

Tugas Besar 2 IF3270 Pembelajaran Mesin

Convolutional Neural Network dan Recurrent Neural Network



Disusun oleh:

Aurelius Justin Philo Fanjaya	(13522020)
Immanuel Sebastian Girsang	(13522058)
Fedrianz Dharma	(13522090)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2025

DAFTAR ISI

BAB I.....	4
DESKRIPSI PERSOALAN.....	4
1.1. Deskripsi Tugas CNN.....	4
Gambar 1. Visualisasi dari Convolutional Neural Network.....	4
Gambar 2. Visualisasi dari Recurrent Neural Network.....	4
1.1.1. Definisi CNN.....	5
1.1.2. Definisi RNN.....	6
1.1.3. Definisi LSTM.....	7
1.1.4. Inisialisasi bobot.....	8
1.1.5. Fungsi Aktivasi.....	10
1.1.6. Fungsi Loss.....	11
1.1.7. Gradient Descent.....	11
1.1.8. Adam.....	12
BAB II.....	13
PEMBAHASAN.....	13
2.1. Deskripsi kelas beserta deskripsi atribut dan methodnya.....	13
2.1.1. Kelas Value.....	13
2.1.2. Kelas Module.....	16
2.1.3. Kelas Layer.....	16
2.1.4. Kelas CNN.....	17
2.1.5. Kelas Conv2D.....	18
2.1.6. Kelas Pooling.....	20
2.1.7. Kelas Flatten.....	21
2.1.8. Kelas Dense.....	22
2.1.9. Kelas LossFunction.....	24
2.1.10. Kelas MSELoss.....	24
2.1.11. Kelas BinaryCrossEntropyLoss.....	24
2.1.12. Kelas CategoricalCrossEntropyLoss.....	25
2.1.13. Kelas RNN.....	25
2.1.14. Kelas EmbeddingLayer.....	25
2.1.15. Kelas DropoutLayer.....	26
2.1.16. Kelas UnidirectionalRNN.....	26
2.1.17. Kelas BidirectionalRNN.....	27
2.1.18. Kelas LSTM.....	27
2.1.19. Kelas UnidirectionalLSTM.....	28
2.1.20. Kelas BidirectionalLSTM.....	28
2.2. Penjelasan Forward Propagation CNN.....	29

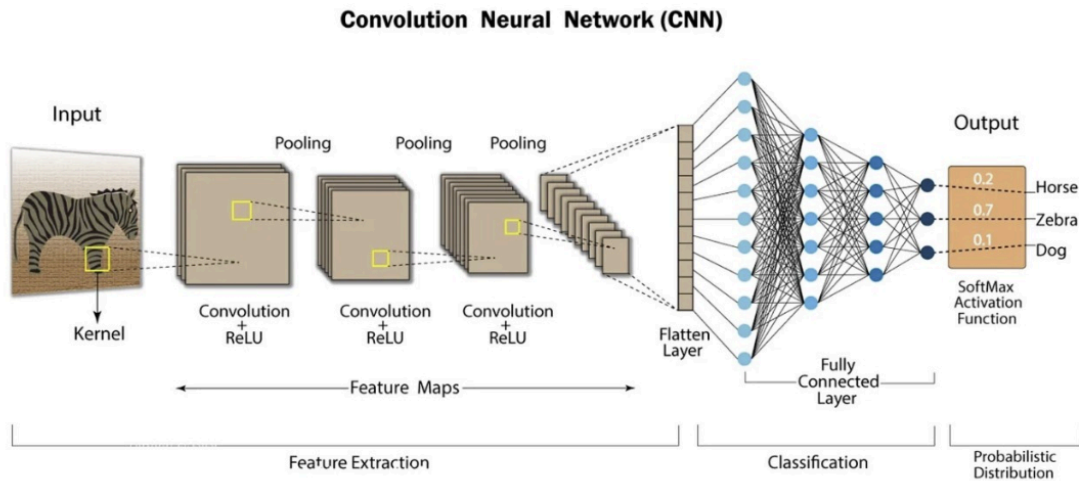
2.3. Penjelasan Forward Propagation RNN.....	30
2.4. Penjelasan Forward Propagation LSTM.....	30
2.5. Penjelasan Backward Propagation dan Weight Update.....	31
2.6. Hasil Pengujian CNN.....	32
2.6.1. Pengaruh jumlah layer konvolusi.....	33
2.6.1.1 Jumlah Layer Konvolusi = 1.....	33
2.6.1.2 Jumlah Layer Konvolusi = 2.....	33
2.6.1.3 Jumlah Layer Konvolusi = 3.....	34
2.6.2. Pengaruh banyak kernel per layer konvolusi.....	35
2.6.2.1 Jumlah Kernel 16 dan 32.....	35
2.6.2.2 Jumlah Kernel 32 dan 64.....	35
2.6.2.3 Jumlah Kernel 64 dan 128.....	36
2.6.3. Pengaruh ukuran kernel per layer konvolusi.....	37
2.6.3.1 Ukuran kernel = 2 x 2.....	37
2.6.3.2 Ukuran kernel = 3 x 3.....	37
2.6.3.3 Ukuran kernel = 4 x 4.....	38
2.6.3.4 Ukuran Kernel 5 x 5.....	39
2.6.3.5 Ukuran kernel = 6 x 6.....	39
2.6.4. Pengaruh Jenis Pooling.....	40
2.6.4.1 Max Pooling.....	40
2.6.4.2. Average Pooling.....	41
2.6.5. Perbandingan dengan Model Keras CNN.....	42
2.6. Hasil Pengujian RNN.....	44
2.6.6. Pengaruh Jumlah layer RNN.....	44
2.6.1.1. 1 Layer RNN.....	44
2.6.1.2. 2 Layer RNN.....	44
2.6.1.3. 3 Layer RNN.....	45
2.6.1.4. F1-Score Keras vs Scratch.....	45
2.6.7. Pengaruh Banyak Cell RNN per Layer.....	46
2.6.2.1. 8 Cell RNN.....	46
2.6.2.2. 16 Cell RNN.....	46
2.6.2.3. 24 Cell RNN.....	46
2.6.2.4. F1-Score Keras vs Scratch.....	47
2.6.8. Pengaruh jenis layer RNN berdasarkan arah.....	48
2.6.3.1. Unidirectional Layer.....	48
2.6.3.2. Bidirectional Layer.....	48
2.6.3.3. F1-Score Keras vs Scratch.....	48
2.7. Hasil Pengujian LSTM.....	49
2.7.1. Pengaruh Jumlah layer LSTM.....	49
2.7.1.1. 1 Layer LSTM.....	49
2.7.1.2. 2 Layer LSTM.....	50

2.7.1.3. 3 Layer LSTM.....	50
2.7.1.4. F1-Score Keras vs Scratch.....	50
2.7.2. Pengaruh banyak cell LSTM per layer.....	51
2.7.2.1. 8 Cell LSTM.....	51
2.7.2.2. 16 Cell LSTM.....	52
2.7.2.3. 24 Cell LSTM.....	52
2.7.3.4. F1-Score Keras vs Scratch.....	52
2.7.3. Pengaruh jenis layer LSTM berdasarkan arah.....	53
2.7.3.1. Unidirectional Layer.....	53
2.7.3.2. Bidirectional Layer.....	54
2.7.3.4. F1-Score Keras vs Scratch.....	54
2.8. Hasil Analisis CNN.....	55
2.8.1. Pengaruh jumlah layer konvolusi.....	55
2.8.2. Pengaruh banyak filter per layer konvolusi.....	55
2.8.3. Pengaruh ukuran filter per layer konvolusi.....	56
2.8.4. Pengaruh jenis pooling layer yang digunakan.....	56
2.8.5. Pengaruh Fungsi Aktivasi terhadap Akurasi.....	57
2.8.6. Perbandingan dengan Library Scikit-Learn.....	58
2.8.7. Pengaruh Regularisasi terhadap Akurasi.....	58
2.9. Hasil Analisis RNN.....	59
2.9.1. Pengaruh jumlah layer RNN.....	59
2.9.2. Pengaruh banyak cell RNN per layer.....	60
2.9.3. Pengaruh jenis layer RNN berdasarkan arah.....	60
2.10. Hasil Analisis LSTM.....	61
2.10.1. Pengaruh jumlah layer LSTM.....	61
2.10.2. Pengaruh banyak cell LSTM per layer.....	61
2.10.3. Pengaruh jenis layer LSTM berdasarkan arah.....	62
BAB III.....	64
KESIMPULAN DAN SARAN.....	64
PEMBAGIAN TUGAS.....	66
REFERENSI.....	67

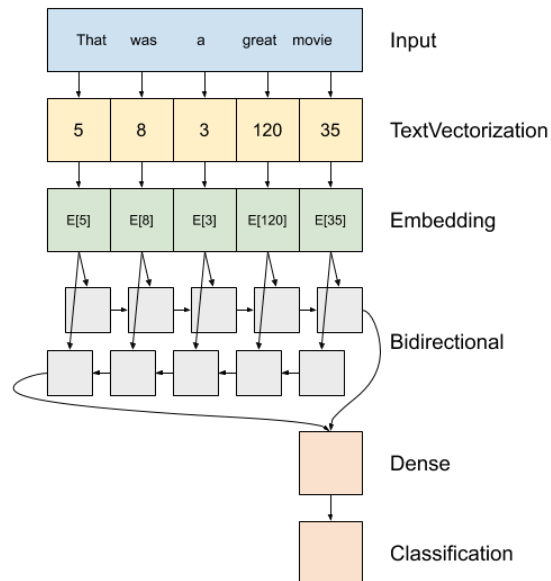
BAB I

DESKRIPSI PERSOALAN

1.1. Deskripsi Tugas CNN



Gambar 1. Visualisasi dari *Convolutional Neural Network*



Gambar 2. Visualisasi dari Recurrent Neural Network

Dalam tugas besar ini, kami membuat metode forward propagation untuk tiga model yang lazim digunakan dalam deep learning, yaitu CNN (*Convolutional Neural Network*), RNN

(*Recurrent Neural Network*), dan LSTM (*Long Short Term Memory*). *Forward propagation* sendiri merupakan proses inti dalam jaringan saraf, di mana data input diproses melalui lapisan-lapisan jaringan untuk menghasilkan output. Proses ini melibatkan perhitungan linear seperti perkalian matriks dan bias, serta penerapan fungsi aktivasi non-linear pada setiap neuron. Hasil dari *forward propagation* kemudian digunakan untuk menghitung *loss*, yang selanjutnya menjadi dasar untuk melakukan proses *backpropagation* dan pembaruan bobot jaringan.

1.1.1. Definisi CNN

Convolutional Neural Network (CNN) merupakan salah satu arsitektur dari *Neural Network* yang secara khusus dirancang untuk mengolah data berdimensi spasial, seperti gambar dan video. CNN sangat populer dalam bidang *computer vision* karena kemampuannya untuk mengenali pola spasial dan struktur lokal secara efektif. Tidak seperti *Feed Forward Neural Network (FFNN)* yang memperlakukan semua input sebagai vektor satu dimensi, CNN dirancang untuk mempertahankan struktur lokal input seperti piksel-piksel yang berdekatan dalam gambar, sehingga informasi spasial tetap terjaga. Terdapat beberapa lapisan utama yang spesial kepada CNN:

1. *Convolutional Layer*

Lapisan *convolution* adalah inti dari CNN. Tujuannya adalah mengekstrak fitur lokal dari input (misalnya gambar) dengan menggunakan filter/kernel yang bergerak (*convolve*) melintasi gambar. Cara Kerja:

1. Filter (misalnya 3×3) akan menggeser secara lokal di area-area kecil dari gambar.
2. Setiap kali filter melewati bagian gambar, dilakukan operasi perkalian *element-wise* dan dijumlahkan untuk menghasilkan satu nilai.
3. Hasil akhirnya adalah *feature map* yang merepresentasikan kehadiran fitur tertentu di posisi tertentu.

Secara strategi, lapisan ini bertugas untuk mengekstraksi "pola" dari data yang ada. Setiap neuron dalam lapisan tersembunyi ini memiliki *weight* dan juga *bias* serta fungsi aktivasi. Fungsi aktivasi digunakan untuk memperkenalkan nonlinearitas pada data yang ada.

2. *Flattening Layer*

Setelah serangkaian *convolution* dan *pooling layer*, hasil *feature maps* yang masih berupa array *multi-dimensi* (misalnya $5 \times 5 \times 64$) diubah menjadi vektor satu dimensi agar bisa diproses oleh *fully connected layer*, misal dari $[5 \times 5 \times 64]$ menjadi $[1600]$. Tidak memiliki parameter untuk dilatih, hanya proses transformasi bentuk data. Ibaratnya, fitur-fitur lokal yang sudah dipelajari di layer sebelumnya sekarang dikemas menjadi satu vektor besar yang berisi representasi citra secara menyeluruh.

3. *Flattening*

Lapisan ini mengurangi dimensi spasial dari *feature map* sambil mempertahankan informasi penting. Contoh paling umum adalah *Max Pooling*, yang mengambil nilai maksimum dari *patch* kecil (misalnya 2×2). Beberapa Fungsi pooling:

1. Mengurangi jumlah parameter
2. Mengurangi *overfitting*
3. Menambah *translation invariance*

4. Fully Connected Layer (Dense)

Lapisan ini mirip dengan lapisan di FFNN dan digunakan di bagian akhir jaringan untuk melakukan klasifikasi berdasarkan fitur-fitur yang telah diekstraksi oleh lapisan sebelumnya. Semua neuron dari layer sebelumnya di-*flatten* dan dihubungkan ke neuron di FC layer. Output-nya bisa berupa probabilitas kelas (menggunakan softmax).

1.1.2. Definisi RNN

Recurrent Neural Network (RNN) merupakan salah satu arsitektur dari Neural Network yang dirancang khusus untuk memproses data berurutan atau sekuensial, seperti teks, suara, dan data time series. Berbeda dengan Feed Forward Neural Network (FFNN) yang memproses input secara independen, RNN memiliki kemampuan untuk mengingat informasi dari langkah waktu sebelumnya melalui koneksi feedback internal sehingga dapat memodelkan dependensi temporal dalam data.

1. Recurrent Layer

Lapisan ini adalah inti dari RNN. Tujuannya adalah untuk memproses input sekuensial satu per satu sambil mempertahankan memori dari langkah waktu sebelumnya.

Cara Kerja:

1. Pada setiap langkah waktu t , RNN menerima input data x_t dan hidden state dari langkah sebelumnya $h_{(t-1)}$.
2. Hidden state baru h_t dihitung sebagai fungsi dari x_t dan $h_{(t-1)}$, biasanya melalui operasi linear dan fungsi aktivasi non-linear (seperti tanh atau ReLU).
3. Hidden state ini menyimpan informasi kontekstual yang “mengalir” ke langkah waktu berikutnya.

Dengan demikian, RNN dapat “mengingat” informasi penting dari masa lalu dan menggunakannya untuk prediksi pada waktu berikutnya.

2. Bidirectional RNN

Untuk menangkap konteks masa depan dan masa lalu secara bersamaan, ada varian RNN yaitu Bidirectional RNN. Terdiri dari dua RNN, satu berjalan maju dari awal ke akhir sekuens, dan satu lagi berjalan mundur dari akhir ke awal

sekuens. Output dari kedua arah ini bisa digabung (misalnya, dengan konkatenasi, penjumlahan, atau perataan) untuk mendapatkan representasi kontekstual dari masa depan dan masa lalu secara bersamaan.

3. **Fully Connected Layer (Dense)**

Lapisan ini umumnya ditempatkan setelah lapisan RNN (atau Bidirectional RNN). Tujuannya adalah untuk mengolah hidden state terakhir (atau semua hidden states jika `return_sequences=True` dari lapisan RNN) menjadi output akhir. Output ini bisa berupa kelas prediksi dalam masalah klasifikasi sekuens (misalnya, analisis sentimen) atau nilai kontinu dalam masalah regresi sekuens (misalnya, prediksi harga saham). Lapisan fully connected menerapkan transformasi linear diikuti oleh fungsi aktivasi (seperti softmax untuk klasifikasi multi-kelas atau sigmoid untuk klasifikasi biner, atau tanpa aktivasi untuk regresi) untuk menghasilkan output yang diinginkan.

1.1.3. Definisi LSTM

Long Short-Term Memory (LSTM) merupakan salah satu arsitektur khusus dari *Recurrent Neural Network (RNN)* yang dirancang untuk mengatasi keterbatasan RNN dalam mempelajari dependensi jangka panjang pada data berurutan atau sekuensial, seperti teks, suara, dan data time series. Berbeda dengan *Feed Forward Neural Network (FFNN)* yang memproses input secara independen, dan berbeda pula dari RNN standar yang kesulitan mengingat informasi dalam jangka waktu lama, LSTM memiliki mekanisme gerbang (gates) yang canggih untuk mengatur aliran informasi sehingga dapat mengingat informasi relevan dari langkah waktu yang jauh sebelumnya.

1. **LSTM Layer**

Lapisan ini adalah inti dari LSTM. Tujuannya adalah untuk memproses input sekuensial satu per satu sambil mempertahankan memori jangka panjang dan pendek dari langkah waktu sebelumnya.

Cara Kerja:

1. Pada setiap langkah waktu t , LSTM menerima input data x_t dan hidden state dari langkah sebelumnya h_{t-1} , serta cell state dari langkah sebelumnya C_{t-1}
2. Hidden state baru h_t dan cell state baru C_t dihitung melalui serangkaian operasi yang melibatkan tiga jenis gerbang utama:
 - *Forget Gate*: Menentukan informasi mana dari cell state sebelumnya (C_{t-1}) yang akan dibuang atau dilupakan.

- *Input Gate*: Menentukan informasi baru mana dari input saat ini (x_t) dan hidden state sebelumnya (h_{t-1}) yang akan disimpan dalam cell state.
- *Output Gate*: Menentukan bagian mana dari cell state saat ini (C_t) yang akan dikeluarkan sebagai hidden state baru (h_t). Cell state (C_t) bertindak sebagai memori jangka panjang yang dapat membawa informasi kontekstual melintasi banyak langkah waktu.
- *Hidden state* (h_t) biasanya merupakan versi terfilter dari cell state yang digunakan untuk prediksi pada langkah waktu saat ini dan sebagai input untuk langkah waktu berikutnya.

2. Bidirectional LSTM Layer

Untuk menangkap konteks masa depan dan masa lalu secara bersamaan dengan lebih efektif, varian LSTM yang umum digunakan adalah *Bidirectional LSTM*. Ini terdiri dari dua lapisan LSTM yang berjalan secara paralel: satu memproses sekuens dari awal ke akhir (maju), dan satu lagi memproses sekuens dari akhir ke awal (mundur). Output dari kedua arah ini kemudian dapat digabungkan (misalnya, dengan konkatenasi, penjumlahan, atau rata-rata) untuk mendapatkan representasi kontekstual yang kaya, mempertimbangkan informasi dari kedua sisi sekuens pada setiap langkah waktu.

3. Dense Layer

Lapisan ini umumnya ditempatkan setelah lapisan LSTM (atau *Bidirectional LSTM*). Tujuannya adalah untuk mengolah hidden state terakhir (atau semua hidden states jika `return_sequences=True` dari lapisan LSTM) menjadi output akhir. Output ini bisa berupa kelas prediksi dalam masalah klasifikasi sekuens (misalnya, analisis sentimen) atau nilai kontinu dalam masalah regresi sekuens (misalnya, prediksi harga saham). Lapisan *fully connected* menerapkan transformasi linear diikuti oleh fungsi aktivasi (seperti softmax untuk klasifikasi multi-kelas atau sigmoid untuk klasifikasi biner, atau tanpa aktivasi untuk regresi) untuk menghasilkan output yang diinginkan.

1.1.4. Inisialisasi bobot

Inisialisasi bobot adalah proses memberikan nilai awal pada bobot sebelum model mulai dilatih. Jika inisialisasi dilakukan dengan tidak tepat, model bisa mengalami *vanishing*

gradient atau *exploding gradient*, yang membuat jaringan sulit untuk belajar secara efektif. Beberapa metode inisialisasi yang digunakan di tugas besar kali ini adalah:

1. Zero Initialization

Inisialisasi dengan membuat semua nilai *weight* dan *bias* menjadi 0.

2. Random Normal Initialization

Random normal initialization dilakukan dengan menerima input *variance* dan juga *mean* untuk nantinya digunakan sebagai standar dalam membagi nilai *weight* dan *bias* secara random namun tetap terdistribusi normal dengan *mean* serta *variance* yang dimasukkan pengguna.

3. Random Uniform Initialization

Random uniform initialization dilakukan dengan menerima input *upper bound* dan juga *lower bound* untuk nantinya digunakan sebagai standar dalam membagi nilai *weight* dan *bias* secara random namun tetap terdistribusi *uniform*.

4. Xavier Initialization

Xavier *initialization* dilakukan dengan menggunakan rumus yang dibuat oleh Xavier Glorot dengan cara menggunakan informasi mengenai berapa input yang akan diterima oleh *input* dan juga berapa *output* yang akan diberikan olehnya. Rumus yang digunakan apabila klasifikasi yang dilakukan adalah *multiclass* sebagai berikut:

$$x = \sqrt{\frac{6}{\text{jumlah output} + \text{input output}}}$$

Apabila klasifikasi yang dilakukan bersifat biner, maka rumus yang dilakukan adalah sebagai berikut:

$$x = \sqrt{\frac{2}{\text{jumlah output} + \text{input output}}}$$

Dari sini, kita akan mendistribusikan bias dan bobot dari range $[-x, +x]$ secara uniform di layer-layer kita.

5. He Initialization

He Initialization merupakan inisialisasi yang dibuat secara spesifik untuk Fungsi aktivasi ReLU dengan merandom angka secara distribusi normal dengan mean = 0 dan juga variansi dengan rumus:

$$\frac{2}{n}$$

Dengan n adalah jumlah input dari layer yang sedang diinisialisasi.

1.1.5. Fungsi Aktivasi

Fungsi aktivasi adalah fungsi non-linear yang menentukan apakah neuron akan diaktifkan berdasarkan input yang diterimanya. Tanpa fungsi aktivasi, jaringan hanya akan merepresentasikan fungsi linear, yang membatasi kemampuannya dalam mempelajari pola kompleks. Beberapa fungsi yang akan digunakan di tugas besar ini adalah:

1. Linear

$$Linear(x) = x$$

2. Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

3. Tanh

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

4. ReLU

$$ReLU(x) = \max(0, x)$$

5. Leaky ReLU

$$Leaky_ReLU(x, \alpha) = x \text{ if } x > 0, \alpha * x \text{ otherwise}$$

6. ELU

$$Elu(x, \alpha) = \alpha * (e^x - 1) \text{ if } x < 0, x \text{ otherwise}$$

7. Softmax

Untuk vector $\vec{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$,

$$\text{softmax}(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

1.1.6. Fungsi Loss

Fungsi loss digunakan untuk mengukur seberapa baik model memprediksi output yang diharapkan. Model dilatih untuk meminimalkan fungsi loss. Beberapa fungsi *loss* yang akan digunakan di tugas besar kali ini adalah:

1. Mean Square Error

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

2. Binary Cross Entropy

$$\mathcal{L}_{BCE} = -\frac{1}{n} \sum_{i=1}^n (y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i))$$

y_i = Actual binary label (0 or 1)

\hat{y}_i = Predicted value of y_i

n = Batch size

3. Categorical Cross Entropy

$$\mathcal{L}_{CCE} = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^C (y_{ij} \log \hat{y}_{ij})$$

y_{ij} = Actual value of instance i for class j

\hat{y}_{ij} = Predicted value of y_{ij}

C = Number of classes

n = Batch size

1.1.7. Gradient Descent

Gradient Descent adalah algoritma optimasi yang digunakan untuk memperbarui bobot dalam jaringan saraf berdasarkan turunan dari fungsi loss. Pada setiap epochnya, bobot yang ada di setiap lapisan akan di update untuk meminimalkan *loss function* dengan cara mengurangnya sebesar *learning rate* tertentu dari gradiennya terhadap *loss function*.

1.1.8. Adam

Adam (*Adaptive Moment Estimation*) adalah algoritma optimasi yang sangat populer dalam pelatihan jaringan saraf karena kemampuannya mempercepat proses konvergensi sekaligus menjaga kestabilan pembelajaran. Adam bekerja dengan menggabungkan dua teknik optimasi, yaitu Momentum dan RMSProp. Momentum mempercepat pergerakan menuju minimum global dengan mempertahankan kecepatan gradien sebelumnya, sementara RMSProp menyesuaikan *learning rate* secara adaptif berdasarkan rata-rata kuadrat gradien terkini. Adam menggunakan dua estimasi statistik: momen pertama (rata-rata gradien) dan momen kedua (variansi gradien), serta memperbaiki bias dari estimasi tersebut di awal iterasi, sehingga update parameter menjadi lebih akurat dan efisien.

Keunggulan utama Adam adalah kemampuannya bekerja dengan baik di berbagai jenis masalah, termasuk yang melibatkan dataset besar dan parameter banyak. Ia tidak memerlukan tuning *learning rate* yang kompleks dan relatif lebih cepat dibanding algoritma lain seperti SGD. Secara umum, Adam cocok untuk pemula karena praktis, namun juga cukup kuat untuk digunakan dalam model deep learning kompleks. Meski begitu, dalam beberapa kasus, Adam bisa overfit atau tidak menghasilkan generalisasi sebaik optimizer lain, sehingga tetap disarankan mengkombinasikannya dengan teknik regularisasi seperti *dropout* atau *early stopping*.

BAB II

PEMBAHASAN

2.1. Deskripsi kelas beserta deskripsi atribut dan methodnya

2.1.1. Kelas Value

Deskripsi Kelas	
<p>Kelas ini berfungsi sebagai wrapper dari struktur data data torch.Tensor yang terlibat di dalam perhitungan FFNN dan memiliki fitur auto-differentiation dengan melakukan pembangunan graf. Definisi operasi-operasi dengan turunan yang khas di-override di kelas ini. Sehingga, setiap kali melakukan operasi, yang terjadi adalah pengembalian objek Value yang baru dengan informasi operasi yang harus dilakukan saat suatu nilai yang memiliki objek tersebut di dalam graf terarahnya ingin mencari nilai turunan parsial dari objek pemanggil terhadap semua operan yang terlibat.</p> <p>Struktur kelas ini mengambil inspirasi dari library micrograd dalam implementasi auto differentiation, serta library autodidact yang digunakan sebagai bahan ajar di University of Toronto untuk mendapatkan referensi perhitungan vector-Jacobian product dan broadcasting jika yang terlibat dalam operasi adalah struktur data dengan dimensi lebih dari 1.</p> <p>Fungsi-fungsi aktivasi didefinisikan pada kelas ini. Jika operasinya merupakan komposisi dari operasi dasar yang sudah terdefinisi pada kelas ini, maka metode untuk implementasi fungsi tersebut tidak akan mendefinisikan ulang atribut <code>_backward</code> dan langsung mengembalikan nilai yang didapatkan dari operasi saja.</p>	
Atribut	
data	Menyimpan data berupa objek torch.Tensor.
requires_grad	Boolean flag yang menandakan apakah objek perlu nilai grad dihitung saat backward propagation dilakukan.
grad	Menyimpan nilai gradien objek, yaitu nilai turunan parsial dari objek Value yang memanggil metode <code>backward()</code> terhadap objek ini.
_backward	Merupakan lambda function yang dapat dipanggil ketika pemanggilan <code>backward()</code> dilakukan oleh objek lain pada elemen lebih akhir dari topological graph operasi-operasi yang terjadi.

<code>_prev</code>	Menyimpan child nodes, yaitu operand dari operasi yang terdefinisi pada kelas Value yang menghasilkan objek yang diacu.
<code>_op</code>	String yang menandakan operasi yang menyebabkan objek terbuat.
Metode	
<code>__init__(self, data, requires_grad=False, _children=(), _op="")</code>	Konstruktor objek.
<code>__repr__(self)</code>	Memberi format yang akan muncul jika objek dicetak.
<code>T(self)</code>	Melakukan transpose matriks.
<code>__add__(self, other)</code>	Override metode penjumlahan ketika operan sebelah kiri adalah objek Value.
<code>__mul__(self, other)</code>	Override metode perkalian antarelemen ketika operan sebelah kiri adalah objek Value.
<code>__matmul__(self, other)</code>	Override metode perkalian matriks ketika operan sebelah kiri adalah objek Value.
<code>__truediv__(self, other)</code>	Override metode pembagian ketika operan sebelah kiri adalah objek Value.
<code>__pow__(self, other)</code>	Override metode pemangkatan untuk operan objek Value terhadap int atau float.
<code>exp(self)</code>	Pemangkatan bilangan euler terhadap objek Value.
<code>log(self)</code>	Perhitungan logaritma terhadap objek Value.
<code>abs(self)</code>	Perhitungan nilai mutlak terhadap objek Value.
<code>sum(self)</code>	Mencari jumlah semua elemen atribut data.
<code>mean(self)</code>	Mencari rata-rata semua elemen atribut data.
<code>linear(self)</code>	Fungsi aktivasi, mengembalikan diri sendiri.
<code>relu(self)</code>	Fungsi aktivasi, mengembalikan nilai

	maksimum dari nol dan tiap elemen atribut data dalam dimensi atribut data.
<code>tanh(self)</code>	Fungsi aktivasi, mengembalikan nilai tanh dari tiap elemen atribut data dalam dimensi atribut data.
<code>sigmoid(self)</code>	Fungsi aktivasi, mengembalikan nilai sigmoid dari tiap elemen atribut data dalam dimensi atribut data.
<code>leaky_relu(self, negative_slope=0.01)</code>	Fungsi aktivasi, mengembalikan nilai Leaky ReLU dari tiap elemen atribut data dalam dimensi atribut data.
<code>elu(self, alpha=1.0)</code>	Fungsi aktivasi, mengembalikan nilai ELU dari tiap elemen atribut data dalam dimensi atribut data.
<code>softmax(self)</code>	Fungsi aktivasi, mengembalikan nilai proporsi eksponensial dari tiap elemen atribut data dalam dimensi atribut data.
<code>clamp(self, min_val, max_val)</code>	Memotong nilai tiap elemen atribut data yang lebih kecil dari <code>min_val</code> menjadi <code>min_val</code> dan yang lebih besar dari <code>max_val</code> menjadi <code>max_val</code> .
<code>backward(self)</code>	Membangun topological graph, kemudian dari elemen terakhir pada graf memanggil fungsi yang disimpan pada atribut <code>_backward</code> elemen tersebut sampai pada elemen pertama.
<code>__neg__(self)</code>	Override ketika suatu objek Value memiliki prefiks '- '.
<code>__radd__(self, other)</code>	Override metode penjumlahan ketika operan sebelah kanan adalah objek Value.
<code>__sub__(self, other)</code>	Override metode pengurangan ketika operan sebelah kiri adalah objek Value.
<code>__rsub__(self, other)</code>	Override metode pengurangan ketika operan sebelah kanan adalah objek Value.
<code>__rmul__(self, other)</code>	Override metode perkalian antarelemen ketika operan sebelah kanan adalah objek Value.

<code>__rtruediv__(self, other)</code>	Override metode pembagian ketika operan sebelah kanan adalah objek Value.
<code>_reduce_grad(self, grad, target_shape)</code>	Melakukan "reverse broadcasting." Setiap kali melakukan operasi antara dua tensor dengan ukuran yang berbeda namun kompatibel untuk broadcasting, maka torch akan secara otomatis melakukan broadcasting. Namun, karena auto differentiation diimplementasikan secara mandiri, metode ini yang memastikan reduksi nilai dari operasi bersama nilai gradien objek pemanggil metode <code>_backward</code> dilakukan dengan benar.

2.1.2. Kelas Module

Deskripsi Kelas	
Kelas general yang akan diturunkan oleh kelas Layer dan kelas FFNN, sesuai API PyTorch.	
Metode	
<code>zero_grad(self)</code>	Membuat semua nilai yang diberikan metode parameters menjadi nol.
<code>parameters(self)</code>	Mengembalikan semua parameter objek.

2.1.3. Kelas Layer

Deskripsi Kelas	
Unit terkecil Feed Forward Neural Network. Untuk mendukung paralelisasi, semua weights dan bias model disimpan dalam bentuk matriks (2D-tensor), sehingga, tidak terdapat kelas neuron.	
Atribut	
<code>activation</code>	Fungsi aktivasi layer.
<code>W</code>	Bobot layer.
<code>b</code>	Bias layer.
<code>input_shape</code>	Dimensi masukan layer.

output_shape	Dimensi keluaran layer.
init_method	Metode inisialisasi bobot dan bias layer.
Metode	
<code>__init__(self, output_shape, activation, init_method, input_shape)</code>	Menginisialisasi bentuk layer, termasuk berapa neuron (output_shape), fungsi aktivasi, ingin diinisialisasi dengan cara apa, dan dimensi masukkan.
parameters(self)	Implementasi metode yang sama dari kelas Module.
<code>__call__(self, X)</code>	Menghitung nilai net dan output aktivasi berdasarkan input X.

2.1.4. Kelas CNN

Deskripsi Kelas	
Kelas ini menyediakan API untuk <i>Convolutional Neural Network</i> dan merupakan implementasi dari kelas <i>Module</i> yang terinspirasi dari API TensorFlow, PyTorch, dan micrograd.	
Atribut	
seed	Nilai random untuk seeding pengacakan agar eksperimen dapat diulang.
lr	Konstanta yang mengatur proporsi perubahan bobot saat melakukan optimisasi.
verbose	Verbose bernilai satu artinya detil akan ditampilkan saat pelatihan.
epochs	Besaran ini menyatakan berapa kali seluruh dataset ingin dikunjungi secara keseluruhan ketika pelatihan model.
batch_size	Menyatakan banyaknya baris dari dataset latih yang ingin dihitung sebelum melakukan pembaruan nilai bobot/bias.
layers	List dari objek Layer.
loss_fn	Objek fungsi loss yang akan digunakan untuk perhitungan loss.

optimizer	Optimizer yang akan digunakan oleh kelas, secara <i>default</i> yang digunakan adalah 'adam'.
Metode	
<code>__init__(self, layers_list, random_seed, learning_rate, verbose, epochs, batch_size)</code>	Konstruktor kelas FFNN, menerima list objek Layer yang akan menjadi hidden layers dan output layer dari Feed Forward Neural Network.
<code>fit(self, X, Y, X_val=None, y_val=None)</code>	Melakukan pelatihan (memanggil metode untuk forward dan backward propagation) dari dataset X dan target y yang diberikan dan menerima validation set X_val dan y_val. Jika X_val dan y_val diberikan, maka setiap epoch akan dilakukan perhitungan loss validation set.
<code>predict(self, X)</code>	Memanggil forward untuk setiap <i>layer</i> yang ada sehingga dapat melakukan prediksi atas nilai yang
<code>load_weights(self, weights_list)</code>	<i>Load Weights</i> digunakan untuk melakukan <i>load</i> terhadap bobot-bobot yang ada di model keras ke setiap layer yang memiliki bobot dan juga bias.

2.1.5. Kelas Conv2D

Deskripsi Kelas	
Kelas ini merupakan kelas yang merepresentasikan layer konvolusi dari CNN. Kelas ini memiliki metode yang dapat digunakan untuk melakukan <i>forward</i> dan juga <i>backward propagation</i> dari suatu <i>layer</i> .	
Atribut	
filters	Seberapa banyak kernel yang digunakan untuk <i>layer</i> ini.
kernel_size	Ukuran kernel yang digunakan untuk <i>layer</i> ini.
strides	Seberapa besar <i>stride</i> atau pergeseran yang digunakan untuk <i>layer</i> ini
padding	Seberapa besar <i>padding</i> yang ingin

	digunakan untuk <i>layer</i> ini
activation	<i>Activation function</i> yang digunakan untuk <i>layer</i> ini
use_bias	Boolean value yang digunakan untuk menandakan apakah <i>layer</i> ini menggunakan bias
kernel_initializer	Metode yang ingin digunakan sebagai strategi untuk melakukan inisialisasi bobot dari setiap kernel yang ada.
bias_initializer	Metode yang ingin digunakan sebagai strategi untuk melakukan inisialisasi bias
weights	Bobot yang digunakan di <i>layer</i> ini
bias	Bias yang digunakan di <i>layer</i> ini
grad_weights	Gradien dari tiap bobot di <i>layer</i> ini
grad_bias	Gradien dari tiap bias di <i>layer</i> ini
input	Input yang digunakan di <i>layer</i> ini
input_padded	Hasil dari input yang sudah diberikan <i>padding</i>
input_channels	Jumlah <i>channel</i> dari input <i>layer</i> ini
padding_h	Tinggi dari <i>padding</i> yang digunakan
padding_w	Lebar dari <i>padding</i> yang digunakan
Metode	
<code>__init__(self, filters, kernel_size, strides, padding, activation, use_bias, kernel_initializer, bias_initializer)</code>	Metode yang digunakan untuk melakukan inisialisasi terhadap kelas ini.
<code>_calculate_same_padding(self, input_height,</code>	Metode yang digunakan untuk menentukan

input_width):	<i>padding</i> apabila pengguna menggunakan <i>padding = "same"</i>
_pad_input(self, x, padding_h, padding_w):	Metode untuk melakukan <i>padding</i> terhadap input yang dimasukkan oleh pengguna
_calculate_output_size(self, H, W, padding_h, padding_w):	Metode untuk melakukan kalkulasi terhadap ukuran dari <i>output</i> yang diharapkan oleh layer ini
_initialize_parameters(self, input_channels):	Metode untuk melakukan inisialisasi parameter tergantung dari berapa banyak <i>channel</i> yang ada
forward(self, x):	Metode yang digunakan untuk melakukan <i>backward propagation</i> di <i>layer</i> ini untuk kemudian melakukan propagasi terhadap gradien-gradien di sini.
backward(self, dA):	Metode yang digunakan untuk melakukan <i>backward propagation</i> di <i>layer</i> ini untuk kemudian melakukan propagasi terhadap gradien-gradien di sini.
load_parameters(self, weights, bias=None):	Metode untuk melakukan <i>load</i> dari bobot dan juga bias di <i>layer</i> ini.

2.1.6. Kelas Pooling

Deskripsi Kelas	
<p>Kelas ini merupakan kelas yang merepresentasikan layer Pooling dari CNN. Kelas ini memiliki metode yang dapat digunakan untuk melakukan <i>forward</i> dan juga <i>backward propagation</i> dari suatu <i>layer</i>. Kelas ini juga memiliki pilihan berikut sebagai jenis <i>pooling</i> nya:</p> <ol style="list-style-type: none"> 1. <i>Max Pooling</i> 2. <i>Average Pooling</i> 3. <i>Global Max Pooling</i> 4. <i>Global Average Pooling</i> 	
Atribut	
pool_size	Seberapa besar ukuran <i>kernel</i> yang digunakan untuk melakukan <i>pooling</i>
pool_type	Tipe pooling yang digunakan di <i>layer</i> ini
strides	Seberapa besar <i>stride</i> atau pergeseran yang

	digunakan untuk <i>layer</i> ini
padding	Seberapa besar <i>padding</i> yang ingin digunakan untuk <i>layer</i> ini
input_shape	Ukuran dari data <i>input</i>
output_shape	Ukuran dari data <i>output</i>
max_indices	Index dari nilai-nilai maksimal yang dapat digunakan nanti ketika <i>backpropagation</i>
input_padded	Hasil input yang sudah diberikan <i>padding</i> yang sesuai
Metode	
<code>__init__(self, pool_size, pool_type, strides, padding):</code>	Metode untuk melakukan inisialisasi terhadap kelas <i>Pooling</i>
<code>_pad_input(self, x, padding_h, padding_w):</code>	Metode untuk melakukan <i>padding</i> terhadap input yang dimasukkan oleh pengguna
<code>_calculate_output_size(self, H, W, padding_h, padding_w):</code>	Metode untuk melakukan kalkulasi terhadap ukuran dari <i>output</i> yang diharapkan oleh layer ini
<code>forward(self, x):</code>	Metode yang digunakan untuk melakukan <i>backward propagation</i> di <i>layer</i> ini untuk kemudian melakukan propagasi terhadap gradien-gradien di sini.
<code>backward(self, dA):</code>	Metode yang digunakan untuk melakukan <i>backward propagation</i> di <i>layer</i> ini untuk kemudian melakukan propagasi terhadap gradien-gradien di sini.
<code>load_parameters(self, weights, bias=None):</code>	Metode untuk melakukan <i>load</i> dari bobot dan juga bias di <i>layer</i> ini.

2.1.7. Kelas Flatten

Deskripsi Kelas
Kelas ini merupakan kelas yang merepresentasikan layer <i>Flatten</i> dari CNN. Kelas ini memiliki metode yang dapat digunakan untuk melakukan <i>forward</i> dan juga <i>backward propagation</i> dari suatu <i>layer</i> . Kelas ini bertugas untuk melakukan <i>flattening</i> dimensi input yang awalnya dimensinya bisa banyak menjadi satu dimensi saja agar bisa menjadi <i>input</i> di <i>dense layer</i> .

Atribut	
input_shape	Ukuran input yang nantinya akan di <i>reshape</i>
X	Input yang dimasukkan oleh pengguna
Metode	
__init__(self):	Metode ini digunakan untuk melakukan inisialisasi dari <i>layer flatten</i> yang digunakan untuk menjadi <i>input</i> dari <i>dense</i> layer.
forward(self, x):	Metode yang digunakan untuk melakukan <i>backward propagation</i> di <i>layer</i> ini untuk kemudian melakukan propagasi terhadap gradien-gradien di sini.
backward(self, dA):	Metode yang digunakan untuk melakukan <i>backward propagation</i> di <i>layer</i> ini untuk kemudian melakukan propagasi terhadap gradien-gradien di sini.

2.1.8. Kelas Dense

Deskripsi Kelas	
Kelas ini merepresentasikan <i>dense</i> layer dari model. Kelas ini memiliki metode untuk melakukan perkalian matriks antara <i>input</i> dan juga bobot serta bias yang ada di layer tersebut.	
Atribut	
weights	Bobot yang digunakan di <i>layer</i> ini

bias	Bias yang digunakan di <i>layer</i> ini
grad_weights	Gradien dari tiap bobot di <i>layer</i> ini
grad_bias	Gradien dari tiap bias di <i>layer</i> ini
Metode	
def __init__(self, output_shape, activation, init_method, input_shape):	Metode untuk melakukan inisialisasi layer dengan parameter yang ada.
_initialize_parameters(self, input_dim)	Metode untuk melakukan inisialisasi bobot di layer ini
forward(self, x):	Metode yang digunakan untuk melakukan <i>backward propagation</i> di <i>layer</i> ini untuk kemudian melakukan propagasi terhadap gradien-gradien di sini.
backward(self, dA):	Metode yang digunakan untuk melakukan <i>backward propagation</i> di <i>layer</i> ini untuk kemudian melakukan propagasi terhadap gradien-gradien di sini.
load_parameters(self, weights, bias=None):	Metode untuk melakukan <i>load</i> dari bobot dan juga bias di <i>layer</i> ini.

2.1.9. Kelas LossFunction

Deskripsi Kelas	
Kelas umum yang harus diimplementasikan oleh kelas yang mengimplementasikan fungsi-fungsi loss yang akan digunakan.	
Metode	
<code>__call__(self, y_pred, y_true)</code>	Secara internal metode ini memanggil metode forward.
<code>forward(self, y_pred, y_true)</code>	Metode ini akan raise <code>NotImplementedError</code>

2.1.10. Kelas MSELoss

Deskripsi Kelas	
Kelas ini mengimplementasikan Mean Squared Error Loss untuk permasalahan multi label yang setiap labelnya adalah binary classification, single label binary classification, dan regression.	
Metode	
<code>forward(self, y_pred, y_true)</code>	Menghitung rata-rata dari selisih kuadrat label prediksi dan label sebenarnya. Menerima input dengan dimensi <code>(batch_size,)</code> , <code>(batch_size, 1)</code> , atau <code>(batch_size, n_label)</code> .

2.1.11. Kelas BinaryCrossEntropyLoss

Deskripsi Kelas	
Kelas ini mengimplementasikan Binary Cross Entropy Loss untuk permasalahan multi label yang setiap labelnya adalah binary classification dan single label binary classification.	
Metode	
<code>forward(self, y_pred, y_true)</code>	Menghitung cross entropy dan menghitung rata-ratanya, baik terhadap seluruh baris atau terhadap seluruh label. Menerima input dengan dimensi <code>(batch_size,)</code> , <code>(batch_size, 1)</code> , atau <code>(batch_size, n_label)</code> .

2.1.12. Kelas CategoricalCrossEntropyLoss

Deskripsi Kelas	
Kelas ini mengimplementasikan Categorical Cross Entropy Loss untuk permasalahan multiclass classification.	
Metode	
forward(self, y_pred, y_true)	Menghitung categorical cross entropy dan menghitung rata-ratanya terhadap semua baris data.

2.1.13. Kelas RNN

Deskripsi Kelas	
Kelas ini mengimplementasikan RNN untuk membangun model RNN dengan layer-layer yang ada	
Metode	
__init__(self, layers=None, batch_size=None)	Mengkonstruksi dan menginisialisasi objek RNN dengan parameter layer dan batch size
add(self, layer)	Menambahkan layer ke model RNN
forward(self, x: Value)	Melakukan pemanggilan forward pada tiap layer pada model RNN
batch_generator(self, X, batch_size)	Memproses X secara batch
predict(self, x)	Melakukan inferensi secara batch
load_weights(self, weights)	Memuat weight yang didapatkan dari model Keras untuk setiap layer
__call__(self, x)	Melakukan pemanggilan forward atau predict

2.1.14. Kelas EmbeddingLayer

Deskripsi Kelas	
Kelas ini mengimplementasikan Embedding Layer	
Metode	
__init__(self, vocab_size, embedding_dim, initializer='normal')	Mengkonstruksi dan menginisialisasi objek EmbeddingLayer

<code>forward(self, x: Value)</code>	Melakukan training atau inferensi
<code>load_weights(self, weights)</code>	Memuat weight yang didapatkan dari model Keras
<code>__call__(self, x)</code>	Melakukan pemanggilan forward

2.1.15. Kelas DropoutLayer

Deskripsi Kelas	
Kelas ini mengimplementasikan Dropout Layer	
Metode	
<code>__init__(self, rate, seed=None)</code>	Mengkonstruksi dan menginisialisasi objek Dropout
<code>forward(self, inputs):</code>	Melakukan training atau inferensi

2.1.16. Kelas UnidirectionalRNN

Deskripsi Kelas	
Kelas ini mengimplementasikan Unidirectional RNN	
Metode	
<code>__init__(self, units, input_dim=None, batch_size=None, activation="tanh", kernel_initializer="glorot_uniform", recurrent_initializer="orthogonal", bias_initializer="zeros", return_sequences=False)</code>	Mengkonstruksi dan menginisialisasi objek Unidirectional RNN
<code>build(self, input_dim)</code>	Menginisialisasi weight
<code>init_hidden_state(self, batch_size)</code>	Menginisialisasi hidden state pertama
<code>forward(self, x: Value)</code>	Melakukan training atau inferensi
<code>load_weights(self, weights)</code>	Memuat weight yang didapatkan dari model Keras
<code>__call__(self, x)</code>	Melakukan pemanggilan forward

2.1.17. Kelas BidirectionalRNN

Deskripsi Kelas	
Kelas ini mengimplementasikan Bidirectional RNN	
Metode	
<code>__init__(self, units, input_dim=None, batch_size=None, activation="tanh", kernel_initializer="glorot_uniform", recurrent_initializer="orthogonal", bias_initializer="zeros", return_sequences=False)</code>	Mengkonstruksi dan menginisialisasi objek Bidirectional RNN
<code>forward(self, x: Value)</code>	Melakukan training atau inferensi
<code>load_weights(self, weights)</code>	Memuat weight yang didapatkan dari model Keras
<code>__call__(self, x)</code>	Melakukan pemanggilan forward

2.1.18. Kelas LSTM

Deskripsi Kelas	
Kelas ini mengimplementasikan LSTM untuk membangun model LSTM dengan layer-layer yang ada	
Metode	
<code>__init__(self, layers=None, batch_size=None)</code>	Mengkonstruksi dan menginisialisasi objek LSTM dengan parameter layer dan batch size
<code>add(self, layer)</code>	Menambahkan layer ke model LSTM
<code>forward(self, x: Value)</code>	Melakukan pemanggilan forward pada tiap layer pada model LSTM
<code>batch_generator(self, X, batch_size)</code>	Memproses X secara batch
<code>predict(self, x)</code>	Melakukan inferensi secara batch
<code>load_weights(self, weights)</code>	Memuat weight yang didapatkan dari model

	Keras untuk setiap layer
<code>__call__(self, x)</code>	Melakukan pemanggilan forward atau predict

2.1.19. Kelas UnidirectionalLSTM

Deskripsi Kelas	
Kelas ini mengimplementasikan Unidirectional LSTM	
Metode	
<code>__init__(self, units, input_dim=None, batch_size=None, activation="tanh", kernel_initializer="glorot_uniform", recurrent_initializer="orthogonal", bias_initializer="zeros", return_sequences=False)</code>	Mengkonstruksi dan menginisialisasi objek Unidirectional LSTM
<code>build(self, input_dim)</code>	Menginisialisasi weight
<code>init_hidden_state(self, batch_size)</code>	Menginisialisasi hidden state pertama
<code>forward(self, x: Value)</code>	Melakukan training atau inferensi
<code>load_weights(self, weights)</code>	Memuat weight yang didapatkan dari model Keras
<code>__call__(self, x)</code>	Melakukan pemanggilan forward

2.1.20. Kelas BidirectionalLSTM

Deskripsi Kelas	
Kelas ini mengimplementasikan Bidirectional LSTM	
Metode	
<code>__init__(self, units, input_dim=None, batch_size=None, activation="tanh",</code>	Mengkonstruksi dan menginisialisasi objek Bidirectional LSTM

<pre> kernel_initializer="glorot_uniform", recurrent_initializer="orthogonal", bias_initializer="zeros", return_sequences=False) </pre>	
forward(self, x: Value)	Melakukan training atau inferensi
load_weights(self, weights)	Memuat weight yang didapatkan dari model Keras
__call__(self, x)	Melakukan pemanggilan forward

2.2. Penjelasan Forward Propagation CNN

Secara umum, forward propagation untuk dilakukan dengan sekuens perhitungan sebagai berikut

<ol style="list-style-type: none"> 1. Input layer menerima data X dalam bentuk <i>batch</i> dengan dimensi (<i>batch_size</i>, <i>height</i>, <i>width</i>, <i>channels</i>). 2. Untuk setiap convolutional layer, hitung nilai <i>feature map</i> dengan operasi konvolusi antara input dan kernel, sesuai dengan langkah-langkah yang telah dijelaskan di bagian 1. $Z = \text{conv2d}(X, W, \text{stride}, \text{padding}) + b$ <p>dengan W dan b masing-masing adalah bobot kernel dan bias. Jika layer memiliki fungsi aktivasi f, maka output layer adalah $f(Z)$.</p> <ol style="list-style-type: none"> 3. Untuk setiap pooling layer, lakukan <i>downsampling</i> dari <i>feature map</i> menggunakan <i>max pooling</i> atau <i>average pooling</i>. <i>Output pooling layer</i> menjadi <i>input</i> untuk layer berikutnya. 4. Setelah semua layer konvolusi dan pooling selesai, <i>Flatten output</i> menjadi bentuk vektor (<i>batch_size</i>, <i>n_flat_features</i>) untuk digunakan oleh <i>fully connected layer</i> (<i>dense</i>). 5. Untuk setiap dense layer (<i>fully connected</i>), hitung nilai $Z = X @ W.T + b$ seperti pada FFNN. Jika layer memiliki fungsi aktivasi f, maka output layer adalah $f(Z)$. Jika bukan layer terakhir, hasil $f(Z)$ digunakan sebagai input ke dense layer berikutnya. 6. Hasil akhir dari forward propagation adalah output dengan dimensi (<i>batch_size</i>, <i>n_label</i>) dari dense layer terakhir.

Perhitungan ini melibatkan beberapa kelas, yaitu:

1. Kelas CNN

Menginisiasi forward propagation dan mengirim output dari satu layer ke layer lainnya, termasuk urutan konvolusi → pooling → flatten → dense.

2. Kelas Conv2D dan Pooling

Melakukan perhitungan konvolusi dan pooling serta memanggil fungsi aktivasi jika ada.

3. Kelas Flatten

Melakukan *flattening* terhadap hasil konvolusi dan juga *pooling*

4. Kelas DenseLayer

Melakukan perhitungan matriks biasa seperti pada FFNN.

2.3. Penjelasan Forward Propagation RNN

Secara umum, forward propagation dengan perhitungan sebagai berikut

1. Input layer menerima instans data X (batch_size, seq_len, feature_size)
2. Pada RNN layer untuk setiap seq,
 - a. Hitung nilai $h = f(W_{xh} @ x_t + W_{hh} @ h.T + b_{xh}).T$, dengan W_{xh} dan W_{hh} adalah bobot dengan dimensi (output_size, input_size) dan (output_size, output_size) dan b_{xh} adalah bias dengan dimensi (output_size, 1), f adalah fungsi aktivasi.
 - b. Simpan nilai h dan gunakan untuk perhitungan h pada seq selanjutnya.
 - c. Jika return sequence True, maka akan mengembalikan semua hasil perhitungan h pada setiap timestep. Hasil forward propagation adalah output (batch_size, seq_len, units).
 - d. Jika return sequence False, maka akan mengembalikan hasil perhitungan h pada timestep terakhir. Hasil forward propagation adalah output (batch_size, units).
3. Pada output layer, hasil forward propagation adalah (batch_size, n_label).

Perhitungan ini melibatkan beberapa kelas, yaitu:

1. Kelas FFNN yang menginisiasi forward propagation dan mengirim output suatu layer ke layer lain
2. Kelas Layer yang memanggil perhitungan fungsi aktivasi
3. Kelas Value yang melakukan perhitungan fungsi aktivasi

2.4. Penjelasan Forward Propagation LSTM

Secara umum, forward propagation dengan perhitungan sebagai berikut

1. Input layer menerima data dengan bentuk:
 $X.shape = (batch_size, seq_len, feature_size)$
2. LSTM Layer:
Untuk setiap langkah waktu (timestep) t dalam seq_len , dilakukan perhitungan sebagai berikut:
 - a. Perhitungan Gate
Hitung empat komponen utama LSTM:
 - Input Gate:
 $i_t = \text{sigmoid}(W_{xi} @ x_t + W_{hi} @ h_{t-1} + b_i)$
 - Forget Gate:
 $f_t = \text{sigmoid}(W_{xf} @ x_t + W_{hf} @ h_{t-1} + b_f)$

- Output Gate:

$$o_t = \text{sigmoid}(W_{xo} @ x_t + W_{ho} @ h_{t-1} + b_o)$$
 - Candidate Cell State:

$$\tilde{c}_t = \tanh(W_{xc} @ x_t + W_{hc} @ h_{t-1} + b_c)$$
 - b. Update States
 - Update cell state:

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$
 - Update hidden state:

$$h_t = o_t * \tanh(c_t)$$
 - c. Return Output
 - Jika return_sequences=True:
Simpan semua h_t pada setiap timestep $\rightarrow \text{output.shape} = (\text{batch_size}, \text{seq_len}, \text{units})$
 - Jika return_sequences=False:
Ambil hanya h_t pada timestep terakhir $\rightarrow \text{output.shape} = (\text{batch_size}, \text{units})$
3. Output Layer
Output dari LSTM bisa diteruskan ke dense layer atau layer lainnya.
Bentuk output: $\text{output.shape} = (\text{batch_size}, \text{n_label})$

Perhitungan ini melibatkan beberapa kelas, yaitu:

1. Kelas FFNN yang menginisiasi forward propagation dan mengirim output suatu layer ke layer lain
2. Kelas Layer yang memanggil perhitungan fungsi aktivasi
3. Kelas Value yang melakukan perhitungan fungsi aktivasi

2.5. Penjelasan Backward Propagation dan Weight Update

Implementasi backward propagation dilakukan pada kelas Value pada modul autodiff. Backward propagation dilakukan setiap kali pemrosesan forward propagation suatu batch selesai dilakukan. Tahapan backward propagation dan weight update adalah sebagai berikut:

1. Hitung nilai loss dari nilai prediksi hasil forward propagation dan true label. Nilai loss bisa saja memiliki regularization term.
2. Panggil metode backward() terhadap nilai loss yang merupakan objek Value. Di dalam metode backward(),
 - a. Bangun topological graph dari semua operasi.
 - b. Untuk setiap objek Value elemen topological graph, dari elemen paling belakang
 - i. Panggil metode _backward() yang disimpan pada atribut _backward. Metode ini akan mengisi nilai atribut grad dari child node objek pemanggil _backward() dengan memanfaatkan aturan rantai, secara spesifik menambahkan nilai gradien child node dengan nilai vector-Jacobian product.
3. Update nilai bobot/bias sebesar negatif gradien dengan proporsi learning rate.

2.6. Hasil Pengujian CNN

Berikut merupakan kode yang digunakan untuk melakukan pengujian terhadap model CNN

```
def create_cnn_model(
    input_shape,
    num_classes,
    conv_layers=2,
    filters_list=[32, 64],
    kernel_size_list=[3, 3],
    pooling_type="max"
):
    model = models.Sequential()

    for i in range(conv_layers):
        filters = filters_list[i] if i < len(filters_list) else
filters_list[-1]
        kernel_size = kernel_size_list[i] if i <
len(kernel_size_list) else kernel_size_list[-1]

        if i == 0:
            model.add(layers.Conv2D(filters, (kernel_size,
kernel_size), activation='relu', input_shape=input_shape,
padding='same'))
        else:
            model.add(layers.Conv2D(filters, (kernel_size,
kernel_size), activation='relu', padding='same'))

        if pooling_type == "max":
            model.add(layers.MaxPooling2D(pool_size=(2, 2)))
        else:
            model.add(layers.AveragePooling2D(pool_size=(2, 2)))

    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(num_classes, activation='softmax'))

    model.compile(
        optimizer='adam',
        loss=tf.keras.losses.SparseCategoricalCrossentropy(),
        metrics=['accuracy']
    )
    return model
```

Secara *default*, model yang dibuat akan memiliki:

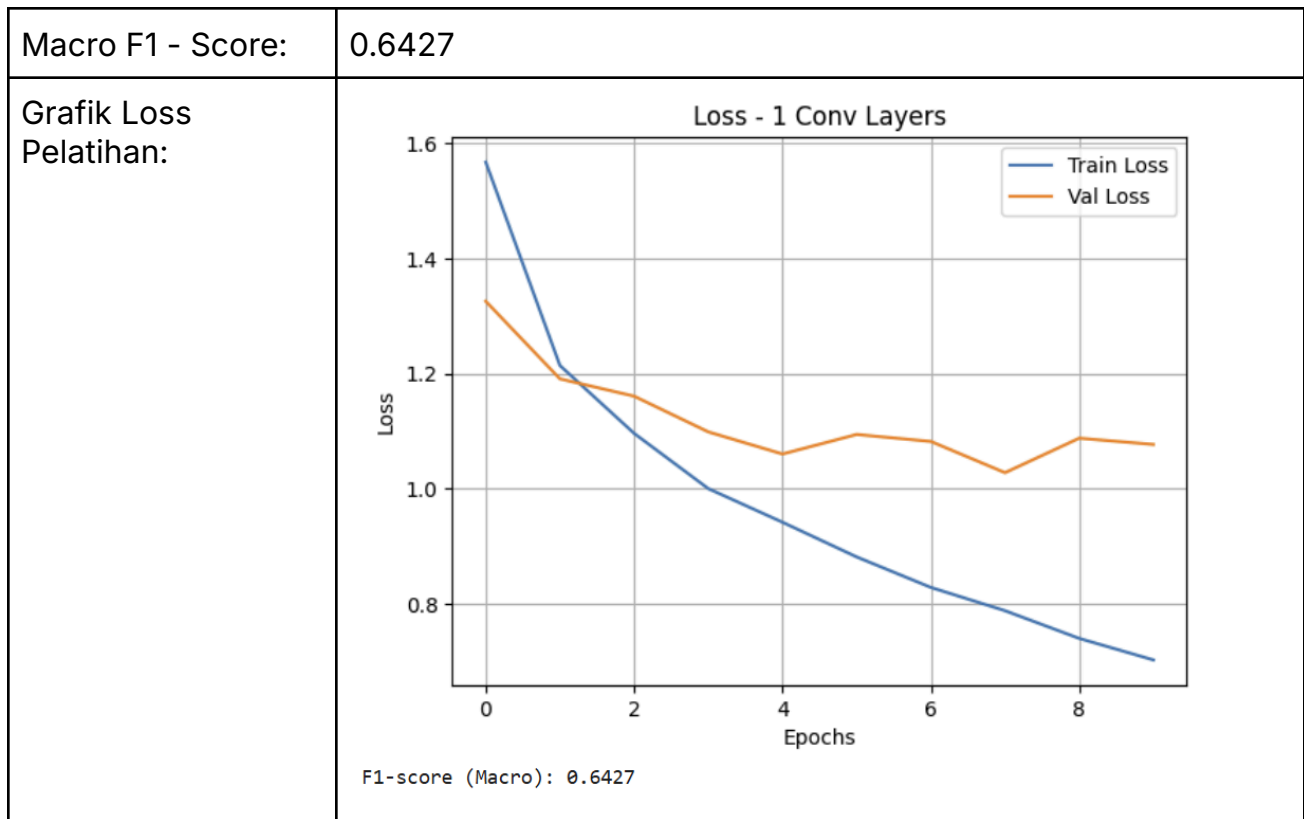
1. 2 layer konvolusi dengan ukuran kernel 3 x 3, padding "same" yang artinya menjaga ukuran input, jumlah kernel 32 dan 64, serta fungsi aktivasi relu

2. 2 layer *MaxPooling* dengan *pool size* 2 x 2
3. 1 layer Flattening
4. 1 layer dense dengan aktivasi relu dan 64 neuron
5. 1 layer terakhir dengan fungsi aktivasi softmax dengan 10 neuron

2.6.1. Pengaruh jumlah layer konvolusi

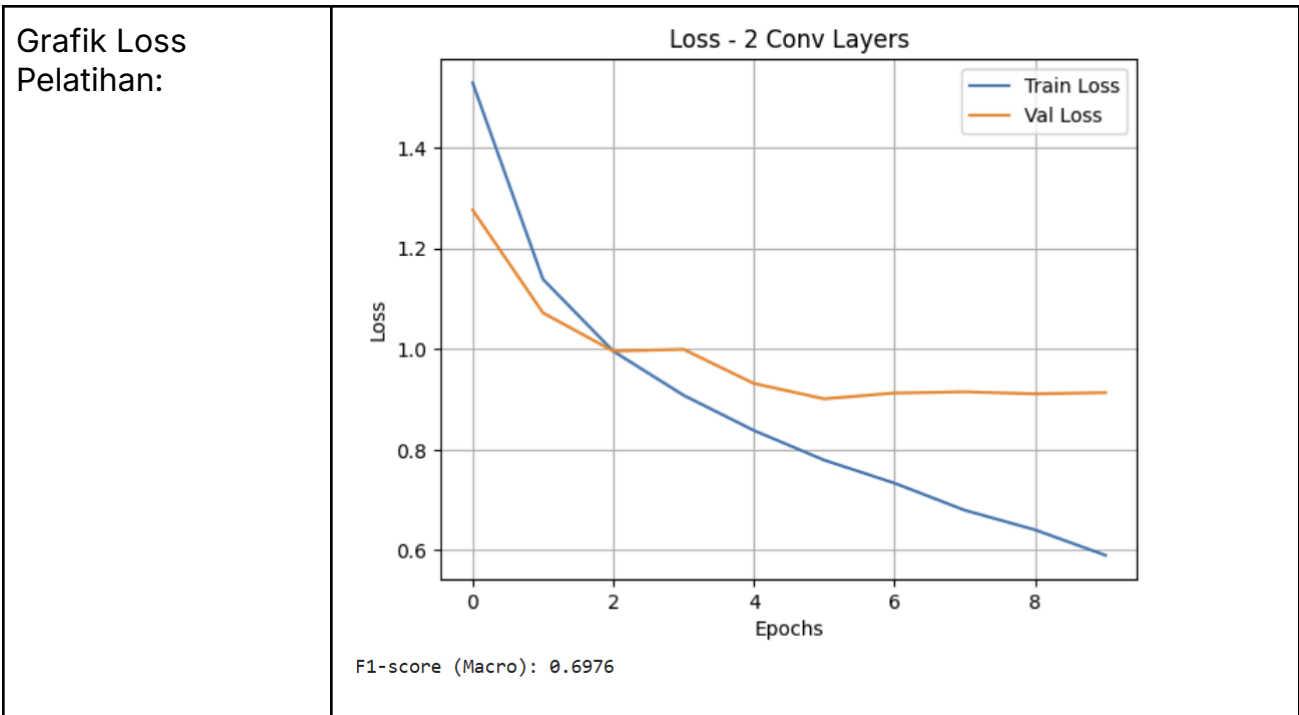
Pada pengujian ini, akan diuji 3 jumlah layer konvolusi yaitu 1,2,3 dan jumlah kernel untuk masing-masing layer adalah 64, dengan ukuran kernel adalah 3 x 3.

2.6.1.1 Jumlah Layer Konvolusi = 1

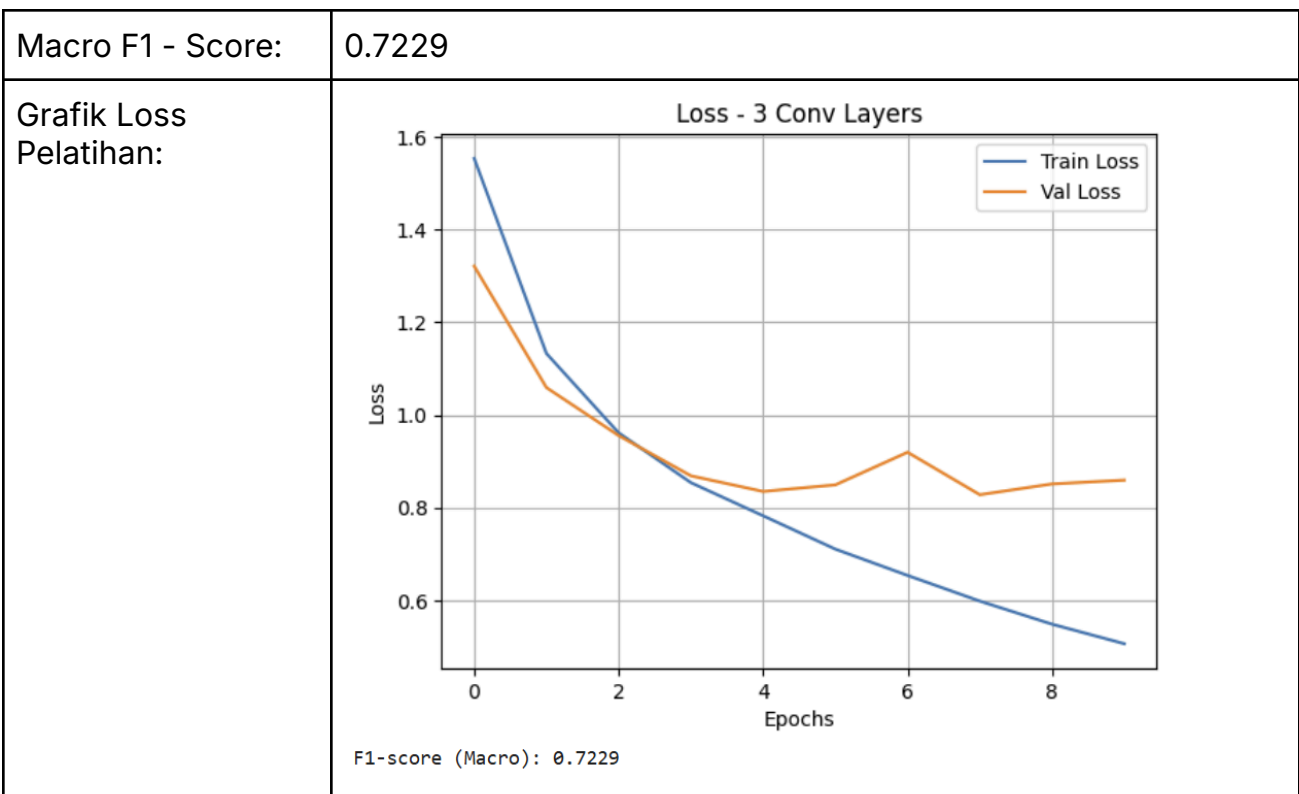


2.6.1.2 Jumlah Layer Konvolusi = 2

Macro F1 - Score:	0.6976
-------------------	--------



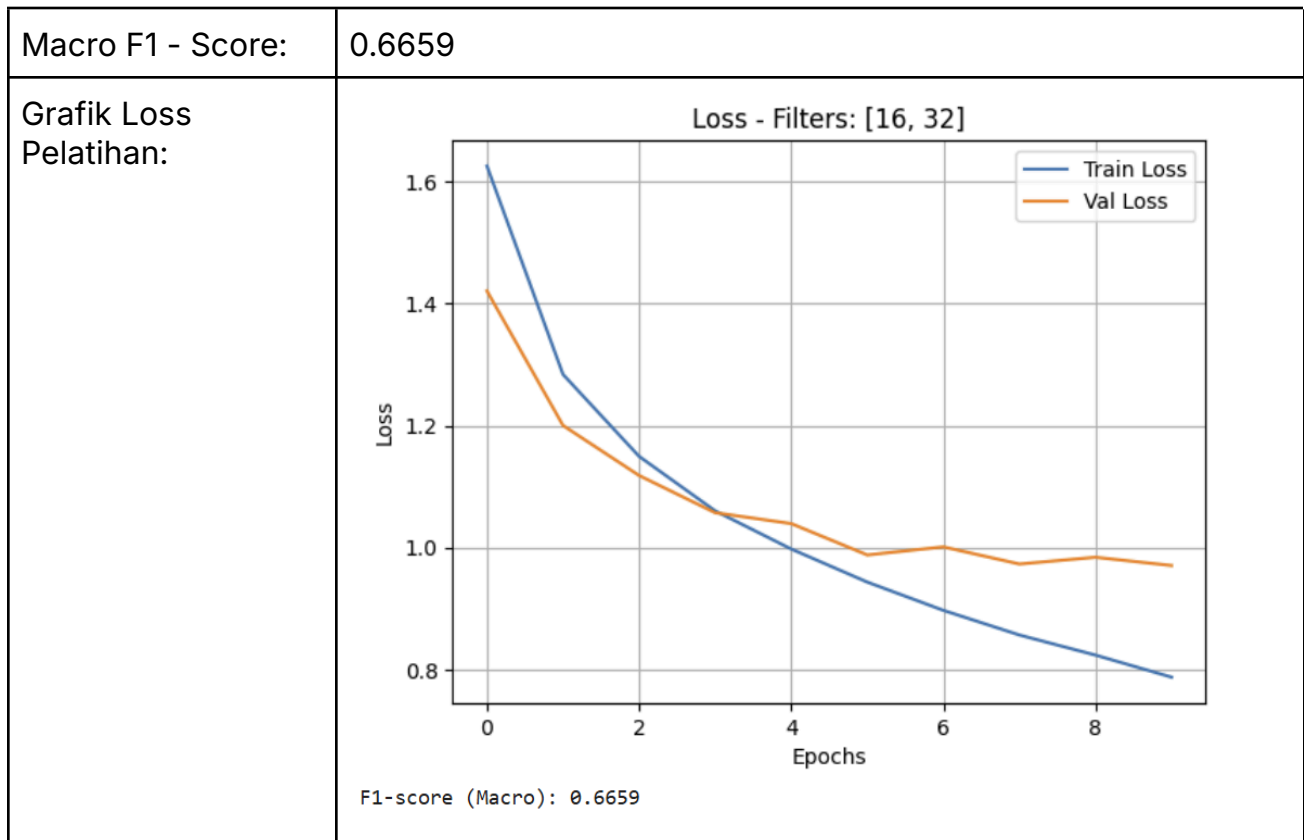
2.6.1.3 Jumlah Layer Konvolusi = 3



2.6.2. Pengaruh banyak kernel per layer konvolusi

Pada pengujian ini, akan digunakan jumlah kernel yang berbeda-beda. Dari model default yang digunakan, akan diganti jumlah kernel di layernya (2 layer konvolusi).

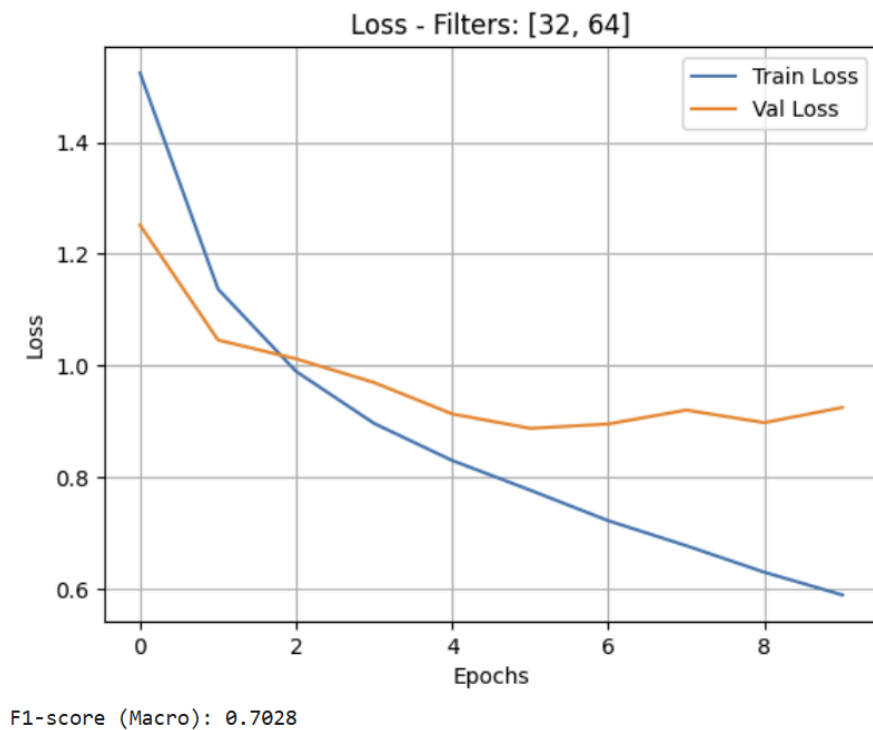
2.6.2.1 Jumlah Kernel 16 dan 32



2.6.2.2 Jumlah Kernel 32 dan 64

Macro F1 - Score:	0.7028
-------------------	--------

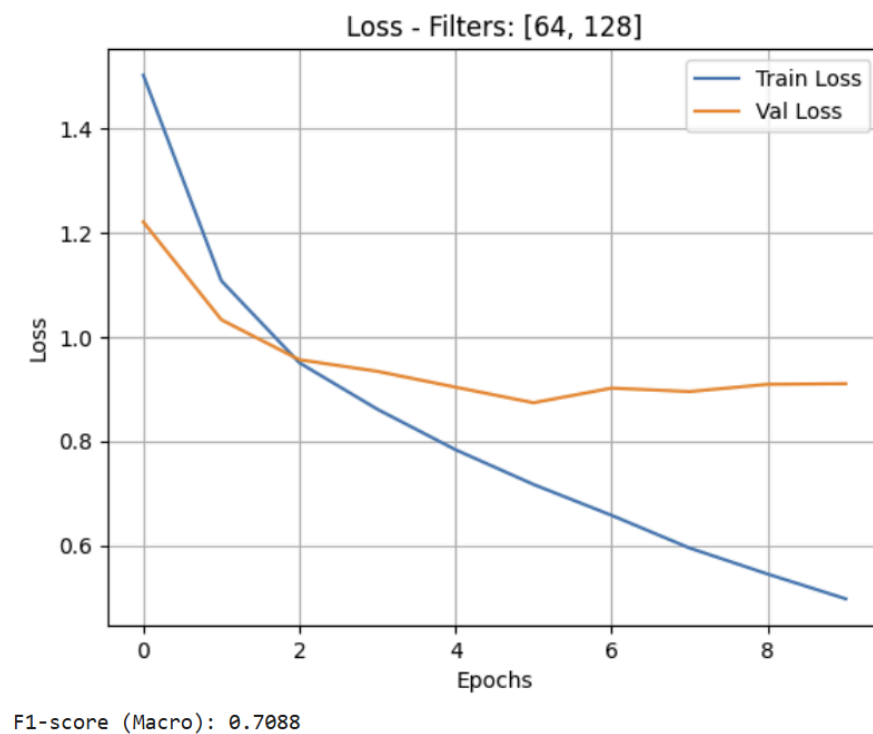
Grafik Loss Pelatihan:



2.6.2.3 Jumlah Kernel 64 dan 128

Macro F1 - Score: 0.7088

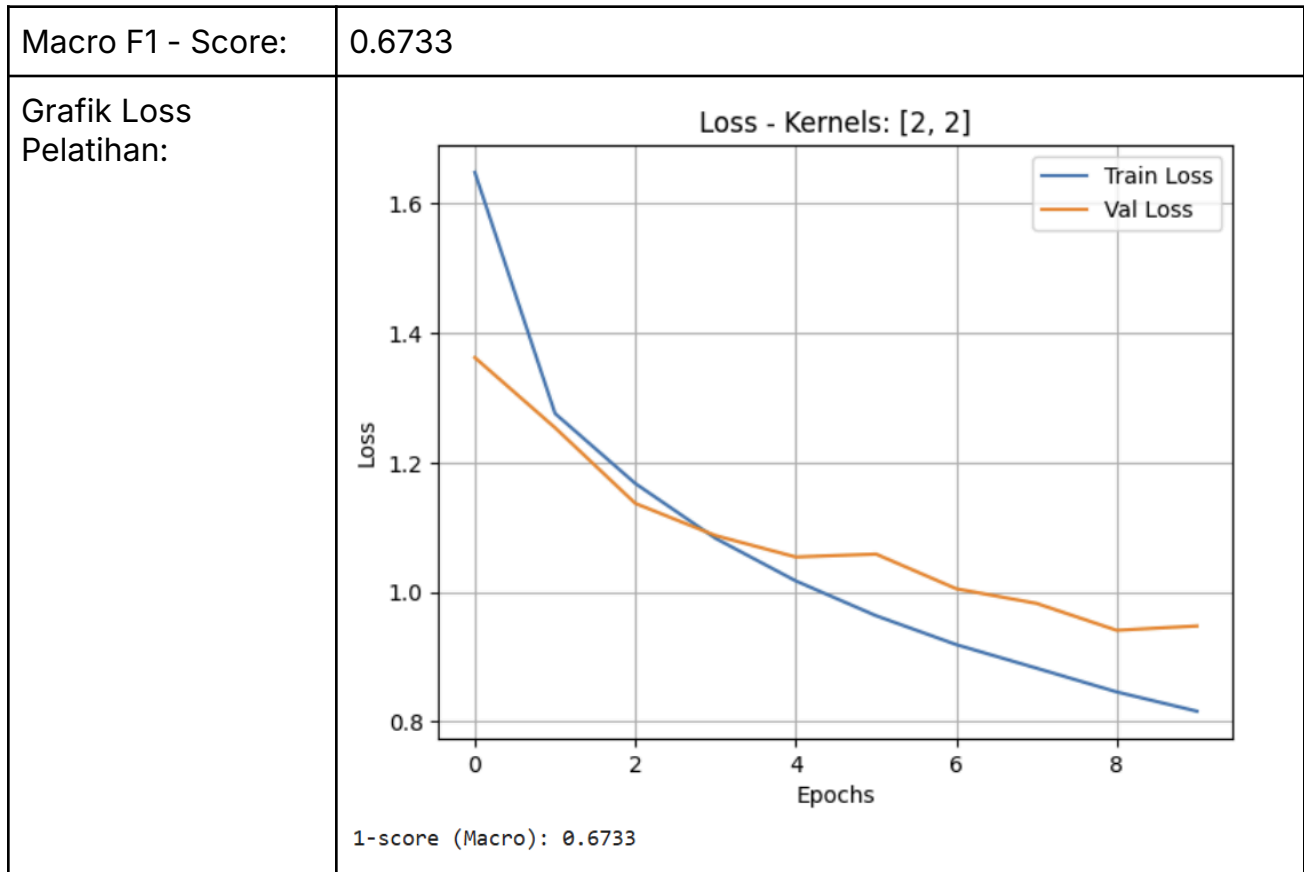
Grafik Loss Pelatihan:



2.6.3. Pengaruh ukuran kernel per layer konvolusi

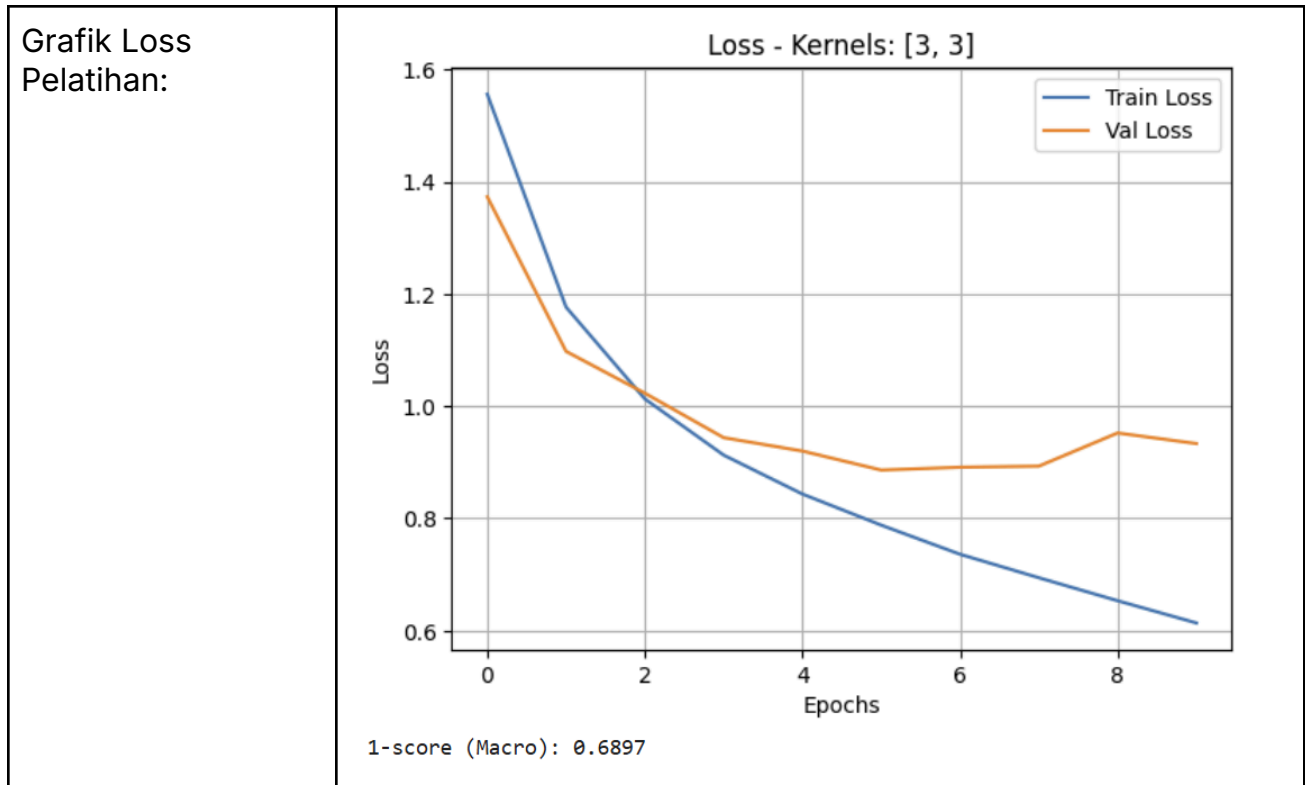
Pada pengujian ini, akan digunakan variasi dari ukuran kernel. Akan digunakan ukuran kernel 2 x 2 hingga 6 x 6.

2.6.3.1 Ukuran kernel = 2 x 2

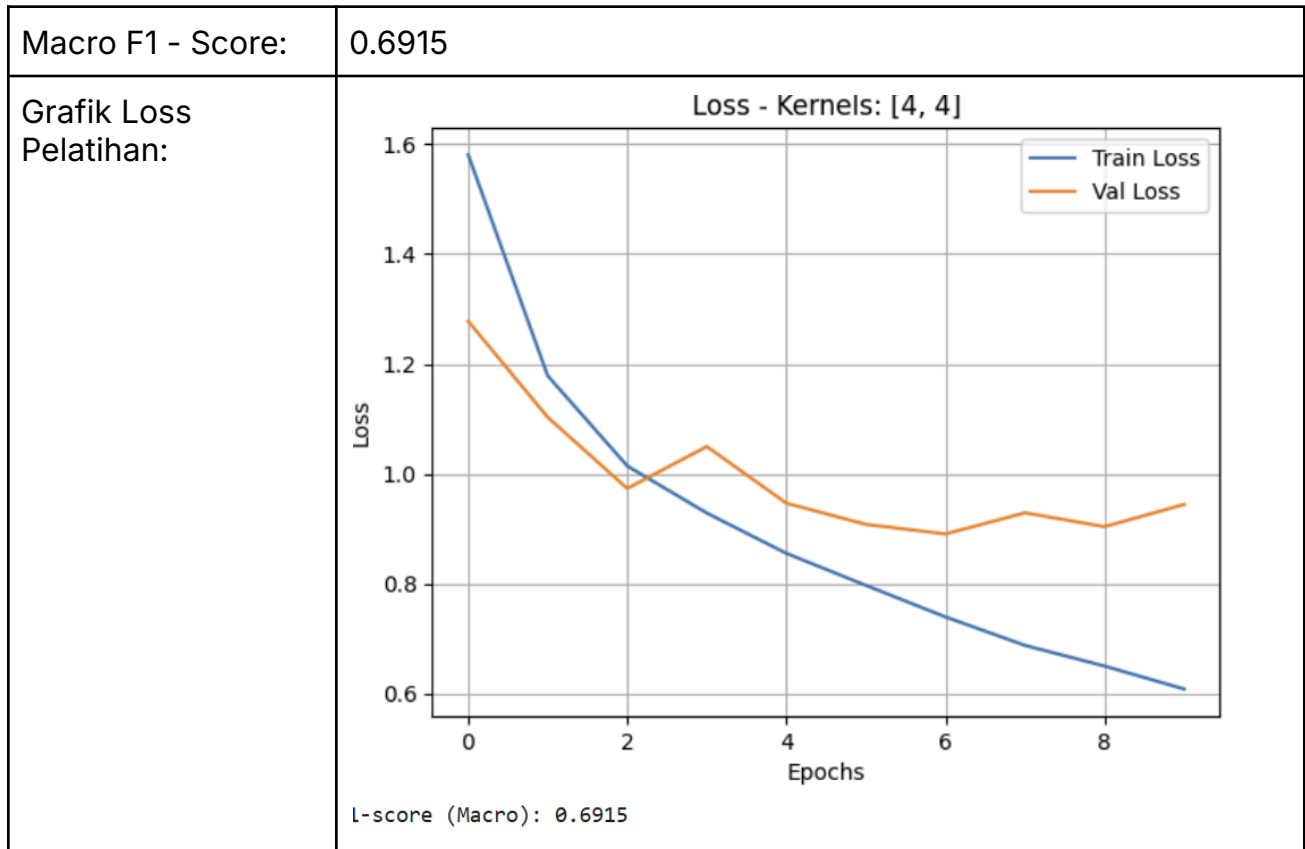


2.6.3.2 Ukuran kernel = 3 x 3

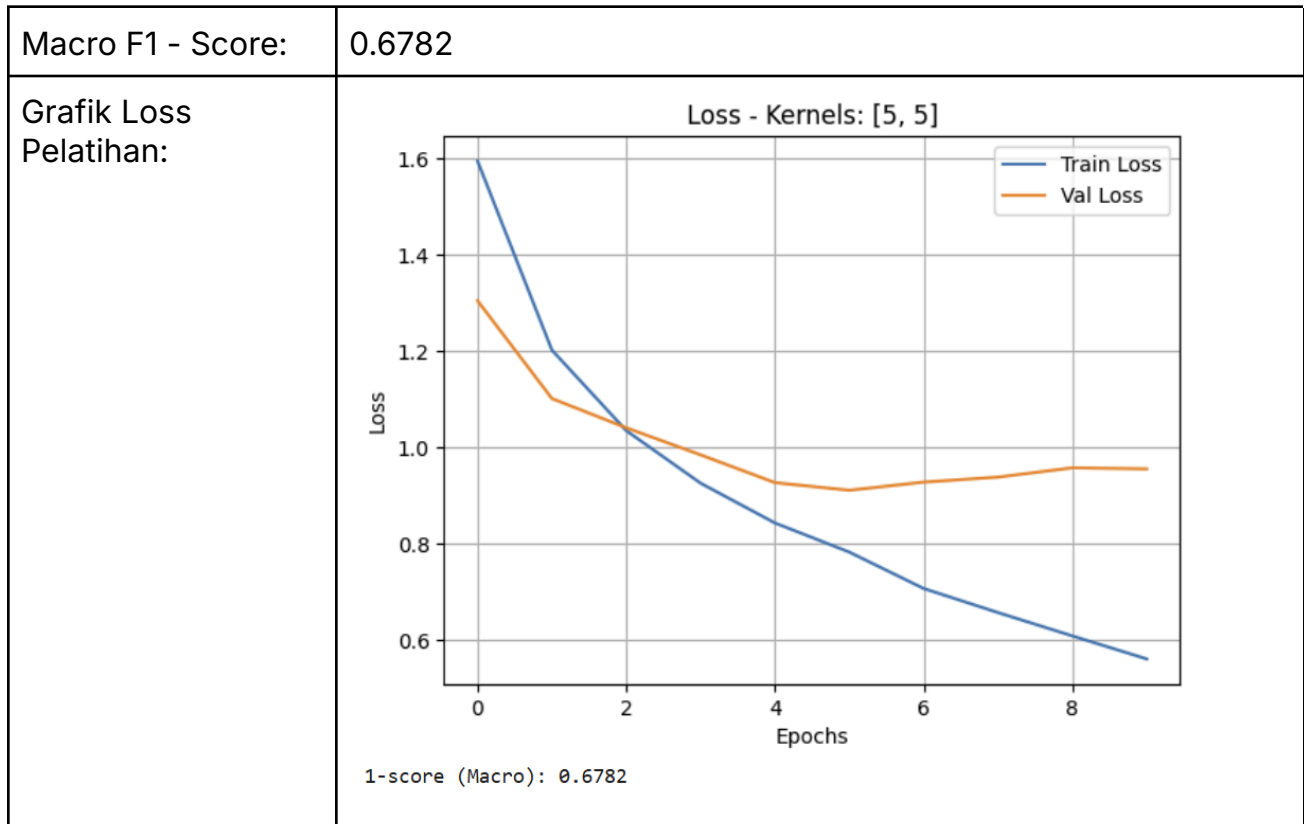
Macro F1 - Score:	0.6897
-------------------	--------



2.6.3.3 Ukuran kernel = 4 x 4



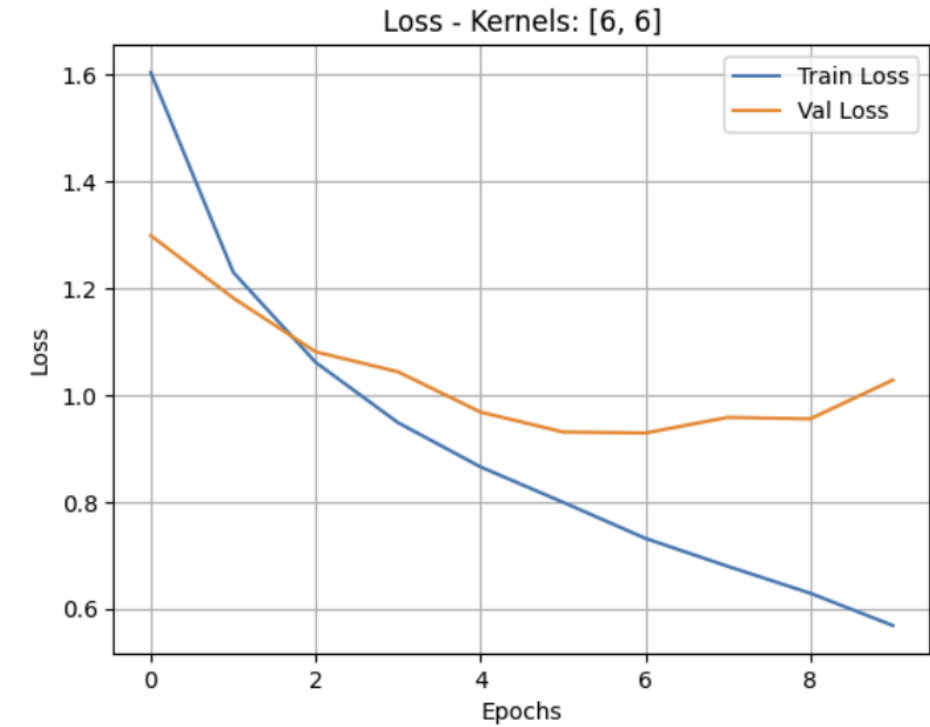
2.6.3.4 Ukuran Kernel 5 x 5



2.6.3.5 Ukuran kernel = 6 x 6

Macro F1 - Score:	0.6710
-------------------	--------

Grafik Loss Pelatihan:



F1-score (Macro): 0.6710

2.6.4. Pengaruh Jenis Pooling

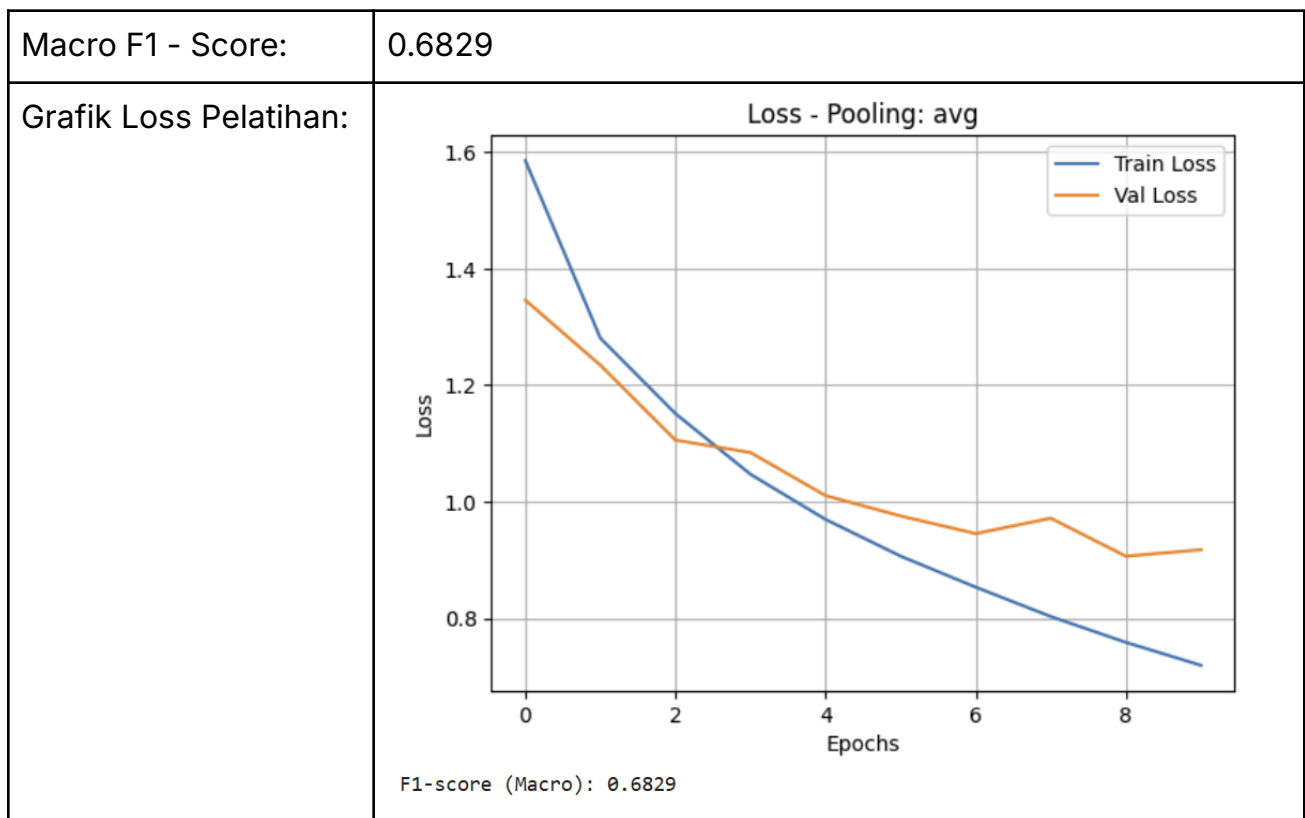
Akan diuji 2 jenis *pooling* yang ada yaitu *average* dan juga *max pooling*.

2.6.4.1 Max Pooling

Macro F1 - Score:	0.7014
-------------------	--------



2.6.4.2. Average Pooling



2.6.5. Perbandingan dengan Model Keras CNN

```
def final_cnn_model(input_shape=(32, 32, 3), num_classes=10):
    model = models.Sequential()

    model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=input_shape))
    model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D(pool_size=(2, 2)))

    model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D(pool_size=(2, 2)))

    model.add(layers.Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D(pool_size=(2, 2)))

    model.add(layers.Flatten())
    model.add(layers.Dense(512, activation='relu'))
    model.add(layers.Dense(num_classes, activation='softmax'))

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

    return model

model = final_cnn_model()
history = model.fit(
    x_train, y_train,
    epochs=15,
    validation_data=(x_val, y_val),
    batch_size=64,
    verbose=1
)
```

Gambar 2.5.5.x Gambar Model Keras yang digunakan untuk *training*

Karena kedalaman arsitektur model serta keterbatasan sumber daya GPU yang tersedia, terutama akibat adanya enam lapisan konvolusi, kami tidak dapat melakukan proses forward terhadap seluruh data uji menggunakan model kami sendiri. Oleh karena itu, kami memutuskan untuk melakukan pengujian hanya pada 1000 data pertama dari dataset uji yang disediakan.

```

You, 1 second ago | 1 author (You)
1 import numpy as np      Import "numpy" could not be resolved
2 import tensorflow as tf  Import "tensorflow" could not be resolved
3 from sklearn.metrics import f1_score      Import "sklearn.metrics" could not be resolved from
4 from src.models.cnn_normal import CNNModel
5 from src.layers.cnn.usenumpy.conv import Conv2D
6 from src.layers.cnn.usenumpy.dense import DenseLayer
7 from src.layers.cnn.usenumpy.flatten import Flatten
8 from src.layers.cnn.usenumpy.pooling import Pooling
9 from src.test.cnn.load_cnn_data import load_cifar10_custom
10 from src.utils.loss import CategoricalCrossEntropyLoss
11
12 (x_train, y_train), (x_test, y_test) = load_cifar10_custom()
13 x_input = x_test[:10] # Use one sample for comparison
14
15 x_input_keras = x_input.transpose(0, 2, 3, 1) # Convert from NCHW to NHWC for Keras
16
17 # === Step 2: Load weights ===
18 keras_weights_path = "./src/test/cnn/keras_cnn_weights.npz"
19 weights_np = np.load(keras_weights_path, allow_pickle=True)
20 weights_list = [weights_np[k] for k in weights_np.files]
21
You, 2 hours ago | 1 author (You)
22 class CNNModelWithIntermediates(CNNModel):
23     def predict_with_intermediates(self, x):
24         intermediates = []
25         for layer in self.layers:      You, 2 hours ago • feat: cnn forward working ...
26             x = layer(x)
27             intermediates.append(x)
28     return intermediates

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS VESSEL LOGS GITLENS

```

Processing batch 201/1000
Processing batch 301/1000
Processing batch 401/1000
Processing batch 501/1000
Processing batch 601/1000
Processing batch 701/1000
Processing batch 801/1000
Processing batch 901/1000
Processing batch 1/1000
Processing batch 101/1000
Processing batch 201/1000
Processing batch 301/1000
Processing batch 401/1000
Processing batch 501/1000
Processing batch 601/1000
Processing batch 701/1000
Processing batch 801/1000
Processing batch 901/1000
Processing batch 1/1000
Processing batch 101/1000
Processing batch 201/1000
Processing batch 301/1000
Processing batch 401/1000
Processing batch 501/1000
Processing batch 601/1000
Processing batch 701/1000
Processing batch 801/1000
Processing batch 901/1000
32/32 3s 86ms/step
Custom Model F1 Score (macro) [For 1k data]: 0.7361
Keras Model F1 Score (macro) [For 1k data]: 0.7361
PS C:\Users\Imanuel Girsang\OneDrive - Institut Teknologi Bandung\Documents\IF-General\Semester-6\ML\IF3270-

```

Dari hasil percobaan didapat Macro F1 Score dari 1000 dataset pertama menggunakan model sendiri maupun keras sama - sama berada di angka 0.7361.

```

1 operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI
Epoch 1/10
1/1 1s 1s/step - accuracy: 0.0000e+00 - loss: 7.1528
Epoch 2/10
1/1 0s 69ms/step - accuracy: 0.0000e+00 - loss: 3.6319
Epoch 3/10
1/1 0s 63ms/step - accuracy: 1.0000 - loss: 0.7803
Epoch 4/10
1/1 0s 61ms/step - accuracy: 1.0000 - loss: 0.0477
Epoch 5/10
1/1 0s 69ms/step - accuracy: 1.0000 - loss: 0.0036
Epoch 6/10
1/1 0s 79ms/step - accuracy: 1.0000 - loss: 4.0308e-04
Epoch 7/10
1/1 0s 70ms/step - accuracy: 1.0000 - loss: 6.3298e-05
Epoch 8/10
1/1 0s 71ms/step - accuracy: 1.0000 - loss: 1.2755e-05
Epoch 9/10
1/1 0s 79ms/step - accuracy: 1.0000 - loss: 3.2186e-06
Epoch 10/10
1/1 0s 89ms/step - accuracy: 1.0000 - loss: 8.3446e-07

```

```

PS C:\Users\Imanuel Girsang\OneDrive - Institut Teknologi Bandung\Documents\IF-General\Semester-6\ML\IF3270- ---CMI-RNN---50b) python -m src.test.cnn.cnn
Processing batch 1/1
Epoch 1/10 - loss: 7.1528
Processing batch 1/1
Epoch 2/10 - loss: 3.6319
Processing batch 1/1
Epoch 3/10 - loss: 0.7803
Processing batch 1/1
Epoch 4/10 - loss: 0.0477
Processing batch 1/1
Epoch 5/10 - loss: 0.0036
Processing batch 1/1
Epoch 6/10 - loss: 0.0477
Processing batch 1/1
Epoch 7/10 - loss: 0.0036
Processing batch 1/1
Epoch 8/10 - loss: 0.0036
Processing batch 1/1
Epoch 9/10 - loss: 0.0036
Processing batch 1/1
Epoch 10/10 - loss: 0.0000
Processing batch 1/1
Epoch 10/10 - loss: 0.0000

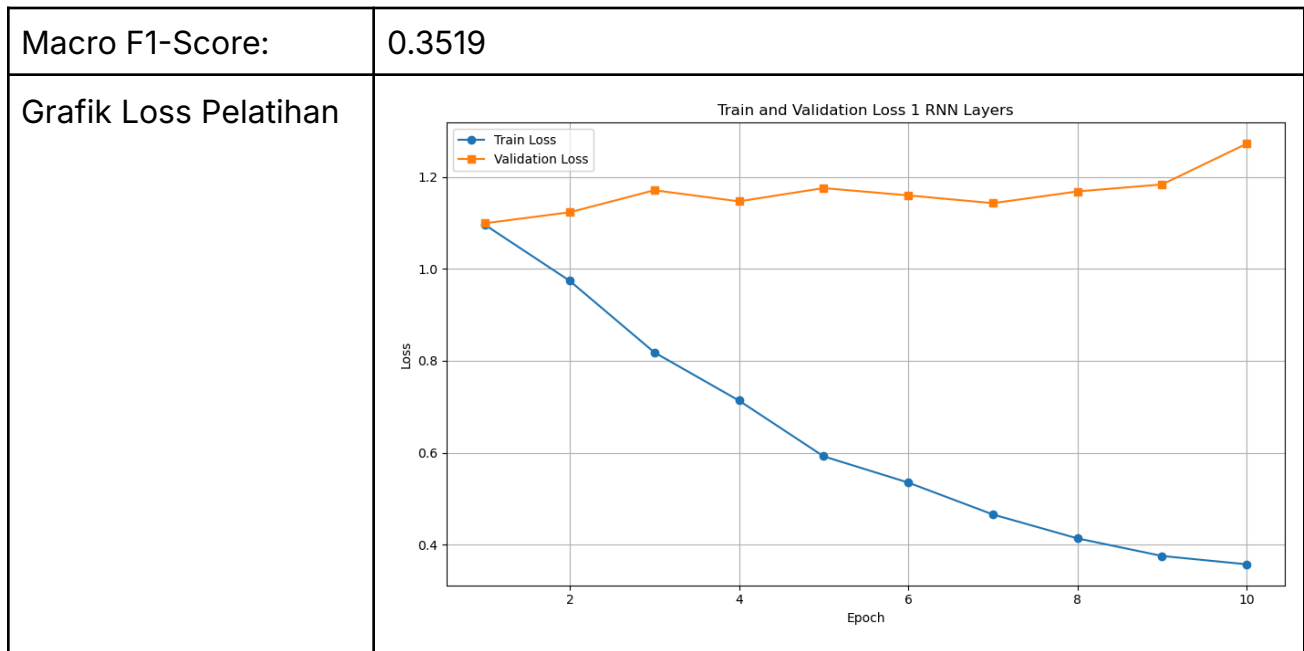
```

Dari hasil percobaan berdasarkan gambar di atas, didapatkan bahwa nilai loss di tiap epoch pada training Keras dan implementasi from scratch sama.

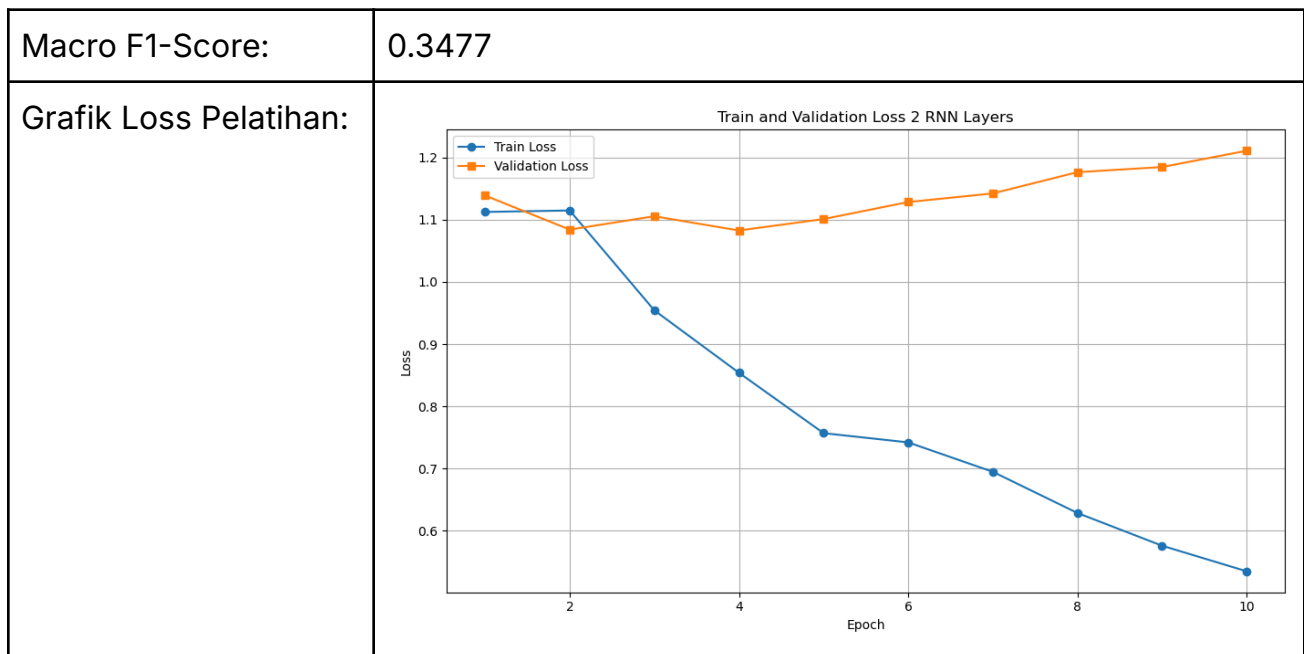
2.6. Hasil Pengujian RNN

2.6.6. Pengaruh Jumlah layer RNN

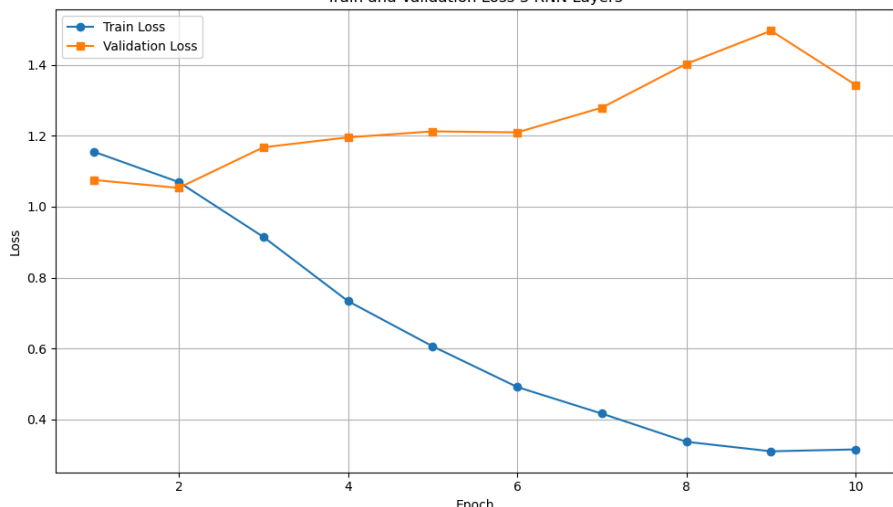
2.6.1.1. 1 Layer RNN



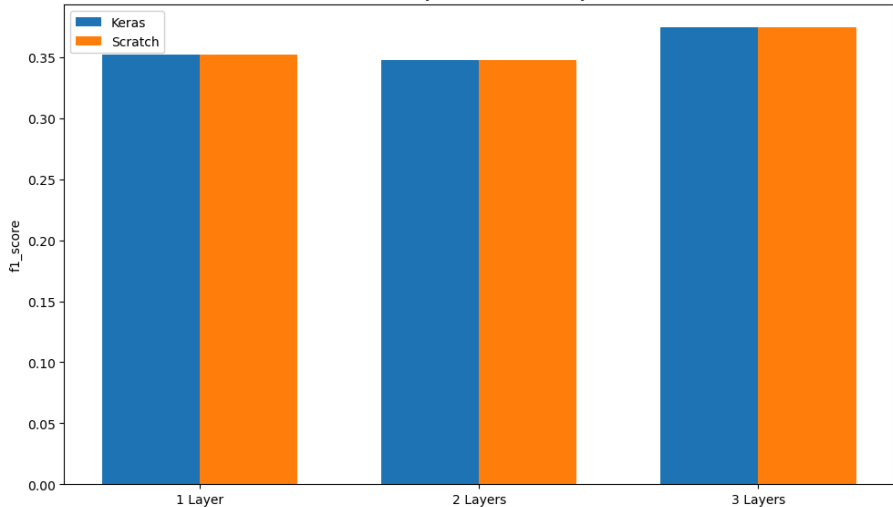
2.6.1.2. 2 Layer RNN



2.6.1.3. 3 Layer RNN

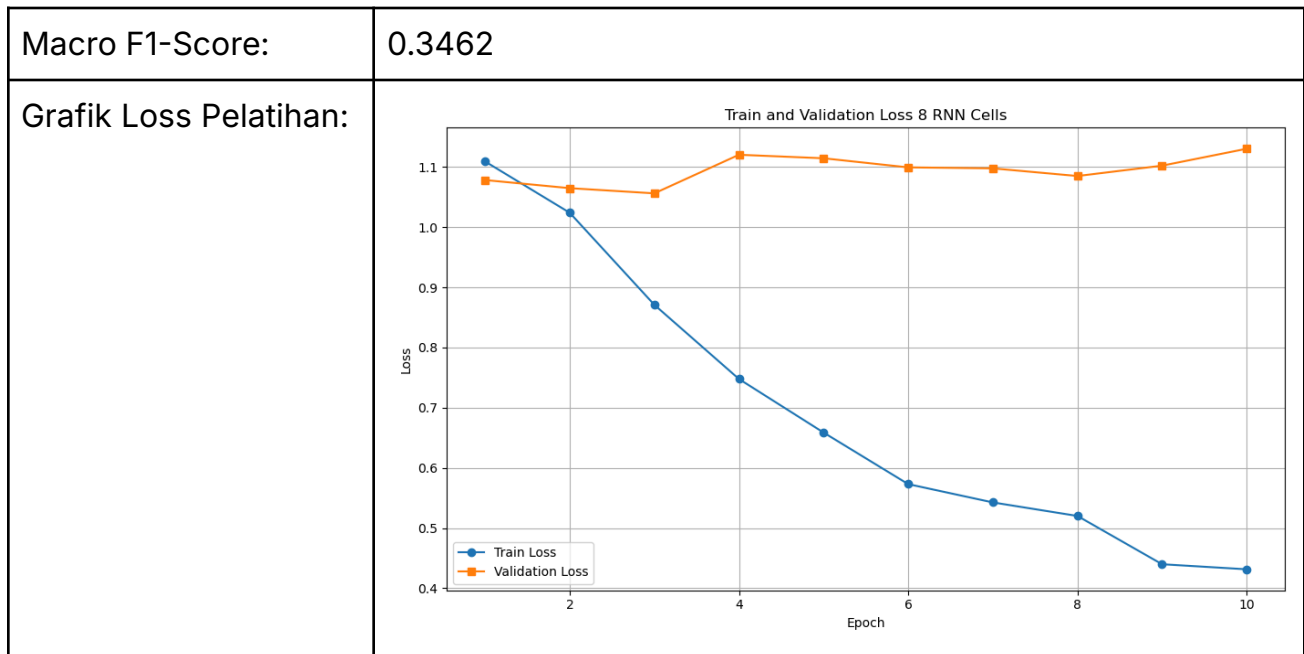
Macro F1-Score:	0.3744																														
Grafik Loss Pelatihan:	<div><p>Train and Validation Loss 3 RNN Layers</p><table><thead><tr><th>Epoch</th><th>Train Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>2</td><td>1.15</td><td>1.08</td></tr><tr><td>3</td><td>1.08</td><td>1.18</td></tr><tr><td>4</td><td>0.92</td><td>1.20</td></tr><tr><td>5</td><td>0.80</td><td>1.22</td></tr><tr><td>6</td><td>0.70</td><td>1.22</td></tr><tr><td>7</td><td>0.62</td><td>1.28</td></tr><tr><td>8</td><td>0.55</td><td>1.40</td></tr><tr><td>9</td><td>0.52</td><td>1.35</td></tr><tr><td>10</td><td>0.32</td><td>1.32</td></tr></tbody></table></div>	Epoch	Train Loss	Validation Loss	2	1.15	1.08	3	1.08	1.18	4	0.92	1.20	5	0.80	1.22	6	0.70	1.22	7	0.62	1.28	8	0.55	1.40	9	0.52	1.35	10	0.32	1.32
Epoch	Train Loss	Validation Loss																													
2	1.15	1.08																													
3	1.08	1.18																													
4	0.92	1.20																													
5	0.80	1.22																													
6	0.70	1.22																													
7	0.62	1.28																													
8	0.55	1.40																													
9	0.52	1.35																													
10	0.32	1.32																													

2.6.1.4. F1-Score Keras vs Scratch

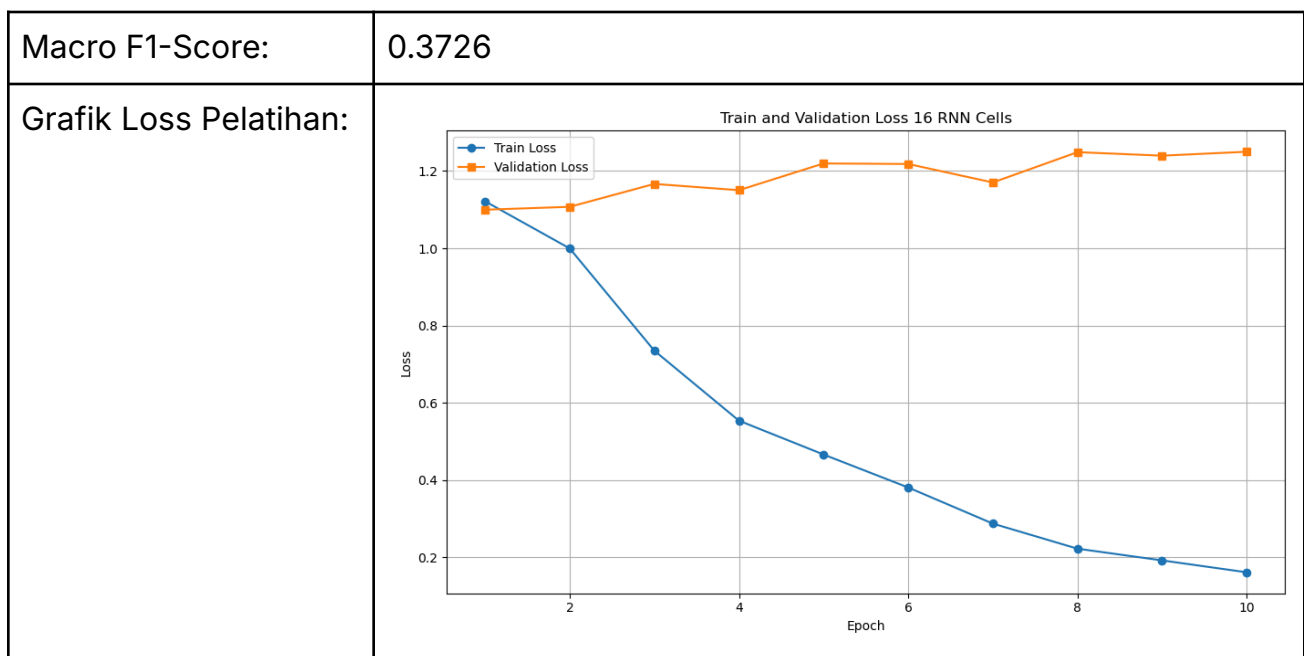
Macro F1-Score Keras:	0.3519, 0.3477, 0.3744												
Macro F1-Score Scratch:	0.3519, 0.3477, 0.3744												
Grafik Macro F1-Score:	<div><p>F1 Score by Number of RNN Layers</p><table><thead><tr><th>Number of RNN Layers</th><th>Keras</th><th>Scratch</th></tr></thead><tbody><tr><td>1 Layer</td><td>0.3519</td><td>0.3519</td></tr><tr><td>2 Layers</td><td>0.3477</td><td>0.3477</td></tr><tr><td>3 Layers</td><td>0.3744</td><td>0.3744</td></tr></tbody></table></div>	Number of RNN Layers	Keras	Scratch	1 Layer	0.3519	0.3519	2 Layers	0.3477	0.3477	3 Layers	0.3744	0.3744
Number of RNN Layers	Keras	Scratch											
1 Layer	0.3519	0.3519											
2 Layers	0.3477	0.3477											
3 Layers	0.3744	0.3744											

2.6.7. Pengaruh Banyak Cell RNN per Layer

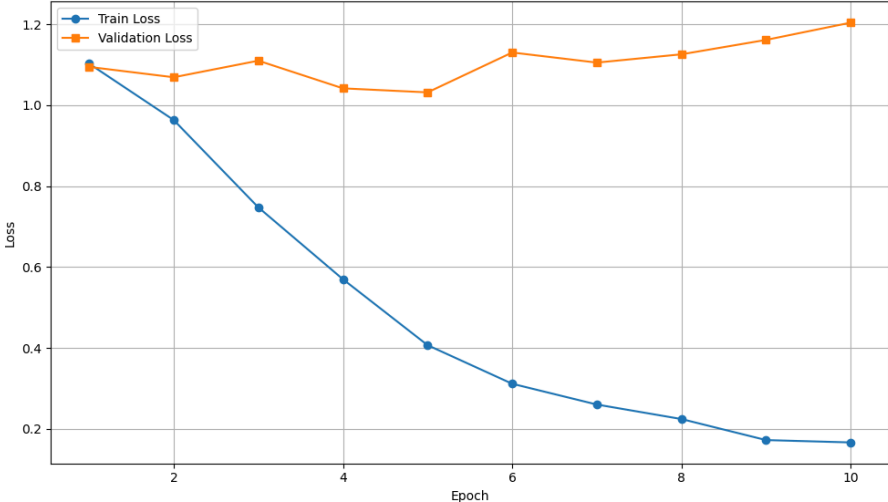
2.6.2.1. 8 Cell RNN



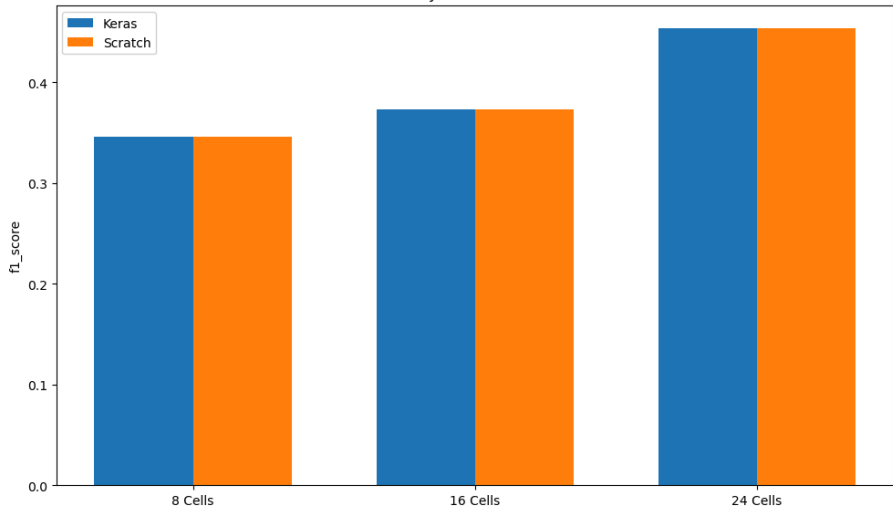
2.6.2.2. 16 Cell RNN



2.6.2.3. 24 Cell RNN

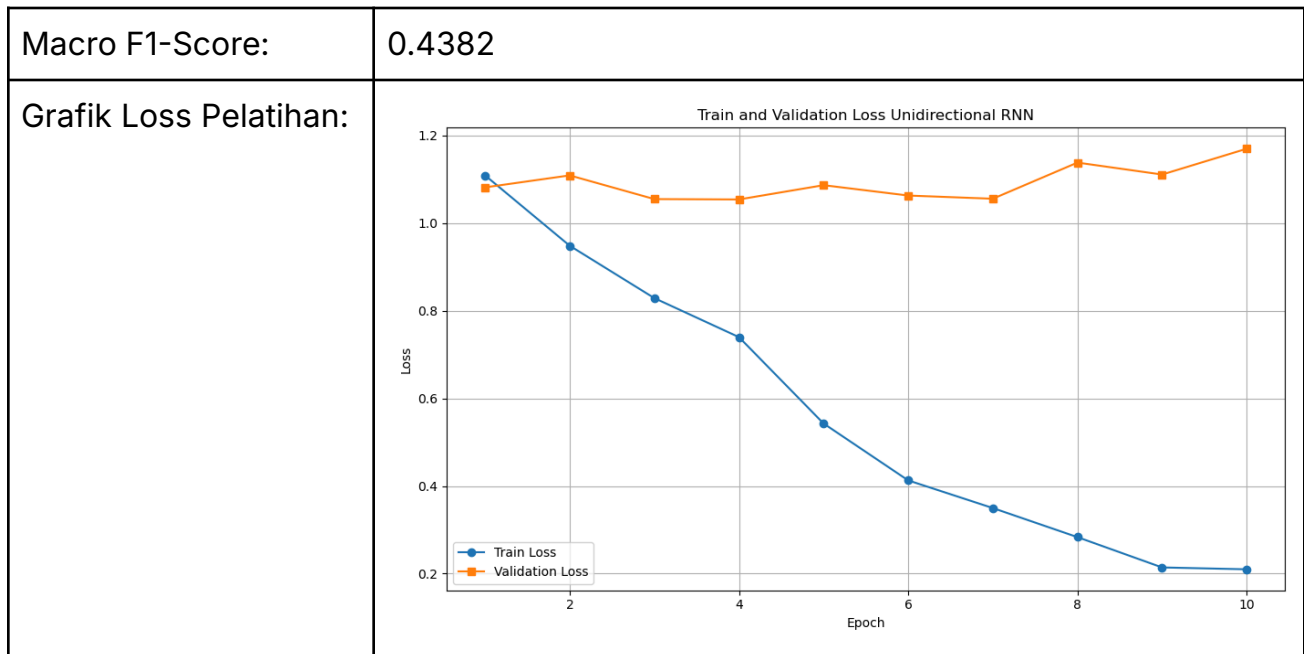
Macro F1-Score:	0.4532																																	
Grafik Loss Pelatihan:	<div><div>Train and Validation Loss 24 RNN Cells</div><table border="1"><thead><tr><th>Epoch</th><th>Train Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>1</td><td>1.1</td><td>1.1</td></tr><tr><td>2</td><td>0.95</td><td>1.05</td></tr><tr><td>3</td><td>0.75</td><td>1.1</td></tr><tr><td>4</td><td>0.55</td><td>1.05</td></tr><tr><td>5</td><td>0.4</td><td>1.05</td></tr><tr><td>6</td><td>0.3</td><td>1.15</td></tr><tr><td>7</td><td>0.25</td><td>1.1</td></tr><tr><td>8</td><td>0.2</td><td>1.15</td></tr><tr><td>9</td><td>0.15</td><td>1.2</td></tr><tr><td>10</td><td>0.15</td><td>1.2</td></tr></tbody></table></div>	Epoch	Train Loss	Validation Loss	1	1.1	1.1	2	0.95	1.05	3	0.75	1.1	4	0.55	1.05	5	0.4	1.05	6	0.3	1.15	7	0.25	1.1	8	0.2	1.15	9	0.15	1.2	10	0.15	1.2
Epoch	Train Loss	Validation Loss																																
1	1.1	1.1																																
2	0.95	1.05																																
3	0.75	1.1																																
4	0.55	1.05																																
5	0.4	1.05																																
6	0.3	1.15																																
7	0.25	1.1																																
8	0.2	1.15																																
9	0.15	1.2																																
10	0.15	1.2																																

2.6.2.4. F1-Score Keras vs Scratch

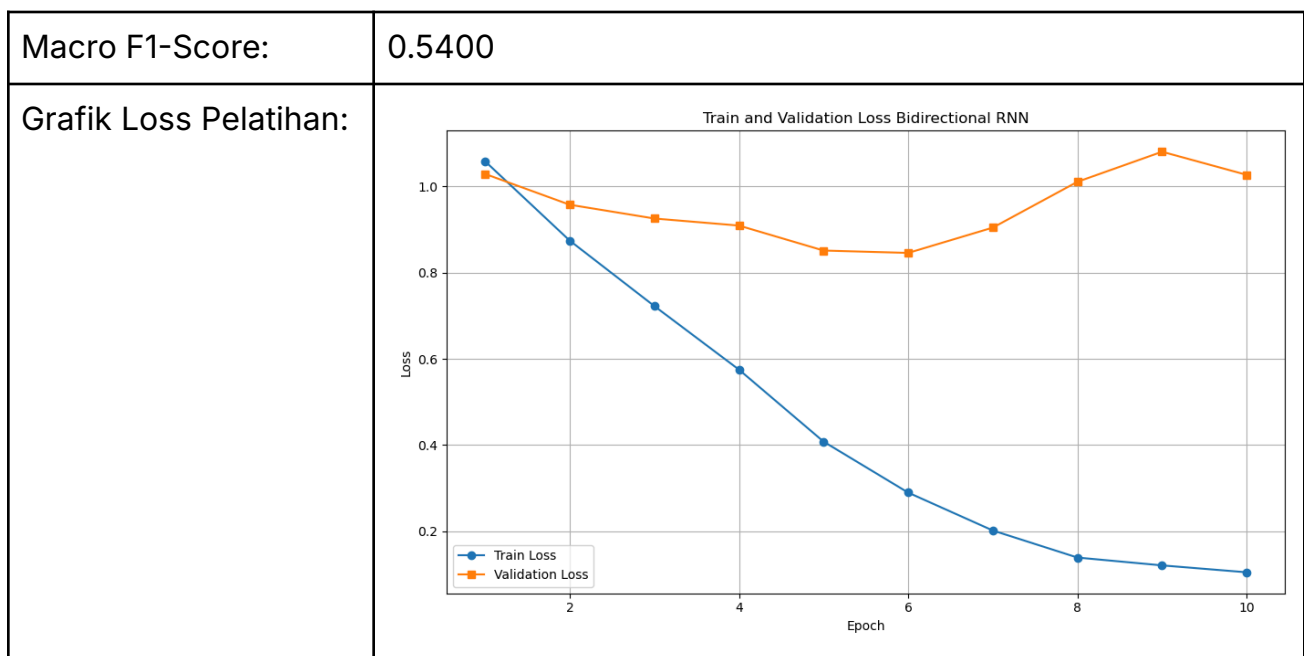
Macro F1-Score Keras:	0.3462, 0.3726, 0.4532												
Macro F1-Score Scratch:	0.3462, 0.3726, 0.4532												
Grafik Macro F1-Score:	<div><p>F1 Score by Number of RNN Cells</p><table><thead><tr><th>Number of RNN Cells</th><th>Keras</th><th>Scratch</th></tr></thead><tbody><tr><td>8 Cells</td><td>0.3462</td><td>0.3462</td></tr><tr><td>16 Cells</td><td>0.3726</td><td>0.3726</td></tr><tr><td>24 Cells</td><td>0.4532</td><td>0.4532</td></tr></tbody></table></div>	Number of RNN Cells	Keras	Scratch	8 Cells	0.3462	0.3462	16 Cells	0.3726	0.3726	24 Cells	0.4532	0.4532
Number of RNN Cells	Keras	Scratch											
8 Cells	0.3462	0.3462											
16 Cells	0.3726	0.3726											
24 Cells	0.4532	0.4532											

2.6.8. Pengaruh jenis layer RNN berdasarkan arah

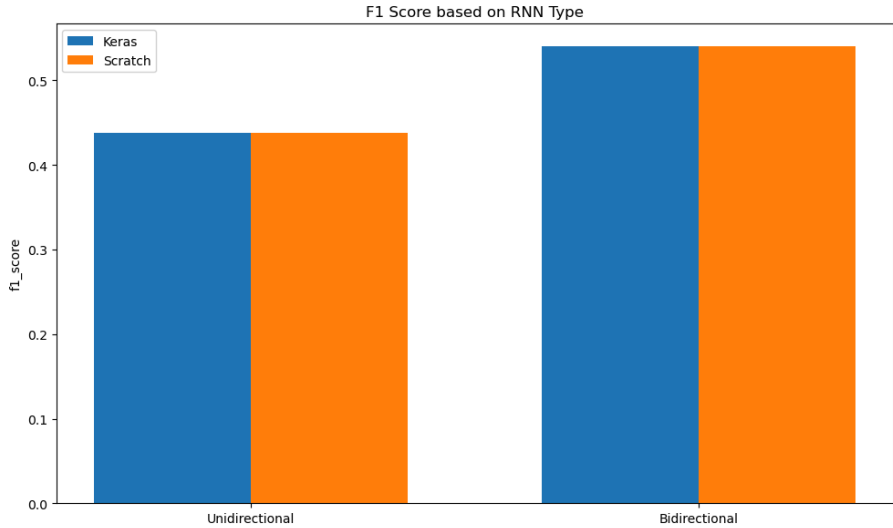
2.6.3.1. Unidirectional Layer



2.6.3.2. Bidirectional Layer



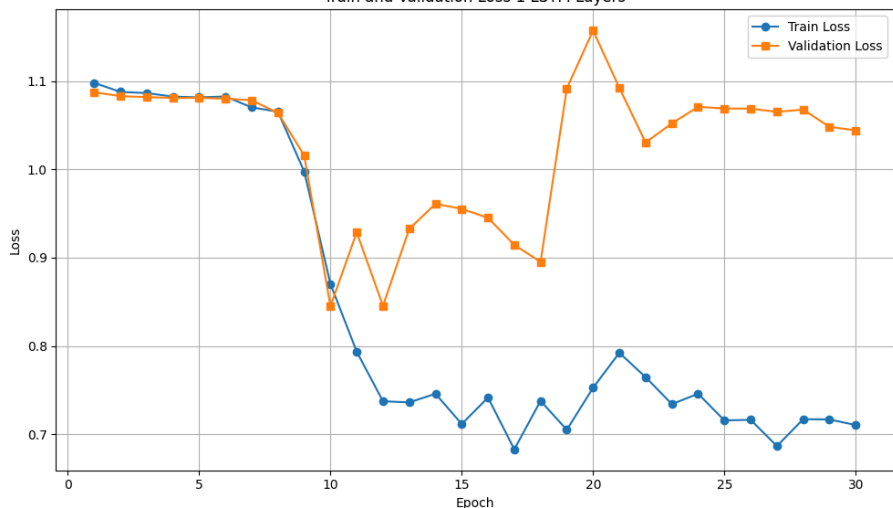
2.6.3.3. F1-Score Keras vs Scratch

Macro F1-Score Keras:	0.4382, 0.5400									
Macro F1-Score Scratch:	0.4382, 0.5400									
Grafik Macro F1-Score:	 <table><caption>F1 Score based on RNN Type</caption><thead><tr><th>RNN Type</th><th>Keras</th><th>Scratch</th></tr></thead><tbody><tr><td>Unidirectional</td><td>0.4382</td><td>0.4382</td></tr><tr><td>Bidirectional</td><td>0.5400</td><td>0.5400</td></tr></tbody></table>	RNN Type	Keras	Scratch	Unidirectional	0.4382	0.4382	Bidirectional	0.5400	0.5400
RNN Type	Keras	Scratch								
Unidirectional	0.4382	0.4382								
Bidirectional	0.5400	0.5400								

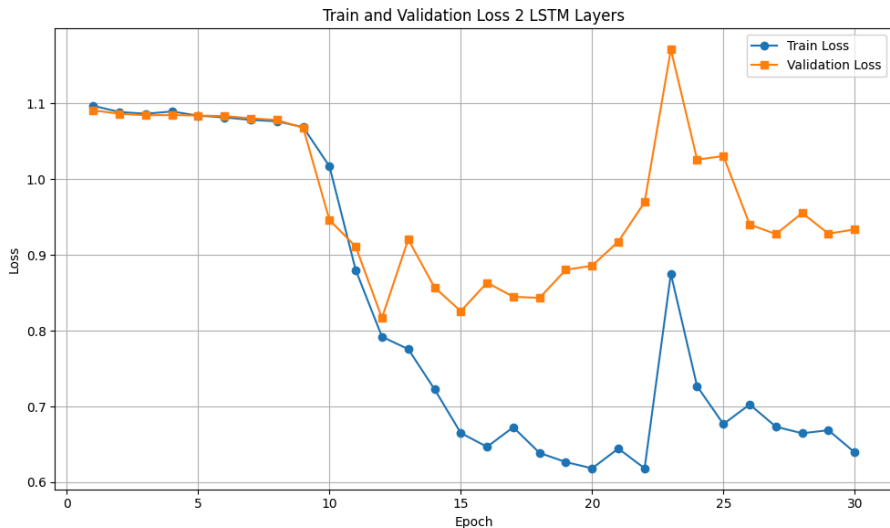
2.7. Hasil Pengujian LSTM

2.7.1. Pengaruh Jumlah layer LSTM

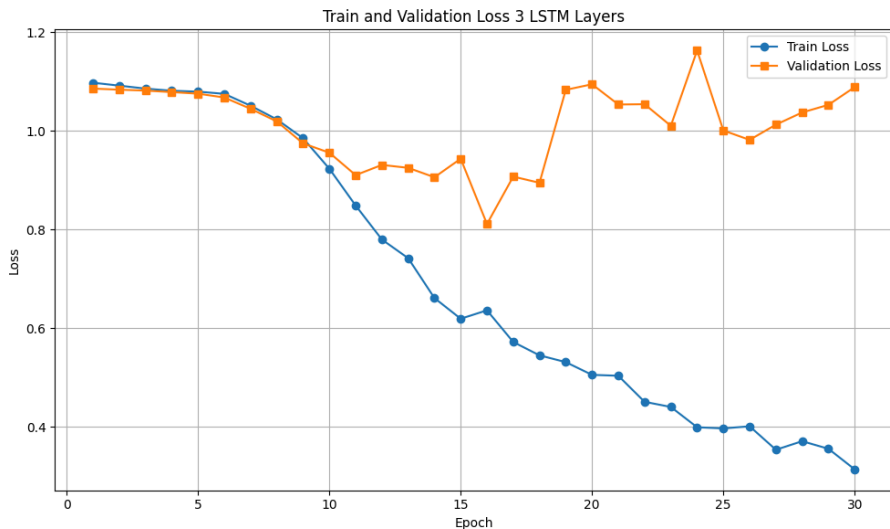
2.7.1.1. 1 Layer LSTM

Macro F1-Score	0.4525																																																																																																
Grafik Loss Pelatihan:	<div><div>Train and Validation Loss 1 LSTM Layers</div><table><thead><tr><th>Epoch</th><th>Train Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.10</td><td>1.08</td></tr><tr><td>1</td><td>1.09</td><td>1.08</td></tr><tr><td>2</td><td>1.08</td><td>1.08</td></tr><tr><td>3</td><td>1.08</td><td>1.08</td></tr><tr><td>4</td><td>1.08</td><td>1.08</td></tr><tr><td>5</td><td>1.08</td><td>1.08</td></tr><tr><td>6</td><td>1.07</td><td>1.08</td></tr><tr><td>7</td><td>1.07</td><td>1.07</td></tr><tr><td>8</td><td>1.00</td><td>1.06</td></tr><tr><td>9</td><td>0.87</td><td>0.85</td></tr><tr><td>10</td><td>0.80</td><td>0.85</td></tr><tr><td>11</td><td>0.74</td><td>0.93</td></tr><tr><td>12</td><td>0.74</td><td>0.85</td></tr><tr><td>13</td><td>0.74</td><td>0.93</td></tr><tr><td>14</td><td>0.75</td><td>0.96</td></tr><tr><td>15</td><td>0.71</td><td>0.95</td></tr><tr><td>16</td><td>0.74</td><td>0.94</td></tr><tr><td>17</td><td>0.68</td><td>0.91</td></tr><tr><td>18</td><td>0.74</td><td>0.89</td></tr><tr><td>19</td><td>0.71</td><td>1.09</td></tr><tr><td>20</td><td>0.75</td><td>1.15</td></tr><tr><td>21</td><td>0.79</td><td>1.09</td></tr><tr><td>22</td><td>0.76</td><td>1.03</td></tr><tr><td>23</td><td>0.73</td><td>1.05</td></tr><tr><td>24</td><td>0.75</td><td>1.07</td></tr><tr><td>25</td><td>0.72</td><td>1.07</td></tr><tr><td>26</td><td>0.72</td><td>1.07</td></tr><tr><td>27</td><td>0.69</td><td>1.07</td></tr><tr><td>28</td><td>0.72</td><td>1.07</td></tr><tr><td>29</td><td>0.72</td><td>1.04</td></tr><tr><td>30</td><td>0.71</td><td>1.04</td></tr></tbody></table></div>	Epoch	Train Loss	Validation Loss	0	1.10	1.08	1	1.09	1.08	2	1.08	1.08	3	1.08	1.08	4	1.08	1.08	5	1.08	1.08	6	1.07	1.08	7	1.07	1.07	8	1.00	1.06	9	0.87	0.85	10	0.80	0.85	11	0.74	0.93	12	0.74	0.85	13	0.74	0.93	14	0.75	0.96	15	0.71	0.95	16	0.74	0.94	17	0.68	0.91	18	0.74	0.89	19	0.71	1.09	20	0.75	1.15	21	0.79	1.09	22	0.76	1.03	23	0.73	1.05	24	0.75	1.07	25	0.72	1.07	26	0.72	1.07	27	0.69	1.07	28	0.72	1.07	29	0.72	1.04	30	0.71	1.04
Epoch	Train Loss	Validation Loss																																																																																															
0	1.10	1.08																																																																																															
1	1.09	1.08																																																																																															
2	1.08	1.08																																																																																															
3	1.08	1.08																																																																																															
4	1.08	1.08																																																																																															
5	1.08	1.08																																																																																															
6	1.07	1.08																																																																																															
7	1.07	1.07																																																																																															
8	1.00	1.06																																																																																															
9	0.87	0.85																																																																																															
10	0.80	0.85																																																																																															
11	0.74	0.93																																																																																															
12	0.74	0.85																																																																																															
13	0.74	0.93																																																																																															
14	0.75	0.96																																																																																															
15	0.71	0.95																																																																																															
16	0.74	0.94																																																																																															
17	0.68	0.91																																																																																															
18	0.74	0.89																																																																																															
19	0.71	1.09																																																																																															
20	0.75	1.15																																																																																															
21	0.79	1.09																																																																																															
22	0.76	1.03																																																																																															
23	0.73	1.05																																																																																															
24	0.75	1.07																																																																																															
25	0.72	1.07																																																																																															
26	0.72	1.07																																																																																															
27	0.69	1.07																																																																																															
28	0.72	1.07																																																																																															
29	0.72	1.04																																																																																															
30	0.71	1.04																																																																																															

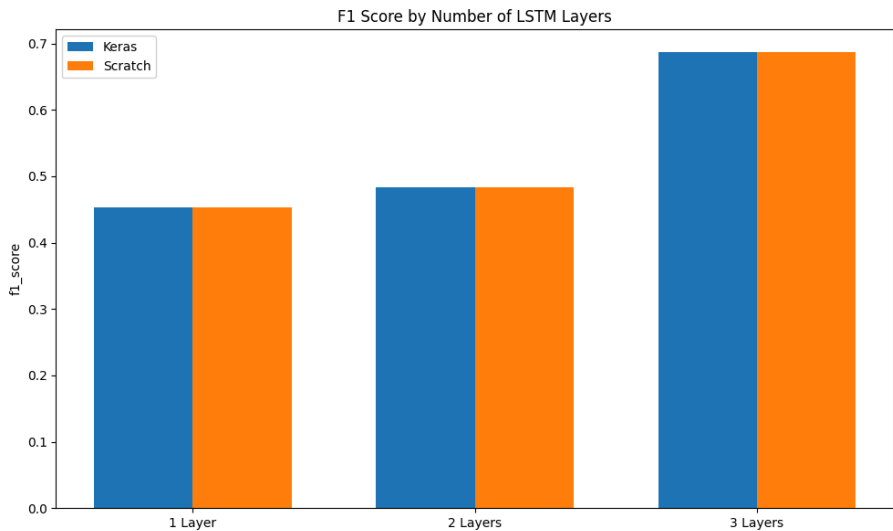
2.7.1.2. 2 Layer LSTM

Macro F1-Score	0.4828																								
Grafik Loss Pelatihan:	<div><p>Train and Validation Loss 2 LSTM Layers</p><table><thead><tr><th>Epoch</th><th>Train Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.10</td><td>1.08</td></tr><tr><td>5</td><td>1.08</td><td>1.08</td></tr><tr><td>10</td><td>1.02</td><td>0.95</td></tr><tr><td>15</td><td>0.68</td><td>0.83</td></tr><tr><td>20</td><td>0.62</td><td>0.88</td></tr><tr><td>25</td><td>0.68</td><td>1.03</td></tr><tr><td>30</td><td>0.65</td><td>0.93</td></tr></tbody></table></div>	Epoch	Train Loss	Validation Loss	0	1.10	1.08	5	1.08	1.08	10	1.02	0.95	15	0.68	0.83	20	0.62	0.88	25	0.68	1.03	30	0.65	0.93
Epoch	Train Loss	Validation Loss																							
0	1.10	1.08																							
5	1.08	1.08																							
10	1.02	0.95																							
15	0.68	0.83																							
20	0.62	0.88																							
25	0.68	1.03																							
30	0.65	0.93																							

2.7.1.3. 3 Layer LSTM

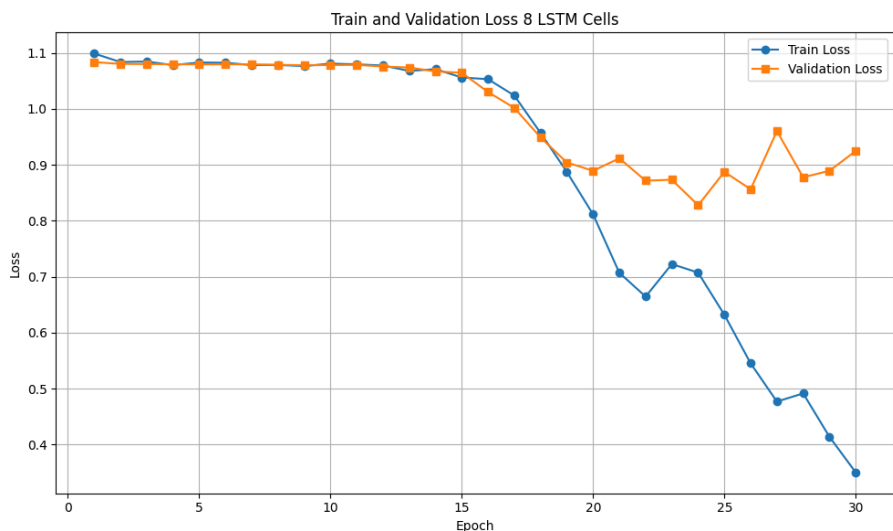
Macro F1-Score	0.6866																																																																																																
Grafik Loss Pelatihan:	<div><p>Train and Validation Loss 3 LSTM Layers</p><table border="1"><thead><tr><th>Epoch</th><th>Train Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.10</td><td>1.08</td></tr><tr><td>1</td><td>1.09</td><td>1.08</td></tr><tr><td>2</td><td>1.08</td><td>1.08</td></tr><tr><td>3</td><td>1.08</td><td>1.08</td></tr><tr><td>4</td><td>1.08</td><td>1.08</td></tr><tr><td>5</td><td>1.08</td><td>1.08</td></tr><tr><td>6</td><td>1.07</td><td>1.07</td></tr><tr><td>7</td><td>1.05</td><td>1.04</td></tr><tr><td>8</td><td>1.02</td><td>1.02</td></tr><tr><td>9</td><td>0.98</td><td>0.98</td></tr><tr><td>10</td><td>0.92</td><td>0.95</td></tr><tr><td>11</td><td>0.85</td><td>0.92</td></tr><tr><td>12</td><td>0.78</td><td>0.93</td></tr><tr><td>13</td><td>0.74</td><td>0.92</td></tr><tr><td>14</td><td>0.66</td><td>0.90</td></tr><tr><td>15</td><td>0.62</td><td>0.94</td></tr><tr><td>16</td><td>0.64</td><td>0.81</td></tr><tr><td>17</td><td>0.57</td><td>0.91</td></tr><tr><td>18</td><td>0.54</td><td>0.89</td></tr><tr><td>19</td><td>0.53</td><td>1.08</td></tr><tr><td>20</td><td>0.50</td><td>1.09</td></tr><tr><td>21</td><td>0.50</td><td>1.05</td></tr><tr><td>22</td><td>0.45</td><td>1.05</td></tr><tr><td>23</td><td>0.44</td><td>1.01</td></tr><tr><td>24</td><td>0.40</td><td>1.16</td></tr><tr><td>25</td><td>0.40</td><td>1.00</td></tr><tr><td>26</td><td>0.40</td><td>0.98</td></tr><tr><td>27</td><td>0.35</td><td>1.02</td></tr><tr><td>28</td><td>0.37</td><td>1.04</td></tr><tr><td>29</td><td>0.35</td><td>1.05</td></tr><tr><td>30</td><td>0.30</td><td>1.09</td></tr></tbody></table></div>	Epoch	Train Loss	Validation Loss	0	1.10	1.08	1	1.09	1.08	2	1.08	1.08	3	1.08	1.08	4	1.08	1.08	5	1.08	1.08	6	1.07	1.07	7	1.05	1.04	8	1.02	1.02	9	0.98	0.98	10	0.92	0.95	11	0.85	0.92	12	0.78	0.93	13	0.74	0.92	14	0.66	0.90	15	0.62	0.94	16	0.64	0.81	17	0.57	0.91	18	0.54	0.89	19	0.53	1.08	20	0.50	1.09	21	0.50	1.05	22	0.45	1.05	23	0.44	1.01	24	0.40	1.16	25	0.40	1.00	26	0.40	0.98	27	0.35	1.02	28	0.37	1.04	29	0.35	1.05	30	0.30	1.09
Epoch	Train Loss	Validation Loss																																																																																															
0	1.10	1.08																																																																																															
1	1.09	1.08																																																																																															
2	1.08	1.08																																																																																															
3	1.08	1.08																																																																																															
4	1.08	1.08																																																																																															
5	1.08	1.08																																																																																															
6	1.07	1.07																																																																																															
7	1.05	1.04																																																																																															
8	1.02	1.02																																																																																															
9	0.98	0.98																																																																																															
10	0.92	0.95																																																																																															
11	0.85	0.92																																																																																															
12	0.78	0.93																																																																																															
13	0.74	0.92																																																																																															
14	0.66	0.90																																																																																															
15	0.62	0.94																																																																																															
16	0.64	0.81																																																																																															
17	0.57	0.91																																																																																															
18	0.54	0.89																																																																																															
19	0.53	1.08																																																																																															
20	0.50	1.09																																																																																															
21	0.50	1.05																																																																																															
22	0.45	1.05																																																																																															
23	0.44	1.01																																																																																															
24	0.40	1.16																																																																																															
25	0.40	1.00																																																																																															
26	0.40	0.98																																																																																															
27	0.35	1.02																																																																																															
28	0.37	1.04																																																																																															
29	0.35	1.05																																																																																															
30	0.30	1.09																																																																																															

2.7.1.4. F1-Score Keras vs Scratch

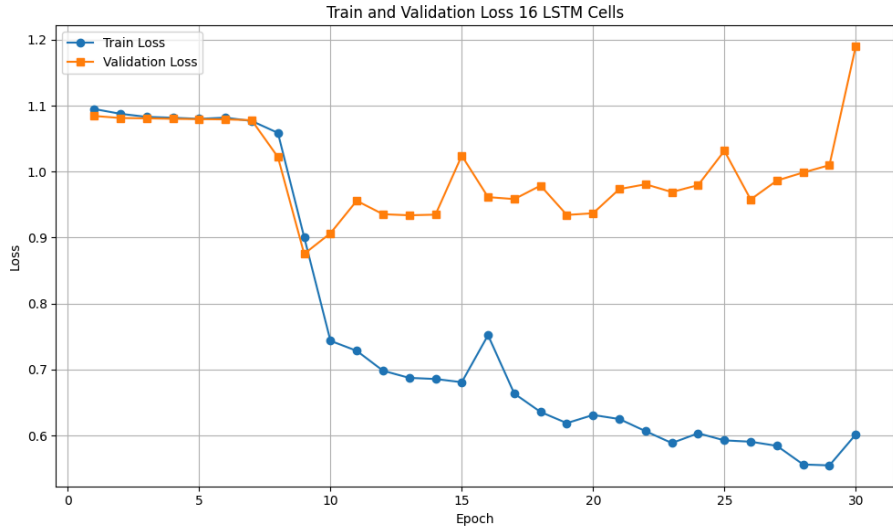
Macro F1-Score Keras:	0.4525, 0.4828, 0.6866												
Macro F1-Score Scratch:	0.4525, 0.4828, 0.6866												
Grafik Macro F1-Score:	 <table><caption>F1 Score by Number of LSTM Layers</caption><thead><tr><th>Number of Layers</th><th>Keras</th><th>Scratch</th></tr></thead><tbody><tr><td>1 Layer</td><td>0.4525</td><td>0.4525</td></tr><tr><td>2 Layers</td><td>0.4828</td><td>0.4828</td></tr><tr><td>3 Layers</td><td>0.6866</td><td>0.6866</td></tr></tbody></table>	Number of Layers	Keras	Scratch	1 Layer	0.4525	0.4525	2 Layers	0.4828	0.4828	3 Layers	0.6866	0.6866
Number of Layers	Keras	Scratch											
1 Layer	0.4525	0.4525											
2 Layers	0.4828	0.4828											
3 Layers	0.6866	0.6866											

2.7.2. Pengaruh banyak cell LSTM per layer

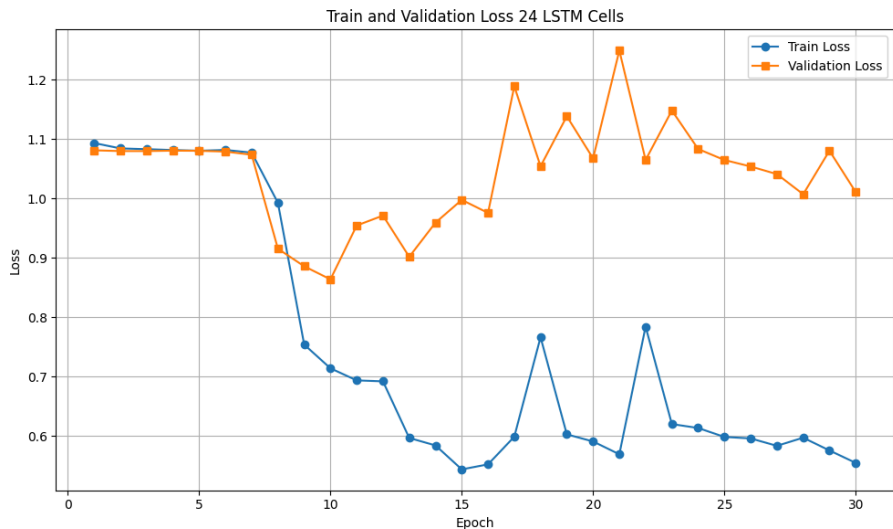
2.7.2.1. 8 Cell LSTM

Macro F1-Score	0.5553																																																																																																
Grafik Loss Pelatihan:	<div><p>Train and Validation Loss 8 LSTM Cells</p><table><thead><tr><th>Epoch</th><th>Train Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.10</td><td>1.08</td></tr><tr><td>1</td><td>1.09</td><td>1.08</td></tr><tr><td>2</td><td>1.09</td><td>1.08</td></tr><tr><td>3</td><td>1.09</td><td>1.08</td></tr><tr><td>4</td><td>1.08</td><td>1.08</td></tr><tr><td>5</td><td>1.08</td><td>1.08</td></tr><tr><td>6</td><td>1.08</td><td>1.08</td></tr><tr><td>7</td><td>1.08</td><td>1.08</td></tr><tr><td>8</td><td>1.08</td><td>1.08</td></tr><tr><td>9</td><td>1.08</td><td>1.08</td></tr><tr><td>10</td><td>1.08</td><td>1.08</td></tr><tr><td>11</td><td>1.08</td><td>1.08</td></tr><tr><td>12</td><td>1.07</td><td>1.08</td></tr><tr><td>13</td><td>1.07</td><td>1.07</td></tr><tr><td>14</td><td>1.07</td><td>1.07</td></tr><tr><td>15</td><td>1.05</td><td>1.07</td></tr><tr><td>16</td><td>1.02</td><td>1.04</td></tr><tr><td>17</td><td>1.00</td><td>1.00</td></tr><tr><td>18</td><td>0.95</td><td>0.95</td></tr><tr><td>19</td><td>0.89</td><td>0.91</td></tr><tr><td>20</td><td>0.81</td><td>0.89</td></tr><tr><td>21</td><td>0.71</td><td>0.91</td></tr><tr><td>22</td><td>0.67</td><td>0.87</td></tr><tr><td>23</td><td>0.72</td><td>0.88</td></tr><tr><td>24</td><td>0.71</td><td>0.83</td></tr><tr><td>25</td><td>0.63</td><td>0.89</td></tr><tr><td>26</td><td>0.55</td><td>0.85</td></tr><tr><td>27</td><td>0.48</td><td>0.96</td></tr><tr><td>28</td><td>0.49</td><td>0.88</td></tr><tr><td>29</td><td>0.42</td><td>0.89</td></tr><tr><td>30</td><td>0.35</td><td>0.92</td></tr></tbody></table></div>	Epoch	Train Loss	Validation Loss	0	1.10	1.08	1	1.09	1.08	2	1.09	1.08	3	1.09	1.08	4	1.08	1.08	5	1.08	1.08	6	1.08	1.08	7	1.08	1.08	8	1.08	1.08	9	1.08	1.08	10	1.08	1.08	11	1.08	1.08	12	1.07	1.08	13	1.07	1.07	14	1.07	1.07	15	1.05	1.07	16	1.02	1.04	17	1.00	1.00	18	0.95	0.95	19	0.89	0.91	20	0.81	0.89	21	0.71	0.91	22	0.67	0.87	23	0.72	0.88	24	0.71	0.83	25	0.63	0.89	26	0.55	0.85	27	0.48	0.96	28	0.49	0.88	29	0.42	0.89	30	0.35	0.92
Epoch	Train Loss	Validation Loss																																																																																															
0	1.10	1.08																																																																																															
1	1.09	1.08																																																																																															
2	1.09	1.08																																																																																															
3	1.09	1.08																																																																																															
4	1.08	1.08																																																																																															
5	1.08	1.08																																																																																															
6	1.08	1.08																																																																																															
7	1.08	1.08																																																																																															
8	1.08	1.08																																																																																															
9	1.08	1.08																																																																																															
10	1.08	1.08																																																																																															
11	1.08	1.08																																																																																															
12	1.07	1.08																																																																																															
13	1.07	1.07																																																																																															
14	1.07	1.07																																																																																															
15	1.05	1.07																																																																																															
16	1.02	1.04																																																																																															
17	1.00	1.00																																																																																															
18	0.95	0.95																																																																																															
19	0.89	0.91																																																																																															
20	0.81	0.89																																																																																															
21	0.71	0.91																																																																																															
22	0.67	0.87																																																																																															
23	0.72	0.88																																																																																															
24	0.71	0.83																																																																																															
25	0.63	0.89																																																																																															
26	0.55	0.85																																																																																															
27	0.48	0.96																																																																																															
28	0.49	0.88																																																																																															
29	0.42	0.89																																																																																															
30	0.35	0.92																																																																																															

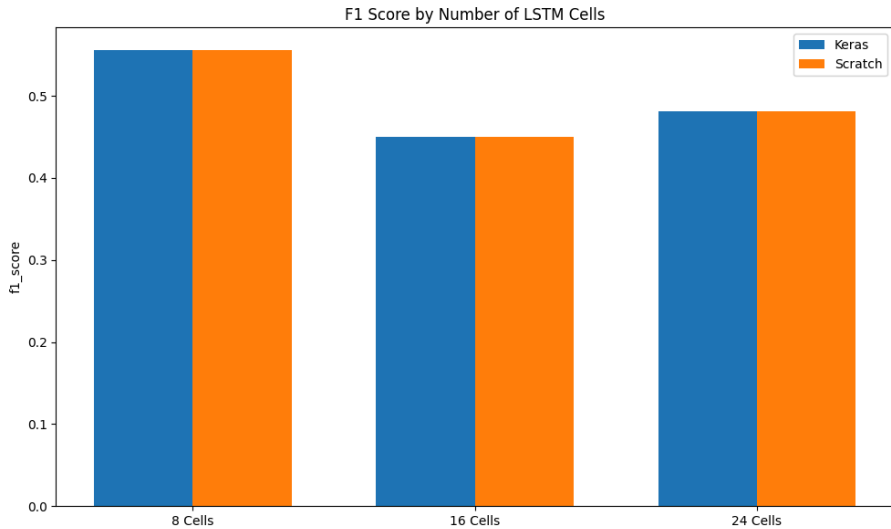
2.7.2.2. 16 Cell LSTM

Macro F1-Score	0.4501																																																																																																
Grafik Loss Pelatihan:	<div><div>Train and Validation Loss 16 LSTM Cells</div><table><thead><tr><th>Epoch</th><th>Train Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.10</td><td>1.10</td></tr><tr><td>1</td><td>1.09</td><td>1.09</td></tr><tr><td>2</td><td>1.08</td><td>1.08</td></tr><tr><td>3</td><td>1.08</td><td>1.08</td></tr><tr><td>4</td><td>1.08</td><td>1.08</td></tr><tr><td>5</td><td>1.08</td><td>1.08</td></tr><tr><td>6</td><td>1.08</td><td>1.08</td></tr><tr><td>7</td><td>1.07</td><td>1.07</td></tr><tr><td>8</td><td>1.05</td><td>1.02</td></tr><tr><td>9</td><td>0.90</td><td>0.88</td></tr><tr><td>10</td><td>0.75</td><td>0.90</td></tr><tr><td>11</td><td>0.73</td><td>0.95</td></tr><tr><td>12</td><td>0.70</td><td>0.93</td></tr><tr><td>13</td><td>0.69</td><td>0.93</td></tr><tr><td>14</td><td>0.69</td><td>0.93</td></tr><tr><td>15</td><td>0.68</td><td>1.02</td></tr><tr><td>16</td><td>0.75</td><td>0.96</td></tr><tr><td>17</td><td>0.67</td><td>0.96</td></tr><tr><td>18</td><td>0.64</td><td>0.98</td></tr><tr><td>19</td><td>0.63</td><td>0.93</td></tr><tr><td>20</td><td>0.63</td><td>0.93</td></tr><tr><td>21</td><td>0.63</td><td>0.97</td></tr><tr><td>22</td><td>0.61</td><td>0.98</td></tr><tr><td>23</td><td>0.59</td><td>0.97</td></tr><tr><td>24</td><td>0.60</td><td>0.98</td></tr><tr><td>25</td><td>0.59</td><td>1.03</td></tr><tr><td>26</td><td>0.59</td><td>0.95</td></tr><tr><td>27</td><td>0.58</td><td>0.98</td></tr><tr><td>28</td><td>0.56</td><td>1.00</td></tr><tr><td>29</td><td>0.55</td><td>1.01</td></tr><tr><td>30</td><td>0.60</td><td>1.18</td></tr></tbody></table></div>	Epoch	Train Loss	Validation Loss	0	1.10	1.10	1	1.09	1.09	2	1.08	1.08	3	1.08	1.08	4	1.08	1.08	5	1.08	1.08	6	1.08	1.08	7	1.07	1.07	8	1.05	1.02	9	0.90	0.88	10	0.75	0.90	11	0.73	0.95	12	0.70	0.93	13	0.69	0.93	14	0.69	0.93	15	0.68	1.02	16	0.75	0.96	17	0.67	0.96	18	0.64	0.98	19	0.63	0.93	20	0.63	0.93	21	0.63	0.97	22	0.61	0.98	23	0.59	0.97	24	0.60	0.98	25	0.59	1.03	26	0.59	0.95	27	0.58	0.98	28	0.56	1.00	29	0.55	1.01	30	0.60	1.18
Epoch	Train Loss	Validation Loss																																																																																															
0	1.10	1.10																																																																																															
1	1.09	1.09																																																																																															
2	1.08	1.08																																																																																															
3	1.08	1.08																																																																																															
4	1.08	1.08																																																																																															
5	1.08	1.08																																																																																															
6	1.08	1.08																																																																																															
7	1.07	1.07																																																																																															
8	1.05	1.02																																																																																															
9	0.90	0.88																																																																																															
10	0.75	0.90																																																																																															
11	0.73	0.95																																																																																															
12	0.70	0.93																																																																																															
13	0.69	0.93																																																																																															
14	0.69	0.93																																																																																															
15	0.68	1.02																																																																																															
16	0.75	0.96																																																																																															
17	0.67	0.96																																																																																															
18	0.64	0.98																																																																																															
19	0.63	0.93																																																																																															
20	0.63	0.93																																																																																															
21	0.63	0.97																																																																																															
22	0.61	0.98																																																																																															
23	0.59	0.97																																																																																															
24	0.60	0.98																																																																																															
25	0.59	1.03																																																																																															
26	0.59	0.95																																																																																															
27	0.58	0.98																																																																																															
28	0.56	1.00																																																																																															
29	0.55	1.01																																																																																															
30	0.60	1.18																																																																																															

2.7.2.3. 24 Cell LSTM

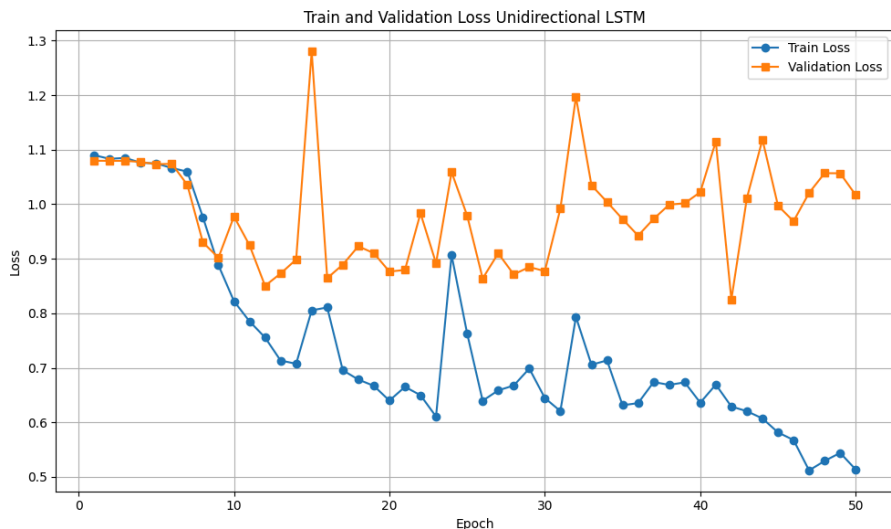
Macro F1-Score	0.4808																																																																																																
Grafik Loss Pelatihan:	<div><div>Train and Validation Loss 24 LSTM Cells</div><table><thead><tr><th>Epoch</th><th>Train Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.10</td><td>1.10</td></tr><tr><td>1</td><td>1.09</td><td>1.09</td></tr><tr><td>2</td><td>1.08</td><td>1.08</td></tr><tr><td>3</td><td>1.08</td><td>1.08</td></tr><tr><td>4</td><td>1.08</td><td>1.08</td></tr><tr><td>5</td><td>1.08</td><td>1.08</td></tr><tr><td>6</td><td>1.08</td><td>1.08</td></tr><tr><td>7</td><td>1.07</td><td>1.07</td></tr><tr><td>8</td><td>1.00</td><td>0.92</td></tr><tr><td>9</td><td>0.75</td><td>0.88</td></tr><tr><td>10</td><td>0.72</td><td>0.85</td></tr><tr><td>11</td><td>0.70</td><td>0.95</td></tr><tr><td>12</td><td>0.70</td><td>0.98</td></tr><tr><td>13</td><td>0.60</td><td>0.90</td></tr><tr><td>14</td><td>0.58</td><td>0.95</td></tr><tr><td>15</td><td>0.55</td><td>1.00</td></tr><tr><td>16</td><td>0.55</td><td>0.98</td></tr><tr><td>17</td><td>0.60</td><td>1.18</td></tr><tr><td>18</td><td>0.77</td><td>1.05</td></tr><tr><td>19</td><td>0.60</td><td>1.15</td></tr><tr><td>20</td><td>0.59</td><td>1.07</td></tr><tr><td>21</td><td>0.57</td><td>1.35</td></tr><tr><td>22</td><td>0.78</td><td>1.07</td></tr><tr><td>23</td><td>0.62</td><td>1.15</td></tr><tr><td>24</td><td>0.62</td><td>1.08</td></tr><tr><td>25</td><td>0.60</td><td>1.07</td></tr><tr><td>26</td><td>0.60</td><td>1.05</td></tr><tr><td>27</td><td>0.58</td><td>1.04</td></tr><tr><td>28</td><td>0.60</td><td>1.01</td></tr><tr><td>29</td><td>0.58</td><td>1.08</td></tr><tr><td>30</td><td>0.55</td><td>1.01</td></tr></tbody></table></div>	Epoch	Train Loss	Validation Loss	0	1.10	1.10	1	1.09	1.09	2	1.08	1.08	3	1.08	1.08	4	1.08	1.08	5	1.08	1.08	6	1.08	1.08	7	1.07	1.07	8	1.00	0.92	9	0.75	0.88	10	0.72	0.85	11	0.70	0.95	12	0.70	0.98	13	0.60	0.90	14	0.58	0.95	15	0.55	1.00	16	0.55	0.98	17	0.60	1.18	18	0.77	1.05	19	0.60	1.15	20	0.59	1.07	21	0.57	1.35	22	0.78	1.07	23	0.62	1.15	24	0.62	1.08	25	0.60	1.07	26	0.60	1.05	27	0.58	1.04	28	0.60	1.01	29	0.58	1.08	30	0.55	1.01
Epoch	Train Loss	Validation Loss																																																																																															
0	1.10	1.10																																																																																															
1	1.09	1.09																																																																																															
2	1.08	1.08																																																																																															
3	1.08	1.08																																																																																															
4	1.08	1.08																																																																																															
5	1.08	1.08																																																																																															
6	1.08	1.08																																																																																															
7	1.07	1.07																																																																																															
8	1.00	0.92																																																																																															
9	0.75	0.88																																																																																															
10	0.72	0.85																																																																																															
11	0.70	0.95																																																																																															
12	0.70	0.98																																																																																															
13	0.60	0.90																																																																																															
14	0.58	0.95																																																																																															
15	0.55	1.00																																																																																															
16	0.55	0.98																																																																																															
17	0.60	1.18																																																																																															
18	0.77	1.05																																																																																															
19	0.60	1.15																																																																																															
20	0.59	1.07																																																																																															
21	0.57	1.35																																																																																															
22	0.78	1.07																																																																																															
23	0.62	1.15																																																																																															
24	0.62	1.08																																																																																															
25	0.60	1.07																																																																																															
26	0.60	1.05																																																																																															
27	0.58	1.04																																																																																															
28	0.60	1.01																																																																																															
29	0.58	1.08																																																																																															
30	0.55	1.01																																																																																															

2.7.3.4. F1-Score Keras vs Scratch

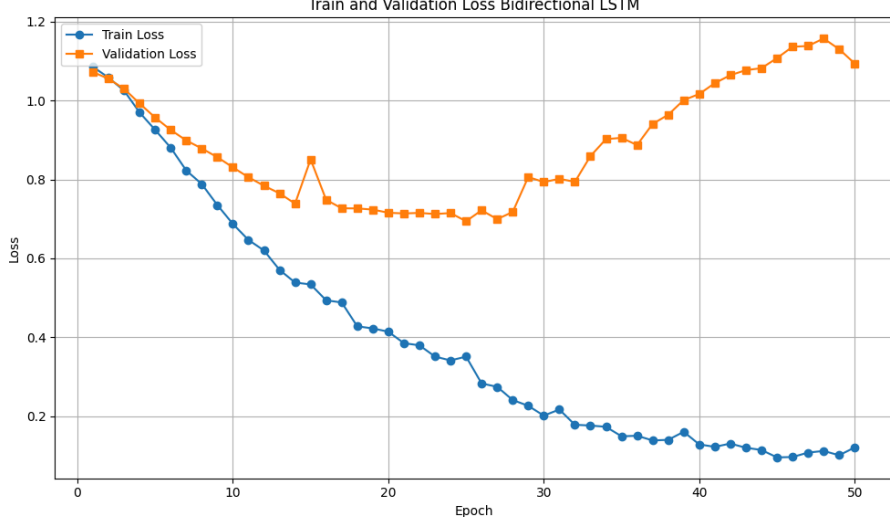
Macro F1-Score Keras:	0.5553, 0.4501, 0.4808												
Macro F1-Score Scratch:	0.5553, 0.4501, 0.4808												
Grafik Macro F1-Score:	<div><div><div>F1 Score by Number of LSTM Cells</div><table><thead><tr><th>Number of LSTM Cells</th><th>Keras</th><th>Scratch</th></tr></thead><tbody><tr><td>8 Cells</td><td>0.5553</td><td>0.5553</td></tr><tr><td>16 Cells</td><td>0.4501</td><td>0.4501</td></tr><tr><td>24 Cells</td><td>0.4808</td><td>0.4808</td></tr></tbody></table></div></div>	Number of LSTM Cells	Keras	Scratch	8 Cells	0.5553	0.5553	16 Cells	0.4501	0.4501	24 Cells	0.4808	0.4808
Number of LSTM Cells	Keras	Scratch											
8 Cells	0.5553	0.5553											
16 Cells	0.4501	0.4501											
24 Cells	0.4808	0.4808											

2.7.3. Pengaruh jenis layer LSTM berdasarkan arah

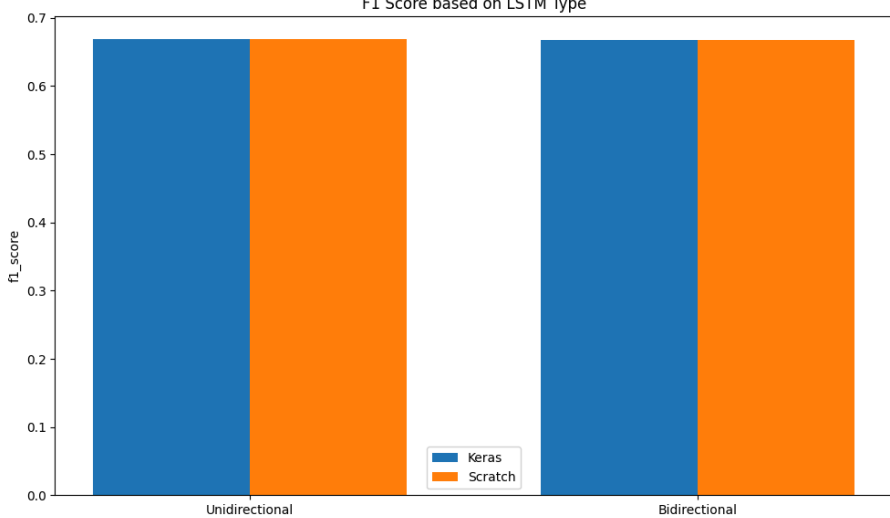
2.7.3.1. Unidirectional Layer

Macro F1-Score	0.6686																																				
Grafik Loss Pelatihan:	<div><div>Train and Validation Loss Unidirectional LSTM</div><table border="1"><caption>Approximate data points from the loss chart</caption><thead><tr><th>Epoch</th><th>Train Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.10</td><td>1.10</td></tr><tr><td>5</td><td>1.08</td><td>1.08</td></tr><tr><td>10</td><td>0.85</td><td>0.95</td></tr><tr><td>15</td><td>0.80</td><td>1.28</td></tr><tr><td>20</td><td>0.65</td><td>0.90</td></tr><tr><td>25</td><td>0.60</td><td>1.05</td></tr><tr><td>30</td><td>0.65</td><td>0.88</td></tr><tr><td>35</td><td>0.68</td><td>1.20</td></tr><tr><td>40</td><td>0.65</td><td>1.00</td></tr><tr><td>45</td><td>0.60</td><td>1.10</td></tr><tr><td>50</td><td>0.52</td><td>1.02</td></tr></tbody></table></div>	Epoch	Train Loss	Validation Loss	0	1.10	1.10	5	1.08	1.08	10	0.85	0.95	15	0.80	1.28	20	0.65	0.90	25	0.60	1.05	30	0.65	0.88	35	0.68	1.20	40	0.65	1.00	45	0.60	1.10	50	0.52	1.02
Epoch	Train Loss	Validation Loss																																			
0	1.10	1.10																																			
5	1.08	1.08																																			
10	0.85	0.95																																			
15	0.80	1.28																																			
20	0.65	0.90																																			
25	0.60	1.05																																			
30	0.65	0.88																																			
35	0.68	1.20																																			
40	0.65	1.00																																			
45	0.60	1.10																																			
50	0.52	1.02																																			

2.6.7.2. Bidirectional Layer

Macro F1-Score	0.6675																					
Grafik Loss Pelatihan:	<div><p>Train and Validation Loss Bidirectional LSTM</p><table border="1"><caption>Approximate data points for Train and Validation Loss</caption><thead><tr><th>Epoch</th><th>Train Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.10</td><td>1.10</td></tr><tr><td>10</td><td>0.70</td><td>0.85</td></tr><tr><td>20</td><td>0.45</td><td>0.75</td></tr><tr><td>30</td><td>0.25</td><td>0.80</td></tr><tr><td>40</td><td>0.15</td><td>1.05</td></tr><tr><td>50</td><td>0.10</td><td>1.10</td></tr></tbody></table></div>	Epoch	Train Loss	Validation Loss	0	1.10	1.10	10	0.70	0.85	20	0.45	0.75	30	0.25	0.80	40	0.15	1.05	50	0.10	1.10
Epoch	Train Loss	Validation Loss																				
0	1.10	1.10																				
10	0.70	0.85																				
20	0.45	0.75																				
30	0.25	0.80																				
40	0.15	1.05																				
50	0.10	1.10																				

2.7.3.4. F1-Score Keras vs Scratch

Macro F1-Score Keras:	0.6686, 0.6675									
Macro F1-Score Scratch:	0.6686, 0.6675									
Grafik Macro F1-Score:	 <p>F1 Score based on LSTM Type</p> <table><thead><tr><th>LSTM Type</th><th>Keras</th><th>Scratch</th></tr></thead><tbody><tr><td>Unidirectional</td><td>0.6686</td><td>0.6675</td></tr><tr><td>Bidirectional</td><td>0.6686</td><td>0.6675</td></tr></tbody></table>	LSTM Type	Keras	Scratch	Unidirectional	0.6686	0.6675	Bidirectional	0.6686	0.6675
LSTM Type	Keras	Scratch								
Unidirectional	0.6686	0.6675								
Bidirectional	0.6686	0.6675								

2.8. Hasil Analisis CNN

2.8.1. Pengaruh jumlah layer konvolusi

No	Banyak layer konvolusi	Macro F1 Score
1.	1	0.6427
2.	2	0.6976
3.	3	0.7229

Berdasarkan hasil yang diperoleh, peningkatan jumlah layer konvolusi memberikan dampak positif terhadap performa model, terlihat dari naiknya nilai Macro F1 Score seiring bertambahnya jumlah layer. Dengan hanya satu layer konvolusi, model tampaknya belum mampu menangkap fitur yang cukup kompleks, sehingga hasilnya masih cukup rendah. Ketika jumlah layer ditambah menjadi dua dan kemudian tiga, model dapat mengekstraksi fitur secara bertahap, mulai dari pola sederhana di layer awal hingga pola yang lebih kompleks di layer berikutnya. Ini memungkinkan model mengenali karakteristik yang lebih dalam dari data.

Penambahan layer konvolusi juga memperluas kedalaman jaringan, yang artinya model memiliki kapasitas lebih besar untuk belajar representasi yang lebih abstrak. Hal ini terbukti dari nilai tertinggi yang dicapai saat menggunakan tiga layer, yaitu 0.7229. Namun, perlu diingat bahwa semakin dalam jaringan, semakin besar pula risiko overfitting jika tidak diimbangi dengan strategi seperti dropout, batch normalization, atau data augmentasi. Dalam eksperimen ini, tiga layer konvolusi tampaknya menjadi titik optimal di mana kompleksitas jaringan cukup untuk meningkatkan akurasi tanpa menyebabkan penurunan performa.

2.8.2. Pengaruh banyak filter per layer konvolusi

No	Banyak Filter	Macro F1 Score
1.	16 dan 32	0.6659
2.	32 dan 64	0.7028
3.	64 dan 128	0.7088

Dari hasil yang ditampilkan, terlihat bahwa semakin banyak jumlah filter yang digunakan di setiap layer konvolusi, semakin tinggi pula nilai Macro F1 Score yang dihasilkan. Penggunaan kombinasi 64 dan 128 filter menghasilkan skor tertinggi, yaitu 0.7088. Hal ini

menunjukkan bahwa dengan jumlah filter yang lebih banyak, model dapat menangkap lebih banyak variasi fitur dari data, baik yang bersifat sederhana maupun kompleks. Setiap filter bertanggung jawab untuk mengenali pola tertentu, sehingga semakin banyak filter, semakin kaya pula representasi fitur yang bisa dipelajari oleh model.

Namun, perlu dicatat bahwa menambah jumlah filter juga meningkatkan kompleksitas model dan jumlah parameter yang harus dilatih. Jika tidak ditangani dengan baik, seperti tanpa regularisasi atau data yang cukup, hal ini bisa menyebabkan overfitting. Tapi dalam kasus ini, peningkatan jumlah filter justru berdampak positif karena kemungkinan besar kompleksitas tambahan tersebut membantu model mengenali perbedaan kelas dengan lebih akurat. Maka, penggunaan 64 dan 128 filter tampaknya menjadi konfigurasi yang paling seimbang antara kemampuan ekstraksi fitur dan generalisasi model.

2.8.3. Pengaruh ukuran filter per layer konvolusi

No	Ukuran Filter	Macro F1 Score
1.	2 x 2	0.6733
2.	3 x 3	0.6897
3.	4 x 4	0.6915
4.	5 x 5	0.6782
5.	6 x 6	0.6710

Berdasarkan hasil pengujian, terlihat bahwa ukuran filter 4×4 memberikan Macro F1 Score tertinggi dibanding ukuran lainnya. Hal ini menunjukkan bahwa filter 4×4 mampu menangkap pola dan fitur penting dari data dengan lebih optimal. Ukuran ini cukup luas untuk mencakup konteks lokal yang lebih besar dibanding 2×2 atau 3×3, tetapi tidak terlalu besar hingga kehilangan detail lokal yang penting. Filter yang terlalu kecil seperti 2×2 cenderung terlalu sempit dan kurang mampu menangkap keterkaitan antar piksel secara menyeluruh, sehingga performa model menjadi kurang maksimal.

Sebaliknya, ukuran filter yang terlalu besar seperti 5×5 atau 6×6 justru menurunkan performa. Ini kemungkinan disebabkan oleh hilangnya informasi detail akibat area pengamatan yang terlalu luas, sehingga fitur halus bisa terabaikan. Selain itu, semakin besar filter, semakin besar juga jumlah parameter yang harus dipelajari, yang bisa menyebabkan model menjadi lebih kompleks dan berisiko overfitting jika tidak diimbangi dengan data yang cukup banyak. Maka dari itu, ukuran 4×4 dapat dikatakan sebagai titik tengah yang pas antara menangkap cukup konteks dan tetap mempertahankan detail penting.

2.8.4. Pengaruh jenis pooling layer yang digunakan

No	Jenis Pooling	Macro F1 Score
----	---------------	----------------

1.	Max Pooling	0.7014
2.	Average Pooling	0.6829

Berdasarkan hasil evaluasi menggunakan metrik Macro F1 Score, penggunaan *Max Pooling* menunjukkan performa yang lebih baik dibandingkan *Average Pooling*. Salah satu alasan utamanya adalah karena *Max Pooling* mampu menangkap fitur yang paling menonjol dalam sebuah area, seperti tepi atau pola yang kuat. Dalam konteks pengenalan pola atau klasifikasi, fitur-fitur seperti ini sering kali sangat menentukan. Dengan hanya mempertahankan nilai tertinggi, *Max Pooling* juga secara tidak langsung membantu mengurangi pengaruh noise atau informasi yang kurang relevan, sehingga hasil ekstraksi fitur menjadi lebih tajam dan fokus.

Di sisi lain, *Average Pooling* merata-ratakan seluruh nilai dalam sebuah area, yang membuatnya cenderung melunakkan atau meredam perbedaan antar fitur. Hal ini bisa mengaburkan informasi penting, terutama jika fitur yang dibutuhkan memiliki nilai kontras tinggi dibanding sekitarnya. Selain itu, *Max Pooling* juga menghasilkan aktivasi yang lebih spars, yang bisa membantu mencegah *overfitting* karena jaringan dipaksa untuk memperhatikan hanya bagian yang paling informatif dari data. Oleh karena itu, dalam banyak kasus, *Max Pooling* lebih efektif dalam membantu model belajar representasi yang lebih kuat, sebagaimana terlihat pada hasil yang lebih tinggi di pengujian ini.

2.8.5. Pengaruh Fungsi Aktivasi terhadap Akurasi

No	Fungsi Aktivasi	Akurasi
1.	Linear	89.55%
2.	ReLU	97.29%
3.	Sigmoid	96.16%
4.	Tanh	96.57%
5.	ELU	96.27%
6.	Leaky_relu	97.18%

ReLU bekerja sangat baik pada dataset MNIST karena kemampuannya menangani pola yang kompleks dengan lebih efisien dibandingkan fungsi aktivasi lain seperti sigmoid dan tanh. MNIST adalah dataset gambar digit yang ditulis menggunakan tangan sehingga mengandung banyak variasi bentuk dan pola. Agar FFNN bisa mengenali pola ini dengan baik, ia harus bisa menangkap hubungan non-linear dalam data. ReLU sangat cocok untuk ini karena ia hanya meneruskan nilai positif dan membiarkan nilai negatif menjadi nol, sehingga jaringan dapat belajar lebih cepat dan lebih stabil. Tidak seperti sigmoid dan tanh yang

membatasi output mereka dalam rentang kecil, ReLU membiarkan nilai positif tetap besar, sehingga jaringan bisa lebih fleksibel dalam memahami variasi data.

Keunggulan lain dari ReLU adalah kemampuannya mengatasi masalah vanishing gradient, yang sering menjadi kendala saat menggunakan sigmoid atau tanh. Dalam sigmoid, jika nilai input sangat besar atau sangat kecil, perubahan bobot menjadi sangat kecil, membuat jaringan sulit belajar, terutama di lapisan yang lebih dalam. Tanh memang sedikit lebih baik, tetapi masih mengalami masalah yang sama. ReLU menghindari hal ini karena gradiennya tetap besar untuk nilai positif, sehingga pembelajaran tetap berjalan efektif bahkan dalam jaringan yang dalam.

Selain itu, ReLU juga membuat jaringan lebih efisien dan tidak mudah overfitting. Karena nilai negatif dipetakan ke nol, tidak semua neuron akan aktif setiap saat, sehingga jaringan menjadi lebih sederhana dan lebih hemat komputasi. Ini berbeda dengan sigmoid dan tanh, yang selalu mengaktifkan semua neuron, sehingga kadang-kadang justru membuat jaringan belajar hal-hal yang tidak terlalu penting dan berisiko overfitting.

2.8.6. Perbandingan dengan Library Scikit-Learn

No	Model yang Digunakan	Akurasi
1.	Buatan Penulis	97.15%
2.	Scikit-Learn	96.27%

Model yang diimplementasikan sudah cukup akurat. Hal tersebut bisa dibuktikan dengan betapa dekatnya akurasi yang dihasilkan model penulis dan model yang disediakan oleh Scikit-Learn. Penulis mencoba melihat secara detail model scikit learn dan kemungkinan perbedaan pada akurasi dan loss dapat disebabkan oleh inisialisasi bobot yang oleh Scikit-Learn dilakukan untuk seluruh model sekaligus dan bukan pada tiap layer seperti yang dilakukan pada program ini.

2.8.7. Pengaruh Regularisasi terhadap Akurasi

No	Metode Regularisasi	Akurasi
1.	Tanpa Regularisasi	97.15%
2.	L1	96.93%
3.	L2	97.15%

Secara umum, metode regularisasi digunakan untuk mencegah overfitting dengan cara menambahkan penalti terhadap loss yang besar. Artinya, setiap kali melakukan backward propagation, besaran yang seharusnya digunakan untuk mengupdate suatu bobot akan berkurang karena adanya penalti. Metode regularisasi L1 akan mendorong model untuk memiliki bobot yang "sparse," yaitu memiliki banyak bobot dengan nilai 0. Sedangkan,

metode regularisasi L2 akan mendorong model memiliki bobot yang kecil namun bukan mendorong sparsity.

Oleh karena itu, berdasarkan tabel pengaruh regularisasi terhadap akurasi yang diberikan, dapat dilihat bahwa akurasi bila pelatihan model menggunakan regularisasi L1 menurun dibandingkan jika tanpa regularisasi. Hal ini dapat diduga disebabkan oleh sparsity sehingga ada fitur-fitur yang menjadi tidak terdeteksi. Melihat akurasi pelatihan model menggunakan regularisasi L2 seakan menandakan bahwa regularisasi tidak memperbaiki kemampuan klasifikasi model.

Namun, karena kemampuan regularisasi masing-masing metode regularisasi, meski hasil akurasi yang diberikan sama saja atau bahkan lebih buruk, selisih dari training loss dan validation loss memiliki perbedaan seperti yang dapat dilihat pada grafik sejarah loss. Model yang dilatih dengan metode regularisasi tidak mengalami overfitting yang besar bila dibandingkan dengan model yang dilatih tanpa metode regularisasi. Hal ini dapat kita lihat pada grafik sejarah loss data latih dan data validasi pelatihan model tanpa regularisasi. Pada suatu epoch tertentu, loss dari data latih mengalami penurunan namun loss dari data validasi malah mengalami kenaikan.

2.9. Hasil Analisis RNN

Berdasarkan grafik Loss Pelatihan RNN pada bagian 2.6., dapat dilihat bahwa pada pelatihan RNN, meskipun training loss turun secara signifikan, tapi validation loss untuk seluruh variasi RNN stagnan atau bahkan naik selama proses pelatihan. Hal ini menunjukkan bahwa model RNN kurang efisien/cocok untuk dilakukan pelatihan dataset NusaX-Sentiment (Bahasa Indonesia) sehingga model tidak dapat mendapatkan pola umum dari data dan terjadi overfit. Hal ini dapat terjadi karena panjang teks/*timestep* dari dataset yang kami gunakan cukup panjang, sehingga terjadi *vanishing gradient* yang menyebabkan proses dan hasil training yang dilakukan kurang baik.

2.9.1. Pengaruh jumlah layer RNN

No	Jumlah Layer RNN	Macro F1 Score Keras	Macro F1 Score Scratch
1.	1	0.3519	0.3519
2.	2	0.3477	0.3477
3.	3	0.3744	0.3744

Berdasarkan hasil evaluasi model terhadap dataset menggunakan F1-Score, jumlah layer RNN memengaruhi hasil, meskipun dalam kasus ini pengaruhnya tidak terlalu signifikan dan variasi tidak terlalu berkorelasi. Hasil ketiga variasi jumlah layer masih menunjukkan nilai Macro F1 Score yang kurang baik, yaitu dalam pada rentang nilai 3.4~3.7. Meskipun begitu,

model dengan jumlah layer paling banyak tetap memiliki Macro F1-Score yang tertinggi dibandingkan yang lain. Hasil ketiga variasi jumlah layer dengan menggunakan Keras dan Scratch menunjukkan hasil yang sama.

2.9.2. Pengaruh banyak cell RNN per layer

No	Banyak Cell RNN per Layer	Macro F1 Score Keras	Macro F1 Score Scratch
1.	8	0.3462	0.3462
2.	16	0.3726	0.3726
3.	24	0.4532	0.4532

Berdasarkan hasil evaluasi model terhadap dataset menggunakan F1-Score, jumlah cell per layer RNN memengaruhi hasil skor secara linier. Semakin banyak jumlah Cell RNN per Layer suatu model, semakin baik Macro F1 Score dari model tersebut. Meskipun begitu, hasil ketiga variasi jumlah cell per layer masih menunjukkan nilai Macro F1 Score yang kurang baik, meskipun lebih baik dari hasil variasi jumlah layer, yaitu dalam pada rentang nilai 3.4~4.5. Model RNN dengan jumlah cell 24 per layer mendapatkan nilai yang lebih tinggi secara signifikan dibandingkan variasi yang lain yaitu pada nilai 0.4532.

Hal ini menunjukkan bahwa model RNN dengan jumlah cell per layer lebih banyak dapat menangkap *meaning*/makna dari teks dengan lebih baik dibandingkan dengan model dengan jumlah cell per layer yang lebih sedikit. Hasil ketiga variasi jumlah cell per layer RNN dengan menggunakan Keras dan Scratch menunjukkan hasil yang sama.

2.9.3. Pengaruh jenis layer RNN berdasarkan arah

No	Arah	Macro F1 Score Keras	Macro F1 Score Scratch
1.	Unidirectional (Forward)	0.4382	0.4382
2.	Bidirectional	0.5400	0.5400

Berdasarkan hasil evaluasi model terhadap dataset menggunakan F1-Score, jenis arah RNN memengaruhi hasil skor secara cukup signifikan. Model RNN dengan arah Bidirectional (2 arah) memiliki Macro F1 Score yang lebih baik secara cukup signifikan dibandingkan model RNN dengan arah Unidirectional (1 arah). Meskipun begitu, hasil kedua variasi arah RNN, masih menunjukkan nilai Macro F1 Score yang belum optimal, meskipun lebih baik dari hasil variasi-variasi sebelumnya. Model RNN dengan arah bidirectional mendapatkan nilai Macro

F1-Score 0.5400 yang lebih tinggi secara signifikan dibandingkan variasi unidirectional yang berada pada nilai 0.4382.

Hal ini menunjukkan bahwa model RNN Bidirectional dapat menangkap *meaning*/makna dari teks secara lebih baik/lebih komprehensif dibandingkan dengan model RNN Unidirectional. Hal ini dapat menunjukkan bahwa training yang dilakukan dari depan ke belakang dan belakang ke depan mungkin menangkap pola-pola atau makna yang tidak dapat didapatkan jika hanya dilakukan dari satu arah saja (unidirectional). Hasil kedua variasi arah dengan menggunakan Keras dan Scratch menunjukkan hasil yang sama.

2.10. Hasil Analisis LSTM

Berdasarkan grafik Loss Pelatihan LSTM pada bagian 2.7., dapat dilihat bahwa berbeda dengan grafik Loss Pelatihan RNN pada bagian sebelumnya, model LSTM dapat melakukan training yang memperkecil train dan validation loss secara bersamaan. Hal ini menunjukkan bahwa model LSTM lebih baik dalam mendapatkan pola umum dari data dibandingkan model RNN. Hal ini dapat terjadi karena meskipun panjang teks/*timestep* dari dataset yang kami gunakan cukup panjang, LSTM dapat atau jauh lebih baik dalam menangani *vanishing gradient* dibandingkan dengan RNN.

2.10.1. Pengaruh jumlah layer LSTM

No	Jumlah Layer LSTM	Macro F1 Score Keras	Macro F1 Score Scratch
1.	1	0.4525	0.4525
2.	2	0.4828	0.4828
3.	3	0.6866	0.6866

Berdasarkan hasil evaluasi model terhadap dataset menggunakan F1-Score, jumlah layer RNN memengaruhi hasil skor secara positif. Semakin banyak jumlah Layer suatu model, semakin baik Macro F1 Score dari model tersebut. Hasil dari variasi Layer berjumlah tiga menunjukkan Macro F1-Score yang secara signifikan lebih baik dari kedua variasi yang lain, yaitu dengan nilai 0.6866 (dibandingkan dengan 0.4525 untuk 1 layer dan 0.4828 untuk 2 layer)

Hal ini menunjukkan bahwa dalam kasus ini, model LSTM dengan jumlah layer lebih banyak (yang di-test hingga 3 layer) dapat menangkap *meaning*/makna dari teks dengan lebih baik dibandingkan dengan model dengan jumlah layer yang lebih sedikit.

2.10.2. Pengaruh banyak cell LSTM per layer

No	Jumlah Cell LSTM	Macro F1 Score Keras	Macro F1 Score Scratch
----	------------------	----------------------	------------------------

1.	8	0.5553	0.5553
2.	16	0.4501	0.4501
3.	24	0.4808	0.4808

Berdasarkan hasil evaluasi model terhadap dataset menggunakan F1-Score, jumlah layer RNN tidak terlalu berkorelasi dengan nilai Macro F1-Score dari hasil yang didapat. Hasil dari variasi jumlah cell per Layer berjumlah 8 (paling sedikit) menunjukkan Macro F1-Score yang lebih baik dari kedua variasi yang lain, yaitu dengan nilai 0.5553 (dibandingkan dengan 0.4501 untuk 16 cell per layer dan 0.4808 untuk 24 cell per layer).

Hal ini menunjukkan bahwa dalam kasus ini, model LSTM dengan jumlah cell per layer lebih banyak (yang di-test hingga 3 layer) tidak menjamin bahwa model LSTM dapat menangkap *meaning*/makna dari teks dengan lebih baik dibandingkan dengan model dengan jumlah layer yang lebih sedikit.

2.10.3. Pengaruh jenis layer LSTM berdasarkan arah

No	Arah	Macro F1 Score Keras	Macro F1 Score Scratch
1.	Unidirectional (Forward)	0.5553	0.5553
2.	Bidirectional	0.4501	0.4501

Berdasarkan hasil evaluasi model terhadap dataset menggunakan F1-Score, jenis arah LSTM memengaruhi hasil skor secara cukup signifikan. Berbeda dengan RNN, model LSTM dengan arah Bidirectional (2 arah) memiliki Macro F1 Score yang lebih buruk dibandingkan model LSTM dengan arah Unidirectional (1 arah). Meskipun begitu, jika dilihat dari grafik training lossnya, dapat dilihat bahwa sebenarnya hingga epoch training ke 20-30 LSTM bidirectional masih mendapatkan penurunan validation loss yang lebih baik dari unidirectional LSTM dan setelah itu validation loss malah naik secara cukup signifikan meskipun training loss semakin kecil sehingga membuat nilai loss bidirectional LSTM menjadi lebih buruk dibandingkan unidirectional LSTM. Hal ini dapat disebabkan karena model bidirectional terlalu *overfit* terhadap training data.

Hal ini menunjukkan bahwa model RNN Bidirectional dapat menangkap *meaning*/makna dari teks secara lebih baik/lebih komprehensif dibandingkan dengan model RNN Unidirectional jika training dilakukan secara optimal (epoch secukupnya sehingga tidak terjadi overfit). Hal ini juga dapat berarti bahwa bahwa training yang dilakukan dari depan ke belakang dan belakang ke depan mungkin menangkap pola-pola atau makna yang tidak dapat didapatkan jika hanya dilakukan dari satu arah saja (unidirectional), meskipun pola-pola tersebut mungkin saja tidak umum sehingga membuat data overfit terhadap training data.

BAB III

KESIMPULAN DAN SARAN

3.1 Kesimpulan

Pada tugas ini, kami berhasil mengimplementasikan model Convolutional Neural Network (CNN) untuk klasifikasi gambar menggunakan dataset CIFAR-10. Model diuji menggunakan 1000 data uji pertama dan dibandingkan dengan model Keras yang memiliki arsitektur identik dan bobot yang sama. Hasil evaluasi menunjukkan bahwa model buatan sendiri dapat memberikan prediksi yang sebanding, ditunjukkan dengan skor F1 makro yang kompetitif.

Dari hasil percobaan, diketahui bahwa **penambahan jumlah layer konvolusi** dan **peningkatan jumlah kernel** dapat membantu model dalam mengekstraksi fitur yang lebih kompleks dan mendalam. Ukuran filter yang konsisten (3×3) efektif dalam menangkap pola lokal pada gambar. Penggunaan **max pooling dengan ukuran 2×2** juga terbukti penting dalam mengurangi dimensi data secara bertahap tanpa kehilangan informasi penting. Namun demikian, peningkatan kompleksitas jaringan perlu diimbangi dengan pertimbangan *overfitting* dan efisiensi komputasi.

Model kedua yang kami implementasikan adalah RNN yang dapat digunakan untuk permasalahan klasifikasi. Model dilatih dan diuji terhadap dataset klasifikasi sentimen pada Bahasa Indonesia NusaX-Sentiment. Model yang dikembangkan dapat menerima input satuan, maupun secara batch, namun tidak terdapat *backpropagation*. Bobot yang digunakan pada model diambil dari model Keras dengan arsitektur serupa (jumlah dan jenis layer yang sama) yang telah dilatih. Pengujian model dilakukan dengan variasi jumlah layer RNN, jumlah cell RNN, dan tipe arah RNN. Hasil pengujian menunjukkan bahwa semakin banyak jumlah layer RNN, semakin banyak jumlah cell RNN, dan Bidirectional RNN memberikan hasil yang lebih baik. Hasil pengujian juga menunjukkan bahwa hasil prediksi dari model berhasil memberikan skor yang sama dengan hasil prediksi dari model Keras.

Pada tugas ini, kami mengimplementasikan LSTM yang dapat digunakan untuk permasalahan klasifikasi. Implementasi LSTM mirip dengan RNN yang dijelaskan

sebelumnya, yaitu model dilatih dan diuji terhadap dataset klasifikasi sentimen pada Bahasa Indonesia NusaX. Model yang dikembangkan dapat menerima input satuan, maupun secara batch, namun tidak terdapat *backpropagation*. Bobot yang digunakan pada model diambil dari model Keras dengan arsitektur serupa (jumlah dan jenis layer yang sama) yang telah dilatih. Pengujian model dilakukan dengan variasi jumlah layer LSTM, jumlah cell LSTM, dan tipe arah LSTM. Hasil pengujian menunjukkan bahwa semakin banyak jumlah layer LSTM, semakin baik nilai Macro F1-Score. Banyak jumlah cell LSTM kurang berkorelasi dengan nilai Macro F1-Score. Jenis LSTM Bidirectional memberikan hasil yang lebih baik hingga *epoch* tertentu, kemudian akan menjadi lebih buruk secara signifikan jika epoch terlalu banyak. Hasil pengujian juga menunjukkan bahwa hasil prediksi dari model berhasil memberikan skor yang sama dengan hasil prediksi dari model Keras.

3.2 Saran

Berikut beberapa saran untuk pengembangan ke depan:

- 1) Pemanfaatan GPU: Untuk efisiensi yang lebih tinggi, implementasi CNN sebaiknya mempertimbangkan penggunaan GPU, terutama saat model diperbesar atau digunakan pada dataset yang lebih kompleks. Karena keterbatasan perangkat, implementasi CNN pada proyek ini tidak menggunakan GPU dan lebih difokuskan pada pemrograman berbasis **NumPy array** untuk kemudahan eksplorasi dan pembelajaran
- 2) Optimalisasi Perhitungan Konvolusi: Disarankan untuk menggunakan pendekatan berbasis perkalian matriks seperti *im2col* sebagai alternatif dari metode konvolusi konvensional (*sliding kernel*). Pendekatan ini dapat mempercepat komputasi bahkan tanpa GPU, terutama ketika menggunakan pustaka seperti NumPy atau framework lain yang telah mengoptimalkan operasi linear algebra.
- 3) Implementasi *backpropagation* pada model RNN dan LSTM agar tidak perlu melakukan pelatihan pada model Keras terlebih dahulu dan mengambil bobot.

PEMBAGIAN TUGAS

Kegiatan	Nama (NIM)
CNN	Immanuel Sebastian Girsang (13522058)
RNN	Fedrianz Dharma (13522090)
LSTM	Aurelius Justin Philo Fanjaya (13522020)

REFERENSI

<https://github.com/karpathy/micrograd/blob/master/micrograd/engine.py>

https://github.com/mattij/autodidact/blob/master/autograd/numpy/numpy_vjps.py

<https://douglasorr.github.io/2021-11-autodiff/article.html>

▶ Softmax - Pullback/vJp rule

<https://builtin.com/data-science/l2-regularization>