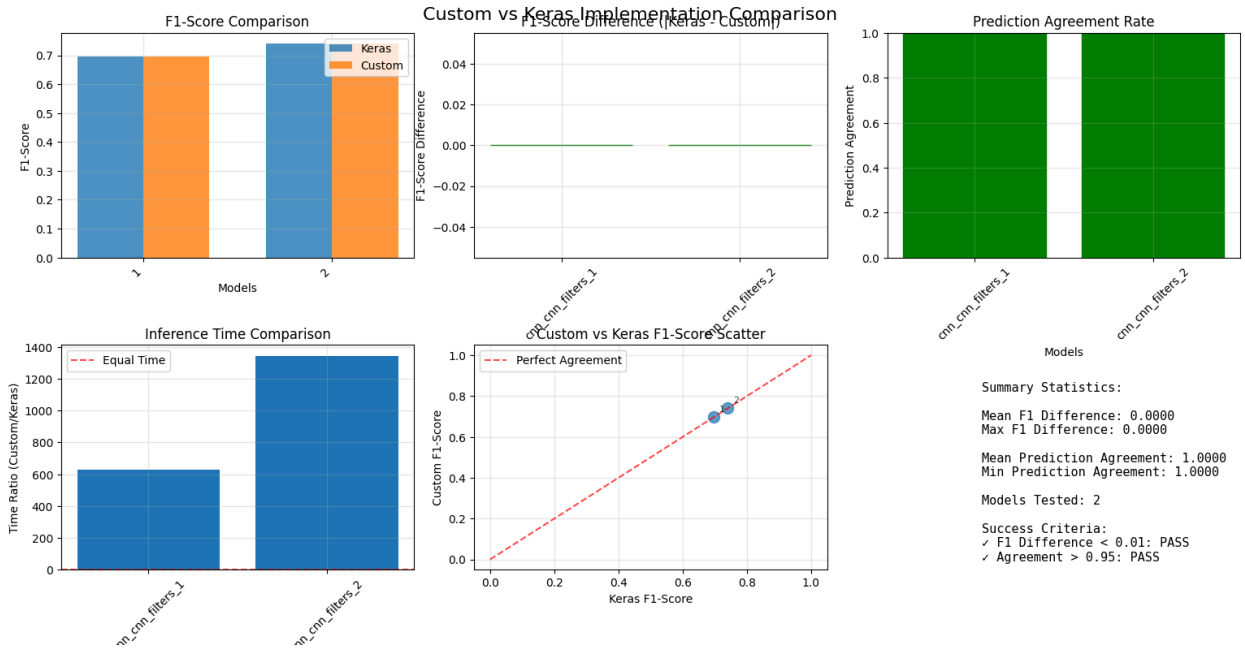
Ukuran filter yang lebih besar, yaitu 5, skor f1_macro meningkat menjadi sekitar 0,7363. Ini menunjukkan bahwa peningkatan ukuran filter secara seragam di semua lapisan dari 3 menjadi 5 meningkatkan kinerja model. Konfigurasi ketiga menghasilkan skor f1_macro tertinggi, yaitu sekitar 0,7431. Ini menunjukkan bahwa kombinasi ukuran filter yang bervariasi, berpotensi meningkat seiring kedalaman, mungkin lebih efektif daripada menggunakan ukuran filter yang seragam.

3.1.4. Pengaruh jenis pooling layer yang digunakan

pool_type	f1_macro
pooling max	0.7307859531772575
Pooling avg	0.7534377656055535

Dari data ini, terlihat bahwa Pooling avg menghasilkan skor f1_macro yang lebih tinggi (sekitar 0.7534) dibandingkan dengan pooling max (sekitar 0.7308). Ini menunjukkan bahwa dalam konteks eksperimen ini, penggunaan average pooling cenderung memberikan kinerja model yang lebih baik dibandingkan dengan max pooling berdasarkan metrik f1-macro.

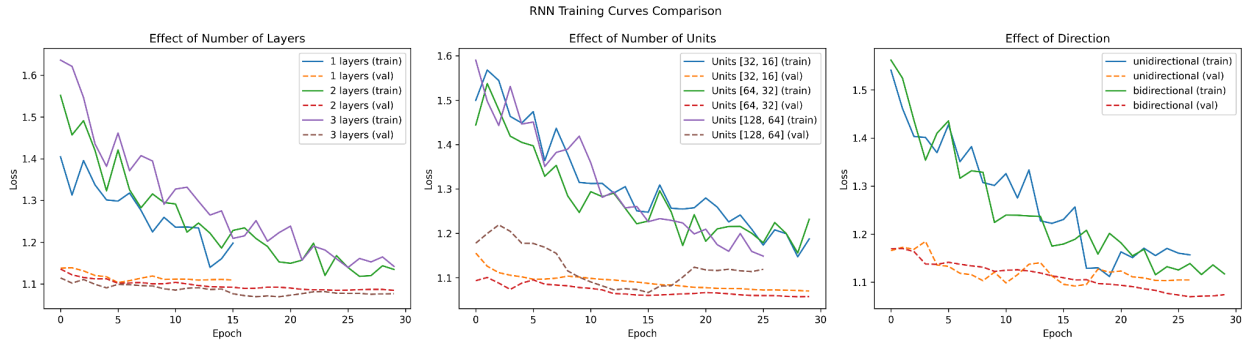
3.1.5. Perbandingan Scratch dengan library Keras



Secara keseluruhan,visualisasi ini menunjukkan bahwa implementasi "Scratch" berhasil mencapai tingkat kesepakatan prediksi yang hampir sempurna dan F1-Score yang sangat mirip dengan implementasi "Keras". Ini adalah pencapaian yang baik dalam hal ekuivalensi fungsional.

Namun, kelemahan besar dari implementasi "Scratch" adalah efisiensi waktu inferensinya. Implementasi "Scratch" lebih lambat dibandingkan dengan Keras. Ini menunjukkan bahwa meskipun secara fungsional setara, optimasi dan efisiensi yang ditawarkan oleh Keras yang mungkin dibangun di atas backend yang lebih cepat seperti TensorFlow atau PyTorch belum bisa ditandingi oleh implementasi Custom ini.

3.2. Simple RNN



3.2.1. Pengaruh Jumlah Layer RNN

n_layer	f1_macro
1	0.3824577022139577
2	0.30840372536278965
3	0.2719716416086053

Dari data ini, terlihat jelas bahwa semakin banyak jumlah lapisan RNN yang digunakan, skor f1_macro cenderung menurun. Ini menunjukkan bahwa dalam konteks eksperimen ini, menambahkan lebih banyak lapisan pada Simple RNN tidak meningkatkan kinerja model, bahkan justru menguranginya.

3.2.2. Pengaruh banyak cell RNN per layer

n_cell_layer_1	n_cell_layer_2	f1_macro
32	16	0.3048372804476487
64	32	0.2803921236141938
128	64	0.344050243761929

Peningkatan jumlah sel dari konfigurasi 1 (32, 16) ke konfigurasi 2 (64, 32) justru menurunkan skor f1-macro. Ini mungkin menunjukkan bahwa model menjadi terlalu kompleks atau mulai overfitting pada titik ini. Namun, peningkatan lebih lanjut ke konfigurasi 3 (128, 64) meningkatkan skor f1-macro secara substansial, bahkan menjadi yang tertinggi di antara ketiga konfigurasi tersebut. Ini menunjukkan bahwa untuk mencapai kinerja optimal, model mungkin membutuhkan kapasitas yang lebih besar (jumlah sel yang lebih banyak) untuk mempelajari

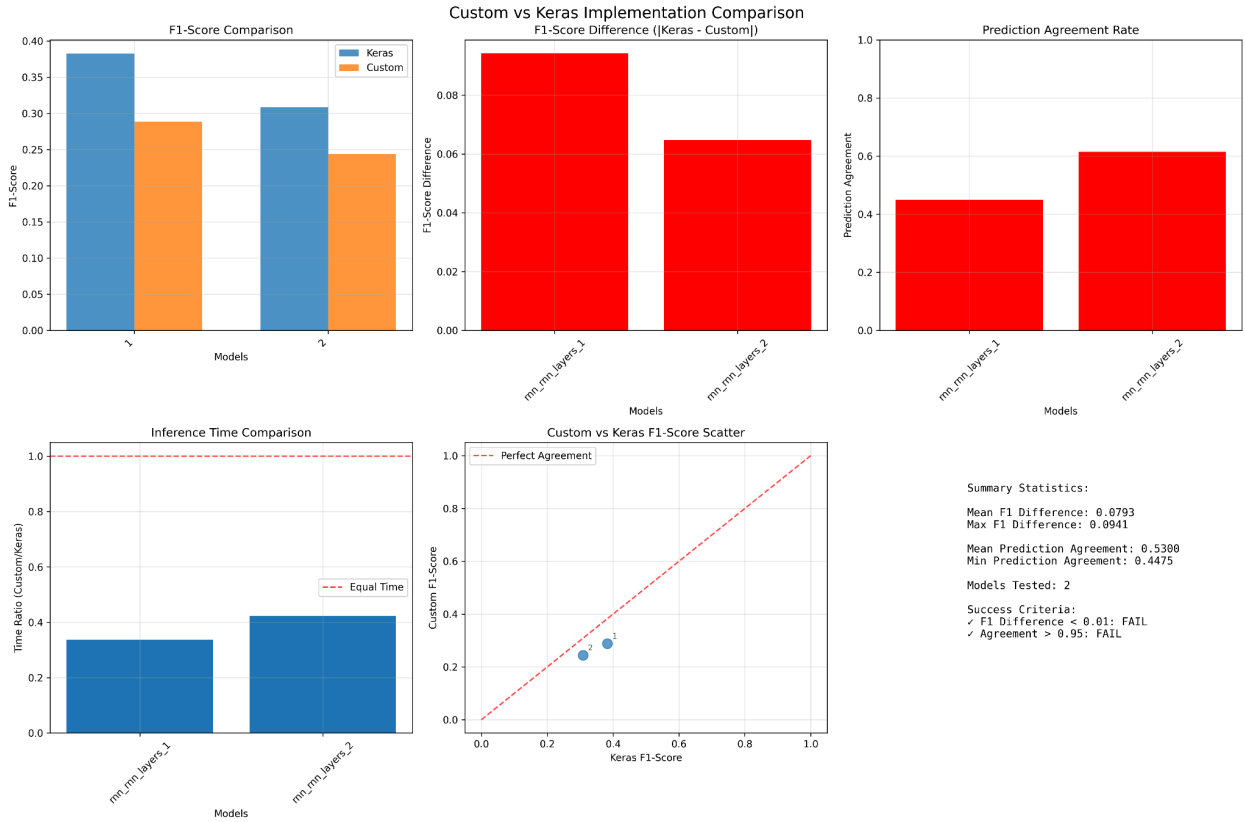
pola-pola dalam data, tetapi ada titik di mana penambahan kapasitas yang tidak tepat dapat memperburuk kinerja.

3.2.3. Pengaruh jenis layer RNN berdasarkan arah

pool_type	f1_macro
unidirectional	0.3824577022139577
bidirectional	0.30840372536278965

Dari data ini, terlihat bahwa penggunaan RNN searah (unidirectional) menghasilkan skor f1_macro yang lebih tinggi (sekitar 0.3825) dibandingkan dengan RNN dua arah (bidirectional) (sekitar 0.3084).Ini adalah hasil yang menarik, karena dalam banyak kasus, Bidirectional RNN seringkali diharapkan untuk berkinerja lebih baik karena dapat menangkap konteks dari masa lalu dan masa depan dalam urutan data.

3.2.4. Perbandingan Scratch dengan library Keras

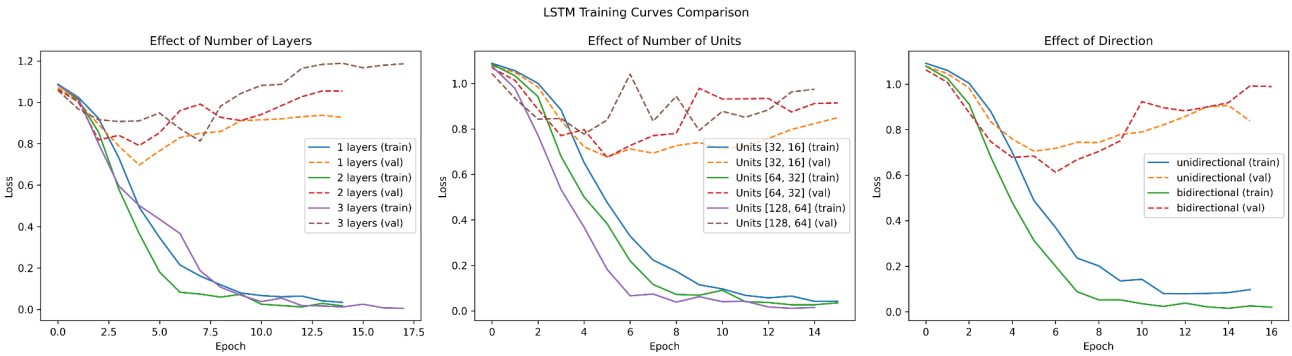


Visualisasi ini menunjukkan bahwa implementasi "Scratch" dalam skenario ini tidak berhasil menyamai kinerja implementasi "Keras" dalam hal akurasi (F1-Score)..

Meskipun implementasi "Scratch" secara signifikan lebih cepat dalam hal waktu inferensi dibandingkan Keras, keunggulan kecepatan ini datang dengan mengorbankan akurasi dan konsistensi prediksi yang cukup besar. Ini menunjukkan adanya trade-off yang jelas antara kecepatan dan akurasi/kesetaraan fungsional dalam implementasi "Scratch" ini dibandingkan dengan Keras.

Jadi, meskipun "Scratch" unggul dalam kecepatan, ia tidak dapat direkomendasikan sebagai pengganti Keras jika akurasi dan kesetaraan prediksi menjadi prioritas utama. Perlu ada pekerjaan lebih lanjut pada implementasi "Scratch" untuk meningkatkan F1-Score dan tingkat persetujuan prediksinya agar mendekati Keras, jika tujuannya adalah mencapai kinerja yang setara sekaligus lebih cepat.

3.3. LSTM



3.3.1. Pengaruh jumlah layer LSTM

n_layer	f1_macro
1	0.734143711899405
2	0.7171092796092796
3	0.7217897940871838

Dari data ini, terlihat bahwa penggunaan 1 lapisan LSTM menghasilkan skor f1_macro tertinggi di antara konfigurasi yang diuji. Menambahkan lebih banyak lapisan (2 atau 3) justru menyebabkan penurunan atau stagnasi kinerja (dibandingkan dengan 1 lapisan).

3.3.2. Pengaruh banyak cell LSTM per layer

n_cell_layer_1	n_cell_layer_2	f1_macro

32	16	0.710359659942919
64	32	0.7266367742356502
128	64	0.7030721966205838

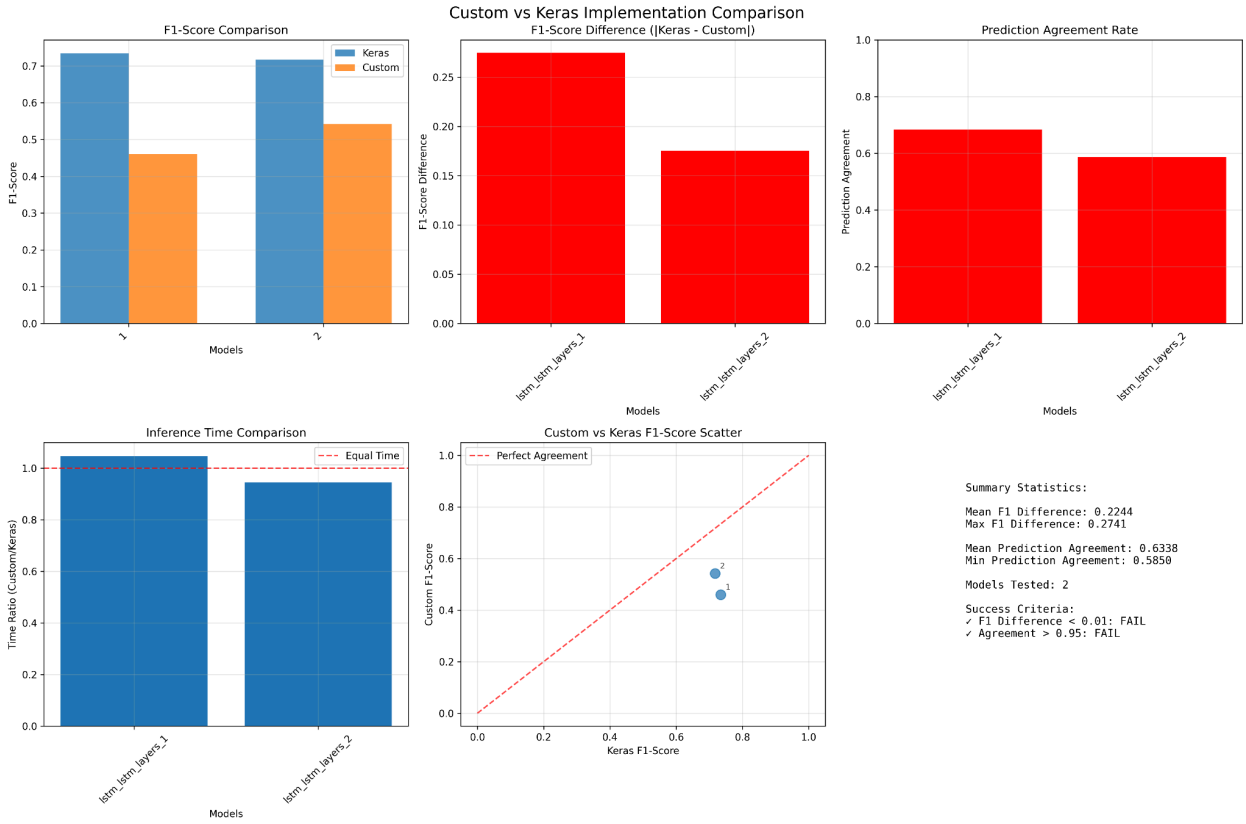
Dari data ini, terlihat bahwa penggunaan 64 sel di lapisan pertama dan 32 sel di lapisan kedua menghasilkan skor f1_macro tertinggi (sekitar 0.7266) di antara ketiga konfigurasi yang diuji. Peningkatan jumlah sel dari konfigurasi 1 (32, 16) ke konfigurasi 2 (64, 32) meningkatkan skor f1-macro, menunjukkan bahwa kapasitas model yang lebih besar di sini membantu menangkap pola yang lebih baik. Namun, peningkatan lebih lanjut ke konfigurasi 3 (128, 64) justru menurunkan skor f1-macro.

3.3.3. Pengaruh jenis layer LSTM berdasarkan arah

direction	f1_macro
unidirectional	0.7307859531772575
bidirectional	0.7504294788907595

Dari data ini, terlihat bahwa penggunaan LSTM dua arah (bidirectional) menghasilkan skor f1_macro yang lebih tinggi (sekitar 0.7504) dibandingkan dengan LSTM searah (unidirectional) (sekitar 0.7308). Ini adalah hasil yang lebih umum diharapkan, karena Bidirectional LSTM dapat menangkap konteks dari masa lalu dan masa depan dalam urutan data, yang seringkali sangat bermanfaat untuk tugas-tugas yang melibatkan pemahaman urutan (seperti pemrosesan bahasa alami atau deret waktu).

3.3.4. Perbandingan Scratch dengan library Keras



Visualisasi ini secara jelas menunjukkan bahwa implementasi "Scratch" dalam skenario ini kurang memenuhi kriteria keberhasilan dalam hal akurasi (F1-Score) dan kesetaraan prediksi dibandingkan dengan implementasi "Keras". Perbedaan F1-Score besar, dan tingkat persetujuan prediksi relatif rendah. Meskipun implementasi "Scratch" menunjukkan waktu inferensi yang sebanding dengan Keras (yang lebih baik daripada contoh RNN sebelumnya yang jauh lebih lambat), keunggulan ini tidak sepadan dengan penurunan performa akurasi yang signifikan.

Singkatnya, implementasi "Scratch" ini tidak sebanding atau lebih baik dari implementasi Keras untuk model LSTM yang diuji, karena kurang bisa memenuhi metrik kinerja inti (akurasi dan kesetaraan prediksi). Perlu ada revisi lebih pada implementasi Custom untuk meningkatkan akurasinya agar dapat bersaing dengan Keras.

BAB 4

PENUTUP

4.1 Kesimpulan

Berdasarkan serangkaian tabel dan data, kami telah menganalisis berbagai aspek pengaruh hiperparameter dan arsitektur model terhadap kinerja (terutama F1-macro score) serta membandingkan implementasi "Scratch" dengan "Keras". Dari analisis hyperparameter model (ukuran filter konvolusi, jenis pooling, jumlah dan sel layer RNN/LSTM), ditemukan bahwa pemilihan yang tepat sangat krusial. Misalnya, pada CNN, filter size 5x5 atau kombinasi 3-5-7 menghasilkan F1-macro tertinggi, dan average pooling mengungguli max pooling. Untuk Simple RNN dan LSTM, umumnya ditemukan bahwa jumlah lapisan yang lebih sedikit (1 lapisan) seringkali memberikan kinerja terbaik, dan terdapat titik optimal untuk jumlah sel per lapisan. Menariknya, pada Simple RNN, unidirectional mengalahkan bidirectional, sementara pada LSTM, bidirectional justru menunjukkan hasil yang lebih baik, menegaskan bahwa pilihan arsitektur terbaik sangat tergantung pada jenis model dan konteks tugas.

Selanjutnya, perbandingan implementasi "Scratch" dengan "Keras" mengungkapkan gambaran yang bervariasi. Pada kasus CNN, implementasi "Scratch" berhasil mencapai kesetaraan fungsional yang hampir sempurna dalam hal F1-score dan kesepakatan prediksi, namun dengan biaya waktu inferensi yang jauh lebih lambat (ratusan hingga ribuan kali lebih lama). Sebaliknya, pada kasus RNN dan LSTM, implementasi "Scratch" menunjukkan kecepatan inferensi yang lebih baik atau sebanding dengan Keras, namun sayangnya dengan penurunan signifikan pada F1-score dan tingkat persetujuan prediksi. Ini menunjukkan bahwa meskipun mungkin ada upaya untuk mengoptimalkan kecepatan pada implementasi "Scratch", masih ada tantangan besar dalam mencapai akurasi dan kesetaraan fungsional yang setara dengan kerangka kerja yang telah teroptimasi seperti Keras.

4.2 Saran

Untuk selanjutnya manajemen waktu harus dapat dijaga dengan baik karena pengerjaan masih serba dadakan sehingga kurang tereksekusi maksimal. Selain itu pembagian kerja juga harus lebih merata lagi karena beban kerja yang sekarang masih tidak seimbang sehingga tidak

tereksekusi secara efisien. Dalam hal komunikasi juga masih kurang optimal sehingga terdapat miskomunikasi yang menghambat pengerjaan. Penggunaan bahasa pemrograman serta kakas yang relatif baru untuk tugas besar kali ini juga menjadi tantangan tersendiri sehingga membuat kami berlatih lebih lagi.

4.3 Refleksi

Selama pengerjaan tugas besar ini dirasa masih kurang dalam hal manajemen waktu karena terdapat banyak tugas besar lain dan pengalokasian waktu untuk tugas besar ini masih kurang sehingga tidak terselesaikan dengan optimal. Selain itu pembagian beban kerja yang kurang seimbang dan terdapat pula masalah komunikasi dalam kelompok.

Lampiran

Github :