IF3270 Pembelajaran Mesin

Convolutional Neural Network dan Recurrent Neural Network

Tugas Besar 2

Disusun untuk memenuhi tugas mata kuliah Pembelajaran Mesin
pada Semester 2 (dua) Tahun Akademik 2024/2025
Tugas Besar IF3270 Pembelajaran Mesin

Oleh

Auralea Alvinia Syaikha	**13522148**
Muhammad Fauzan Azhim	**13522153**
Pradipta Rafa Mahesa	**13522162**

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2025

DAFTAR ISI

BAB 1 DESKRIPSI PERSOALAN

Convolutional Neural Network (CNN) dan Recurrent Neural Network (RNN) adalah dua arsitektur deep learning fundamental yang banyak digunakan dalam berbagai aplikasi machine learning, khususnya untuk pemrosesan citra dan teks. CNN sangat efektif dalam mengekstraksi fitur hirarkis dari data spasial seperti gambar, sementara RNN (termasuk varian LSTM) unggul dalam menangani data sekuensial dengan mengingat informasi dari langkah waktu sebelumnya. Dalam tugas besar ini, mahasiswa ditugaskan untuk mengimplementasikan modul forward propagation dari CNN, Simple RNN, dan LSTM secara mandiri (from scratch), tanpa bergantung pada library deep learning tingkat tinggi untuk komputasi inti. Implementasi ini harus mampu membaca bobot model yang telah dilatih menggunakan library Keras, dengan perhitungan matematika yang terbatas hanya pada library dasar seperti NumPy.

Modul yang dikembangkan harus diawali dengan pelatihan model CNN untuk klasifikasi citra menggunakan dataset CIFAR-10, yang harus dibagi menjadi training, validation, dan test set dengan rasio 4:1:1. Model CNN minimal harus mencakup layer Conv2D, Pooling layers, Flatten/Global Pooling, dan Dense, serta menggunakan loss function Sparse Categorical Crossentropy dan optimizer Adam. Demikian pula, untuk tugas klasifikasi teks, model Simple RNN dan LSTM akan dilatih menggunakan dataset NusaX-Sentiment (Bahasa Indonesia). Preprocessing data teks akan dilakukan dengan TextVectorization layer untuk tokenisasi dan Embedding layer dari Keras untuk embedding function. Model RNN dan LSTM minimal harus memiliki Embedding layer, Bidirectional/Unidirectional RNN/LSTM layer, Dropout layer, dan Dense layer, juga dengan Sparse Categorical Crossentropy dan Adam sebagai loss function dan optimizer.

Implementasi juga harus mencakup serangkaian eksperimen untuk menganalisis pengaruh hyperparameter tertentu. Untuk CNN, variasi akan dilakukan pada jumlah layer konvolusi, banyak filter per layer, ukuran filter per layer, dan jenis pooling layer. Sementara itu, untuk Simple RNN dan LSTM, analisis akan berfokus pada jumlah layer RNN/LSTM, banyak cell per layer, dan jenis layer berdasarkan arah (bidirectional atau unidirectional). Setiap variasi akan dievaluasi berdasarkan macro f1-score serta perbandingan grafik training loss dan validation loss. Hasil forward propagation from scratch akan divalidasi dengan membandingkannya terhadap hasil dari Keras pada test data, dengan macro f1-score sebagai metrik utama. Tugas ini tidak hanya menguji kemampuan implementasi teknis, tetapi juga analisis kritis terhadap kinerja model deep learning di bawah berbagai konfigurasi.

BAB 2 PEMBAHASAN

2.1 Penjelasan Implementasi

2.1.1 Deskripsi Kelas, Atribut, dan Method

Berikut adalah penjelasan dari kelas, atribut dari masing-masing kelas, dan method dari masing-masing kelas yang digunakan dalam implementasi model CNN, RNN, dan LSTM

Berikut daftar kelas yang digunakan untuk implementasi model CNN.

Class ActivationFunction (ActivationFunction.py)	
Kelas ini menyediakan implementasi statis untuk berbagai fungsi aktivasi yang digunakan dalam CNN (relu dan softmax). Metode-metodenya menerima input numerik (output dari layer sebelumnya) dan mengembalikan hasil setelah penerapan	
Fungsi	**Deskripsi**
relu	Mengembalikan nilai hasil operasi relu dari input, fungsi aktivasi ini digunakan untuk layer konvolusi dan dense layer.
softmax	Mengembalikan nilai hasil operasi softmax dari input, fungsi aktivasi ini digunakan untuk layer output.

Class Layer (layer.py)	
Kelas dasar (abstrak) yang mendefinisikan antarmuka umum untuk semua jenis layer dalam CNN.	
Fungsi	**Deskripsi**
__init__	Membuat sebuah instansi kelas layer tanpa input apa-apa
forward	Fungsi virtual untuk forward propagation dari sebuah layer yang nantinya diimplementasi pada anak dari kelas layer.

Class Convolutional (convolutional.py)	
Mewakili layer konvolusi dalam Convolutional Neural Network (CNN). Kelas ini menginisialisasi filter (kernel) dan bias, serta melakukan operasi konvolusi pada input selama propagasi maju. Jika bobot (kernels dan bias) tidak disediakan saat inisialisasi, kelas ini akan menginisialisasinya menggunakan metode Xavier/Glorot. Kelas ini juga menangani fungsi aktivasi jika ada.	
Fungsi	**Deskripsi**
__init__	Menginisialisasi Layer convolutional dengan parameter seperti , kernel size, bobot bias fungsi aktivasi (relu), dan bobot kernel
forward	Melakukan operasi konvolusi terhadap sebuah input matriks, berdasarkan kernel dan bias yang sudah di set saat inisialisasi. Fungsi ini mengembalikan hasil konvolusi dalam bentuk matriks juga.

Class MaxPooling2D (pool.py)	
Mewakili layer maxpooling dalam Convolutional Neural Network (CNN). Kelas ini menginisialisasi pool size, serta melakukan operasi pooling pada input selama propagasi maju.	
Fungsi	**Deskripsi**
__init__	Menginisialisasi Layer convolutional dengan parameter seperti , kernel size, bobot bias fungsi aktivasi (relu), dan bobot kernel
forward	Melakukan operasi max pooling terhadap sebuah input matriks. Fungsi ini bertujuan untuk memperkecil input dengan cara memilih nilai maksimum dari setiap jendela yang ditentukan oleh ukuran pool dan stride saat inisialisasi. Fungsi ini mengembalikan matriks hasil pooling dengan dimensi yang lebih kecil

Class AvgPooling2D (pool.py)	
Mewakili layer averagepooling dalam Convolutional Neural Network (CNN). Kelas ini menginisialisasi pool size, serta melakukan operasi pooling pada input selama propagasi maju dengan algoritma average (rata-rata)	
Fungsi	**Deskripsi**
__init__	Menginisialisasi Layer convolutional dengan parameter seperti , kernel size, bobot bias fungsi aktivasi (relu), dan bobot kernel
forward	Melakukan operasi average pooling terhadap sebuah input matriks . Fungsi ini bertujuan untuk memperkecil input dengan cara menghitung nilai rata-rata dari setiap jendela yang ditentukan oleh ukuran pool dan stride saat inisialisasi. Fungsi ini mengembalikan matriks hasil pooling dengan dimensi yang lebih kecil.

Class Dense (dense.py)	
Mewakili layer dense dalam Convolutional Neural Network (CNN). Kelas ini menginisialisasi bobot dari setiap neuron dan melakukan forward propagation dengan fungi forward.	
Fungsi	**Deskripsi**
__init__	Menginisialisasi Layer convolutional dengan parameter seperti, kernel size, bobot bias fungsi aktivasi (relu), dan bobot kernel
forward	Melakukan operasi dense terhadap sebuah input vektor. Fungsi ini mengalikan input dengan matriks bobot dan menambahkan vektor bias, yang telah diatur saat inisialisasi. Fungsi ini mengembalikan output dalam bentuk matriks dan kemudian dilewatkan ke fungsi aktivasi.

Class Flatten (flatten.py)	
Kelas yang mewakili layer flatten tidak memiliki atribut hanya fungsi forward untuk forward propagation.	
Fungsi	**Deskripsi**
__init__	Menginisialisasi Layer convolutional dengan parameter seperti , kernel size, bobot bias fungsi aktivasi (relu), dan bobot kernel
forward	Melakukan operasi flatten terhadap sebuah input matriks.

load.py	
File berisi fungsi untuk ngeload sebuah model keras ke dalam format model scratch	
Fungsi	**Deskripsi**
newConvLayer	Membuat convolution layer baru berdasarkan config layer input
newMaxPoolLayer	Membuat maxpool layer baru berdasarkan config layer input
newAvgPoolLayer	Membuat averagepool layer baru berdasarkan config layer input
newFlattenLayer	Membuat Flatten layer baru
newDenseLayer	Membuat dense layer baru berdasarkan config layer input
load_scratch_model	Ngeload model keras dari filepath lalu iterasi setiap layer pada konfig model kerasnya dan memanggil fungsi di atas yang sesuai.

Berikut daftar kelas yang digunakan untuk implementasi model RNN.

Class SimpleRNN (simpleRNN.py)	
Kelas ini dibuat untuk implementasi dasar Recurrent Neural Network	
Fungsi	**Deskripsi**
__init__	Berfungsi untuk menginisialisasi komponen-komponen dasar dari layer RNN. Constructor menerima beberapa parameter penting yaitu kernel (matriks bobot input), recurrent_kernel (matriks bobot hidden state), dan bias (nilai bias).
forward	Metode forward melakukan komputasi utama dari SimpleRNN. Metode ini menerima input berbentuk array 3D dengan dimensi (batch_size, timesteps, input_features). Proses komputasi dimulai dengan menginisialisasi hidden state awal dengan nilai nol. Kemudian untuk setiap timestep, dilakukan perhitungan kombinasi linear dari input saat ini dan hidden state sebelumnya menggunakan persamaan W_x * x_t + W_h * h_{t-1} + b. Kemudian, diterapkan fungsi aktivasi tanh pada hasil perhitungan.

Class Bidirectional (bidirectional.py)	
Kelas ini dibuat sebagai wrapper untuk bidirectional RNN	
Fungsi	**Deskripsi**
__init__	Berfungsi untuk menginisialisasi layer bidirectional RNN. Constructor ini menerima beberapa parameter yaitu forward_layer, backward_layer, dan input (untuk menyimpan input)
forward	Metode forward melakukan komputasi forward pass bidirectional yang akan menjalankan forward RNN pada input normal dan membalikan input untuk backward RNN, serta menjalankan backward RNN pada input yang dibalik.

Class Dense (dense.py)

Kelas ini dibuat untuk membuat dense layer	
Fungsi	**Deskripsi**
__init__	Constructor Dense layer merupakan turunan dari class Layer dasar. Constructor ini menerima parameter weights (matriks bobot), bias (nilai bias), dan activation (fungsi aktivasi, default="softmax")
forward	Metode forward melakukan komputasi utama dari Dense layer dengan melakukan komputasi linear standard dengan formula inputs @ weights + bias, kemudian menerapkan fungsi aktivasi sesuai parameter yang diberikan.

Class Dropout (dropout.py)	
Kelas ini dibuat sebagai regularisasi layer	
Fungsi	**Deskripsi**
__init__	Constructor ini menerima parameter rate yang menentukan proporsi neuron yang akan di-nonaktifkan selama training. Constructor menginisialisasi tipe layer sebagai "dropout", menyimpan rate dropout, dan menyiapkan variabel mask yang akan digunakan untuk mematikan neuron-neuron secara acak.
forward	Metode forward melakukan proses dropout dengan menghitung probabilitas keep (1- dropout_rate), membuat mask menggunakan distribusi binomial, melakukan scaling, dan mengaplikasikan mask ke input
forward_inference	Metode ini adalah helper method yang memanggil forward pass dalam mode inferensi (training=False). Ini memastikan bahwa dropout tidak aktif saat melakukan prediksi, yang merupakan perilaku standar dari dropout layer.

Class Layer (layer.py)	
Kelas ini dibuat sebagai base class untuk layer-layer	
Fungsi	**Deskripsi**

__init__	Berfungsi untuk menginisialisasi dua atribut yaitu input (menyimpan input yang diterima layer) dan output (menyimpan hasil komputasi layer)
forward	Berfungsi sebagai metode abstrak yang digunakan sebagai template untuk proses forward propagation

loadRNN (loadRNN.py)	
Fungsi-fungsi pada file ini digunakan untuk mengkonversi model Keras ke implementasi scrat	
Fungsi	**Deskripsi**
newEmbeddingLayer	Fungsi ini bertugas untuk membuat layer embedding baru dari layer Keras Embedding yang sudah ada.
newSimpleRNNLayer	Fungsi ini mengkonversi layer SimpleRNN dari Keras menjadi implementasi SimpleRNN dari scratch.
newBidirectionalLayer	Fungsi ini membuat layer bidirectional RNN dari implementasi scratch berdasarkan layer Keras Bidirectional.
newDropoutLayer	Fungsi ini membuat layer dropout dari implementasi scratch berdasarkan layer Keras Dropout.
newDenseLayer	Fungsi ini mengkonversi layer Dense dari Keras menjadi implementasi Dense dari scratch.
load_scratch_model	Fungsi ini yang mengkoordinasikan konversi seluruh model Keras menjadi implementasi scratch. Fungsi ini mengiterasi setiap layer dalam model Keras (kecuali TextVectorization), mengidentifikasi tipe layer, dan memanggil fungsi konversi yang sesuai. Hasil konversi setiap layer disimpan dalam list ScratchModel yang kemudian dikembalikan sebagai model lengkap dalam format implementasi scratch.

Class EmbeddingWrapper & Predict Method (predictRNN.py)
Kelas ini dibuat untuk menjembatani Keras Embedding layer dengan implementasi

scratch dan fungsi ‘predict’ dibuat untuk melakukan inferensi dengan model scratch	
Fungsi	**Deskripsi**
__init__	Menginisialisasi wrapper dengan menyimpan instance Keras Embedding layer
forward	Metode forward menghandle konversi input dari numpy array ke tensor TensorFlow, kemudian memastikan tipe data input adalah int32 yang diperlukan untuk embedding, dan mengembalikan hasil embedding dalam format numpy array
predict	Fungsi ini memiliki beberapa tahapan yaitu memvalidasi input, lalu melakukan batch processing, melakukan forward pass per batch, dan kemudian menggabungkan hasil dari semua batch.

Berikut daftar kelas yang digunakan untuk implementasi model LSTM.

Class LSTMCell (Layers.py)	
Implementasi dari satu buah sel LSTM yang menangani operasi dasar LSTM seperti gerbang input, forget, output, dan cell state.	
Fungsi	**Deskripsi**
__init__	Inisialisasi sel LSTM dengan parameter seperti units, activation, bias, dan dropout
_get_activation	Mengambil fungsi aktivasi berdasarkan string parameter (tanh, sigmoid, relu, linear)
_sigmoid	Implementasi fungsi sigmoid dengan clipping untuk mencegah overflow

Class LSTMLayer (Layers.py)	
Layer LSTM utama yang mendukung mode unidirectional dan bidirectional, dengan berbagai konfigurasi seperti return_sequences, return_state, dan go_backwards.	
Fungsi	**Deskripsi**
__init__	Inisialisasi layer LSTM dengan konfigurasi lengkap
_sigmoid	Implementasi fungsi sigmoid dengan clipping untuk mencegah overflow
set_weights	Mengatur bobot untuk kernel, recurrent kernel, dan bias
_clip_value	Melakukan clipping nilai berdasarkan threshold
call	Wrapper untuk fungsi forward
forward	Fungsi utama untuk forward pass, mengarahkan ke bidirectional atau unidirectional
_forward_unidirectional_lstm	Implementasi forward pass untuk LSTM satu arah
_forward_bidirectional_lstm	Implementasi forward pass untuk LSTM dua arah
units	Untuk mendapatkan jumlah unit LSTM

Class UnidirectionalLSTM (Layers.py)	
Wrapper untuk LSTMLayer yang secara eksplisit dikonfigurasi sebagai LSTM unidirectional.	
Fungsi	**Deskripsi**
__init__	Inisialisasi dengan bidirectional=False

Class BidirectionalLSTM (Layers.py)

Wrapper untuk LSTMLayer yang secara eksplisit dikonfigurasi sebagai LSTM bidirectional.	
Fungsi	**Deskripsi**
__init__	Inisialisasi dengan bidirectional=True

Class Embedding (Layers.py)	
Layer embedding yang mengkonversi token ID menjadi vektor dense dengan tambahan masking untuk padding token.	
Fungsi	**Deskripsi**
__init__	Inisialisasi dengan vocab_size, embedding_dim, dan mask_zero
set_weights	Mengatur weight matrix untuk embedding
forward	Melakukan lookup embedding dan menghasilkan mask jika diperlukan

Class Dense (Layers.py)	
Layer fully connected dengan dukungan berbagai fungsi aktivasi dan bias opsional.	
Fungsi	**Deskripsi**
__init__	Inisialisasi dengan units, activation, dan use_bias
set_weights	Mengatur weight matrix dan bias
apply_activation	Menerapkan fungsi aktivasi (relu, softmax, sigmoid, tanh, linear)
forward	Melakukan operasi linear transformation dan aktivasi

Class Dropout (Layers.py)	
Layer dropout untuk regularisasi yang secara acak mengatur beberapa input menjadi 0 selama training.	
Fungsi	**Deskripsi**
__init__	Inisialisasi dengan rate dropout dan seed
forward	Menerapkan dropout jika dalam mode training

Class LSTMModel (LSTMModel.py)	
Model LSTM lengkap yang menggabungkan embedding, multiple LSTM layers, dropout, dan dense layers untuk klasifikasi teks.	
Fungsi	**Deskripsi**
__init__	Inisialisasi model dengan konfigurasi termasuk vocab_size, embedding_dim, lstm_layers, dll
set_weights_from_keras	Mengatur bobot dari model Keras yang sudah dilatih
forward	Forward pass lengkap melalui semua layer
predict	Melakukan prediksi dengan batch processing

2.1.2 Forward Propagation

2.1.2.1 CNN

Proses propagasi maju diinisiasi melalui pemanggilan fungsi network.predict(model, inputs). Fungsi ini menerima model, yang direpresentasikan sebagai daftar objek layer kustom, beserta data input. Data input kemudian diproses dalam batch untuk meningkatkan efisiensi. Setiap batch data dilewatkan secara berurutan melalui setiap layer yang menyusun model.

Setiap objek layer dalam model memiliki metode forward(input) yang bertugas memproses input yang diterima dan menghasilkan output spesifik untuk layer tersebut. Berikut adalah mekanisme kerja metode forward untuk setiap jenis layer yang diimplementasikan:

1. Convolutional Layer:

Metode forward pada layer konvolusi menerima input berupa feature map dari layer sebelumnya, atau gambar asli jika merupakan layer pertama. Jika kernel dan bias belum diinisialisasi (misalnya, saat model dibangun dari awal tanpa bobot yang dimuat), layer akan melakukan inisialisasi terlebih dahulu. Bentuk kernel, terutama kedalaman inputnya, harus sesuai dengan kedalaman input yang diterima. Proses konvolusi kemudian berlangsung sebagai berikut: untuk setiap sampel dalam batch dan untuk setiap filter pada layer:

- Sebuah patch dari feature map input diambil, dengan ukuran yang sama dengan kernel.
- Operasi perkalian elemen-demi-elemen dilakukan antara patch dan kernel yang sedang aktif, diikuti dengan penjumlahan hasilnya untuk menghasilkan satu nilai.
- Proses ini diulang dengan menggeser patch melintasi seluruh feature map input, sehingga membentuk feature map baru untuk filter tersebut.
- Bias yang bersesuaian dengan filter ditambahkan ke feature map yang baru terbentuk. Setelah semua filter diaplikasikan, feature map yang dihasilkan dari setiap filter digabungkan. Jika fungsi aktivasi (misalnya, ReLU) telah ditentukan, fungsi tersebut diterapkan pada feature map gabungan. Output akhir dari layer konvolusi adalah feature map yang telah diproses ini, yang selanjutnya menjadi input untuk layer berikutnya.

2. MaxPooling2D / AvgPooling2D Layer:

Layer pooling, baik max pooling maupun average pooling, berfungsi untuk mereduksi dimensi spasial (tinggi dan lebar) dari feature map dengan tetap mempertahankan informasi esensial. Saat metode forward dieksekusi:

- Layer menerima feature map dari layer sebelumnya.
- Untuk setiap sampel dalam batch dan untuk setiap channel pada feature map:

    - Feature map dibagi menjadi beberapa patch yang tidak tumpang tindih, sesuai dengan pool_size yang telah ditentukan.
    - Pada MaxPooling2D, nilai maksimum dari setiap patch diambil. Pada AvgPooling2D, nilai rata-rata dari setiap patch yang diambil.
- Hasil operasi pooling ini membentuk feature map baru dengan dimensi spasial yang lebih kecil, namun dengan jumlah channel yang sama dengan input. Output ini kemudian diteruskan ke layer berikutnya.

3. Flatten Layer:

Layer Flatten berfungsi untuk mengubah feature map multi-dimensi menjadi vektor satu dimensi. Ini merupakan langkah transisi yang krusial, terutama dari layer konvolusional atau pooling menuju layer dense. Saat metode forward dieksekusi:

- Layer menerima feature map multi-dimensi (umumnya dengan bentuk batch_size, height, width, channels).
- Feature map ini kemudian diubah bentuknya (diratakan) menjadi vektor satu dimensi untuk setiap sampel dalam batch, menghasilkan output dengan bentuk batch_size, height * width * channels. Vektor hasil perataan ini diteruskan ke layer berikutnya, yang umumnya adalah layer Dense.

4. Dense (Fully Connected) Layer:

Layer Dense melakukan transformasi linear terhadap input yang diterimanya. Saat metode forward dieksekusi:

- Layer menerima input, yang umumnya berupa vektor hasil perataan dari layer Flatten atau output dari layer dense sebelumnya.
- Operasi perkalian matriks dilakukan antara input dan matriks bobot (weights) milik layer dense.
- Vektor bias kemudian ditambahkan pada hasil perkalian matriks tersebut.
- Jika fungsi aktivasi telah ditentukan (misalnya, ReLU atau Softmax untuk layer output), fungsi tersebut diterapkan pada hasil kalkulasi. Output dari layer dense ini diteruskan ke layer dense berikutnya, atau menjadi prediksi akhir model jika merupakan layer terakhir.
- Setelah satu batch data selesai diproses melalui seluruh layer dalam model, output dari batch tersebut disimpan. Proses ini diulang untuk semua batch data input yang tersedia. Akhirnya, fungsi network.predict menggabungkan output dari semua batch untuk menghasilkan prediksi keseluruhan untuk seluruh dataset input.

Dengan demikian, propagasi maju dalam implementasi ini adalah sebuah proses bertahap di mana data mengalir melalui jaringan. Setiap layer melakukan transformasi spesifik terhadap data berdasarkan bobot dan konfigurasi yang dimilikinya, hingga akhirnya menghasilkan prediksi akhir.

2.1.2.2 RNN

Forward pass untuk semua layer dipanggil melalui fungsi predict() pada file predictRNN. Fungsi ini mengimplementasikan pipeline lengkap untuk inferensi model RNN dimulai dari memproses input teks menjadi format yang bisa diproses model, kemudian menjalankan forward pass melalui semua layer secara berurutan sebagai berikut.

1. Embedding Layer

   Forward pass pada EmbeddingWrapper mengkonversi input indeks (integers) menjadi dense vectors. Layer ini mengkonversi input numpy array ke tensor TensorFlow, melakukan lookup pada embedding matrix, dan mengembalikan hasil dalam format numpy array. Layer ini berfungsi sebagai jembatan antara implementasi scratch dan Keras Embedding layer.

2. Bidirectional layer

   Forward pass pada Bidirectional layer memproses input sequence dalam dua arah. Layer ini menjalankan dua SimpleRNN secara paralel: satu memproses sequence dari awal ke akhir (forward), dan satu lagi dari akhir ke awal (backward). Output dari kedua RNN kemudian digabungkan (concatenated) pada dimensi fitur untuk menghasilkan representasi yang menangkap konteks dari kedua arah.

3. Dropout Layer

   Forward pass pada Dropout layer bersifat kondisional berdasarkan mode training. Saat training, layer secara acak "mematikan" (set ke 0) sejumlah neuron sesuai rate yang ditentukan, dan melakukan scaling pada neuron yang tersisa untuk mempertahankan magnitude output. Saat inferensi (training=False), layer melewatkan input tanpa modifikasi.

4. Dense Layer

   Forward pass pada Dense layer melakukan transformasi linear pada input menggunakan matriks bobot dan bias, diikuti dengan penerapan fungsi aktivasi. Layer ini mendukung beberapa fungsi aktivasi:

- Softmax: untuk output probabilistik (normalisasi eksponensial)
- ReLU: mengubah nilai negatif menjadi 0
- Tanh: untuk normalisasi nilai ke range [-1, 1]
- Linear: tanpa fungsi aktivasi

2.1.2.3 LSTM

Model dibuat menggunakan config dengan parameter *embedding_dim, lstm_layers, bidirectional, dropout_rate, num_classes*.

1. Embedding Layer

   Forward pass pada Embedding Layer mengkonversi input indeks (integers) menjadi dense vectors. Layer ini mengkonversi input numpy array ke tensor TensorFlow, melakukan lookup pada embedding matrix, dan mengembalikan hasil dalam format numpy array.

2. Unidirectional / Bidirectional layer

   Forward pass pada Unidirectional / Bidirectional layer memproses input sequence. Layer menerima 3 / 6 weight array, setelah itu layer akan melakukan forward pass tergantung arahnya. Menghitung gate value, mengupdate hidden state dan cell state.

3. Dropout Layer

   Dropout layer tidak digunakan saat tidak melakukan training. Namun saat training layer akan melakukan randomisasi untuk mencari probabilitas suatu neuron akan dimatikan.

4. Dense Layer

   Forward pass pada Dense layer melakukan aktivasi fungsi dengan weight, bias dan input. Fungsi-fungsi aktivasi yang digunakan antara lain adalah:

   - Relu
   - Softmax
   - Sigmoid
   - Tanh
   - Linear

2.2 Hasil Pengujian

2.2.1 CNN

Kami melakukan empat buah pengujian terhadap model CNN dari library tensorflow keras. Pengujian terdiri dari test pengaruh jumlah layer konvolusi, pengaruh banyak filter per layer konvolusi, pengaruh ukuran filter per layer konvolusi, dan jenis pooling layer. Pengujian dilakukan dengan menggunakan dataset CIFAR-10 dengan pembagian dataset berupa 80% training set, 20% validation set (Ratio 4:1).

2.2.1.1 Pengaruh jumlah layer konvolusi

Pengujian ini dilakukan dengan membuat 3 variasi model CNN berdasarkan jumlah layer konvolusi yang digunakan dalam model tersebut.

Base Model

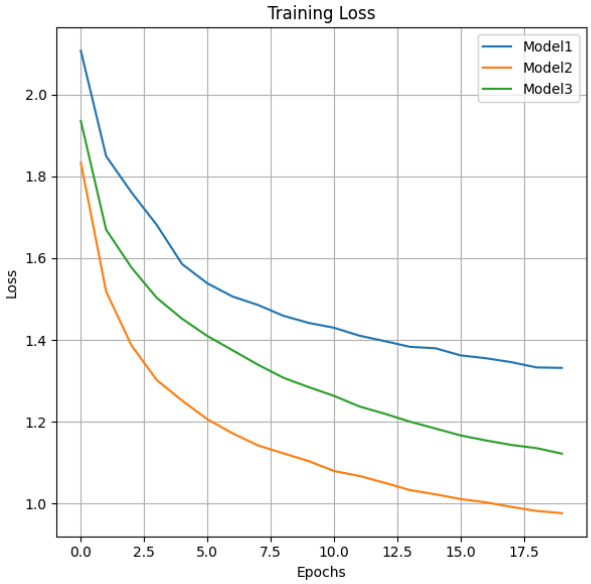
```
model1 = models.Sequential()
model1.add(layers.Conv2D(16, (3, 3), activation='relu'))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Flatten())
model1.add(layers.Dense(16, activation='relu'))
model1.add(layers.Dense(10, activation='softmax'))  # CIFAR-10 → 10 kelas

model1.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

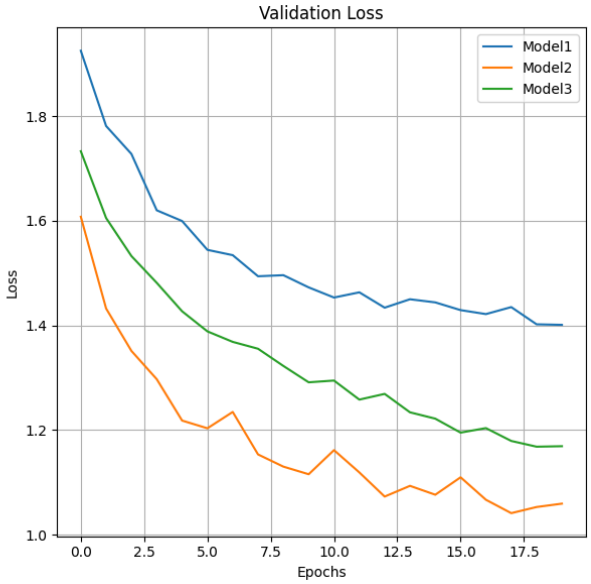
Train

```
def train_model(model):
    history = model.fit(
        x_train, y_train,
        epochs=20,
        validation_data=(x_val, y_val),
        batch_size=64,
        verbose = 1
    )
    return history
```

0.0s

No	Parameter	Accuracy (Macro F1)
1	1 Layer Konvolusi	0.4702511451885433
2	2 Layer Konvolusi	0.6150259309252166
3	3 Layer Konvolusi	0.5805679943317058





Analisis

Dari hasil dapat terlihat bahwa jumlah layer konvolusi model itu memiliki sebuah sweet spot yang tidak terlalu kecil dan terlalu besar. Hal tersebut dapat terlihat dari tingkat berkurangnya training dan validation loss model 2 pada awal iterasi yang jauh lebih bagus daripada model lainnya. Hal tersebut konsisten hingga akhir iterasi walaupun sedikit fluktuatif, model 2 tetap memiliki tingkat akurasi yang terbaik antara model lainnya.

2.2.1.2 Pengaruh banyak filter per layer konvolusi

Pengujian ini dilakukan dengan membuat 3 variasi model CNN berdasarkan banyak filter yang digunakan per layer konvolusi dalam model tersebut.

Base Model

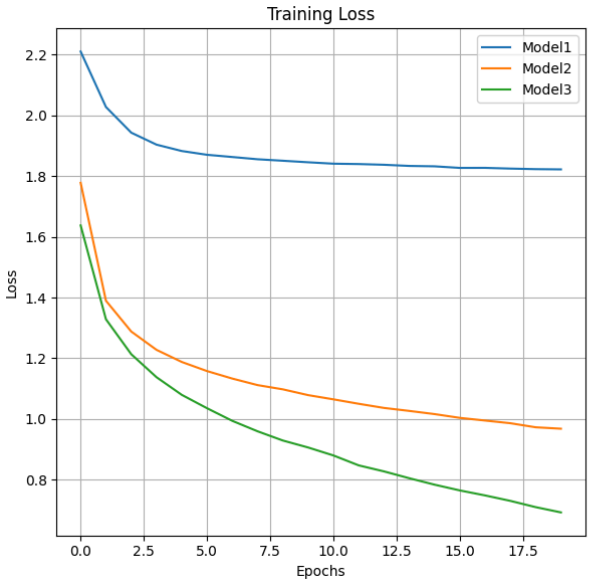
```
model1 = models.Sequential()
model1.add(layers.Conv2D(8, (3, 3), activation='relu'))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Flatten())
model1.add(layers.Dense(8, activation='relu'))
model1.add(layers.Dense(10, activation='softmax'))  # CIFAR-10 → 10 kelas

model1.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

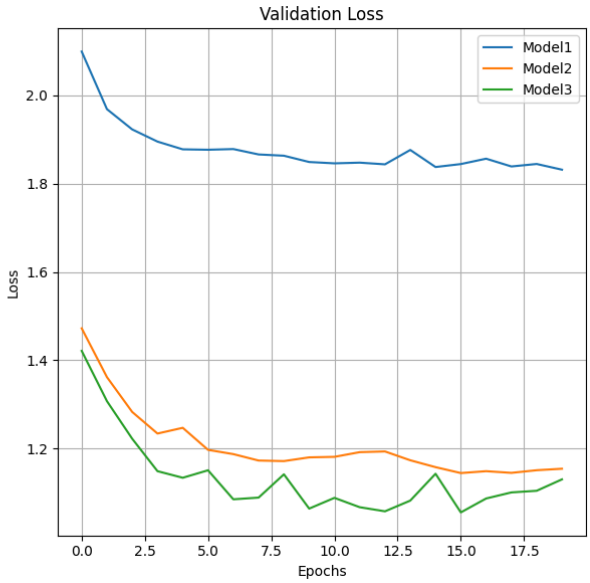
Train

```
def train_model(model):
    history = model.fit(
        x_train, y_train,
        epochs=20,
        validation_data=(x_val, y_val),
        batch_size=64,
        verbose = 1
    )
    return history
```

No	Parameter	Accuracy (Macro F1)
1	8 Filter	0.16218010597130733
2	16 Filter	0.5933649613319342

3	**32 Filter**	**0.6214408358559074**





Analisis

Dari hasil dapat terlihat bahwa semakin banyak filter model akan menjadi lebih akurat dalam iterasi yang lebih sedikit. Dapat terlihat adanya jarak signifikan dalam akurasi dari 8 filter ke 16 dan 32, hal tersebut mengindikasi semakin banyak variabel bobot yang dimiliki suatu model maka peak dari model tersebut dapat meningkat.Namun terlalu banyak juga belum tentu lebih bagus, hal tersebut dapat dilihat pada validation loss dari model 3 yang terlihat lebih fluktuatif setelah iterasi 2.5 dibandingkan model lain, dan pada akhir iterasi tidak beda jauh dari validation loss model 2. Hal tersebut mengindikasi bahwa setelah jumlah iterasi tertentu, model dengan jumlah filter tinggi menjadi sangat mudah untuk terpengaruhi ke 1 arah tertentu dan dapat terjadi overfitting.

2.2.1.3 Pengaruh ukuran filter per layer konvolusi

Pengujian ini dilakukan dengan membuat 3 variasi model CNN berdasarkan ukuran filter per layer konvolusi yang digunakan dalam model tersebut.

Base Model

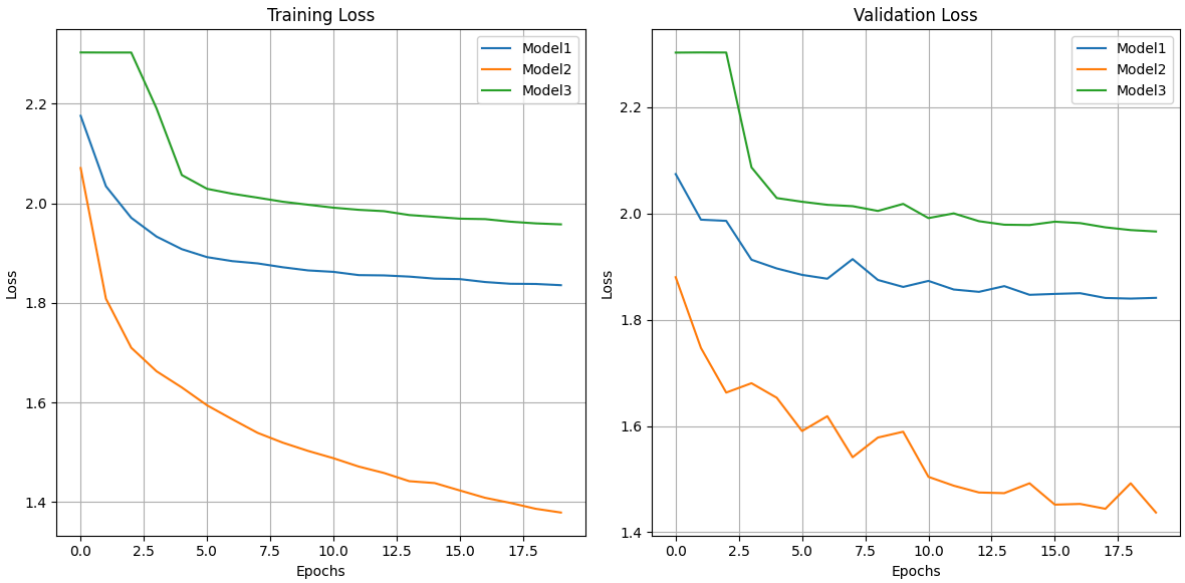
```
model1 = models.Sequential()
model1.add(layers.Conv2D(8, (3, 3), activation='relu'))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Flatten())
model1.add(layers.Dense(8, activation='relu'))
model1.add(layers.Dense(10, activation='softmax'))  # CIFAR-10 → 10 kelas

model1.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

Train

```
def train_model(model):
    history = model.fit(
        x_train, y_train,
        epochs=20,
        validation_data=(x_val, y_val),
        batch_size=64,
        verbose = 1
    )
    return history
```

0.0s

No	Parameter	Accuracy (Macro F1)
1	Ukuran Filter 3x3	0.14648750545247657
2	Ukuran Filter 9x9	0.47086888925546366
3	Ukuran Filter 27x27	0.23728632309578906



Analisis

Dari hasil dapat terlihat bahwa ukuran filter model itu memiliki sebuah sweet spot yang tidak terlalu kecil dan terlalu besar. Hal tersebut dapat terlihat dari tingkat berkurangnya training dan validation loss model 2 pada awal iterasi yang jauh lebih bagus daripada model lainnya. Hal tersebut konsisten hingga akhir iterasi walaupun sedikit fluktuatif, model 2 tetap memiliki tingkat akurasi yang terbaik antara model lainnya.

2.2.1.4 Pengaruh jenis pooling layer

Pengujian ini dilakukan dengan membuat 3 variasi model CNN berdasarkan jenis pooling yang digunakan dalam model tersebut.

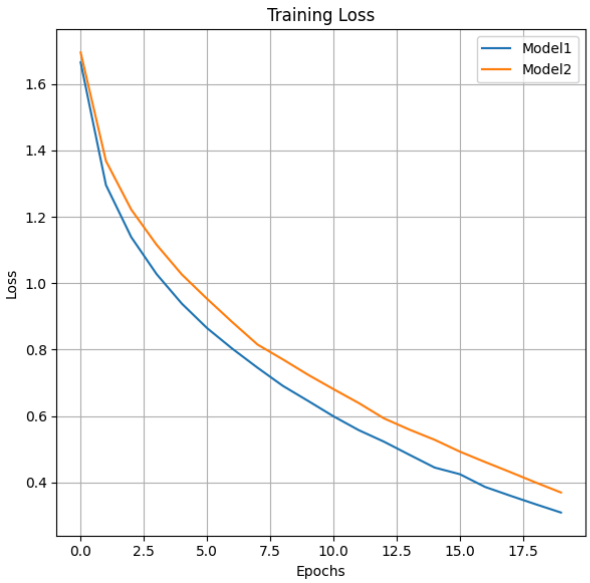
Base Model

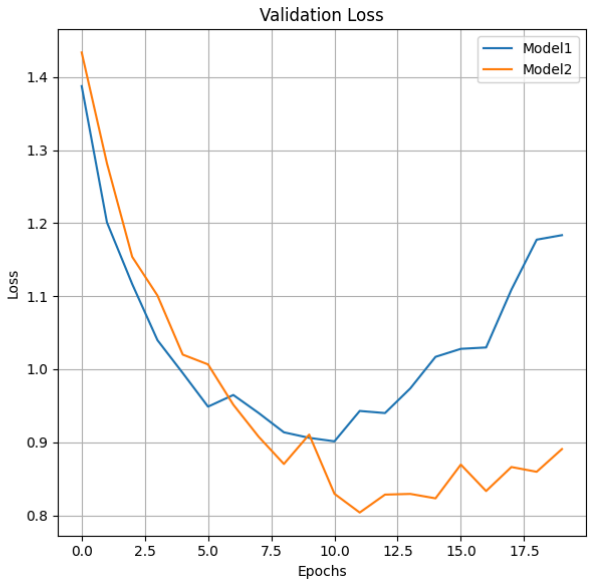
```
model1 = models.Sequential()
model1.add(layers.Conv2D(32, (3, 3), activation='relu'))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Conv2D(64, (3, 3), activation='relu'))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Conv2D(128, (3, 3), activation='relu'))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Flatten())
model1.add(layers.Dense(128, activation='relu'))
model1.add(layers.Dense(64, activation='relu'))
model1.add(layers.Dense(10, activation='softmax'))

model1.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

Train

```
def train_model(model):
    history = model.fit(
        x_train, y_train,
        epochs=20,
        validation_data=(x_val, y_val),
        batch_size=64,
        verbose = 1
    )
    return history
```

No	Parameter	Accuracy (Macro F1)
1	Max Pooling	0.6911504232913771
2	Average Pooling	0.7295960722920937





Analisis

Dari hasil dapat terlihat bahwa average pool memiliki hasil akhir akurasi yang lebih baik dibandingkan maxpooling namun tidak begitu drastis. Walaupun saat dilihat dari graf training loss model 1 dengan maxpooling itu lebih baik dibandingkan averagepooling model 2, saat melihat ke graf validation loss dapat terlihat bahwa keunggulan maxpooling hanya sampai hingga iterasi 5-6 dan setelahnya model 1 mengalami overfitting. Hal tersebut mengindikasikan bahwa maxpooling sebenarnya lebih baik dari pada averagepooling tetapi dalam jumlah iterasi sedikit, apabila jumlah iterasi meningkat maxpooling akan menjadi lebih rentan terhadap overfitting. Dapat disimpulkan juga bahwa averagepooling itu lebih kebal overfitting dibandingkan maxpooling.

2.2.1.4 Hasil Pengujian Scratch dan Model Keras

Pengujian ini dilakukan dengan membuat 1 buah model CNN menggunakan library tensorflow.keras yang setelah itu di-train dan kemudian bobotnya akan digunakan juga untuk model NN from scratch. Keduanya akan dibandingkan hasil prediksinya.

Base Model

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))  # CIFAR-10 → 10 kelas

model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

Train

```
history = model.fit(
    x_train, y_train,
    epochs=10,
    validation_data=(x_val, y_val),
    batch_size=64,
    verbose = 1
)
```

No	Model	Accuracy (Macro F1)
1	Keras	0.6773323374641984
2	Scratch	0.6773323374641984

Analisis

Dari hasil nilai akurasi kedua model dapat disimpulkan bahwa implementasi model scratch sudah dapat melakukan perhitungan yang sangat mirip dengan model keras. Hal yang membedakannya mungkin hanyalah dalam segi efisiensi perhitungannya.

2.2.2 RNN

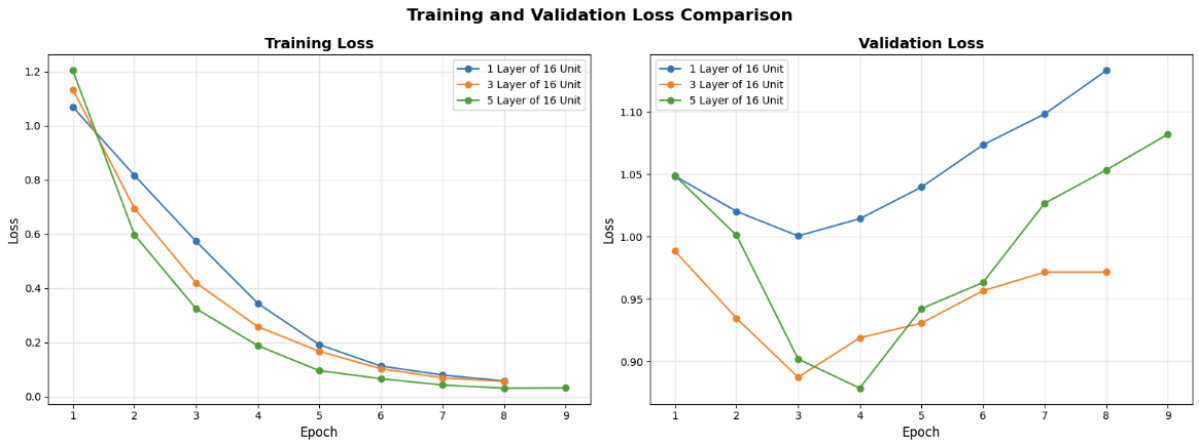
2.2.2.1 Pengaruh jumlah layer RNN

Pengujian ini dilakukan dengan membuat 3 variasi model RNN berdasarkan jumlah layer RNN yang digunakan dalam model tersebut.

Config

```
# Model with 1 RNN layer
model1 = Sequential([
    Input(shape=(1,), dtype=tf.string),
    vectorizer,
    Embedding(input_dim=20000, output_dim=128, mask_zero=True),
    Bidirectional(SimpleRNN(16)),
    Dropout(0.5),
    Dense(3, activation="softmax")
])

# Model with 2 RNN layers
model2 = Sequential([
    Input(shape=(1,), dtype=tf.string),
    vectorizer,
    Embedding(input_dim=20000, output_dim=128, mask_zero=True),
    Bidirectional(SimpleRNN(16, return_sequences=True)),
    Bidirectional(SimpleRNN(16)),
    Dropout(0.5),
    Dense(3, activation="softmax")
])

# Model with 3 RNN layers
model3 = Sequential([
    Input(shape=(1,), dtype=tf.string),
    vectorizer,
    Embedding(input_dim=20000, output_dim=128, mask_zero=True),
    Bidirectional(SimpleRNN(16, return_sequences=True)),
    Bidirectional(SimpleRNN(16, return_sequences=True)),
    Bidirectional(SimpleRNN(16)),
    Dropout(0.5),
    Dense(3, activation="softmax")
])
```

No	Parameter	Accuracy (Macro F1)
1	1 Layer	0.4965
2	2 Layer	0.4911
3	3 Layer	0.5980



Analisis

RNN dengan 3 layer menunjukkan hasil f1-score terbaik. Hal ini, menunjukkan bahwa seiring dengan bertambahnya jumlah layer, model dapat belajar representasi fitur yang lebih kompleks dan hierarkis dari data sekuensial.

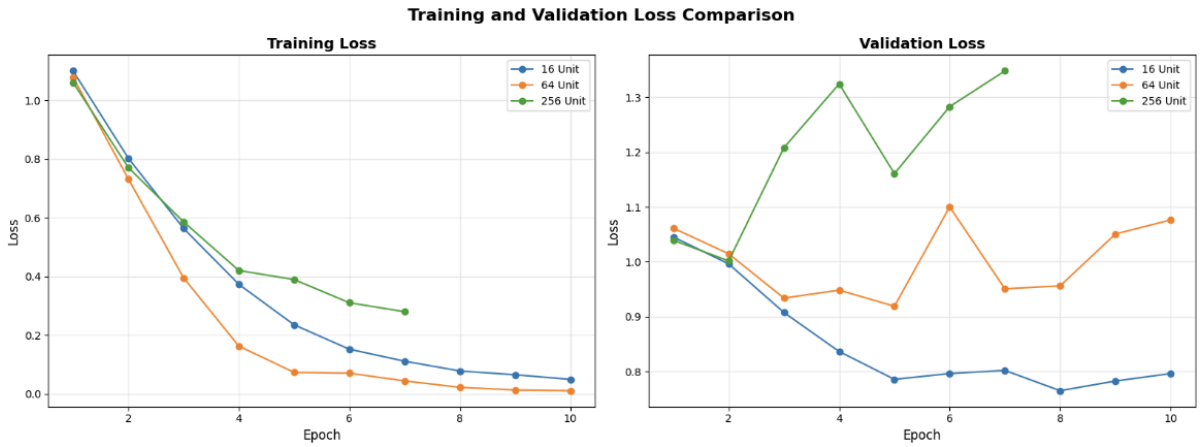
2.2.2.2 Pengaruh banyak cell RNN per layer

Pengujian ini dilakukan dengan membuat 3 variasi model RNN berdasarkan jumlah cell RNN yang digunakan dalam model tersebut.

Config

```
# Model with 2_1 RNN layer
model_2_1 = Sequential([
    Input(shape=(1,), dtype=tf.string),
    vectorizer,
    Embedding(input_dim=20000, output_dim=128, mask_zero=True),
    Bidirectional(SimpleRNN(16)),
    Dropout(0.5),
    Dense(3, activation="softmax")
])

# Model with 2_2 RNN layers
model_2_2 = Sequential([
    Input(shape=(1,), dtype=tf.string),
    vectorizer,
    Embedding(input_dim=20000, output_dim=128, mask_zero=True),
    Bidirectional(SimpleRNN(64)),
    Dropout(0.5),
    Dense(3, activation="softmax")
])

# Model with 2_3 RNN layers
model_2_3 = Sequential([
    Input(shape=(1,), dtype=tf.string),
    vectorizer,
    Embedding(input_dim=20000, output_dim=128, mask_zero=True),
    Bidirectional(SimpleRNN(256)),
    Dropout(0.5),
    Dense(3, activation="softmax")
])
```

No	Parameter	Accuracy (Macro F1)
1	16 Neuron	0.6308
2	64 Neuron	0.5202
3	256 Neuron	0.4015



Analisis

Model RNN dengan jumlah neuron yang terlalu besar (256) memiliki risiko overfitting tinggi, menghasilkan F1-score rendah karena kemungkinan buruk dalam generalisasi. Sementara model dengan neuron lebih sedikit (16) justru memberikan performa paling optimal dan stabil.

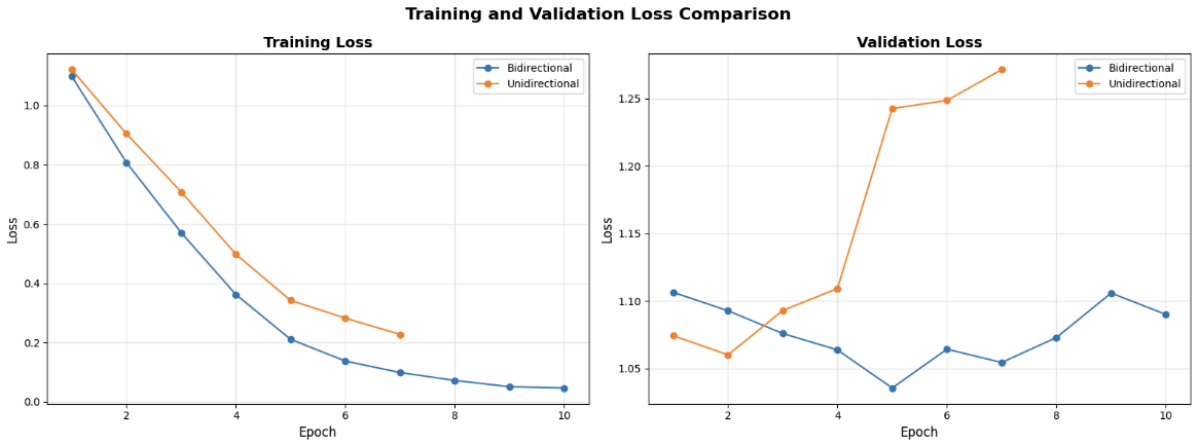
.

2.2.2.3 Pengaruh jenis layer RNN berdasarkan arah

Config

```
# Model with 3_1 RNN layer
model_3_1 = Sequential([
    Input(shape=(1,), dtype=tf.string),
    vectorizer,
    Embedding(input_dim=20000, output_dim=128, mask_zero=True),
    Bidirectional(SimpleRNN(16)),
    Dropout(0.5),
    Dense(3, activation="softmax")
])

# Model with 3_2 RNN layers
model_3_2 = Sequential([
    Input(shape=(1,), dtype=tf.string),
    vectorizer,
    Embedding(input_dim=20000, output_dim=128, mask_zero=True),
    SimpleRNN(16),
    Dropout(0.5),
    Dense(3, activation="softmax")
])
```

No	Parameter	Accuracy (Macro F1)
1	Unidirectional	0.3506
2	Bidirectional	0.4491



Analisis

Penggunaan Bidirectional RNN secara konsisten meningkatkan performa model pada tugas klasifikasi teks seperti ini. Model bidirectional menjadi lebih akurat karena mampu memahami konteks dari dua arah, sehingga menghasilkan macro F1-score lebih tinggi (0.4491) dan validation loss yang lebih stabil.

2.2.2.3 Perbandingan Model Keras dengan Model

a. Parameter

1. Input Layer
   - Input shape: (1,) with dtype=tf.string
   - TextVectorization:
     - max_tokens = 20000
     - output_mode = "int"
     - sequence_length = 100

2. Embedding Layer
   - input_dim = 20000 (vocabulary size)
   - output_dim = 128 (embedding vector size)
   - mask_zero = True

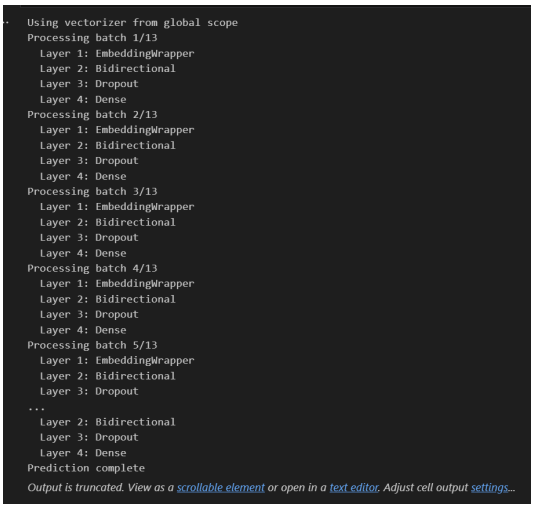
3. RNN Layer (Bidirectional SimpleRNN)
    - units = 64 (per direction)
    - total_units = 128 (2 * 64, due to bidirectional)
    - activation = tanh (default)
    - return_sequences = False

4. Regularization
    - Dropout rate = 0.5

5. Output Layer
    - units = 3 (number of classes)
    - activation = softmax

6. Training Parameters
    - batch_size = 32
    - optimizer = adam
    - loss = sparse_categorical_crossentropy
    - metrics = accuracy
    - Early_stopping:
        - monitor = val_loss
        - patience = 5
        - restore_best_weights = True
        - epochs = 100 (max)

b. Scratch



```
Macro F1-Score on Test Set: 0.38307747879181414
```

c. Sklearn

```
Macro F1-Score: 0.6315

Classification Report:
              precision    recall  f1-score   support

    Negative       0.57      0.67      0.61       153
     Neutral       0.58      0.44      0.50        96
    Positive       0.79      0.77      0.78       151

    accuracy                           0.65       400
   macro avg       0.65      0.63      0.63       400
weighted avg       0.65      0.65      0.65       400
```

d. Hasil Analisis

Berdasarkan hasil f1-score dari kedua model (Keras dan Scratch), model Keras menunjukkan performa yang lebih baik dengan selisih yang cukup signifikan. Perbedaan ini kemungkinan disebabkan oleh optimasi internal yang ada di Keras yang tidak diimplementasikan pada model scratch.

Faktor yang kemungkinan menyebabkan perbedaan ini adalah Keras memiliki mekanisme numerik yang lebih stabil. Model scratch mungkin mengalami masalah pada precision loss dalam komputasi atau radient vanishing/exploding.

Implementasi scratch bisa ditingkatkan dengan menambahkan gradient clipping, implementasi batch normalization, optimasi komputasi matriks, atau perbaikan numerical stability.

2.2.3 LSTM

2.2.2.1 Pengaruh jumlah layer LSTM

Pengujian dilakukan dengan 3 variasi model jumlah layer dengan unit neuron yang tetap. Variasi tersebut adalah:

- 1 Layer
- 3 Layer
- 5 Layer

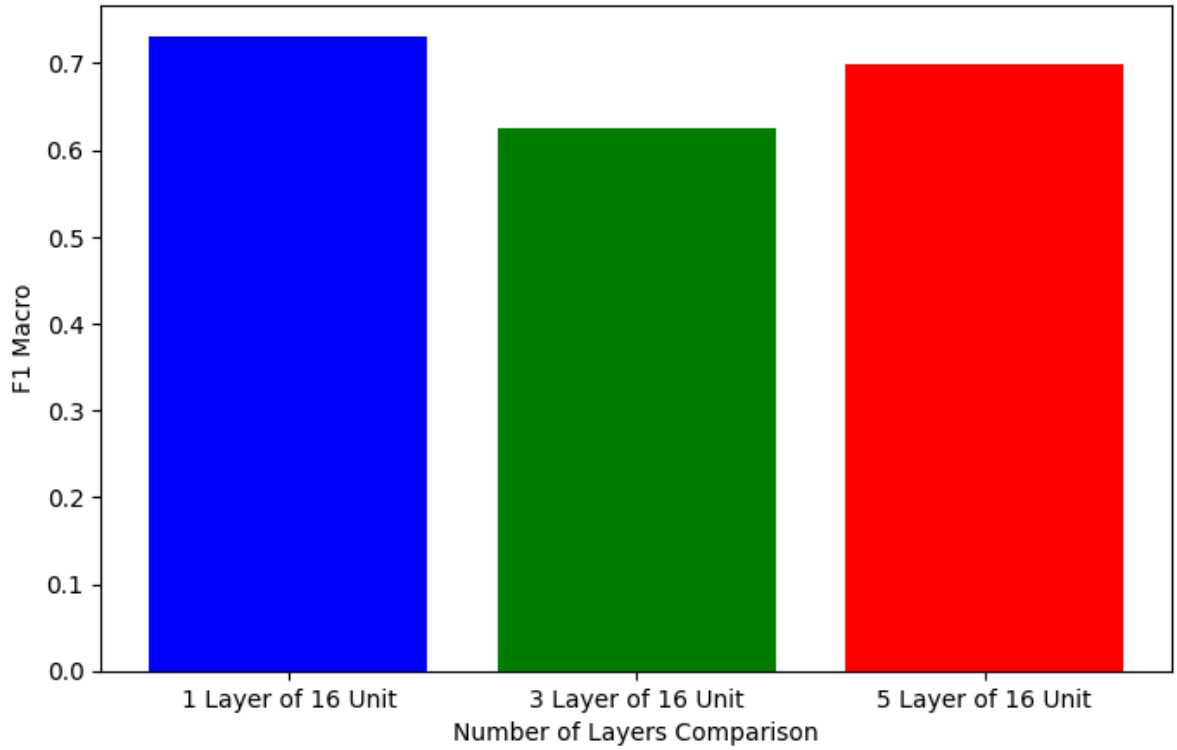
Berikut hasil pengujiannya.

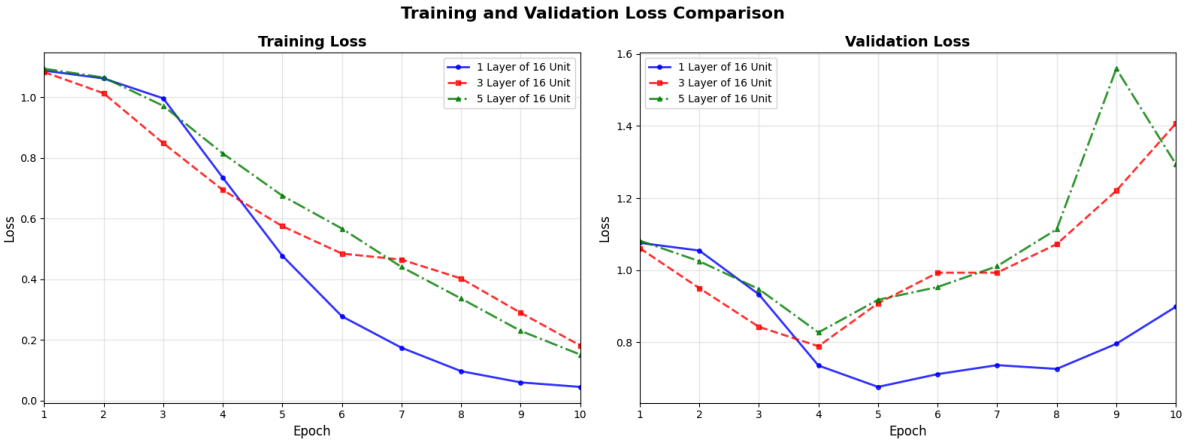
Config

```
config_1_1 = {
    'embedding_dim': 128,
    'lstm_layers': [16],
    'bidirectional': True,
    'dropout_rate': 0.5,
    'num_classes': 3
}

config_1_2 = {
    'embedding_dim': 128,
    'lstm_layers': [16, 16, 16],
    'bidirectional': True,
    'dropout_rate': 0.5,
    'num_classes': 3
}

config_1_3 = {
    'embedding_dim': 128,
    'lstm_layers': [16, 16, 16, 16, 16],
    'bidirectional': True,
    'dropout_rate': 0.5,
    'num_classes': 3
```

```
}
```

No	Parameter	Accuracy (Macro F1)
1	1 Layer	0.7305693442471223
2	3 Layer	0.6247922580057385
3	5 Layer	0.6993139161158061





Analisis

Pada dataset ini jumlah layer yang diberikan sangat bagus pada satu layer. Walau training loss pada 1 layer pada awal-awal epoch adalah yang tertinggi, setelah beberapa epoch training loss ini turun lebih cepat dibandingkan yang lainnya. Selain itu model juga lebih stabil dalam hal overfitting dengan layer yang lebih rendah.

2.2.2.2 Pengaruh banyak cell LSTM per layer

Pengujian dilakukan dengan 3 variasi model unit neuron yang berbeda dengan jumlah layer yang tetap. Variasi tersebut adalah:

- 16 Neuron
- 64 Neuron
- 256 Neuron

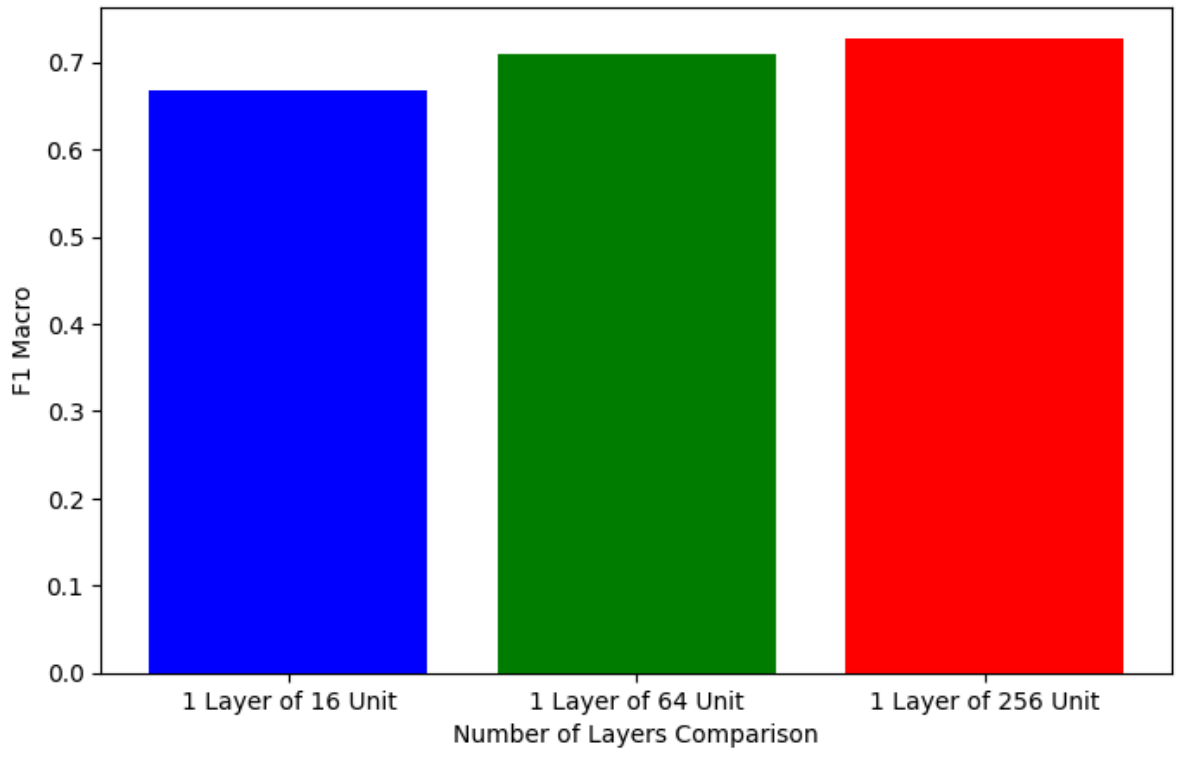
Berikut hasil pengujiannya.

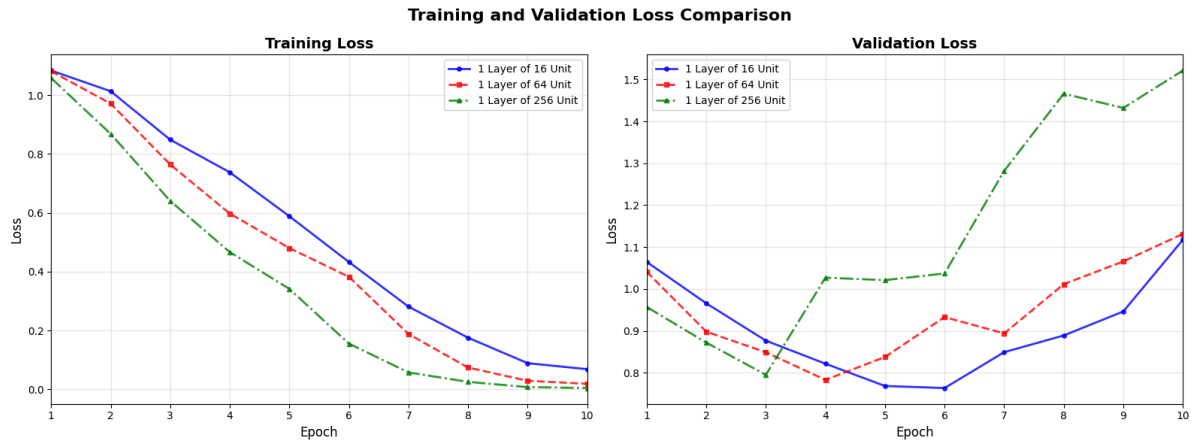
Config

```
config_2_1 = {
    'embedding_dim': 128,
    'lstm_layers': [16],
    'bidirectional': True,
    'dropout_rate': 0.5,
    'num_classes': 3
}

config_2_2 = {
    'embedding_dim': 128,
    'lstm_layers': [64],
    'bidirectional': True,
    'dropout_rate': 0.5,
    'num_classes': 3
}
```

```
config_2_3 = {
    'embedding_dim': 128,
    'lstm_layers': [256],
    'bidirectional': True,
    'dropout_rate': 0.5,
    'num_classes': 3
}
```

No	Parameter	Accuracy (Macro F1)
1	16 Neuron	0.6683373261541624
2	64 Neuron	0.709834462962862
3	256 Neuron	0.7269951840385277





Analisis

Pada dataset ini jumlah neuron yang semakin banyak membuat akurasinya lebih bagus, namun terlihat pada validation loss bahwa model terjadi overfitting. Model dengan jumlah neuron 16 mencapai titik terendah val los pada epoch 6 dan mulai overfit setelahnya.

2.2.2.3 Pengaruh jenis layer LSTM berdasarkan arah

Pengujian dilakukan dengan 2 variasi model directional yang berbeda dengan sisa parameternya sama. Variasi tersebut adalah:

- Unidirectional
- Bidirectional

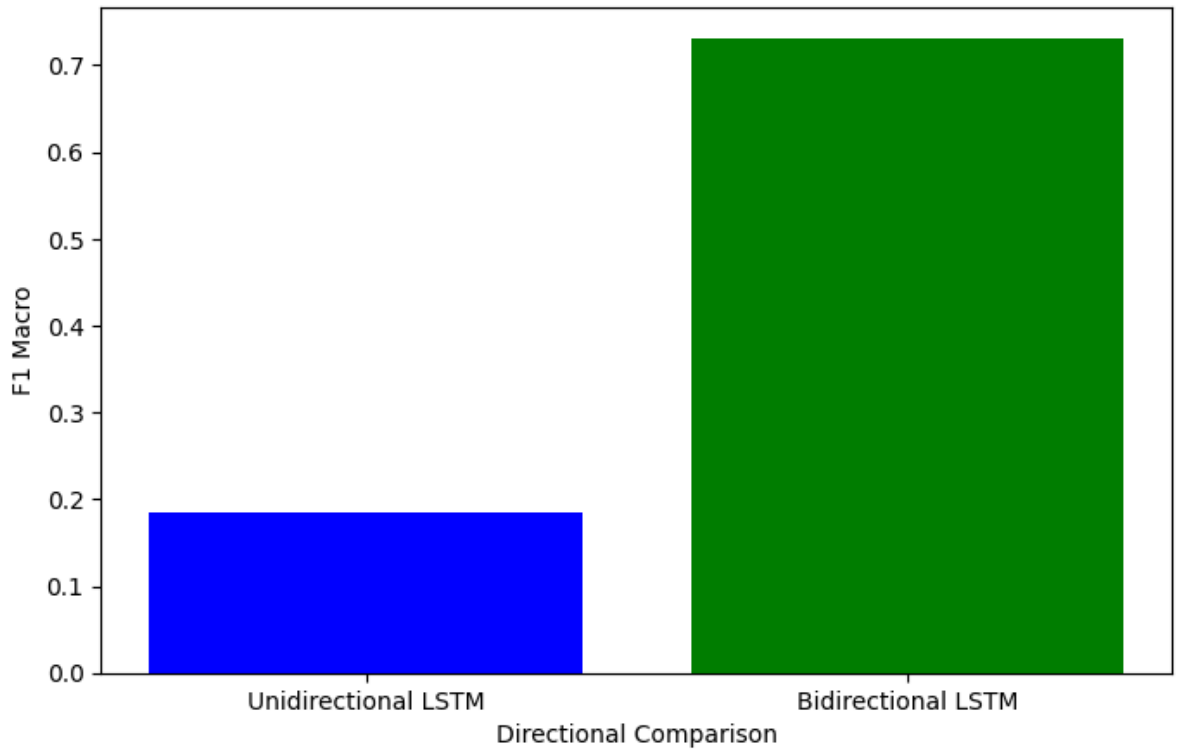
Berikut hasil pengujiannya.

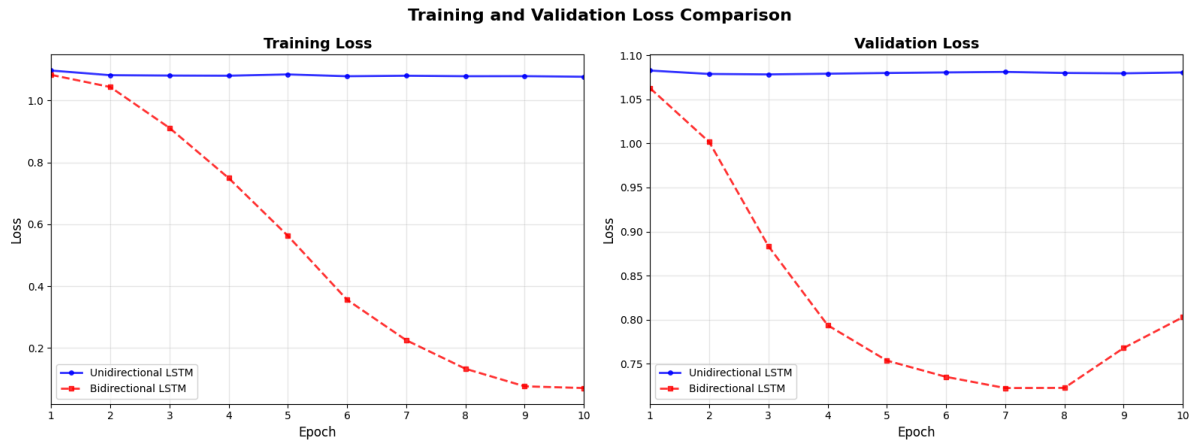
Config

```
config_3_1 = {
    'embedding_dim': 128,
    'lstm_layers': [16],
    'bidirectional': False,
    'dropout_rate': 0.5,
    'num_classes': 3
}
config_3_2 = {
```

```
    'embedding_dim': 128,
    'lstm_layers': [16],
    'bidirectional': True,
    'dropout_rate': 0.5,
    'num_classes': 3
}
```

No	Parameter	Accuracy (Macro F1)
1	Unidirectional	0.1844484629294756
2	Bidirectional	0.7305526113211783





Analisis

Pada dataset ini mengatur arah dari LSTM mempunyai pengaruh yang sangat besar. Model dengan bidirectional mempunyai akurasi dan validation loss yang jauh lebih bagus dibandingkan unidirectional.

2.2.2.4 Perbandingan model Keras dan Scratch

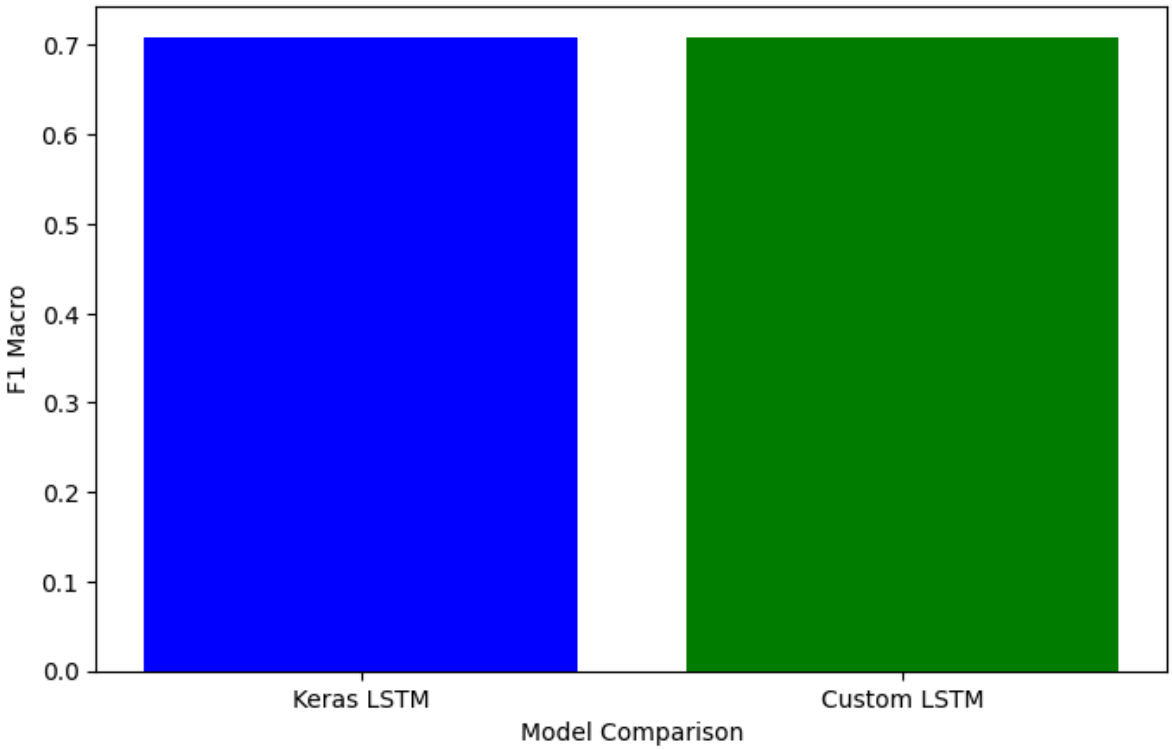
Pengujian dilakukan dengan parameter yang sama. Berikut hasil pengujiannya.

Config

```
config = {
    'data_path': 'data/nusax_sentiment_id',
    'batch_size': 32,
    'max_tokens': 20000,
    'sequence_length': 128,
    'embedding_dim': 128,
    'lstm_layers': [256],
    'bidirectional': True,
    'dropout_rate': 0.5,
    'num_classes': 3,
    'epochs': 10,
}
```

```
 Max difference: 0.00000578
 Mean difference: 0.00000012
 Std difference: 0.00000028

 Dataset prediction agreement: 1.0000

 Class 0 - Mean diff: 0.000000, Agreement: 1.0000
 Class 1 - Mean diff: 0.000000, Agreement: 1.0000
 Class 2 - Mean diff: 0.000000, Agreement: 1.0000
```

```
============================================================
RESULTS COMPARISON
============================================================
Keras LSTM Results:
  Accuracy:    0.7200
  F1 Macro:    0.7086
Custom LSTM Results:
  Accuracy:    0.7200
  F1 Macro:    0.7086
Differences:
  Accuracy Difference:  0.0000
  F1 Macro Difference:  0.0000
```



Analisis

Hasil dari percobaan ini menghasilkan kemiripan yang cukup mirip sekecil $1x10^{-6}$. Dengan F1-Score Makro yang sama

BAB 3 KESIMPULAN DAN SARAN

3.1 Kesimpulan

Berdasarkan implementasi dan serangkaian pengujian yang telah dilakukan terhadap model Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), dan Long Short-Term Memory (LSTM) secara mandiri (from scratch), serta perbandingannya dengan implementasi menggunakan library Keras, dapat ditarik kesimpulan sebagai berikut:

a. Implementasi modul forward propagation untuk CNN, Simple RNN, dan LSTM fromscratch telah berhasil dilakukan. Modul-modul ini mampu memproses input dan menghasilkan output berdasarkan bobot yang telah dilatih sebelumnya menggunakan Keras.
b. Pada model CNN, implementasi from scratch menunjukkan hasil prediksi (Macro F1-score) yang identik dengan model Keras pada dataset CIFAR-10. Hal ini mengindikasikan bahwa perhitungan matematis dalam forward propagation telah diimplementasikan dengan benar.
c. Eksperimen hyperparameter pada CNN menunjukkan bahwa:
    i. Terdapat jumlah layer konvolusi optimal (dalam kasus ini, 2 layer) yang memberikan F1-score tertinggi.
    ii. Peningkatan jumlah filter per layer konvolusi cenderung meningkatkan akurasi, namun dengan risiko overfitting yang juga meningkat seiring bertambahnya filter.
    iii. Ukuran filter konvolusi juga memiliki titik optimal (dalam kasus ini, 9x9) untuk performa terbaik.
    iv. Jenis pooling layer (Max Pooling vs Average Pooling) menunjukkan bahwa Average Pooling memberikan akurasi akhir yang sedikit lebih baik dan lebih tahan terhadap overfitting dalam jumlah epoch yang lebih banyak dibandingkan Max Pooling
d. Pada model LSTM, implementasi from scratch juga berhasil mencapai performa (akurasi dan F1 Macro) yang identik dengan model Keras pada dataset NusaX-Sentiment, dengan perbedaan numerik yang sangat kecil.
e. Eksperimen hyperparameter pada LSTM menunjukkan bahwa:
    i. Satu layer LSTM memberikan F1-score terbaik dibandingkan model dengan 3 atau 5 layer LSTM pada dataset yang digunakan.
    ii. Peningkatan jumlah sel per layer LSTM dapat meningkatkan akurasi, namun juga meningkatkan risiko overfitting.
    iii. Penggunaan arsitektur Bidirectional LSTM secara signifikan meningkatkan performa dibandingkan Unidirectional LSTM.
f. Pada model Simple RNN, terdapat perbedaan performa yang cukup signifikan antara implementasi from scratch dan model Keras, di mana model Keras menunjukkan F1-score yang lebih tinggi. Perbedaan ini kemungkinan disebabkan oleh optimasi internal, stabilitas numerik yang lebih baik, atau mekanisme penanganan gradien yang lebih canggih dalam library Keras yang tidak diimplementasikan pada versi scratch.

g. Eksperimen hyperparameter pada Simple RNN menunjukkan bahwa:
    i. Model dengan 3 layer RNN memberikan F1-score terbaik.
    ii. Jumlah sel RNN yang lebih sedikit (16 neuron) menghasilkan performa yang lebih optimal dan stabil, dengan jumlah sel yang lebih besar (256 neuron) menunjukkan risiko overfitting yang tinggi.
    iii. Sama seperti LSTM, arsitektur Bidirectional pada Simple RNN juga menunjukkan performa yang lebih baik dibandingkan Unidirectional.
h. Secara keseluruhan, tugas besar ini berhasil mendemonstrasikan kemampuan untuk mengimplementasikan dan menganalisis aspek fundamental dari arsitektur deep learning CNN, RNN, dan LSTM, serta memvalidasi kebenaran implementasi forward propagation.

3.2 Saran

Untuk pengembangan dan peningkatan lebih lanjut dari model-model yang telah diimplementasikan, berikut beberapa saran yang dapat dipertimbangkan:

1. Penyempurnaan Implementasi Simple RNN From Scratch

   Mengingat adanya disparitas performa antara model Simple RNN from scratch dan Keras, perlu dilakukan investigasi lebih lanjut. Hal ini bisa mencakup peninjauan ulang perhitungan matematis, eksplorasi teknik untuk meningkatkan stabilitas numerik (misalnya, gradient clipping meskipun tidak diimplementasikan di forward pass, namun kesadaran akan hal ini penting untuk implementasi penuh), atau perbandingan yang lebih detail pada setiap tahapan komputasi.
2. Implementasi Backward Propagation dan Training

   Untuk mendapatkan pemahaman yang lebih menyeluruh, implementasi dapat dilanjutkan dengan modul backward propagation dan mekanisme pembaruan bobot (weight update) secara from scratch. Ini akan memungkinkan pelatihan model sepenuhnya tanpa bergantung pada library Keras untuk proses training.
3. Optimasi Lebih Lanjut

   Meskipun model CNN dan LSTM from scratch telah menunjukkan kesetaraan dengan Keras dalam hal output forward propagation, eksplorasi teknik optimasi komputasi (misalnya, penggunaan operasi matriks yang lebih efisien jika belum maksimal) dapat dilakukan untuk meningkatkan kecepatan inferensi.
4. Eksplorasi Hyperparameter yang Lebih Luas

   Lakukan pengujian dengan rentang hyperparameter yang lebih luas dan kombinasi yang lebih beragam untuk setiap model (CNN, RNN, LSTM) guna mendapatkan pemahaman yang lebih mendalam mengenai sensitivitas model terhadap setiap parameter.
5. Pengujian pada Dataset Lain

   Uji coba model-model from scratch yang telah dikembangkan pada dataset yang berbeda, baik untuk klasifikasi citra maupun teks, untuk mengevaluasi generalisasi dan ketangguhan implementasi.

Pembagian Tugas

Nama	NIM	Tugas
Auralea Alvinia S	13522148	Simple RNN
M. Fauzan Azhim	13522153	LSTM
Pradipta Rafa Mahesa	13522162	CNN

Referensi

- https://www.jasonosajima.com/forwardprop
- https://www.jasonosajima.com/backprop
- https://numpy.org/doc/2.2/
- https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
- https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/
- https://douglasorr.github.io/2021-11-autodiff/article.html
- https://www.cs.toronto.edu/~rgrosse/courses/csc2541_2022/tutorials/tut01.pdf
- https://www.geeksforgeeks.org/training-and-validation-loss-in-deep-learning/
- https://builtin.com/data-science/l2-regularization