

# **LAPORAN TUGAS BESAR II**

IF3270 Pembelajaran Mesin

## **Convolutional Neural Network dan Recurrent Neural Network**



### **Kelompok 9**

Farhan Nafis Rayhan 13522037

Bagas Sambega Rosyada 13522071

Raden Fransisco Trianto B. 13522091

**TEKNIK INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG**

# DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>2</b>
<b>BAB I</b>	
<b>DESKRIPSI PERMASALAHAN.....</b>	<b>3</b>
<b>BAB II</b>	
<b>PEMBAHASAN.....</b>	<b>5</b>
A. Implementasi.....	5
a. Convolutional Neural Network (CNN).....	5
b. Recurrent Neural Network (RNN).....	5
c. Long Short-Term Memory Network (LSTM).....	5
B. Hasil Pengujian.....	6
a. CNN.....	6
b. RNN.....	8
c. LSTM.....	8
<b>BAB III</b>	
<b>KESIMPULAN DAN SARAN.....</b>	<b>15</b>
A. Kesimpulan.....	15
B. Saran.....	15
<b>PEMBAGIAN TUGAS.....</b>	<b>17</b>
<b>REFERENSI.....</b>	<b>18</b>

# BAB I

## DESKRIPSI PERMASALAHAN

**Convolutional Neural Network (CNN)** dan **Recurrent Neural Network (RNN)** adalah model Artificial Neural Network (ANN) awal yang mendorong perkembangan besar di bidang Machine Learning dan kemudian dikenal sebagai model **Deep Learning**. CNN sangat efektif dalam menangani data dengan informasi spasial seperti gambar dengan adanya Convolutional Layer. RNN efektif dalam memproses informasi sequential dari data sehingga unggul dalam mengatasi data sequential seperti teks, video, speech, dsb. Untuk mengatasi masalah pada RNN dikembangkan **Long Short-Term Memory Network (LSTM)** yang dapat menyimpan memori atau konteks dalam melakukan prediksi.

Pada Tugas Besar II ini, kami diberikan permasalahan untuk mengimplementasikan Forward Propagation untuk model CNN dan RNN, termasuk variannya yaitu LSTM, dalam bahasa Python dari scratch. Untuk bagian training atau backward propagation, kami menggunakan model dari library Keras sehingga implementasi scratch harus sesuai dengan struktur model/kelas/fungsi yang digunakan pada Keras.

Dari hasil implementasi, kami diminta untuk melakukan testing dimana untuk model CNN kami menggunakan **CIFAR-10** dan untuk RNN serta LSTM kami menggunakan **NusaX-Sentiment Bahasa Indonesia**. Testing dilakukan beberapa kali dengan hyperparameter yang berbeda-beda untuk mengetahui dampak hyperparameter terhadap model.

Berikut adalah ketentuan utama yang ada pada tugas kami:

### ★ Convolutional Neural Network (CNN):

- Pelatihan model menggunakan CIFAR-10.
- Wajib memiliki layer Conv2D, Pooling, Flatten/Global Pooling, dan Dense.
- Analisis pengaruh jumlah layer konvolusi, banyak filter, ukuran filter, dan jenis pooling.
- Forward propagation dari scratch berdasarkan bobot hasil pelatihan.

### ★ Simple Recurrent Neural Network (RNN) dan LSTM:

- Pelatihan model untuk klasifikasi teks menggunakan dataset NusaX-Sentiment.

- Wajib memiliki layer Embedding, RNN (bidirectional/unidirectional), Dropout, dan Dense.
- Analisis pengaruh jumlah layer, jumlah cell, dan arah propagasi.
- Forward propagation dari scratch berdasarkan bobot hasil pelatihan.

★ **Fitur Bonus:**

- Kemampuan batch input pada semua model from scratch, memungkinkan proses forward propagation untuk lebih dari satu data sekaligus.

Untuk ketentuan lebih lengkap, dapat dilihat pada tautan berikut:

 [Spesifikasi Tugas Besar 2 IF3270 Pembelajaran Mesin](#)

# BAB II

## PEMBAHASAN

### A. Implementasi

#### a. Convolutional Neural Network (CNN)

CNN sendiri terdiri dari beberapa layer berbeda yaitu:

1. Conv2D atau Convolutional layer

*Layer* ini adalah lapisan inti CNN yang bertugas mengekstrak fitur dari *input*. *Layer* ini menerapkan operasi konvolusi menggunakan *filter/kernels* kecil untuk menghasilkan *feature maps*.

Setiap kernel menggeser (*slide*) di atas input dan menghitung hasil *dot product* antara bobot kernel dan bagian dari input.

2. *Pooling* layers

Layer *pooling* berfungsi untuk mereduksi dimensi spasial feature map. Secara umum terdapat dua tipe, yaitu *max pooling* dan *average pooling*. *Max pooling* mengambil nilai maksimum sedangkan *average pooling* menggunakan nilai rata-rata dari semua elemen pada *window* dari kernel.

3. *Flatten* layer

Output dari layer *pooling* yang masih berbentuk tensor  $N$  dimensi diubah menjadi vektor 1D agar dapat diteruskan ke *Dense layer*. Operasi ini tidak mengubah nilai, hanya bentuk dari data.

4. Dense layer

Layer ini melakukan perhitungan linier pada input vektor yaitu:

$$y = \text{activation}(W \cdot x + b)$$

Dense layer berada pada akhir dari model untuk memberikan nilai *output* atau kelas yang dihasilkan.

Implementasi dari CNN untuk model Keras berada di *file* 'model/cnn.py' pada kelas CNNModel. Sementara untuk membantu proses pelatihan, digunakan kelas CNNTrain yang akan diinisialisasi saat fungsi *train* dipanggil pada model CNN. Model CNN yang diimplementasikan dari *scratch* berada pada *file* *scratch/cnn\_forward.py* di kelas ScratchCNN. Setiap *layer* juga diimplementasikan secara modular untuk *layer* konvolusi 2D, PoolingLayer, FlattenLayer, dan DenseLayer. Untuk memudahkan proses perbandingan, dibuat *pipeline* antara CNN Keras dengan CNN dari *scratch* di *file* 'pipeline/cnn\_pipeline.py'.

## b. Simple Recurrent Neural Network (Simple RNN)

RNN (Recurrent Neural Network) adalah arsitektur neural network yang dirancang untuk memproses data sekuensial, di mana urutan dan konteks temporal/spasial sangat penting. Berbeda dengan CNN (Convolutional Neural Network) yang memproses setiap input (misalnya gambar) secara independen, RNN memiliki mekanisme memori internal (dalam bentuk hidden state) yang memungkinkannya menyimpan dan meneruskan informasi dari langkah-langkah sebelumnya dalam sekuens.

Karakteristik Kunci dari RNN adalah adanya Hidden State sebagai Memori Internal. Setiap langkah (time step) dalam RNN menerima input saat ini ( $x_t$ ) dan hidden state sebelumnya ( $h_{t-1}$ ). Hidden state ini berfungsi sebagai "memori" yang menyimpan informasi kontekstual dari langkah-langkah sebelumnya. Secara dasar formulanya adalah

$$h_t = \text{activation}(W_h \cdot h_{t-1} + W_x \cdot x_t + b)$$

Selain itu, karakteristik kunci lain dari RNN adalah pemrosesan data secara sekuensial.

Pada tugas besar ini, Simple RNN yang diimplementasikan terdiri atas 6 layer dengan urutan sebagai berikut:

### 1. Tokenization Layer

Tokenization adalah langkah awal dalam pemrosesan teks yang mengubah input berupa kata atau karakter menjadi representasi numerik. Layer ini membagi teks menjadi unit-unit kecil (token) baik dalam level kata (word-level) maupun karakter (character-level). Setiap token kemudian dipetakan ke indeks unik dalam sebuah vocabulary. Parameter penting

dalam tokenization meliputi ukuran vocabulary (`vocab_size`) dan panjang maksimum sekuens (`max_sequence_length`), yang menentukan berapa banyak token yang akan diproses dalam satu input.

## **2. Embedding Layer**

Bertugas mengkonversi token yang berupa bilangan bulat (integer) menjadi vektor berdimensi tetap yang kaya makna semantik. Setiap kata direpresentasikan sebagai titik dalam ruang vektor multidimensi, di mana kata-kata dengan makna serupa akan memiliki embedding yang mirip. Parameter utama dalam layer ini adalah `embedding_dim` (misalnya 50, 100, atau 300), yang menentukan seberapa besar dimensi vektor embedding. Embedding bisa diinisialisasi secara acak atau menggunakan *pretrained word vectors*.

## **3. Recurrent Layer**

Inti dari arsitektur RNN, di mana setiap langkah waktu (time step) memproses input sambil mempertahankan hidden state sebagai memori kontekstual. Pada RNN dasar, informasi hanya mengalir maju (unidirectional). Pada Simple RNN unit yang digunakan adalah RNN seperti biasa. Parameter pada layer ini adalah apakah mengembalikan output per langkah waktu (`return_sequences=True`) atau hanya output terakhir (`return_sequences=False`).

## **4. Bidirectional Layer**

Bidirectional Layer memperluas RNN tradisional dengan memproses data dalam dua arah: maju (forward) dan mundur (backward). Dengan menggabungkan kedua arah, model dapat memahami konteks yang lebih lengkap, seperti pengaruh kata-kata sebelum dan sesudah suatu token. Karena memproses dua arah, komputasinya lebih berat dibanding RNN satu arah, tetapi pada beberapa kasus dapat memberikan akurasi lebih tinggi.

## **5. Dropout Layer**

Dropout Layer berfungsi sebagai teknik regularisasi untuk mencegah overfitting dengan secara acak menonaktifkan sebagian neuron selama pelatihan. Dengan cara ini, model tidak terlalu bergantung pada fitur tertentu dan menjadi lebih robust. Parameter utamanya adalah `rate` (misalnya 0.2 atau 0.5), yang menentukan persentase neuron yang dinonaktifkan. Dropout biasanya diterapkan setelah layer RNN atau Embedding.

## 6. Dense Layer

Layer akhir yang mengubah representasi dari layer sebelumnya menjadi output yang diinginkan, seperti klasifikasi atau regresi. Pada tugas NLP, layer ini sering menggunakan aktivasi softmax untuk klasifikasi multi-kelas atau sigmoid untuk klasifikasi biner. Parameter utamanya adalah jumlah neuron (units) dan fungsi aktivasi (activation). Misalnya, untuk klasifikasi sentimen positif/negatif, Dense Layer bisa memiliki 1 neuron dengan aktivasi sigmoid, sedangkan untuk klasifikasi multi-label bisa menggunakan softmax.

Implementasi dari RNN untuk model Keras berada di *file* 'model/rnn.py' pada kelas RNNModel. Sementara untuk membantu proses pelatihan, digunakan kelas RNNTrain yang akan diinisialisasi saat fungsi *train* dipanggil pada model RNN. Model RNN yang diimplementasikan dari *scratch* berada pada *file* *scratch/rnn\_forward.py* di kelas ScratchRNN. Setiap *layer* juga diimplementasikan secara modular untuk *layer* Embedding, SingleRNN, Bidirectional, Dense. Untuk memudahkan proses perbandingan, dibuat *pipeline* antara RNN Keras dengan RNN dari *scratch* di *file* 'pipeline/rnn\_pipeline.py'. Eksplorasi menggunakan pipeline yang telah dibuat dilakukan pada notebook 'RNN.ipynb'.

### c. Long Short-Term Memory Network (LSTM)

LSTM merupakan variasi dari RNN yang menggunakan *gates* untuk menyimpan informasi penting jangka panjang dan membuang informasi yang bernilai tidak relevan. LSTM digunakan untuk memecahkan masalah *vanishing gradient*, di mana jaringan *neural network* mendapatkan nilai gradien yang sangat kecil sehingga perubahan gradien malah tidak terlalu signifikan. LSTM tersusun atas 3 gerbang, yaitu *input gate*, *forget gate*, dan *output gate*.

#### 1. Cell State

Merupakan memori jangka panjang yang tidak banyak dimodifikasi oleh setiap *gate* saat bertemu *unit* LSTM. *Cell state* berfungsi untuk membuat gradien tidak mudah hilang dan mengatasi *vanishing gradient*. Formula *cell state* dapat diformulasikan dengan,

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

Dengan  $\tilde{C}_t$  diformulasikan,

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

#### 2. Forget Gate



*Gate* yang digunakan untuk menentukan seberapa banyak informasi lama yang perlu dibuang. Nilai *gate* berada di antara 0 dan 1, di mana saat *gate* bernilai 1, informasi akan dipertahankan, dan jika mendekati 0, informasi akan dibuang. *Forget gate* dirumuskan dengan,

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

### 3. Input Gate

*Gate* yang digunakan untuk menentukan seberapa banyak informasi baru yang akan digunakan untuk memperbarui *cell state*. *Input gate* diformulasikan dengan,

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i)$$

### 4. Output Gate

*Gate* yang digunakan sebagai *output* untuk unit selanjutnya. *Output gate* diformulasikan dengan,

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o)$$

$$h_t = o_t \cdot \tanh(c_t)$$

Implementasi dari LSTM untuk model Keras berada di *file* ‘model/lstm.py’ pada kelas LSTMModel. Model LSTM yang diimplementasikan dari *scratch* berada pada *file* *scratch/lstm\_forward.py* di kelas ScratchLSTM. Setiap *layer* juga diimplementasikan secara modular untuk *layer* LSTMLayer yang berisi fungsi-fungsi saat *unidirectional* LSTM, DenseLayer dan BidirectionalLayer untuk *bidirectional* LSTM. Untuk memudahkan proses perbandingan, dibuat *pipeline* antara LSTM Keras dengan LSTM dari *scratch* di *file* ‘pipeline/lstm\_pipeline.py’.

## B. Hasil Pengujian

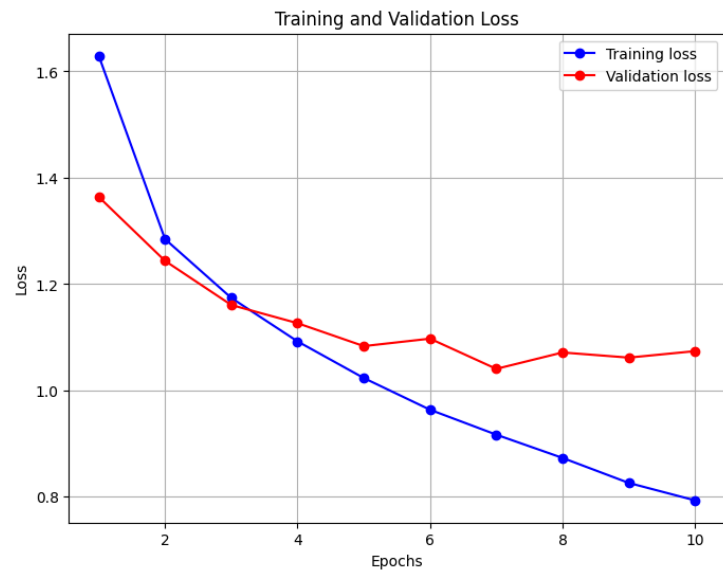
### a. CNN

#### 1. Variasi Jumlah Layer Konvolusi

Jumlah Layer	Hasil
Constant Parameter	
Epoch: 30	

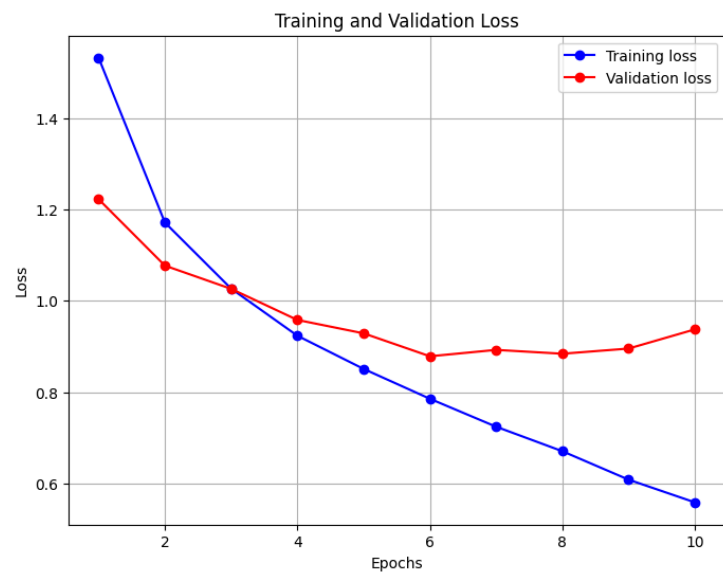
Batch size: 64

1



F1 Score:  
Keras F1 = 0.630284  
Scratch F1 = 0.630284

2



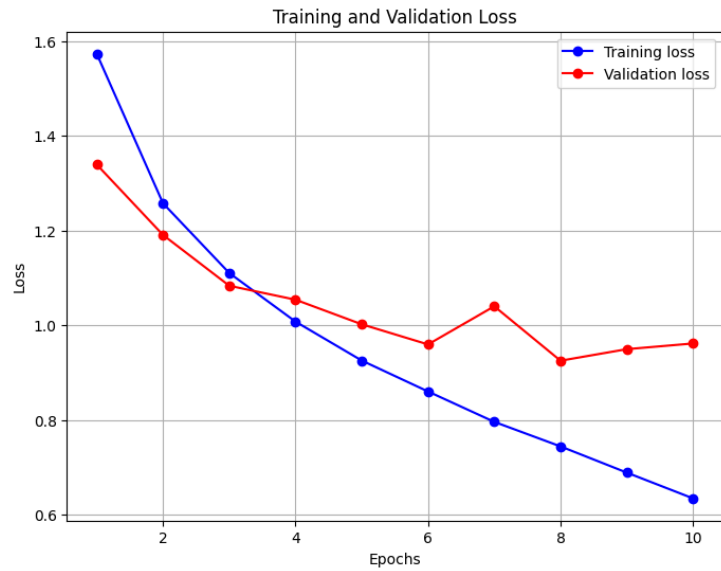
F1 Score:  
Keras F1 = 0.688381  
Scratch F1 = 0.688381

3	<p>Training and Validation Loss</p> <table border="1"><thead><tr><th>Epochs</th><th>Training loss</th><th>Validation loss</th></tr></thead><tbody><tr><td>1</td><td>1.58</td><td>1.33</td></tr><tr><td>2</td><td>1.23</td><td>1.11</td></tr><tr><td>3</td><td>1.07</td><td>1.00</td></tr><tr><td>4</td><td>0.96</td><td>0.92</td></tr><tr><td>5</td><td>0.88</td><td>0.98</td></tr><tr><td>6</td><td>0.82</td><td>0.88</td></tr><tr><td>7</td><td>0.77</td><td>0.84</td></tr><tr><td>8</td><td>0.73</td><td>0.86</td></tr><tr><td>9</td><td>0.70</td><td>0.84</td></tr><tr><td>10</td><td>0.68</td><td>0.83</td></tr></tbody></table> <p>F1 Score: Keras F1 = 0.711548 Scratch F1 = 0.711548</p>	Epochs	Training loss	Validation loss	1	1.58	1.33	2	1.23	1.11	3	1.07	1.00	4	0.96	0.92	5	0.88	0.98	6	0.82	0.88	7	0.77	0.84	8	0.73	0.86	9	0.70	0.84	10	0.68	0.83
Epochs	Training loss	Validation loss																																
1	1.58	1.33																																
2	1.23	1.11																																
3	1.07	1.00																																
4	0.96	0.92																																
5	0.88	0.98																																
6	0.82	0.88																																
7	0.77	0.84																																
8	0.73	0.86																																
9	0.70	0.84																																
10	0.68	0.83																																
Analisis	<p>Berdasarkan percobaan yang dilakukan, semakin tinggi jumlah <i>layer</i> yang digunakan, nilai F1 <i>score</i> yang didapat semakin bagus. Hal ini karena jumlah <i>layer</i> yang banyak memungkinkan pola direpresentasikan secara lebih detail.</p>																																	

## 2. Variasi Banyak Filter

Jumlah Filter	Hasil
<p><b>Constant Parameter</b></p> <p>Epoch: 10 Layer konvolusi: 2 Batch size: 64</p>	

[16,32]

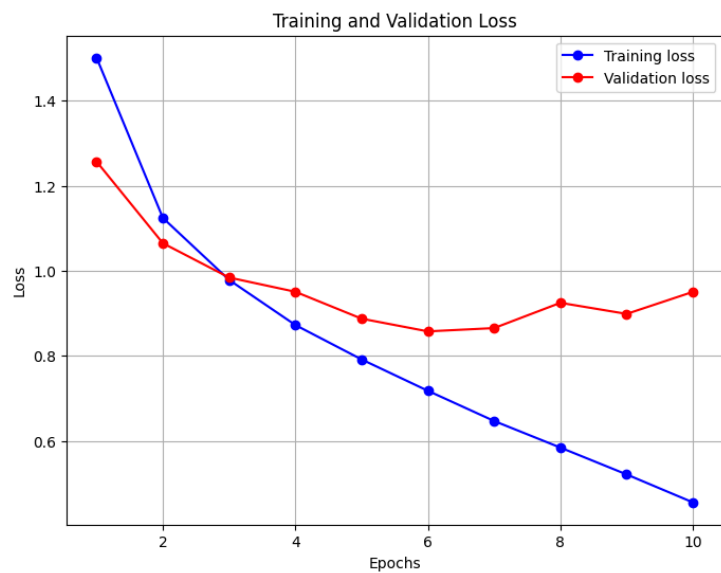


F1 Score:

Keras F1: 0.674614

Scratch F1: 0.674614

[32,64]



F1 Score:

Keras F1: 0.689333

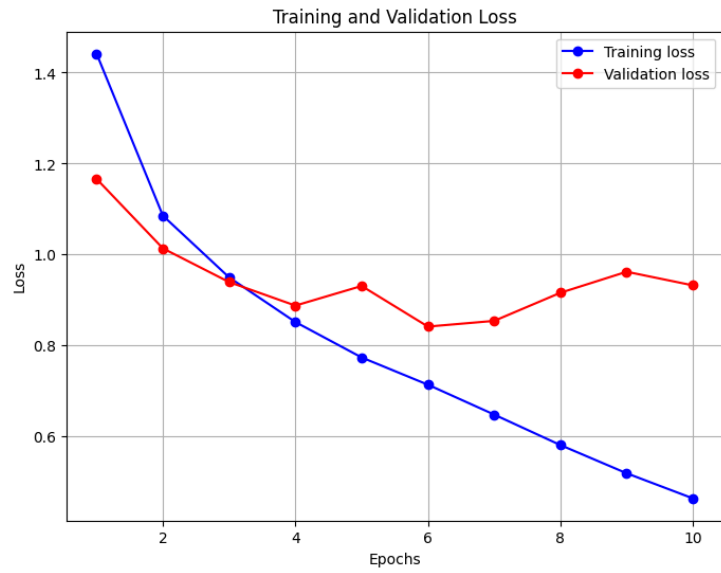
Scratch F1: 0.689333

[64,128]	<div><p>Training and Validation Loss</p><table border="1"><thead><tr><th>Epochs</th><th>Training loss</th><th>Validation loss</th></tr></thead><tbody><tr><td>1</td><td>1.45</td><td>1.15</td></tr><tr><td>2</td><td>1.05</td><td>0.98</td></tr><tr><td>3</td><td>0.88</td><td>0.97</td></tr><tr><td>4</td><td>0.78</td><td>0.90</td></tr><tr><td>5</td><td>0.68</td><td>0.86</td></tr><tr><td>6</td><td>0.60</td><td>0.91</td></tr><tr><td>7</td><td>0.52</td><td>0.88</td></tr><tr><td>8</td><td>0.44</td><td>0.93</td></tr><tr><td>9</td><td>0.37</td><td>0.98</td></tr><tr><td>10</td><td>0.30</td><td>1.04</td></tr></tbody></table></div> <div><p>F1 Score: Keras F1: 0.702659 Scratch F1: 0.702659</p></div>	Epochs	Training loss	Validation loss	1	1.45	1.15	2	1.05	0.98	3	0.88	0.97	4	0.78	0.90	5	0.68	0.86	6	0.60	0.91	7	0.52	0.88	8	0.44	0.93	9	0.37	0.98	10	0.30	1.04
Epochs	Training loss	Validation loss																																
1	1.45	1.15																																
2	1.05	0.98																																
3	0.88	0.97																																
4	0.78	0.90																																
5	0.68	0.86																																
6	0.60	0.91																																
7	0.52	0.88																																
8	0.44	0.93																																
9	0.37	0.98																																
10	0.30	1.04																																
Analisis	<p>Berdasarkan percobaan yang dilakukan, semakin tinggi jumlah <i>filter</i> yang digunakan, nilai F1 score semakin tinggi. Hal ini karena semakin banyak jumlah filter, semakin banyak fitur yang dapat diidentifikasi.</p>																																	

### 3. Variasi Ukuran Filter

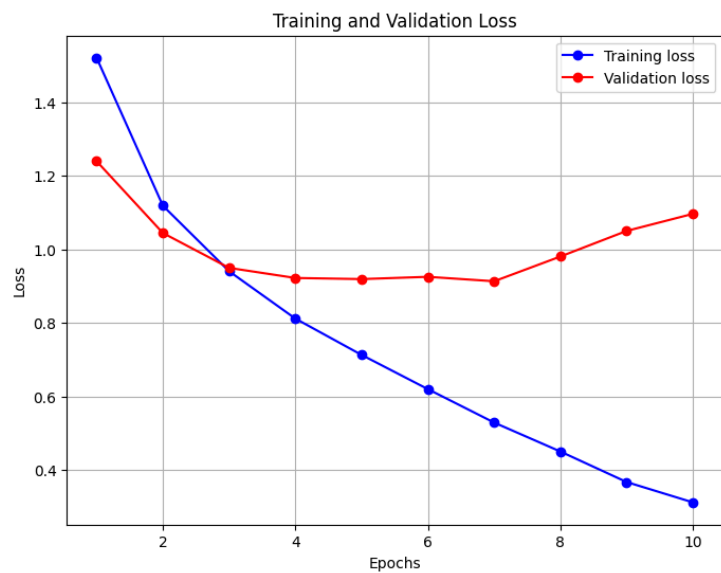
Ukuran Filter	Hasil
<p style="text-align: center;"><b>Constant Parameter</b></p> <p>Epoch: 10  Layer konvolusi: 2  Batch size: 64</p>	

3x3



F1 Score:  
Keras F1 = 0.706905  
Scratch F1 = 0.706905

5x5

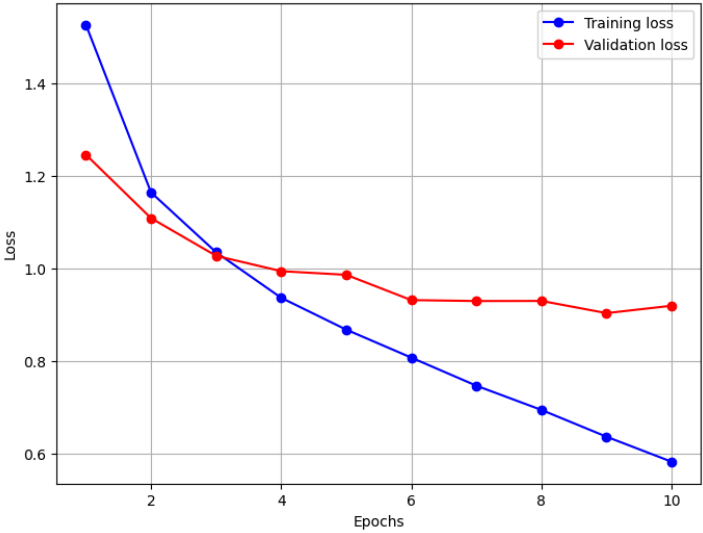
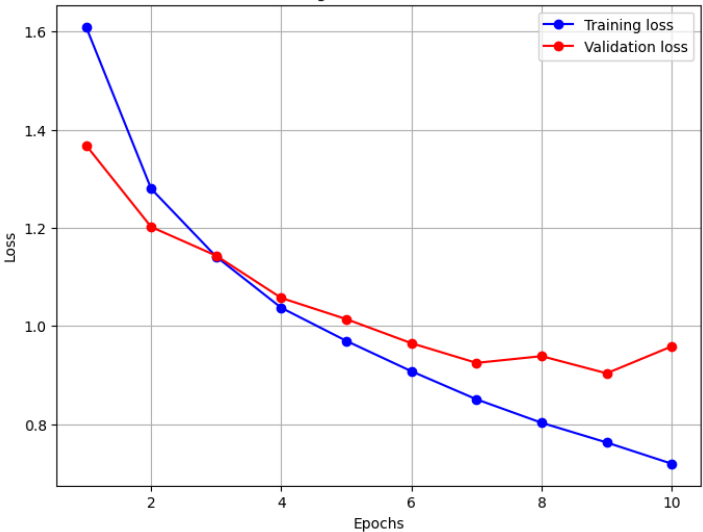


F1 Score:  
Keras F1 = 0.696354  
Scratch F1 = 0.696354

7x7	<p>Training and Validation Loss</p> <table><thead><tr><th>Epochs</th><th>Training loss</th><th>Validation loss</th></tr></thead><tbody><tr><td>1</td><td>1.5</td><td>1.3</td></tr><tr><td>2</td><td>1.2</td><td>1.1</td></tr><tr><td>3</td><td>1.0</td><td>1.1</td></tr><tr><td>4</td><td>0.9</td><td>0.95</td></tr><tr><td>5</td><td>0.8</td><td>0.95</td></tr><tr><td>6</td><td>0.7</td><td>0.9</td></tr><tr><td>7</td><td>0.6</td><td>1.0</td></tr><tr><td>8</td><td>0.5</td><td>1.0</td></tr><tr><td>9</td><td>0.45</td><td>1.05</td></tr><tr><td>10</td><td>0.35</td><td>1.15</td></tr></tbody></table> <p>F1 Score: Keras F1 = 0.679140 Scratch F1 = 0.679140</p>	Epochs	Training loss	Validation loss	1	1.5	1.3	2	1.2	1.1	3	1.0	1.1	4	0.9	0.95	5	0.8	0.95	6	0.7	0.9	7	0.6	1.0	8	0.5	1.0	9	0.45	1.05	10	0.35	1.15
Epochs	Training loss	Validation loss																																
1	1.5	1.3																																
2	1.2	1.1																																
3	1.0	1.1																																
4	0.9	0.95																																
5	0.8	0.95																																
6	0.7	0.9																																
7	0.6	1.0																																
8	0.5	1.0																																
9	0.45	1.05																																
10	0.35	1.15																																
Analisis	<p>Berdasarkan percobaan yang dilakukan, semakin besar ukuran <i>kernel</i> yang digunakan, nilai F1 score yang didapat semakin kecil. Hal ini karena semakin besar kernel yang digunakan, kernel melakukan <i>pooling</i> untuk bagian gambar yang lebih besar pula, sehingga dapat kehilangan detail dan nilai spasial.</p>																																	

#### 4. Variasi Jenis Pooling Layer

Pooling	Hasil
<p>Epoch: 30 Layer: 3 Batch size: 32 Unit: 10</p>	<p><b>Constant Parameter</b></p>

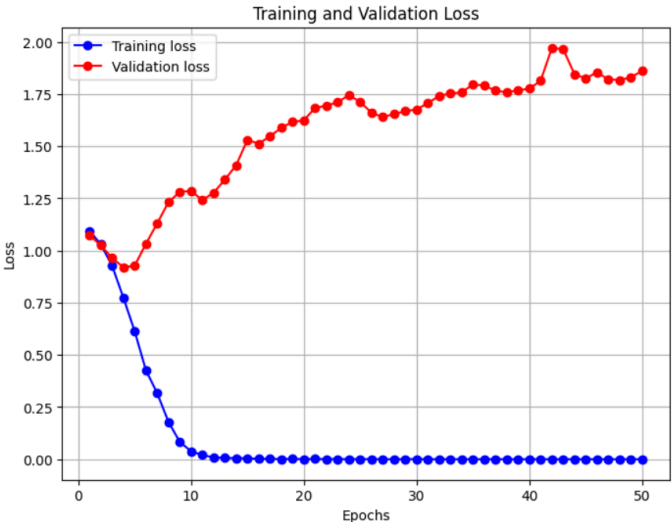
MaxPooling	<p data-bbox="971 218 1211 239">Training and Validation Loss</p>  <p data-bbox="699 785 995 890"> F1 Score:  Keras F1 = 0.686536  Scratch F1 = 0.686536 </p>
Average Pooling	<p data-bbox="971 932 1211 953">Training and Validation Loss</p>  <p data-bbox="699 1499 995 1604"> F1 Score:  Keras F1 = 0.666649  Scratch F1 = 0.666649 </p>
Analisis	<p data-bbox="699 1640 1414 1850"> Berdasarkan percobaan yang dilakukan, <i>max pooling</i> memberikan F1 score yang sedikit lebih baik dari <i>average pooling</i> karena F1 score dihitung berdasarkan kemampuan model membedakan antarkelas, sehingga <i>maxpooling</i> memiliki kemampuan membedakan setiap fiturnya dengan mengambil nilai maksimum fitur, </p>


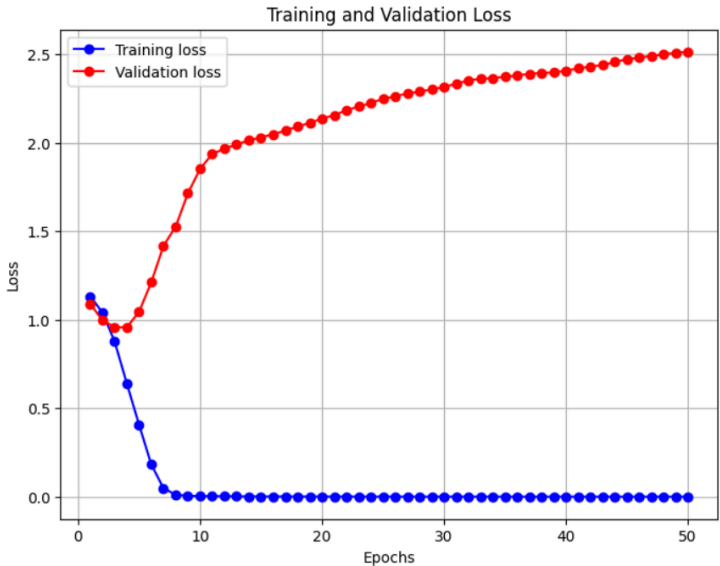


	sementara <i>average pooling</i> merata-ratakan semua nilai, bahkan termasuk <i>noise</i> sekalipun.
--	--

**b. Simple RNN**


1. Variasi Jumlah Layer RNN

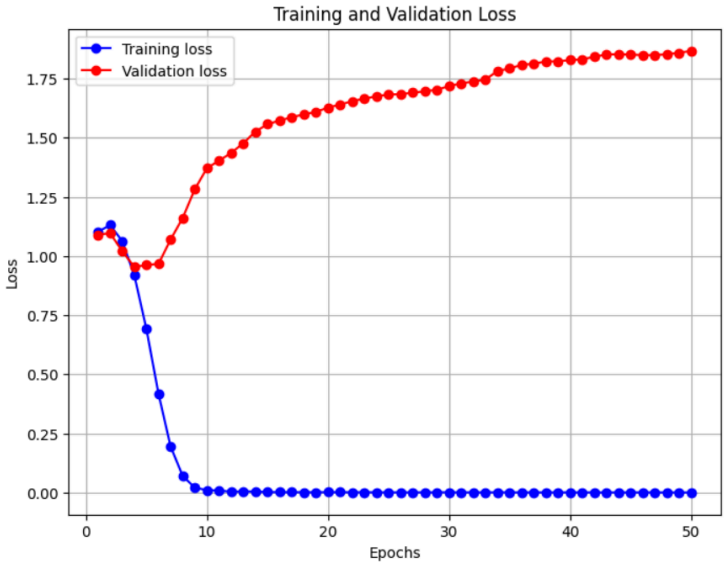
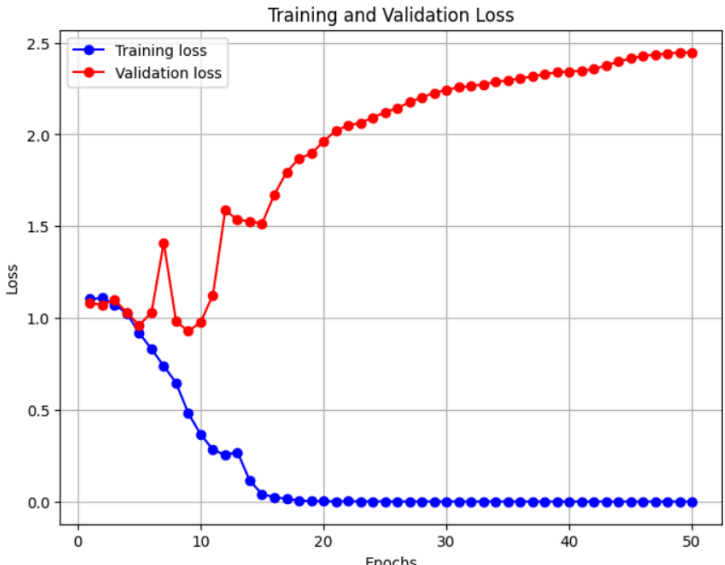
Jumlah Layer	Hasil
<p align="center"><b>Constant Parameter</b></p> <p>Epoch: 50 Batch size: 64</p>	
1	 <p>F1 Score: Keras F1 = 0.515824 Scratch F1 = 0.515824</p>

2	 <p>F1 Score: Keras F1 = 0.555796 Scratch F1 = 0.555796</p>
3	 <p>F1 Score: Keras F1 = 0.532152 Scratch F1 = 0.532152</p>
Analisis	<p>Model dengan satu lapisan RNN mencapai F1-score sebesar 0.5158, menunjukkan kemampuan dasar dalam melakukan klasifikasi sentimen. Ketika jumlah lapisan ditambah menjadi dua, terjadi peningkatan performa yang signifikan dengan F1-score mencapai 0.5558, mengindikasikan bahwa arsitektur yang lebih dalam mampu menangkap pola-pola kontekstual yang lebih</p>

	<p>kompleks dalam data. Namun, ketika lapisan ditambah menjadi tiga, justru terjadi penurunan performa dengan F1-score 0.5321, kemungkinan disebabkan oleh overfitting karena model menjadi terlalu kompleks atau masalah vanishing gradient saat informasi melewati banyak lapisan. Dari hasil ini dapat disimpulkan bahwa dua lapisan RNN merupakan konfigurasi optimal untuk tugas analisis sentimen ini, karena memberikan keseimbangan terbaik antara akurasi dan kompleksitas model.</p>
--	--

## 2. Variasi Jumlah Cell RNN

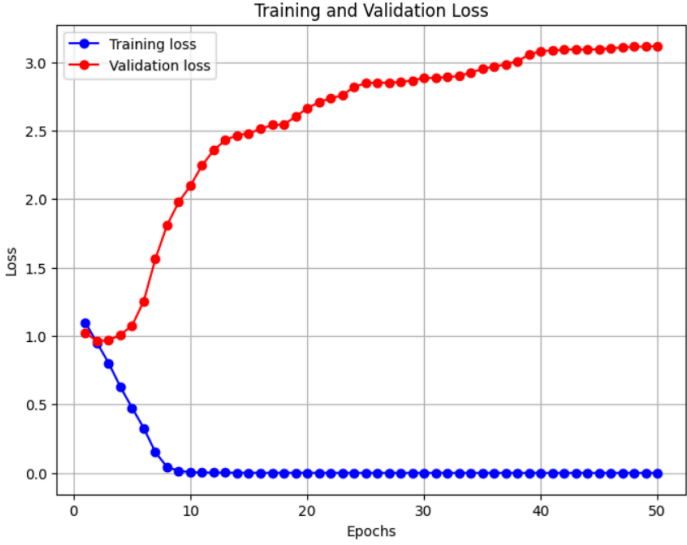
Banyak Cell	Hasil
<p align="center"><b>Constant Parameter</b></p> <p>Epoch: 50 Batch size: 64</p>	
32	<p align="center">Training and Validation Loss</p>  <p>F1 Score: Keras F1 = 0.498722 Scratch F1 = 0.498722</p>

64	 <p>F1 Score: Keras F1 = 0.509422 Scratch F1 = 0.509422</p>
128	 <p>F1 Score: Keras F1 = 0.561615 Scratch F1 = 0.561615</p>
Analisis	<p>Kapasitas model yang lebih besar dengan 128 unit mampu menangkap pola dan hubungan yang lebih kompleks dalam data teks untuk tugas analisis sentimen. Namun, perlu dipertimbangkan bahwa peningkatan unit juga berarti peningkatan kompleksitas komputasi dan risiko overfitting jika data training terbatas. Dapat</p>

	disimpulkan bahwa penggunaan 128 unit adalah konfigurasi optimal untuk kasus ini, dengan catatan perlu dilakukan monitoring terhadap overfitting dan waktu komputasi.
--	---

3. Variasi Arah RNN

Arah RNN	Hasil																																				
Constant Parameter																																					
Epoch: 50 Batch size: 64																																					
Unidirectional	<div>Training and Validation Loss</div> <table><thead><tr><th>Epochs</th><th>Training loss</th><th>Validation loss</th></tr></thead><tbody><tr><td>0</td><td>1.1</td><td>1.1</td></tr><tr><td>5</td><td>0.8</td><td>1.5</td></tr><tr><td>10</td><td>0.5</td><td>2.0</td></tr><tr><td>15</td><td>0.2</td><td>2.8</td></tr><tr><td>20</td><td>0.1</td><td>3.5</td></tr><tr><td>25</td><td>0.05</td><td>4.0</td></tr><tr><td>30</td><td>0.05</td><td>4.1</td></tr><tr><td>35</td><td>0.05</td><td>4.1</td></tr><tr><td>40</td><td>0.05</td><td>4.1</td></tr><tr><td>45</td><td>0.05</td><td>4.1</td></tr><tr><td>50</td><td>0.05</td><td>4.2</td></tr></tbody></table> <div>F1 Score: Keras F1 = 0.369200 Scratch F1 = 0.369200</div>	Epochs	Training loss	Validation loss	0	1.1	1.1	5	0.8	1.5	10	0.5	2.0	15	0.2	2.8	20	0.1	3.5	25	0.05	4.0	30	0.05	4.1	35	0.05	4.1	40	0.05	4.1	45	0.05	4.1	50	0.05	4.2
Epochs	Training loss	Validation loss																																			
0	1.1	1.1																																			
5	0.8	1.5																																			
10	0.5	2.0																																			
15	0.2	2.8																																			
20	0.1	3.5																																			
25	0.05	4.0																																			
30	0.05	4.1																																			
35	0.05	4.1																																			
40	0.05	4.1																																			
45	0.05	4.1																																			
50	0.05	4.2																																			

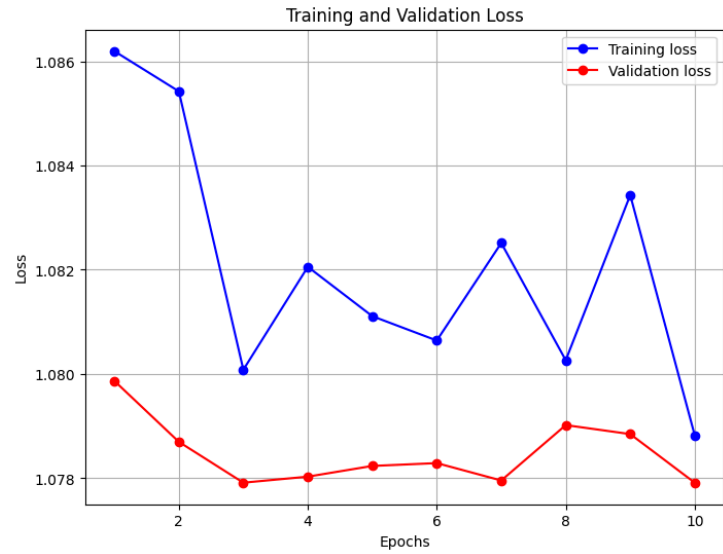
<p>Bidirectional</p>	 <p>Training and Validation Loss</p> <p>F1 Score: Keras F1 = 0.486306 Scratch F1 = 0.486306</p>
<p>Analisis</p>	<p>Pada tugas besar ini, analisis sentimen yang membutuhkan pemahaman konteks utuh (seperti ironi atau negasi), arsitektur bidirectional jelas lebih unggul. Peningkatan komputasi pada bidirectional RNN sebanding dengan peningkatan akurasi. Hasil ini konsisten dengan teori bahwa pemahaman sentimen sering membutuhkan analisis hubungan antar kata dalam seluruh kalimat</p>

### c. LSTM

#### 1. Variasi Jumlah Layer LSTM

Jumlah Layer	Hasil
Epoch: 30 Unit: 64 Batch size: 32	<p><b>Constant Parameter</b></p>

1

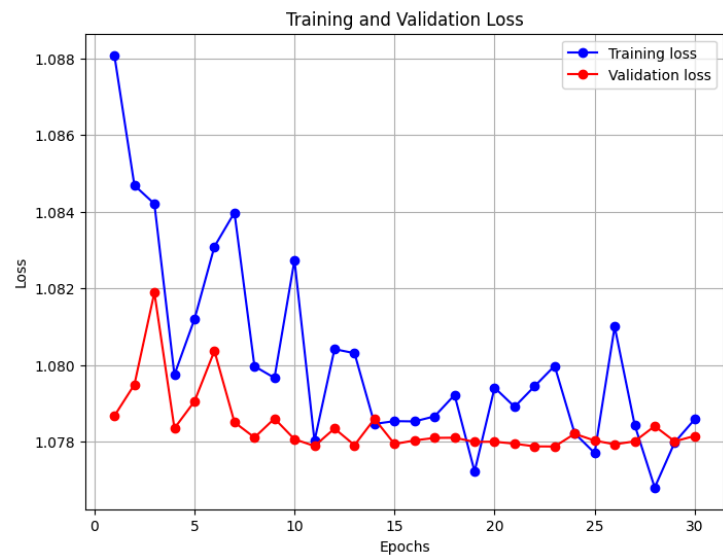


F1 Score:

Keras F1 = 0.184448

Scratch F1 = 0.184448

3



F1 Score:

Keras F1 = 0.189268

Scratch F1 = 0.189268

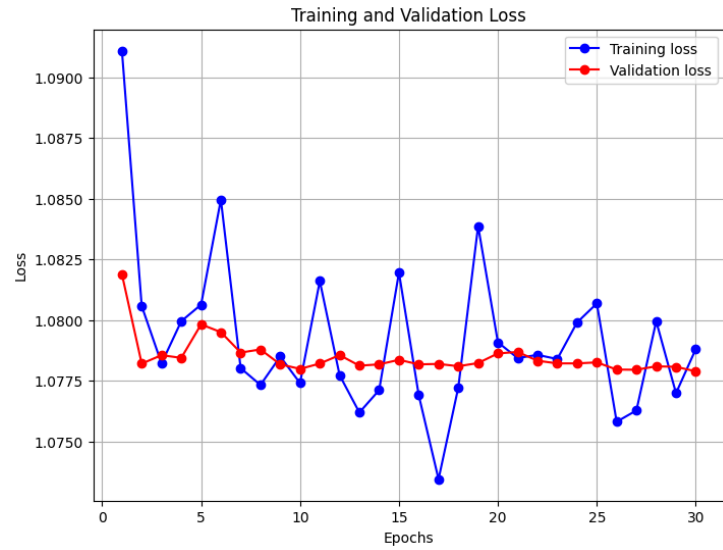
5	<p>Training and Validation Loss</p> <p>F1 Score: Keras F1 = 0.310687 Scratch F1 = 0.310687</p>
Analisis	<p>Berdasarkan percobaan yang dilakukan, semakin tinggi jumlah <i>layer</i> yang digunakan, nilai F1 <i>score</i> yang didapat semakin bagus. Hal ini karena jumlah <i>layer</i> yang banyak memungkinkan pola direpresentasikan secara lebih detail.</p>

## 2. Variasi Jumlah Cells LSTM

Jumlah Cells	Hasil
Epoch: 30 Layer: 1 Batch size: 32	<b>Constant Parameter</b>



8

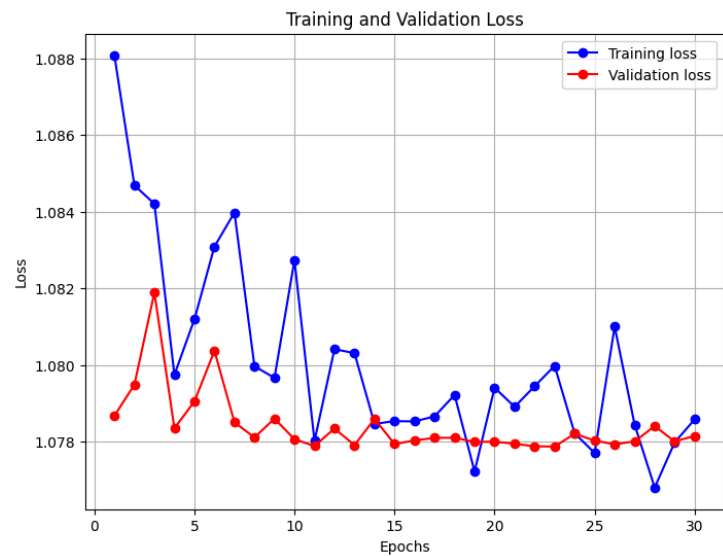


F1 Score:

Keras F1 = 0.182698

Scratch F1 = 0.182698

16



F1 Score:

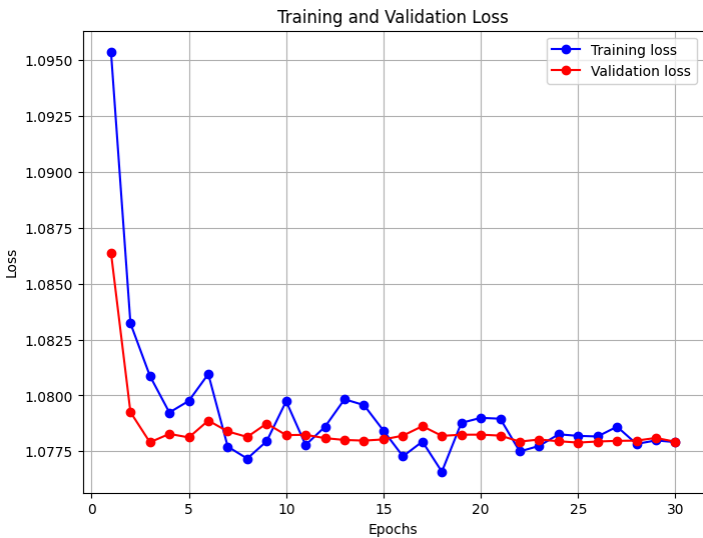
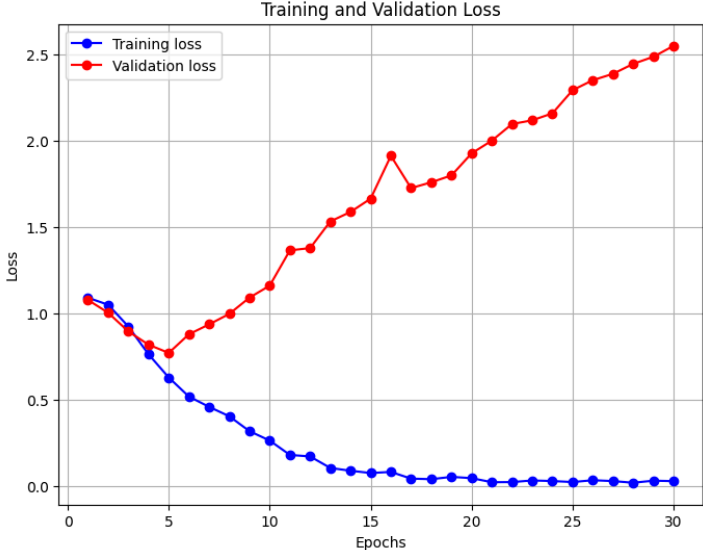
Keras F1 = 0.184448

Scratch F1 = 0.184448

32	<p>Training and Validation Loss</p> <p>F1 Score: Keras F1 = 0.184448 Scratch F1 = 0.184448</p>
Analisis	<p>Berdasarkan percobaan yang dilakukan, semakin tinggi jumlah <i>unit</i> yang digunakan, nilai F1 <i>score</i> yang didapat semakin bagus karena model bisa menyimpan pola yang lebih kompleks. Namun saat jumlah <i>unit</i> mencapai 32, terjadi <i>overfit</i> karena jumlah dataset yang digunakan hanya sedikit, sementara jumlah unit yang digunakan sangat tinggi.</p>

### 3. Variasi Arah LSTM

Arah	Hasil
Epoch: 30 Layer: 3 Batch size: 32 Unit: 10	<b>Constant Parameter</b>

Unidirectional	 <p>Training and Validation Loss</p> <p>F1 Score: Keras F1 = 0.184448 Scratch F1 = 0.184448</p>
Bidirectional	 <p>Training and Validation Loss</p> <p>F1 Score: Keras F1 = 0.636144 Scratch F1 = 0.636144</p>
Analisis	<p>Berdasarkan percobaan yang dilakukan, LSTM <i>bidirectional</i> memberikan nilai F1 score yang lebih tinggi dibandingkan <i>unidirectional</i> karena proses <i>unidirectional</i> hanya memproses urutan dari satu arah saja, sehingga memproses pola hanya dari arah depan ke belakang saja dibandingkan <i>bidirectional</i> yang juga membandingkan pola dari belakang ke depan.</p>



# BAB III

## KESIMPULAN DAN SARAN

### A. Kesimpulan

Pada Tugas Besar II ini, kami berhasil mengimplementasikan *forward propagation* untuk model Convolutional Neural Network (CNN), Simple Recurrent Neural Network (RNN), dan Long Short-Term Memory (LSTM) dari *scratch* menggunakan bahasa Python. Model CNN kami dilatih menggunakan dataset CIFAR-10 untuk klasifikasi gambar, sedangkan model RNN dan LSTM digunakan untuk klasifikasi teks menggunakan dataset NusaX-Sentiment khususnya Bahasa Indonesia. Pelatihan termasuk *backward propagation* dilakukan menggunakan library Keras, kemudian bobot model digunakan kembali dalam implementasi *forward propagation* yang dibuat dari *scratch*.

Dalam prosesnya, kami juga melakukan analisis terhadap beberapa variasi hyperparameter, seperti jumlah *layer*, ukuran dan jumlah filter atau *cell*, serta arah propagasi (*bidirectional* atau *unidirectional*). Evaluasi dilakukan menggunakan metrik macro F1-score untuk mengukur kinerja masing-masing konfigurasi model. Dari analisis ini kami berhasil menentukan dampak dari perubahan hyperparameter pada model CNN, RNN, serta LSTM.

Melalui tugas ini, kami memperoleh pemahaman yang lebih mendalam mengenai cara kerja CNN, RNN, dan LSTM, serta tantangan yang muncul ketika harus membangun komponen-komponennya secara manual, tanpa mengandalkan abstraksi dari *library* tinggi, dan korelasi perubahan nilai *hyperparameter* terhadap kinerja dan hasil prediksi dari model.

### B. Saran

Berikut adalah saran untuk kelompok kami pada tugas-tugas selanjutnya:

- Melakukan perencanaan dan pembagian tugas sejak awal agar pengerjaan lebih terstruktur dan merata

- Memperbanyak diskusi selama pengerjaan tugas
- Memperdalam pengetahuan mengenai CNN, RNN, dan LSTM sehingga memiliki pengertian dan konsep yang lebih baik
- Menyediakan waktu lebih banyak untuk validasi hasil perbandingan antara hasil dari Keras dan versi dari *scratch*.

## PEMBAGIAN TUGAS

NIM	NAMA	Tugas
13522037	Farhan Nafis Rayhan	<ul style="list-style-type: none"><li>• RNN dari Scratch + bonus batch</li><li>• Testing RNN</li><li>• Laporan</li></ul>
13522071	Bagas Sambega Rosyada	<ul style="list-style-type: none"><li>• LSTM dari Scratch + bonus batch</li><li>• Testing CNN</li><li>• Laporan</li></ul>
13522091	Radius Fransiskus Trianitus B.	<ul style="list-style-type: none"><li>• CNN dari Scratch + bonus batch</li><li>• Testing CNN</li><li>• Laporan</li></ul>

## REFERENSI

1. TensorFlow. (n.d.). CIFAR-10 dataset. Diakses dari <https://www.tensorflow.org/datasets/catalog/cifar10>
2. Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2023). Dive into Deep Learning: Convolutional Neural Networks. Diakses dari [https://d2l.ai/chapter\\_convolutional-neural-networks/index.html](https://d2l.ai/chapter_convolutional-neural-networks/index.html)
3. Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2023). Dive into Deep Learning: Recurrent Neural Networks. Diakses dari [https://d2l.ai/chapter\\_recurrent-neural-networks/index.html](https://d2l.ai/chapter_recurrent-neural-networks/index.html)
4. Keras Team. (n.d.). Keras layers API. Diakses dari <https://keras.io/api/layers/>
5. Salindia Perkuliahan IF3270 Pembelajaran Mesin - Dosen Perkuliahan Pembelajaran Mesin Institut Teknologi Bandung