

LAPORAN TUGAS BESAR 2 IF3270

PEMBELAJARAN MESIN

Convolutional Neural Network dan Recurrent Neural Network



Dosen Pengampu : Dr. Nur Ulfa Maulidevi, S.T, M.Sc.

Disusun oleh:

Amalia Putri	(13522042)
Venantius Sean Ardi Nugroho	(13522078)
Julian Chandra Sutadi	(13522080)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2025

DAFTAR ISI

DAFTAR ISI.....	2
DAFTAR GAMBAR.....	2
DAFTAR TABEL.....	3
1. Deskripsi Persoalan.....	5
2. Pembahasan.....	5
2.1 Penjelasan implementasi.....	5
2.1.1 Deskripsi kelas beserta deskripsi atribut dan methodnya.....	5
2.1.2 Penjelasan forward propagation.....	15
2.2 Hasil pengujian.....	19
2.2.1 CNN.....	19
2.2.2 Simple RNN.....	23
2.2.3 LSTM.....	28
3. Kesimpulan dan Saran.....	32
3.1 Kesimpulan.....	32
3.2 Saran.....	32
4. Pembagian tugas tiap anggota kelompok.....	33
5. Referensi.....	33

DAFTAR GAMBAR

Gambar 2.1.2.1.1 Metode Forward Conv2D.....	15
Gambar 2.1.2.2.1 Metode Forward SimpleRNN.....	16
Gambar 2.1.2.3.1 Metode Forward LSTM.....	19

DAFTAR TABEL

Tabel 2.1.1.1 Tabel Sequential.....	5
Tabel 2.1.1.2 Tabel Layer.....	6
Tabel 2.1.1.3 Tabel TextVectorizationWrapper.....	6
Tabel 2.1.1.4 Tabel Embedding.....	6
Tabel 2.1.1.5 Tabel Bidirectional.....	7
Tabel 2.1.1.6 Tabel RNN.....	9
Tabel 2.1.1.7 Tabel SimpleRNN.....	9
Tabel 2.1.1.8 Tabel LSTM.....	10
Tabel 2.1.1.9 Tabel Dense.....	11
Tabel 2.1.1.10 Tabel CONV2D CNN.....	12
Tabel 2.1.1.11 Tabel Pooling CNN.....	13
Tabel 2.1.1.12 Tabel Flatten CNN.....	14
Tabel 2.2.2.1.1 Accuracy dan Loss.....	24
Tabel 2.2.2.1.2 Validation dan Test.....	24
Tabel 2.2.2.2.1 Banyak cell RNN per layer.....	26
Tabel 2.2.2.2.2 Validation dan Test.....	27
Tabel 2.2.2.3.1 RNN berdasarkan arah.....	27
Tabel 2.2.2.3.2 Validation dan Test.....	28
Tabel 2.2.3.1.1 Accuracy dan Loss.....	28
Tabel 2.2.3.1.2 Validation dan Test.....	29
Tabel 2.2.3.2.1 Accuracy dan Loss.....	30
Tabel 2.2.3.2.2 Validation dan Test.....	31
Tabel 2.2.3.3.1 Accuracy dan Loss.....	31
Tabel 2.2.3.3.2 Validation dan Test.....	32
Tabel 4.1 Tabel Pembagian Tugas Kelompok.....	33

1. Deskripsi Persoalan

Tugas Besar 2 mata kuliah IF3270 Pembelajaran Mesin ini dirancang untuk memberikan pemahaman praktis dan mendalam kepada mahasiswa mengenai proses forward propagation pada jaringan saraf, serta implementasi dan evaluasi model deep learning untuk klasifikasi citra dan teks. Tugas ini terbagi ke dalam tiga bagian utama: Convolutional Neural Network (CNN), Simple Recurrent Neural Network (Simple RNN), dan Long Short-Term Memory (LSTM).

2. Pembahasan

Bagian ini berisi tentang penjelasan implementasi yang mendeskripsikan atribut serta method pada suatu kelas.

2.1 Penjelasan implementasi

2.1.1 Deskripsi kelas beserta deskripsi atribut dan methodnya

- Sequential

Kelas untuk membangun model dari layer - layer custom yang diimplementasi

Tabel 2.1.1.1 Tabel Sequential

No	Kelas	Atribut/Method	Deskripsi
1.	Sequential	layers	atribut untuk menyimpan semua custom layer yang diimplementasi
2.		add(self, layer)	Digunakan untuk menambah layer custom secara individu. Penambahan layer juga bisa menggunakan konstruktor.
3.		predict(self, initial_x)	Memanggil metode forward pada semua layer pada self.layers secara sekuensial, mengeluarkan prediksi dari suatu dataset (y).
4.		load_weights(self, path_to_h5_file: str)	Menerima sebuah file .h5 model sekuensial Keras dan memuat weight tiap layer pada self.layers secara otomatis dengan memanggil method load_keras_weights yang dimiliki tiap layer

- Layer

Bapak dari semua custom layer yang diimplementasi, memiliki 2 abstract method yaitu forward dan load_keras_weights yang harus diimplementasi pada anak - anaknya.

Tabel 2.1.1.2 Tabel Layer

No	Kelas	Atribut/Method	Deskripsi
1.	Layer	key	Identifier tipe layer, digunakan pada metode load_weights
2.		_get_activation(name)	Menerima nama dari activation function yang diinginkan, mengembalikan pointer to activation function tersebut.

- TextVectorizationWrapper

Wrapper untuk layer text vectorization yang dimiliki Keras, mengubah kata menjadi token. Diperlukan agar bisa dimasukkan ke dalam objek Sequential yang diimplementasi.

Tabel 2.1.1.3 Tabel TextVectorizationWrapper

No	Kelas	Atribut/Method	Deskripsi
1.	TextVectorizationWrapper	vectorizer	Tempat menyimpan vectorizer Keras
2.		forward(self, x)	Memanggil layer vectorizer Keras.

- Embedding

Mengubah token yang diproduksi oleh Text Vectorization layer, menjadi vector berdimensi n.

Tabel 2.1.1.4 Tabel Embedding

No	Kelas	Atribut/Method	Deskripsi
----	-------	----------------	-----------

1.	Embedding	key	Base untuk identifier layer Embedding, bernilai 'embedding'
2.		input_dim, output_dim	Dimensi (size) dari masukan dan keluaran
3.		mask_zero	Menerima sebuah boolean, memberi tahu apakah perlu dikembalikan zero mask atau tidak di metode forward.
4.		weights	Semacam lookup table yang digunakan untuk mendapatkan vektor yang sesuai untuk nilai token tertentu.
5.		forward(self, x)	Mengharapkan inputan berupa list of token (integer, matrix kalau dalam batch), untuk tiap token, akan dicari vektor yang bersesuaian pada weights. Mengembalikan array of vectors serta masknya, tergantung mask_zero == True atau False.
6.		load_keras_weights(self, weights)	Mengisi lookup table dengan weights dari Keras.

- Bidirectional

Digunakan pada layer LSTM dan Simple RNN, mengubah kedua layer tersebut menjadi bidirectional LSTM atau bidirectional RNN.

Tabel 2.1.1.5 Tabel Bidirectional

No	Kelas	Atribut/Method	Deskripsi
----	-------	----------------	-----------

1.	Bidirectional	key	Base untuk identifier layer Bidirectional, bernilai bidirectional'
2.		merge_mode	String untuk menentukan tipe merge, cara penggabungan forward layer dan backward layer.
3.		forward_layer	Layer RNN atau LSTM yang menjadi dasar dari layer bidirectional.
4.		backward_layer	<i>Deepcopy</i> dari forward layer dengan urutan pemrosesan timestep terbalik dari forward layer.
5.		forward(self, x, mask=None)	Melakukan forward propagation menggunakan forward layer dan backward layer, hasil tersebut digabungkan dengan cara yang ditentukan oleh merge_mode.
6.		load_keras_weights(self, weights)	Pada dasarnya mengisi weight ketiga pertama ke dalam forward layer dan tiga terakhir ke backward layer.

- Dropout

Sebenarnya tidak begitu digunakan bila hanya mengimplementasikan forward propagation, layer ini kami bikin hanya untuk penggambaran yang lebih jelas saat pembuatan model. bagian forward pada layer ini hanya mengembalikan x karena dropout hanya digunakan pada training.

- RNN

Bapak dari layer LSTM dan RNN, digunakan untuk membagi beberapa atribut yang dimiliki keduanya.

Tabel 2.1.1.6 Tabel RNN

No	Kelas	Atribut/Method	Deskripsi
1.	RNN	return_sequences	Digunakan untuk memberi tahu method forward, apakah akan mengembalikan seluruh h_t atau hanya h_t terakhir.
2.		go_backwards	Digunakan untuk menandakan apakah layer ini backward layer atau tidak. Membalik pemrosesan timestep.
3.		key	Base untuk identifier layer RNN, bernilai 'rnn'. Digunakan untuk load_weights

- SimpleRNN

Implementasi forward propagation dari recurrent neural network (RNN). Pada dasarnya digunakan untuk “meringkas” arti dari suatu kalimat menjadi suatu value, dapat menerima masukkan batch.

Tabel 2.1.1.7 Tabel SimpleRNN

No	Kelas	Atribut/Method	Deskripsi
1.	SimpleRNN	return_sequences	Digunakan untuk memberi tahu method forward, apakah akan mengembalikan seluruh h_t atau hanya h_t terakhir.
2.		go_backwards	Digunakan untuk menandakan apakah layer ini backward layer atau tidak. Membalik pemrosesan timestep.
3.		key	Base untuk identifier layer simple RNN, bernilai

			'simple_rnn'. Digunakan untuk load_weights
4.		kernel	Weight yang berinteraksi dengan input.
5.		recurrent_kernel	Weight yang berinteraksi dengan hidden state.
6.		bias	Weight tambahan yang diperlukan.
7.		activation	Activation function yang digunakan.
8.		forward(self, x, mask=None)	Melakukan forward pass Simple RNN dengan mengiterasi setiap timestep dan memperbarui hidden state menggunakan input saat ini, hidden state sebelumnya, dan bobot yang dipelajari. Jika mask disediakan, mask mencegah hidden state diperbarui untuk langkah waktu yang padded. Metode ini mengembalikan urutan lengkap status tersembunyi atau hanya yang terakhir, tergantung pada atribut return_sequences.
9.		load_keras_weights	Mengassign kernel, recurrent kernel, dan bias dengan weight yang sesuai.

- LSTM

Modifikasi dari RNN yang menggabungkan *long-term memory* (cell state) dengan *short-term memory* (hidden state) untuk mencegah vanishing atau exploding gradient problem.

Tabel 2.1.1.8 Tabel LSTM

No	Kelas	Atribut/Method	Deskripsi
1.	LSTM	units	Output size dari layer ini.

2.		key	Base untuk identifier layer LSTM, bernilai lstm'. Digunakan untuk load_weights
3.		kernel	Weight yang berinteraksi dengan input.
4.		recurrent_kernel	Weight yang berinteraksi dengan hidden state.
5.		bias	Weight tambahan yang diperlukan.
6.		activation	Activation function yang digunakan.
7.		recurrent_activation	
8.		forward(self, x, mask=None)	
9.		load_keras_weights	Mengassign kernel, recurrent kernel , dan bias dengan weight yang sesuai.

- Dense

Fully connected layer yang biasa digunakan di akhir model untuk melakukan multi-class classification.

Tabel 2.1.1.9 Tabel Dense

No	Kelas	Atribut/Method	Deskripsi
1.	Dense	units	Output size dari layer ini.
2.		key	Base untuk identifier layer Dense, bernilai 'dense'. Digunakan untuk load_weights
3.		kernel	Weight yang berinteraksi dengan input.
5.		bias	Weight tambahan yang diperlukan.
6.		activation	Activation function yang

			digunakan.
8.		<code>forward(self, x, mask=None)</code>	Melakukan perkalian matrix dan menggunakan hasilnya pada activation function untuk mendapatkan klasifikasi.
9.		<code>load_keras_weights</code>	Mengassign kernel dan bias dengan weight yang sesuai.

- `conv2d.py` (CNN)

Tabel 2.1.1.10 Tabel CONV2D CNN

No	Kelas	Atribut/Method	Deskripsi
1.	Conv2D	<code>__init__(self, filters, kernel_size, strides=(1, 1), padding='valid', groups=1, activation=None)</code>	Konstruktor untuk inisialisasi layer konvolusi 2D. Menerima parameter jumlah filter, ukuran kernel, stride, padding, jumlah grup, dan fungsi aktivasi.
2.		<code>load_keras_weights(self, weights)</code>	Method untuk memuat bobot (kernel) dan bias dari model Keras ke layer konvolusi ini. Digunakan untuk forward propagation manual setelah pelatihan.
3.		<code>_pad_input(self, input)</code>	Method internal untuk menambahkan padding ke input sesuai dengan jenis padding (valid atau same). Padding digunakan agar dimensi output sesuai harapan.
4.		<code>forward(self, input)</code>	Melakukan operasi konvolusi 2D pada input menggunakan kernel dan bias yang telah disimpan. Menjalankan perhitungan sliding window dan mengaplikasikan aktivasi.

- pooling.py (CNN)

Tabel 2.1.1.11 Tabel Pooling CNN

No	Kelas	Atribut/Method	Deskripsi
1.	MaxPooling1D	<code>__init__(self, pool_size=2, strides=None, padding="valid")</code>	Inisialisasi pooling 1D dengan ukuran window (pool_size), langkah (stride), dan jenis padding.
2.		<code>forward(self, input)</code>	Melakukan operasi maksimum pada setiap jendela (window) 1D sepanjang waktu (timesteps).
3.	MaxPooling2D	<code>__init__(self, pool_size=2, strides=None, padding="valid")</code>	Inisialisasi pooling 2D dengan ukuran window (pool_size), langkah (stride), dan jenis padding.
4.		<code>forward(self, input)</code>	Melakukan operasi maksimum pada setiap jendela 2D dari input citra.
5.	AveragePooling1D	<code>__init__(self, pool_size=2, strides=None, padding="valid")</code>	Inisialisasi average pooling 1D. Parameter mirip MaxPooling1D.
6.		<code>forward(self, input)</code>	Melakukan operasi rata-rata pada setiap jendela (window) 1D sepanjang waktu (timesteps).
7.	AveragePooling2D	<code>__init__(self, pool_size=2, strides=None, padding="valid")</code>	Inisialisasi average pooling 2D. Parameter mirip MaxPooling2D.
8.		<code>forward(self, input)</code>	Melakukan operasi rata-rata pada setiap jendela 2D dari input citra.
9.	GlobalAveragePooling1D	<code>forward(self, input)</code>	Menghitung rata-rata semua nilai sepanjang dimensi waktu (timesteps) dari input 1D. Digunakan untuk menyederhanakan output sequence.

10.	GlobalMax	forward(self, input)	Mengambil nilai maksimum sepanjang dimensi waktu (timesteps) dari input 1D. Digunakan sebagai alternatif dari GlobalAveragePooling1D.
11.	GlobalAveragePooling2D	forward(self, input)	Menghitung rata-rata dari seluruh area spasial (tinggi dan lebar) dari input 2D (biasanya dari citra).
12.	GlobalMaxPooling2D	forward(self, input)	Mengambil nilai maksimum dari seluruh area spasial (tinggi dan lebar) dari input 2D.

- flatten.py (CNN)

Tabel 2.1.1.12 Tabel Flatten CNN

No	Kelas	Atribut/Method	Deskripsi
1.	Flatten	__init__(self)	Konstruktor untuk inisialisasi kelas Flatten. Menetapkan self.key sebagai identifikasi layer.
2.		forward(self, input)	Mengubah (reshape) input multidimensi menjadi vektor 1D per sampel, mempertahankan dimensi batch.

2.1.2 Penjelasan forward propagation

2.1.2.1 CNN

```
1 def forward(self, input):
2     batch_size = input.shape[0]
3     kernel_h, kernel_w = self.kernel_size
4     stride_h, stride_w = self.strides
5
6     input_padded = self._pad_input(input)
7     padded_h, padded_w = input_padded.shape[1:3]
8
9     out_h = (padded_h - kernel_h) // stride_h + 1
10    out_w = (padded_w - kernel_w) // stride_w + 1
11
12    output = np.zeros((batch_size, out_h, out_w, self.filters))
13
14    for b in range(batch_size):
15        for f in range(self.filters):
16            kernel = self.weights[:, :, :, f]
17            bias = self.bias[f]
18            for i in range(out_h):
19                for j in range(out_w):
20                    h_start = i * stride_h
21                    h_end = h_start + kernel_h
22                    w_start = j * stride_w
23                    w_end = w_start + kernel_w
24
25                    patch = input_padded[b, h_start:h_end, w_start:w_end, :]
26                    output[b, i, j, f] = np.sum(patch * kernel) + bias
27
28    self.key = "layer"
29    return self.activation(output)
```

Gambar 2.1.2.1.1 Metode Forward Conv2D

Metode forward pada kelas `Conv2D` bertugas untuk melakukan operasi konvolusi dua dimensi terhadap input citra. Pertama, input akan diproses sesuai jenis padding (`valid` atau `same`). Lalu, berdasarkan ukuran kernel dan stride, fungsi ini menghitung dimensi output yang dihasilkan. Untuk setiap batch dan setiap filter, dilakukan proses pengambilan patch dari input dan dilakukan perkalian elemen-per-elemen dengan kernel filter, lalu dijumlahkan dan ditambahkan bias. Hasilnya disimpan di posisi output yang sesuai. Setelah semua posisi selesai diproses, fungsi menerapkan fungsi aktivasi dan mengembalikan hasilnya.

2.1.2.2 RNN

```
def forward(self, x, mask=None):
    batch_size, time_steps, input_dim = x.shape
    units = self.kernel.shape[1]
    h_t = np.zeros((batch_size, units))

    if self.return_sequences:
        outputs = np.zeros((batch_size, time_steps, units))

    time_indices = list(range(time_steps))
    if self.go_backwards:
        time_indices = list(reversed(time_indices))

    for step_idx, t in enumerate(time_indices):
        x_t = x[:, t, :]

        h_t_new = self.activation(
            np.dot(x_t, self.kernel) +
            np.dot(h_t, self.recurrent_kernel) +
            self.bias
        )

        if mask is not None:
            mask_t = mask[:, t]
            mask_t = mask_t[:, np.newaxis]
            h_t = mask_t * h_t_new + (1 - mask_t) * h_t
        else:
            h_t = h_t_new

        if self.return_sequences:
            if self.go_backwards:
                outputs[:, time_steps - 1 - step_idx, :] = h_t
            else:
                outputs[:, t, :] = h_t

    if self.return_sequences:
        return outputs
    return h_t
```

Gambar 2.1.2.2.1 Metode Forward SimpleRNN

Awalnya, inisialisasi h_t (hidden state) sebagai matrix berisi 0 dengan ukuran $batch_size * units$, kita bisa anggap ini sebagai kumpulan ringkasan dari tiap kalimat, yang mana tiap baris adalah kalimat yang berbeda. Selanjutnya, bila $return_sequence == True$, kita akan menginisialisasi outputs yang merupakan output tiap timesteps pada tiap kalimat, ini digunakan bila ingin membuat model dengan lebih dari 1 RNN, RNN pertama sampai dengan RNN terakhir - 1 harus mereturn sequence. Selanjutnya kita akan menginisialisasi index time step, hal ini membantu kita *keep track* kata apa yang sedang diproses karena bila kita ingin membuat layer backward, kita hanya perlu me-reverse list index timestep ini.

Setelah semua variabel yang diperlukan sudah diinisialisasi kita bisa melakukan forward propagation. Kita akan mengiterasi berdasarkan index timestep yang telah disiapkan sebelumnya, line $x_t = x[:, t, :]$, pada dasarnya ingin mengatakan bahwa program akan memilih kata pertama dari himpunan kalimat. Selanjutnya h_t yang baru bisa dikalkulasi dengan menggunakan x_t dan h_t yang lama. Setelah h_t yang baru ditemukan bila ini iterasi terakhir, maka akan mengembalikan h_t , bila tidak h_t baru dijadikan h_t lama untuk iterasi berikutnya. Pada tahap ini pula list output yang sesuai akan diisi dengan h_t .

2.1.2.3 LSTM

```
1 def forward(self, x, mask=None):
2     if self.kernel is None or self.recurrent_kernel is None or self.bias is None:
3         raise ValueError("Weights not initialized. Call load_keras_weights() first.")
4
5     original_ndim = x.ndim
6     if original_ndim == 2:
7         timesteps, _ = x.shape
8         batch_size = 1
9         # Reshape x_sequence to (batch_size, timesteps, input_dim)
10        x_batched = x[np.newaxis, :, :]
11        if mask is not None:
12            mask = mask[np.newaxis, :]
13    elif original_ndim == 3:
14        batch_size, timesteps, _ = x.shape
15        x_batched = x
16    else:
17        raise ValueError("Input sequence must be 2D or 3D.")
18
19    h_t = np.zeros((batch_size, self.units))
20    c_t = np.zeros((batch_size, self.units))
21
22    outputs = np.zeros((batch_size, timesteps, self.units))
23
24    W_i = self.kernel[:, :self.units]
25    W_f = self.kernel[:, self.units:self.units * 2]
26    W_c = self.kernel[:, self.units * 2:self.units * 3]
27    W_o = self.kernel[:, self.units * 3:]
28
29    U_i = self.recurrent_kernel[:, :self.units]
30    U_f = self.recurrent_kernel[:, self.units:self.units * 2]
31    U_c = self.recurrent_kernel[:, self.units * 2:self.units * 3]
32    U_o = self.recurrent_kernel[:, self.units * 3:]
33
34    b_i = self.bias[self.units]
35    b_f = self.bias[self.units:self.units * 2]
36    b_c = self.bias[self.units * 2:self.units * 3]
37    b_o = self.bias[self.units * 3:]
38
39    time_range = reversed(range(timesteps)) if self.go_backwards else range(timesteps)
40
41    for step_idx, t in enumerate(time_range):
42        x_t = x_batched[:, t, :]
43
44        if mask is not None:
45            mask_t = mask[:, t:t+1]
46
47            if np.all(mask_t == 0):
48                if self.return_sequences:
49                    if self.go_backwards:
50                        outputs[:, timesteps - 1 - step_idx, :] = h_t
51                    else:
52                        outputs[:, t, :] = h_t
53                continue
54
55        f_gate = self.recurrent_activation(np.dot(x_t, W_f) + np.dot(h_t, U_f) + b_f)
56        i_gate = self.recurrent_activation(np.dot(x_t, W_i) + np.dot(h_t, U_i) + b_i)
57        c_candidate = self.activation(np.dot(x_t, W_c) + np.dot(h_t, U_c) + b_c)
58        o_gate = self.recurrent_activation(np.dot(x_t, W_o) + np.dot(h_t, U_o) + b_o)
59
60        c_t_new = f_gate * c_t + i_gate * c_candidate
61        h_t_new = o_gate * self.activation(c_t_new)
62
63        if mask is not None:
64            mask_expanded = np.broadcast_to(mask_t, h_t.shape)
65
66            h_t = mask_expanded * h_t_new + (1 - mask_expanded) * h_t
67            c_t = mask_expanded * c_t_new + (1 - mask_expanded) * c_t
68        else:
69            h_t = h_t_new
70            c_t = c_t_new
71
72        if self.return_sequences:
73            if self.go_backwards:
74                outputs[:, timesteps - 1 - step_idx, :] = h_t
75            else:
76                outputs[:, t, :] = h_t
77
78    if self.return_sequences:
79        if original_ndim == 2:
80            return outputs[0]
81        return outputs
82    else:
83        if original_ndim == 2:
84            return h_t[0]
85        return h_t
```

Gambar 2.1.2.3.1 Metode Forward LSTM

Metode forward pada kelas LSTM digunakan untuk melakukan proses forward propagation terhadap input urutan data, baik berdimensi dua (tanpa batch) maupun tiga (dengan batch). Cara kerjanya serupa dengan apa yang diimplementasikan pada SimpleRNN namun memiliki state-state tambahan, walau tidak disimpan sebagai atribut kelas.

State awal untuk hidden state (h_t) dan cell state (c_t) diinisialisasi ke nol. Fungsi ini lalu memisahkan bobot menjadi empat bagian sesuai dengan gate LSTM: input gate, forget gate, candidate (cell input), dan output gate. Untuk setiap timestep, input x_t akan digunakan bersama dengan h_t sebelumnya untuk menghitung masing-masing gate. Iterasi timestep bergantung pada apakah parameter `go_backwards` bernilai benar, yang akan berguna bagi implementasi Bidirectional Layer.

Pada implementasi LSTM, cell state diperbarui menggunakan kombinasi antara cell state lama dan kandidat baru, sedangkan hidden state diperbarui menggunakan output gate dan cell state baru. Jika terdapat mask, maka perhitungan untuk timestep tersebut dapat dilewati atau dikondisikan. Terakhir, jika `return_sequences` bernilai True, fungsi akan mengembalikan seluruh hasil hidden state per timestep, jika tidak, hanya hidden state terakhir yang dikembalikan.

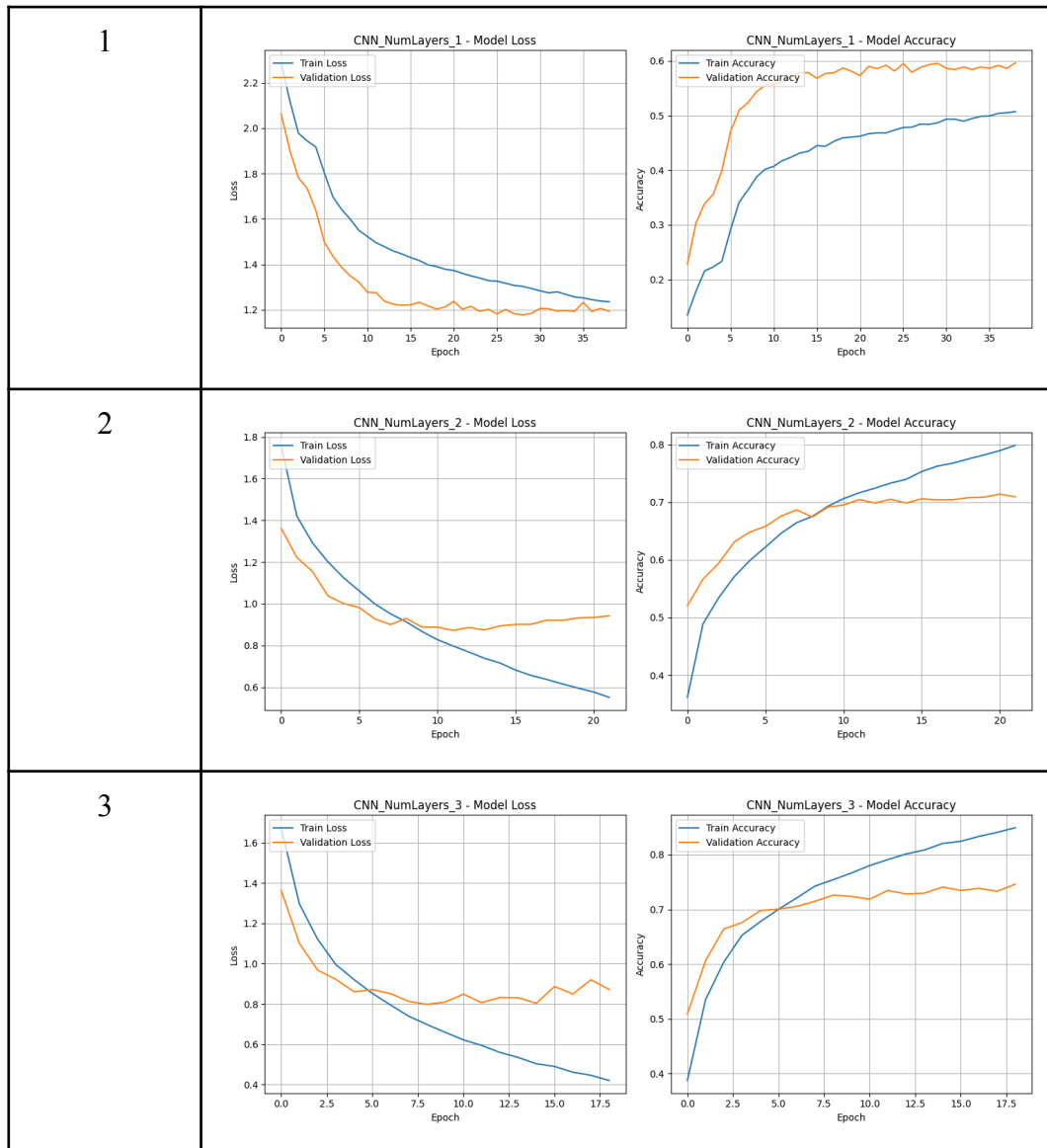
2.2 Hasil pengujian

2.2.1 CNN

2.2.1.1 Pengaruh Jumlah Layer Konvolusi

Tabel 2.2.1.1.1 Accuracy dan Loss

Jumlah layer	Grafik Accuracy dan Loss terhadap Epoch
--------------	---



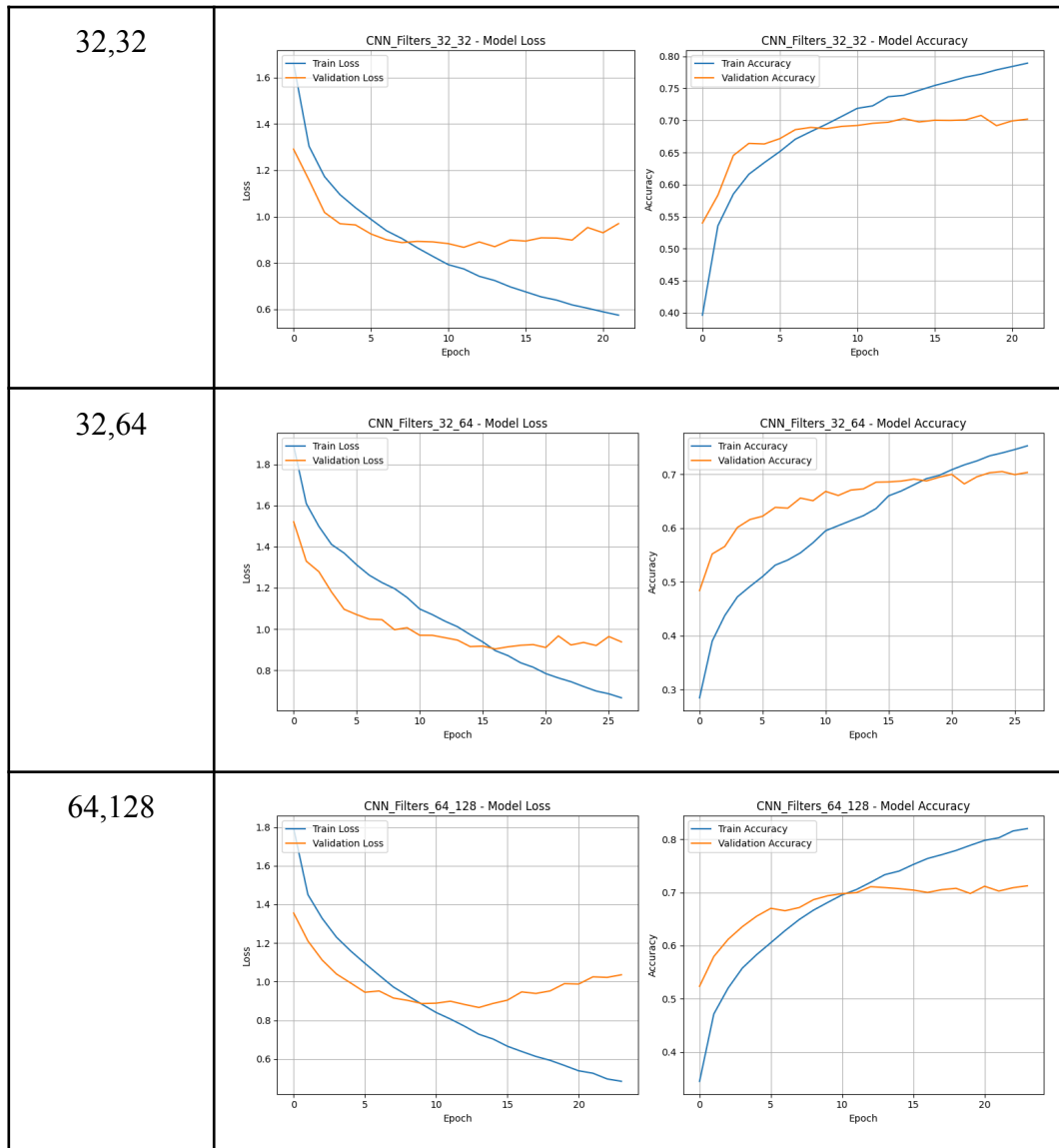
Tabel 2.2.1.1.2 Validation dan Test

Jumlah Layer	Validation Macro F1 Score	Test Macro F1 Score
1	0.5915	0.5917
2	0.7057	0.7059
3	0.7237	0.7221

2.2.1.2 Pengaruh Banyak Filter per Layer Konvolusi

Tabel 2.2.1.2.1 Accuracy dan Loss

Banyak Filter	Grafik Accuracy dan Loss terhadap Epoch
---------------	---



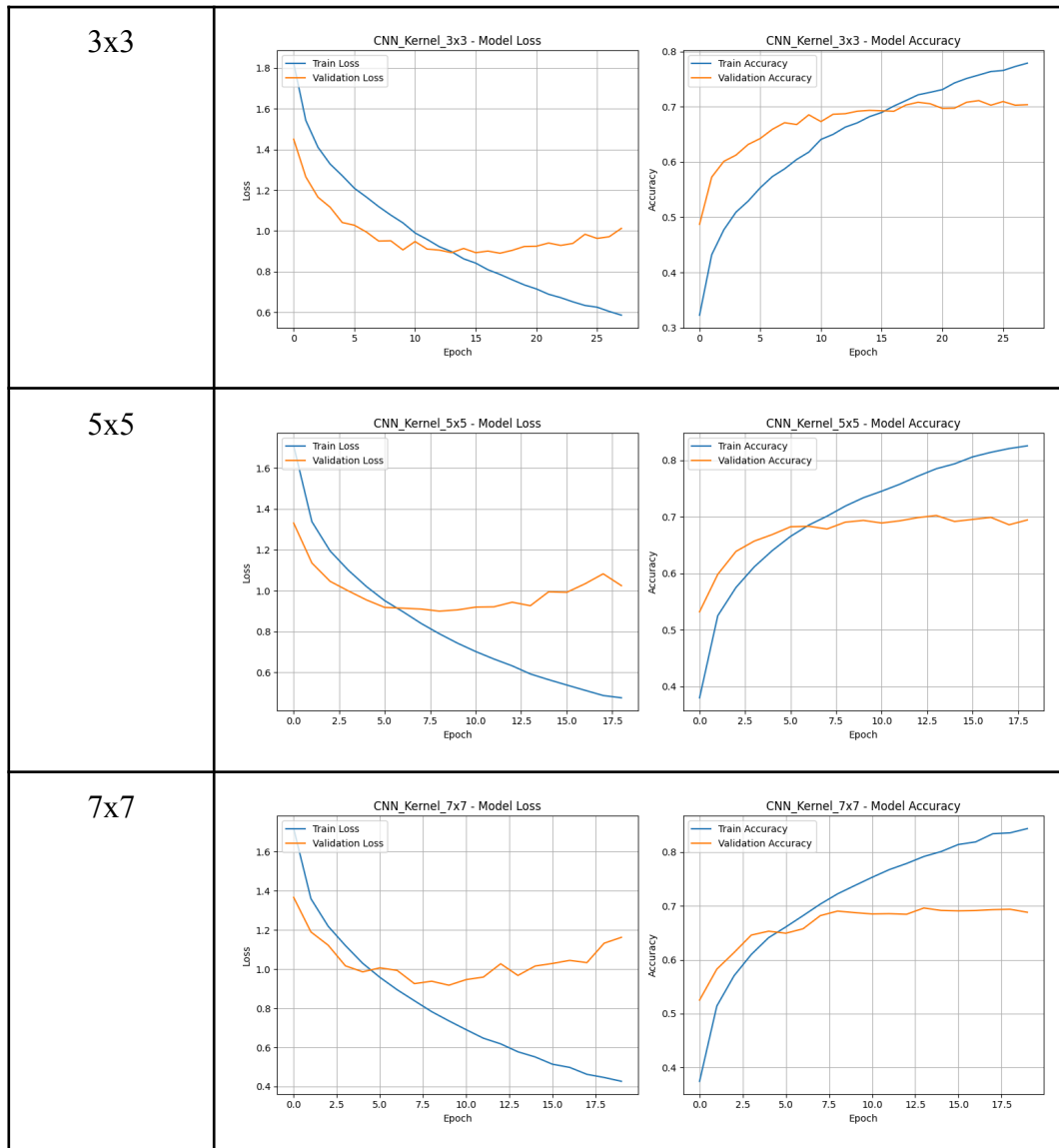
Tabel 2.2.1.2.2 Validation dan Test

Banyak Filter	Validation Macro F1 Score	Test Macro F1 Score
32,32	0.6960	0.7014
32,64	0.6867	0.6911
64,128	0.7081	0.7092

2.2.1.3 Pengaruh Ukuran Filer per Layer Konvolusi

Tabel 2.2.1.3.1 Accuracy dan Loss

Ukuran Filer	Grafik Accuracy dan Loss terhadap Epoch
--------------	---



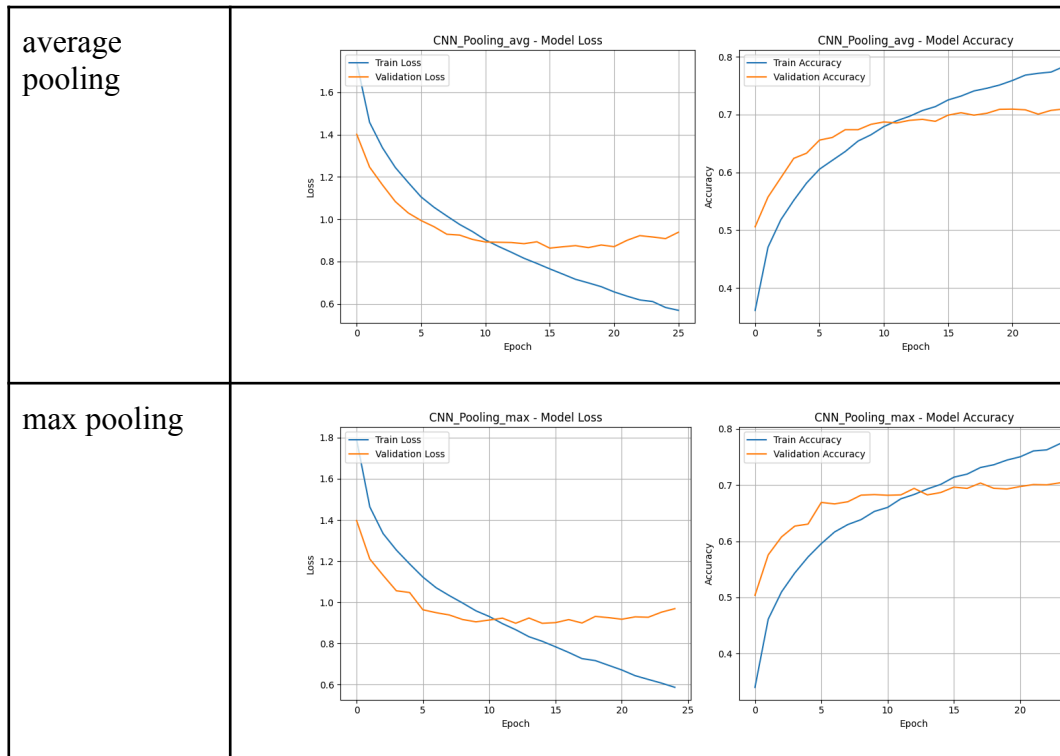
Tabel 2.2.1.3.2 Validation dan Test

Ukuran Filter	Validation Macro F1 Score	Test Macro F1 Score
3x3	0.7029	0.7063
5x5	0.6909	0.6927
7x7	0.6845	0.6826

2.2.1.4 Pengaruh Jenis Pooling Layer

Tabel 2.2.1.4.1 Accuracy dan Loss

Jenis	Grafik Accuracy dan Loss terhadap Epoch
-------	---



Jenis	Validation Macro F1 Score	Test Macro F1 Score
average pooling	0.6984	0.6991
max pooling	0.6855	0.6914

2.2.2 Simple RNN

2.2.2.1 Pengaruh jumlah layer RNN

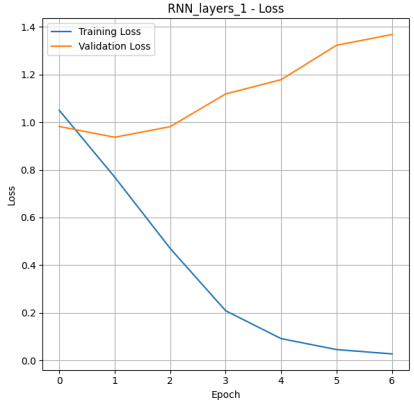
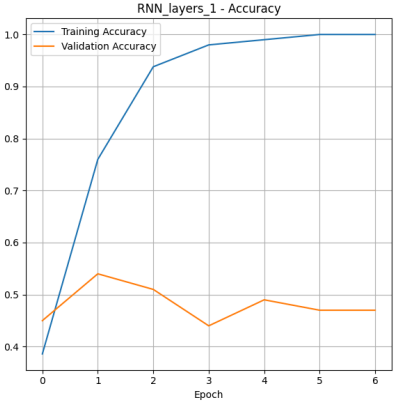
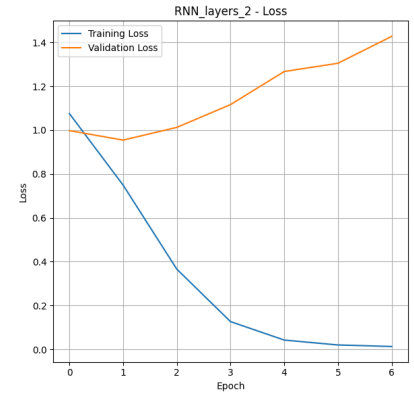
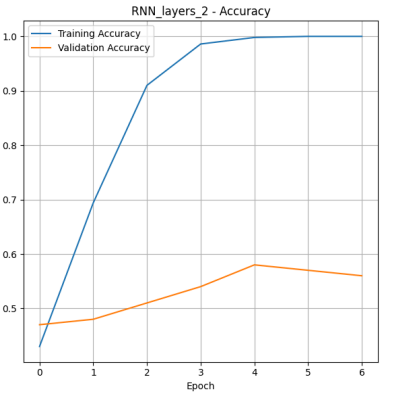
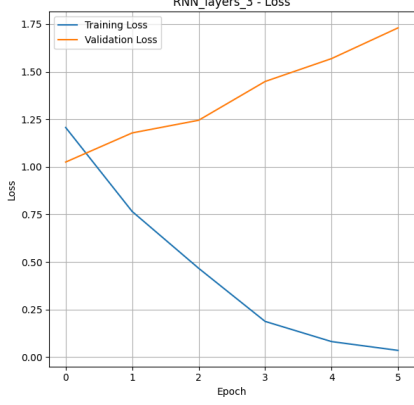
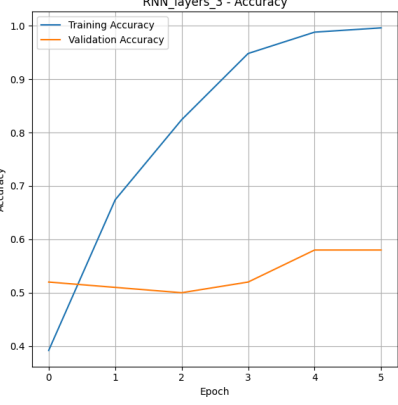
Untuk menguji pengaruh jumlah layer RNN, digunakan model yang terdiri atas:

1. Text Vectorization Layer
2. Embedding Layer
3. Simple RNN Layer
4. Dropout layer (0.3)
5. Dense Layer

Perlu diingat bahwa layer 3 dan 4 berulang sebanyak layer RNN yang ditentukan. Konfigurasi model untuk eksperimen - eksperimen RNN berikutnya akan kurang lebih sama dengan di atas sehingga tidak akan ditulis lagi. Tiap layer

terdiri Simple RNN terdiri dari 64 units serta arah propagasinya unidirectional. Variabel bebasnya yaitu banyak layer terdiri atas 1 , 2 , dan 3 layer.

Tabel 2.2.2.1.1 Accuracy dan Loss

Jumlah layer	Grafik Accuracy dan Loss terhadap Epoch	
1	 	
2	 	
3	 	

Tabel 2.2.2.1.2 Validation dan Test

Jumlah Layer	Validation Macro F1 Score	Test Macro F1 Score
--------------	---------------------------	---------------------

1	0.4314	0.5119
2	0.5542	0.5281
3	0.5422	0.5314

Terlihat bahwa validation macro F1 score memuncak pada saat layer = 2, dari sana kita bisa menyimpulkan bahwa saat layer = 1 terjadi underfitting, sedangkan pada saat layer = 3 terjadi overfitting. Bukti bahwa layer 3 overfitting juga terlihat pada grafik loss vs epoch pada 3 layer. Peningkatan loss terhadap epoch pada validation set terlihat lebih cepat dari pada 1 layer dan 2 layer.

2.2.2.2 Pengaruh banyak cell RNN per layer

Pada eksperimen ini digunakan 2 layer simple RNN dengan variasi cell berjumlah 32, 64, dan 128.

Tabel 2.2.2.2.1 Banyak cell RNN per layer

Jumlah cell	Grafik Accuracy dan Loss terhadap Epoch																																																	
32	<p>RNN_Units_32 - Loss</p> <table border="1"><thead><tr><th>Epoch</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.15</td><td>1.05</td></tr><tr><td>1</td><td>0.85</td><td>1.02</td></tr><tr><td>2</td><td>0.55</td><td>1.15</td></tr><tr><td>3</td><td>0.30</td><td>1.30</td></tr><tr><td>4</td><td>0.15</td><td>1.35</td></tr><tr><td>5</td><td>0.08</td><td>1.55</td></tr><tr><td>6</td><td>0.05</td><td>1.60</td></tr></tbody></table>	Epoch	Training Loss	Validation Loss	0	1.15	1.05	1	0.85	1.02	2	0.55	1.15	3	0.30	1.30	4	0.15	1.35	5	0.08	1.55	6	0.05	1.60	<p>RNN_Units_32 - Accuracy</p> <table border="1"><thead><tr><th>Epoch</th><th>Training Accuracy</th><th>Validation Accuracy</th></tr></thead><tbody><tr><td>0</td><td>0.35</td><td>0.50</td></tr><tr><td>1</td><td>0.65</td><td>0.48</td></tr><tr><td>2</td><td>0.85</td><td>0.40</td></tr><tr><td>3</td><td>0.95</td><td>0.45</td></tr><tr><td>4</td><td>1.00</td><td>0.46</td></tr><tr><td>5</td><td>1.00</td><td>0.44</td></tr><tr><td>6</td><td>1.00</td><td>0.43</td></tr></tbody></table>	Epoch	Training Accuracy	Validation Accuracy	0	0.35	0.50	1	0.65	0.48	2	0.85	0.40	3	0.95	0.45	4	1.00	0.46	5	1.00	0.44	6	1.00	0.43
Epoch	Training Loss	Validation Loss																																																
0	1.15	1.05																																																
1	0.85	1.02																																																
2	0.55	1.15																																																
3	0.30	1.30																																																
4	0.15	1.35																																																
5	0.08	1.55																																																
6	0.05	1.60																																																
Epoch	Training Accuracy	Validation Accuracy																																																
0	0.35	0.50																																																
1	0.65	0.48																																																
2	0.85	0.40																																																
3	0.95	0.45																																																
4	1.00	0.46																																																
5	1.00	0.44																																																
6	1.00	0.43																																																
64	<p>RNN_Units_64 - Loss</p> <table border="1"><thead><tr><th>Epoch</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.15</td><td>1.15</td></tr><tr><td>1</td><td>0.75</td><td>1.12</td></tr><tr><td>2</td><td>0.40</td><td>1.20</td></tr><tr><td>3</td><td>0.15</td><td>1.28</td></tr><tr><td>4</td><td>0.08</td><td>1.38</td></tr><tr><td>5</td><td>0.05</td><td>1.40</td></tr><tr><td>6</td><td>0.05</td><td>1.42</td></tr></tbody></table>	Epoch	Training Loss	Validation Loss	0	1.15	1.15	1	0.75	1.12	2	0.40	1.20	3	0.15	1.28	4	0.08	1.38	5	0.05	1.40	6	0.05	1.42	<p>RNN_Units_64 - Accuracy</p> <table border="1"><thead><tr><th>Epoch</th><th>Training Accuracy</th><th>Validation Accuracy</th></tr></thead><tbody><tr><td>0</td><td>0.42</td><td>0.42</td></tr><tr><td>1</td><td>0.75</td><td>0.45</td></tr><tr><td>2</td><td>0.92</td><td>0.50</td></tr><tr><td>3</td><td>0.98</td><td>0.55</td></tr><tr><td>4</td><td>1.00</td><td>0.52</td></tr><tr><td>5</td><td>1.00</td><td>0.53</td></tr><tr><td>6</td><td>1.00</td><td>0.52</td></tr></tbody></table>	Epoch	Training Accuracy	Validation Accuracy	0	0.42	0.42	1	0.75	0.45	2	0.92	0.50	3	0.98	0.55	4	1.00	0.52	5	1.00	0.53	6	1.00	0.52
Epoch	Training Loss	Validation Loss																																																
0	1.15	1.15																																																
1	0.75	1.12																																																
2	0.40	1.20																																																
3	0.15	1.28																																																
4	0.08	1.38																																																
5	0.05	1.40																																																
6	0.05	1.42																																																
Epoch	Training Accuracy	Validation Accuracy																																																
0	0.42	0.42																																																
1	0.75	0.45																																																
2	0.92	0.50																																																
3	0.98	0.55																																																
4	1.00	0.52																																																
5	1.00	0.53																																																
6	1.00	0.52																																																
128	<p>RNN_Units_128 - Loss</p> <table border="1"><thead><tr><th>Epoch</th><th>Training Loss</th><th>Validation Loss</th></tr></thead><tbody><tr><td>0</td><td>1.15</td><td>1.00</td></tr><tr><td>1</td><td>0.65</td><td>1.10</td></tr><tr><td>2</td><td>0.30</td><td>1.25</td></tr><tr><td>3</td><td>0.15</td><td>1.35</td></tr><tr><td>4</td><td>0.08</td><td>1.55</td></tr><tr><td>5</td><td>0.05</td><td>1.90</td></tr></tbody></table>	Epoch	Training Loss	Validation Loss	0	1.15	1.00	1	0.65	1.10	2	0.30	1.25	3	0.15	1.35	4	0.08	1.55	5	0.05	1.90	<p>RNN_Units_128 - Accuracy</p> <table border="1"><thead><tr><th>Epoch</th><th>Training Accuracy</th><th>Validation Accuracy</th></tr></thead><tbody><tr><td>0</td><td>0.42</td><td>0.52</td></tr><tr><td>1</td><td>0.78</td><td>0.50</td></tr><tr><td>2</td><td>0.92</td><td>0.51</td></tr><tr><td>3</td><td>0.98</td><td>0.55</td></tr><tr><td>4</td><td>1.00</td><td>0.51</td></tr><tr><td>5</td><td>1.00</td><td>0.53</td></tr></tbody></table>	Epoch	Training Accuracy	Validation Accuracy	0	0.42	0.52	1	0.78	0.50	2	0.92	0.51	3	0.98	0.55	4	1.00	0.51	5	1.00	0.53						
Epoch	Training Loss	Validation Loss																																																
0	1.15	1.00																																																
1	0.65	1.10																																																
2	0.30	1.25																																																
3	0.15	1.35																																																
4	0.08	1.55																																																
5	0.05	1.90																																																
Epoch	Training Accuracy	Validation Accuracy																																																
0	0.42	0.52																																																
1	0.78	0.50																																																
2	0.92	0.51																																																
3	0.98	0.55																																																
4	1.00	0.51																																																
5	1.00	0.53																																																

Tabel 2.2.2.2.2 Validation dan Test

Jumlah cell	Validation Macro F1 Score	Test Macro F1 Score
32	0.4312	0.4906
64	0.4981	0.4759
128	0.5097	0.5028

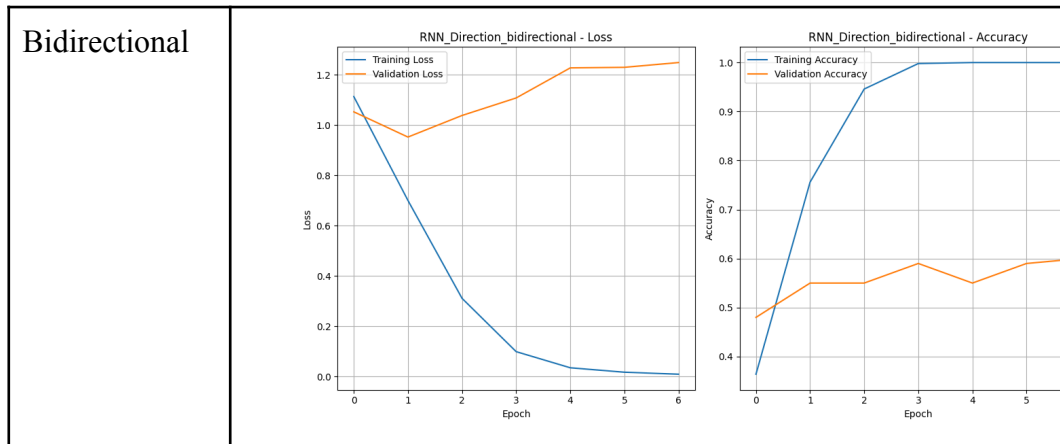
Terlihat bahwa dengan penambahan jumlah cell per layer, validation macro F1 score masih bertambah. Hal tersebut menunjukkan bahwa pada cell unit 128 belum terjadi overfitting. Fakta bahwa pada konfigurasi tersebut validation macro F1 score > test macro F1 score juga mendukung teori tersebut.

2.2.2.3 Pengaruh jenis layer RNN berdasarkan arah

Pada eksperimen ini digunakan 2 layer simple RNN. Pada konfigurasi pertama arah simple RNN layer yang digunakan adalah unidirectional sedangkan pada konfigurasi kedua, digunakan bidirectional RNN.

Tabel 2.2.2.3.1 RNN berdasarkan arah

Arah	Grafik Accuracy dan Loss terhadap Epoch
Unidirectional	<p>The figure contains two line graphs for the Unidirectional RNN configuration. The left graph, titled 'RNN_Direction_unidirectional - Loss', plots Loss (y-axis, 0.0 to 1.6) against Epoch (x-axis, 0 to 5). The Training Loss (blue line) starts at approximately 1.1 and decreases steadily to about 0.05 by epoch 5. The Validation Loss (orange line) starts at approximately 1.05 and increases steadily to about 1.6 by epoch 5. The right graph, titled 'RNN_Direction_unidirectional - Accuracy', plots Accuracy (y-axis, 0.4 to 1.0) against Epoch (x-axis, 0 to 5). The Training Accuracy (blue line) starts at approximately 0.4 and increases steadily to about 1.0 by epoch 5. The Validation Accuracy (orange line) starts at approximately 0.53 and fluctuates slightly, ending at about 0.48 by epoch 5.</p>



Tabel 2.2.2.3.2 Validation dan Test

Arah	Validation Macro F1 Score	Test Macro F1 Score
Unidirectional	0.4696	0.4642
Bidirectional	0.5864	0.5604

Pada dasarnya, penggunaan bidirectional RNN lebih baik daripada unidirectional RNN. Selain macro F1 score yang lebih baik, grafik loss vs epoch juga menunjukkan bahwa kecepatan overfitting lebih lambat di bidirectional RNN.

Selain ketiga eksperimen tersebut, kami menemukan bahwa hasil model forware propaggation pada Simple RNN Keras dan Buatan sendiri, menghasilkan prediction yang sama. Hal tersebut bisa diperiksa di notebook rnn.ipynb pada repository github.

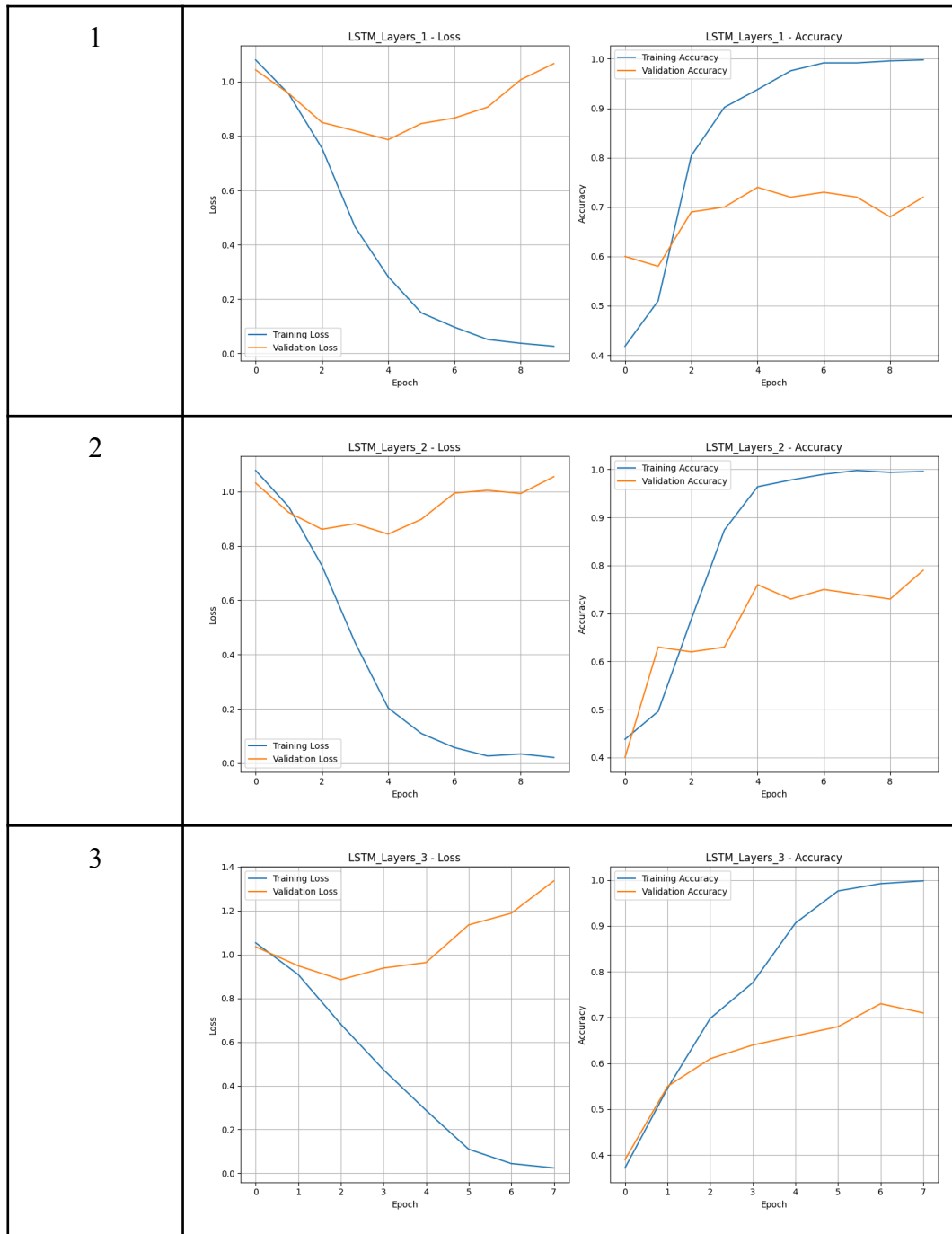
2.2.3 LSTM

2.2.3.1 Pengaruh Jumlah Layer

Konfigurasi tetap yang digunakan adalah 64 unit dan unidirectional.

Tabel 2.2.3.1.1 Accuracy dan Loss

Jumlah layer	Grafik Accuracy dan Loss terhadap Epoch
--------------	---



Tabel 2.2.3.1.2 Validation dan Test

Jumlah Layer	Validation Macro F1 Score	Test Macro F1 Score
1	0.7334	0.7416
2	0.7564	0.7310
3	0.4677	0.4683

Berdasarkan observasi ini, ditemukan bahwa jumlah layer berlebihan malah bisa membuat nilai menjadi sangat buruk pada validasi karena overfitting.

2.2.3.2 Pengaruh Banyak Unit Tiap Layer

Konfigurasi tetap yang digunakan adalah 2 layer dan unidirectional.

Tabel 2.2.3.2.1 Accuracy dan Loss

Banyak Unit	Grafik Accuracy dan Loss terhadap Epoch	
32	<div style="display: flex; justify-content: space-around;"> <div> <p>LSTM_Units_32 - Loss</p> </div> <div> <p>LSTM_Units_32 - Accuracy</p> </div> </div>	
64	<div style="display: flex; justify-content: space-around;"> <div> <p>LSTM_Units_64 - Loss</p> </div> <div> <p>LSTM_Units_64 - Accuracy</p> </div> </div>	
128	<div style="display: flex; justify-content: space-around;"> <div> <p>LSTM_Units_128 - Loss</p> </div> <div> <p>LSTM_Units_128 - Accuracy</p> </div> </div>	

Tabel 2.2.3.2.2 Validation dan Test

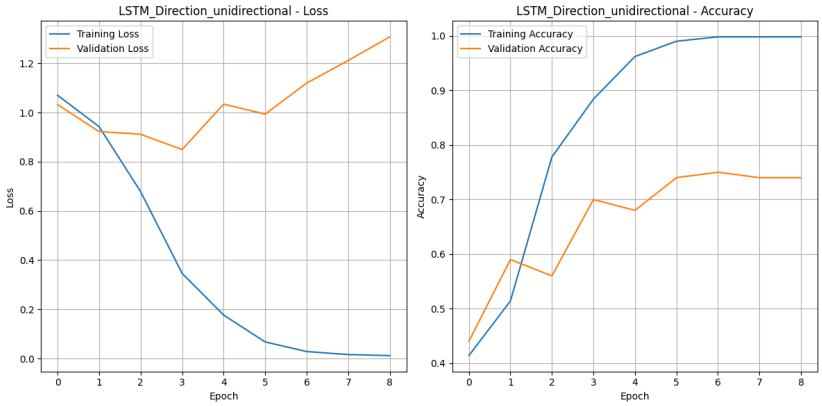
Banyak Unit	Validation Macro F1 Score	Test Macro F1 Score
32	0.5862	0.5031
64	0.7685	0.7464
128	0.7134	0.7350

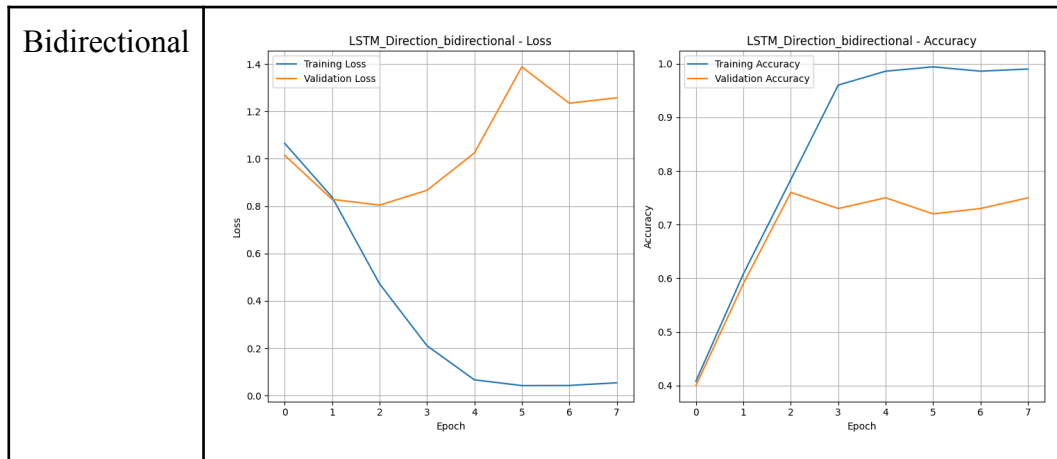
Secara umum, penambahan unit membuat performa model menjadi semakin baik. Kegagalan model pertama tidak disebabkan oleh durasi epoch yang kurang, tapi memang tidak mampu mempelajari fitur yang seharusnya. Namun, terlalu banyak unit, apalagi dalam konteks data yang digunakan kalimatnya tidak terlalu panjang, menyebabkan overfitting.

2.2.3.3 Pengaruh Arah Layer

Konfigurasi tetap yang digunakan adalah 2 layer dan 64 unit.

Tabel 2.2.3.3.1 Accuracy dan Loss

Arah	Grafik Accuracy dan Loss terhadap Epoch
Unidirection	 <p>The figure contains two line graphs for the 'Unidirection' LSTM configuration. The left graph, titled 'LSTM_Direction_unidirectional - Loss', plots Loss (y-axis, 0.0 to 1.2) against Epoch (x-axis, 0 to 8). It shows Training Loss (blue line) decreasing steadily from approximately 1.1 at epoch 0 to near 0.05 at epoch 8. Validation Loss (orange line) starts at approximately 1.05, dips slightly at epoch 3, and then generally increases to approximately 1.3 by epoch 8. The right graph, titled 'LSTM_Direction_unidirectional - Accuracy', plots Accuracy (y-axis, 0.4 to 1.0) against Epoch (x-axis, 0 to 8). It shows Training Accuracy (blue line) increasing steadily from approximately 0.45 at epoch 0 to 1.0 at epoch 6, where it remains. Validation Accuracy (orange line) starts at approximately 0.45, peaks at epoch 3 (approx. 0.7), dips at epoch 4, and then stabilizes around 0.75 from epoch 6 onwards.</p>



Tabel 2.2.3.3.2 Validation dan Test

Arah	Validation Macro F1 Score	Test Macro F1 Score
Unidirectional	0.6967	0.7159
Bidirectional	0.7509	0.7429

Dalam beberapa percobaan yang dilakukan, terdapat perbedaan hasil antara unidirectional dan bidirectional. Terkadang performa unidirectional lebih baik, dan sebaliknya. Model bidirectional lebih cepat mengalami overfitting, dilihat dari grafik loss yang meningkat lebih cepat. Namuni ini bisa disebabkan oleh konfigurasi banyak jumlah unit tiap layer yang menyebabkan overfit, ataupun jumlah layer LSTM.

3. Kesimpulan dan Saran

3.1 Kesimpulan

Dengan selesainya tugas ini, ditemukan bahwa *hyper parameter tuning* sangat berpengaruh pada Macro F1 Score pada CNN, RNN, dan LSTM. Ditemukan juga bahwa forward propagation dengan model Keras dan model yang diciptakan sendiri akan sama bisa menggunakan weight yang sama pula.

3.2 Saran

Pemahaman akan library Keras akan sangat membantu dalam pembuatan tugas ini, terutama pada pembuatan load weights. Metode ini lumayan membingungkan karena banyak edge case penamaan dari Keras yang belum

diketahui. Bila pelajar telah familiar dengan Keras, pembuatan struktur tiap layernya akan lebih rapi dan efisien.

4. Pembagian tugas tiap anggota kelompok

Nama (NIM)	Kontribusi
Amalia Putri (13522042)	CNN (Conv2D), Pooling, Flatten, Dropout
Venantius Sean Ardi Nugroho (13522078)	Text Vectorization Layer, Embedding, Sequential, Simple RNN, dan dokumen,
Julian Chandra Sutadi (13522080)	LSTM, Bidirectional, Dense, notebook template

Tabel 4.1 Tabel Pembagian Tugas Kelompok

5. Referensi

- <https://keras.io/>
- https://www.tensorflow.org/api_docs/python/tf