

Tugas Besar 2

Convolutional Neural Network dan Recurrent Neural Network

IF3270 Pembelajaran Mesin



Elbert Chailes 13522045

Farel Winalda 13522047

Derwin Rustanly 13522115

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2025

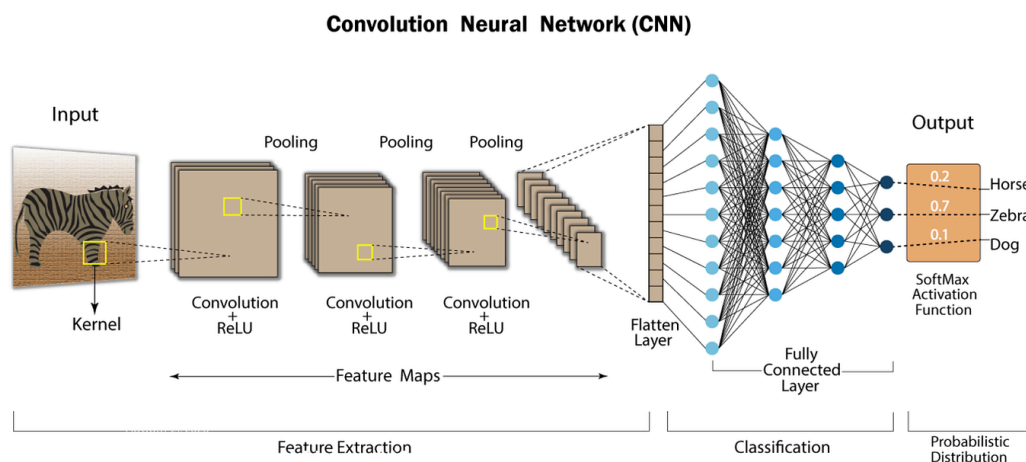
Daftar Isi

Daftar Isi.....	2
BAB 1: Deskripsi Persoalan.....	3
BAB 2: Pembahasan.....	7
2.1. Penjelasan Implementasi.....	7
2.1.1. Deskripsi Kelas.....	7
2.1.2. Penjelasan Forward Propagation.....	16
2.1.2.1. CNN.....	16
2.1.2.2. Simple RNN.....	19
2.1.2.3. LSTM.....	22
2.2. Hasil Pengujian.....	24
2.2.1. CNN.....	24
2.2.2. Simple RNN.....	33
2.2.3. LSTM.....	40
BAB 3: Kesimpulan dan Saran.....	47
3.1. Kesimpulan.....	47
BAB 4: Pembagian Tugas.....	48
Referensi.....	49

BAB 1: Deskripsi Persoalan

Tugas besar 2 pada mata kuliah IF3270 Pembelajaran Mesin bertujuan untuk mendapatkan wawasan tentang cara mengimplementasikan model CNN (*Convolutional Neural Network*), RNN (*Recurrent Neural Network*), dan LSTM (*Long-Short Term Memory*) dimulai dari *scratch*.

a. CNN (*Convolutional Neural Network*)



Gambar 1.1. Visualisasi arsitektur CNN

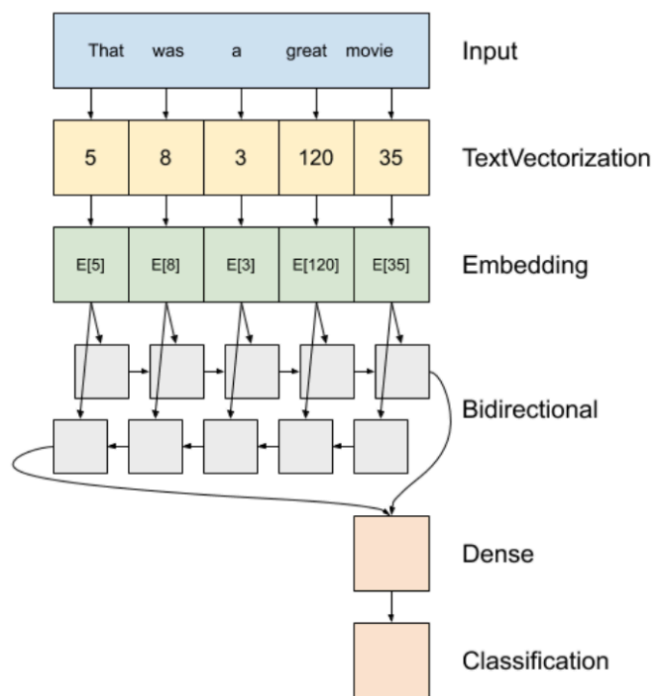
Untuk model CNN dengan tujuan *image classification* dilakukan training terlebih dahulu menggunakan library Keras dan dataset [CIFAR-10](#). Model harus memenuhi kriteria berikut.

- Model minimal memiliki layer [Conv2D layer](#), [Pooling layers](#), [Flatten/Global Pooling](#) layer, dan [Dense layer](#) dengan urutan dan jumlah layer yang dapat disesuaikan sendiri.
- Loss function yang digunakan adalah [Sparse Categorical Crossentropy](#) dan optimizer yang digunakan adalah [Adam](#).
- Dataset CIFAR-10 yang disediakan hanya terdiri dari dua split data saja, yaitu train dan test. Tambahkan split ke-3 (validation set) dengan cara membagi training set yang sudah ada dengan menjadi training set yang lebih kecil dan validation set

dengan rasio 4:1 (jumlah data akhir adalah 40k train data, 10k validation data, dan 10k test data)

- Lakukan variasi pelatihan dan analisis berupa pengaruh hyperparameter pada CNN, berupa 3 variasi jumlah layer konvolusi, 3 variasi kombinasi banyak filter per layer konvolusi, 3 variasi kombinasi banyak filter per layer konvolusi, dan 2 variasi pooling layer.
- Bobot pelatihan disimpan untuk digunakan sebagai bobot ketika melakukan *forward propagation from scratch* dari model CNN.

b. RNN (*Recurrent Neural Network*)



Gambar 1.2. Visualisasi Arsitektur RNN

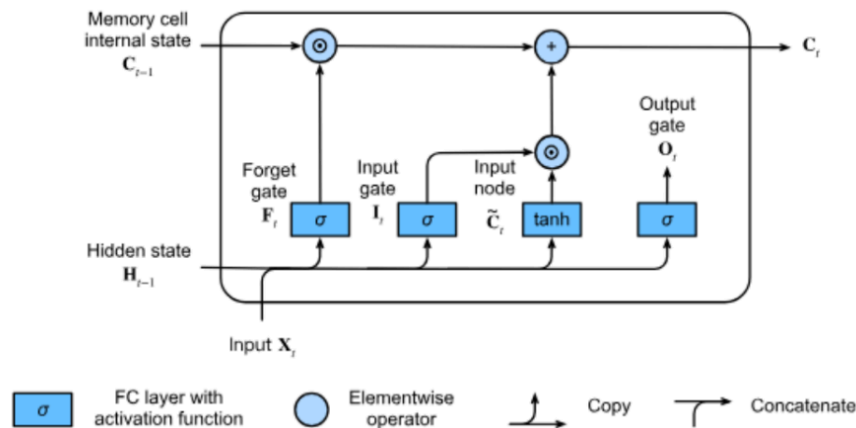
Untuk model RNN dengan tujuan text classification dilakukan training terlebih dahulu menggunakan library Keras dan dataset [NusaX-Sentiment \(Bahasa Indonesia\)](#). Model harus memenuhi kriteria berikut.

- Preprocessing dilakukan dengan [TextVectorization layer](#) untuk melakukan tokenisasi dan representasi numerik terhadap teks. Token-token tersebut

kemudian dipetakan ke ruang vektor berdimensi-n menggunakan Embedding layer dari Keras.

- Model minimal memiliki layer [Embedding layer](#), [Bidirectional RNN layer](#) dan/atau [Unidirectional RNN layer](#), [Dropout layer](#), dan [Dense layer](#) dengan urutan dan jumlah layer yang dapat disesuaikan sendiri.
- Loss function yang digunakan adalah [Sparse Categorical Crossentropy](#) dan optimizer yang digunakan adalah [Adam](#).
- Lakukan variasi pelatihan dan analisis berupa pengaruh hyperparameter pada RNN, berupa 3 variasi jumlah layer RNN, 3 variasi kombinasi jumlah cell RNN per layer, dan 2 variasi arah RNN layer (bidirectional atau unidirectional).
- Bobot pelatihan disimpan untuk digunakan sebagai bobot ketika melakukan *forward propagation from scratch* dari model RNN.

c. LSTM (*Long-Short Term Memory*)



Gambar 1.3. Visualisasi Arsitektur LSTM

Untuk model LSTM dengan tujuan text classification dilakukan training terlebih dahulu menggunakan library Keras dan dataset NusaX-Sentiment (Bahasa Indonesia). Model harus memenuhi kriteria berikut.

- Preprocessing dilakukan dengan [TextVectorization layer](#) untuk melakukan tokenisasi dan representasi numerik terhadap teks. Token-token tersebut

kemudian dipetakan ke ruang vektor berdimensi-n menggunakan Embedding layer dari Keras.

- Model minimal memiliki layer [Embedding layer](#), [Bidirectional RNN layer](#) dan/atau [Bidirectional LSTM layer](#), [Dropout layer](#), dan [Dense layer](#) dengan urutan dan jumlah layer yang dapat disesuaikan sendiri.
- Loss function yang digunakan adalah [Sparse Categorical Crossentropy](#) dan optimizer yang digunakan adalah [Adam](#).
- Lakukan variasi pelatihan dan analisis berupa pengaruh hyperparameter pada LSTM, berupa 3 variasi jumlah layer LSTM, 3 variasi kombinasi jumlah cell LSTM per layer, dan 2 variasi arah LSTM layer (bidirectional atau unidirectional).
- Bobot pelatihan disimpan untuk digunakan sebagai bobot ketika melakukan *forward propagation from scratch* dari model LSTM.

BAB 2: Pembahasan

2.1. Penjelasan Implementasi

2.1.1. Deskripsi Kelas

Pada *source code* yang dibuat oleh pengembang, terdiri dari empat file yang mempunyai kelas-kelasnya masing-masing. File tersebut terdiri dari `Layer.py`, `forward_cnn.py`, `forward_lstm.py`, dan `forward_rnn.py`.

Pembahasan kelas dan atributnya untuk masing-masing file akan dijelaskan secara terpisah untuk memudahkan pembacaan dan pemahaman.

a. `Layer.py`

File ini berisi kelas `Layer` yang sebelumnya telah diimplementasikan pada tugas besar sebelumnya. Dalam file ini, terdapat metode-metode global berupa metode-metode aktivasi yang mungkin untuk digunakan, beserta dengan metode global inisialisasi bobot.

Kelas `Layer` memiliki atribut yang dijelaskan pada **Tabel 2.1.1.** dan method yang dijelaskan pada **Tabel 2.1.2.**

Tabel 2.1.1. Atribut Kelas *Layer*

Atribut	Deskripsi
<code>input_size</code>	Jumlah fitur input yang masuk ke layer.
<code>output_size</code>	Jumlah neuron (output) pada layer.
<code>activation_name</code>	Nama fungsi aktivasi yang digunakan (dikonversi ke huruf kecil).
<code>activation</code>	Fungsi aktivasi yang diterapkan pada output (misal: ReLU, sigmoid, tanh, dll.).
<code>activation_deriv</code>	Fungsi turunan dari fungsi aktivasi, digunakan untuk perhitungan backpropagation.

W	Matriks bobot (weights) yang menghubungkan input ke neuron pada layer.
b	Vektor bias yang ditambahkan ke hasil perkalian bobot dengan input.
use_rmsnorm	Boolean untuk menentukan apakah RMS normalization digunakan pada layer.
epsilon	Nilai kecil untuk menghindari pembagian dengan nol saat perhitungan RMS normalization.
g	Parameter skala yang digunakan dalam RMS normalization (jika diaktifkan).
dg	Gradient dari parameter g, dihitung selama backpropagation (jika RMS normalization digunakan).
dW	Gradient untuk bobot W, dihitung saat proses backward.
db	Gradient untuk bias b, dihitung saat proses backward.
X	Input yang disimpan dari forward pass untuk digunakan saat backpropagation.
z_raw	Hasil perkalian linear ($X * W + b$) sebelum normalisasi (jika RMS normalization diaktifkan).
rms	Nilai Root Mean Square dari z_raw, digunakan dalam normalisasi (jika diaktifkan).
z	Nilai pre-aktivasi yang akan diberikan ke fungsi aktivasi (hasil normalisasi atau z_raw).
a	Output akhir layer setelah diterapkan fungsi aktivasi.

Tabel 2.1.2. Method Kelas *Layer*

Method	Deskripsi
init	Konstruktor yang menginisialisasi layer dengan menentukan ukuran input/output, memilih fungsi aktivasi, inisialisasi bobot dan bias, serta mengatur penggunaan RMS normalization jika diperlukan.
forward	Melakukan forward propagation: menghitung nilai linear (z_{raw}), melakukan normalisasi (jika diaktifkan), menerapkan fungsi aktivasi, dan menghasilkan output (a).
backward	Melakukan backward propagation: menghitung gradien (dW , db , dan dg jika RMS normalization digunakan), mengupdate parameter (W , b , dan g), serta mengembalikan error propagasi ke layer sebelumnya.

b. `forward_cnn.py`

File ini berisi beberapa kelas dan juga metode global yang keseluruhannya digunakan dan diperlukan untuk menjalankan CNN *forward propagation from scratch* yang telah dibuat. Kelas-kelas tersebut terdiri dari kelas Conv2DLayer, PoolingLayer, FlattenLayer dan CNN.

Kelas Conv2DLayer dibuat dengan tujuan sebagai implementasi manual dari layer 2D Convolution (Conv2D) dengan padding 'same' dan aktivasi ReLU. Kelas ini memiliki atribut berupa W sebagai bobot konvolusi dan b sebagai bias per filter. Selain itu, terdapat juga metode forward yang melakukan padding sesuai dengan filter *size*, melakukan *sliding window convolution* untuk setiap posisi dan channel output, dan terakhir melakukan aktivasi menggunakan ReLU.

Kelas PoolingLayer dibuat dengan tujuan sebagai implementasi manual dari layer pooling (max atau average), dengan kernel 2×2 dan stride 2. Kelas ini mempunyai atribut *mode* yang dapat disesuaikan tergantung apakah *max pooling*

atau *average pooling*. Metode pada kelas ini adalah forward dengan melakukan pada setiap blok 2x2, dilakukan operasi max atau mean sesuai dengan atribut *mode*.

Kelas FlattenLayer dibuat dengan tujuan untuk mengubah output 4D dari CNN menjadi 2D sebelum masuk ke layer Dense. Terdapat metode forward pada kelas ini untuk mengubah shape menjadi 1D.

Kelas CNN dibuat dengan tujuan untuk mewakili keseluruhan model CNN yang dibangun dari konfigurasi layer ketika kelas pertama kali diinisialisasi. Penjelasan atribut dan metode dapat dilihat pada tabel berikut.

Tabel 2.1.3. Atribut Kelas *CNN*

Atribut	Deskripsi
layers	list of layers (Conv2DLayer, PoolingLayer, FlattenLayer, Layer)

Tabel 2.1.4. Metode Kelas *CNN*

Method	Deskripsi
<code>__init__(weights, config)</code>	<ul style="list-style-type: none"> - Bangun model dari config layer, sambil memuat bobot dari dictionary weights - config adalah list tuple berisi spesifikasi layer, misalnya: <pre>[('conv', 'layer_name'), ('pool', 'max'), ('flatten', None), ('dense', 'dense_name', 'activation')]</pre>
forward	Forward input melalui semua layer dalam urutan

predict	Menjalankan forward, lalu mendapatkan prediksi kelas
---------	--

Terakhir, terdapat metode `global load_weights(h5_path)` yang bertujuan untuk memuat bobot model Keras dari file .h5 secara manual.

c. `forward_rnn.py`

File ini berisi beberapa kelas dan juga metode global yang keseluruhannya digunakan dan diperlukan untuk menjalankan RNN *forward propagation from scratch* yang telah dibuat. Kelas-kelas tersebut terdiri dari kelas `Embedding`, `RNN`, dan `RNNBidirectional`.

Kelas `Embedding` dibuat dengan tujuan untuk merepresentasikan layer `Embedding` seperti pada Keras, yaitu mengubah token menjadi vektor representasi berdimensi tetap.

- Atribut yang dimiliki oleh kelas `Embedding` hanya `W` dengan shape `(vocab_size, emb_dim)`.
- Metode yang dimiliki juga hanya `forward(X)` yang bertujuan untuk mengambil token (`X` dengan shape `(batch, seq_len)`) menjadi representasi vektor `W[X]`, dengan shape `(batch, seq_len, emb_dim)`.

Kelas `RNN` dibuat dengan tujuan untuk mewakili layer `RNN Unidirectional` untuk implementasi *forward propagation by scratch*. Penjelasan setiap atribut dapat dilihat pada **Tabel 2.1.5.** dan metode pada **Tabel 2.1.6.**

Tabel 2.1.5. Atribut Kelas *RNN*

Atribut	Deskripsi
W	weight input → hidden (input_dim, units)
U	weight hidden → hidden (units, units)

b_xh	bias hidden layer (units,)
units	jumlah unit neuron
return_sequences	apakah mengembalikan semua output per time-step (True) atau hanya akhir (False)

Tabel 2.1.6. Metode Kelas *RNN*

Metode	Deskripsi
forward(X)	<ul style="list-style-type: none"> - Input: X dengan shape (batch, seq_len, input_dim) - Loop per time-step, hitung hidden state menggunakan fungsi aktivasi tanh (<i>default</i>-nya Keras) - Output: <ul style="list-style-type: none"> • (batch, seq_len, units) jika return_sequences=True • (batch, units) jika return_sequences=False

Kemudian, terdapat kelas *RNNBidirectional* dengan tujuan untuk Menggabungkan dua instance RNN (maju dan mundur) untuk mensimulasikan bidirectional RNN seperti pada Keras. Penjelasan setiap atribut dapat dilihat pada **Tabel 2.1.7.** dan metode pada **Tabel 2.1.8.**

Tabel 2.1.7. Atribut Kelas *RNNBidirectional*

Atribut	Deskripsi
fw	RNN untuk arah maju
bw	RNN untuk arah mundur

return_sequences	apakah mengembalikan semua output per time-step (True) atau hanya akhir (False)
------------------	---

Tabel 2.1.8. Metode Kelas *RNNBidirectional*

Metode	Deskripsi
forward(X)	<ul style="list-style-type: none"> - Memanggil fw.forward(X) dan bw.forward(X[:, ::-1, :]) (<i>reverse sequence</i>) - Jika return_sequences, hasilnya adalah konkatenasi dua output per time-step (batch, seq_len, 2*units) - Jika tidak, konkatenasi hasil terakhir (batch, 2*units)

Selain itu, terdapat 2 metode global yang didefinisikan pada file ini, yaitu.

- build_pipeline(keras_model: Model) dengan tujuan untuk mengonversi model Keras (yang sudah dilatih) menjadi pipeline dari layer-layer custom (Embedding, RNN, RNNBidirectional, Layer) untuk diproses secara manual.
- predict(pipeline: List[Layer], X_tok: np.ndarray) -> np.ndarray dengan tujuan melakukan forward propagation manual dari input yang sudah di-tokenisasi (X_tok) melalui pipeline hasil build_pipeline.

d. forward_lstm.py

File ini berisi beberapa kelas dan juga metode global yang keseluruhannya digunakan dan diperlukan untuk menjalankan LSTM *forward propagation from scratch* yang telah dibuat. Kelas-kelas tersebut terdiri dari kelas Embedding, LSTM, dan Bidirectional..

Kelas Embedding dibuat dengan tujuan untuk merepresentasikan layer Embedding seperti pada Keras, yaitu mengubah token menjadi vektor representasi berdimensi tetap.

- Atribut yang dimiliki oleh kelas Embedding hanya W dengan shape (vocab_size, emb_dim).
- Metode yang dimiliki juga hanya forward(X) yang bertujuan untuk mengambil token (X dengan shape (batch, seq_len)) menjadi representasi vektor W[X], dengan shape (batch, seq_len, emb_dim).

Kelas LSTM dibuat dengan tujuan untuk mewakili layer LSTM Unidirectional untuk implementasi *forward propagation by scratch*. Penjelasan setiap atribut dapat dilihat pada **Tabel 2.1.9.** dan metode pada **Tabel 2.1.10.**

Tabel 2.1.9. Atribut Kelas *LSTM*

Atribut	Deskripsi
W	weight input → hidden (input_dim, 4 * units)
U	weight hidden → hidden (units, 4 * units)
b_xh	bias hidden layer (4 * units,)
units	jumlah unit LSTM
return_sequences	apakah mengembalikan semua output per time-step (True) atau hanya akhir (False)

Tabel 2.1.10. Metode Kelas *LSTM*

Metode	Deskripsi
forward(X)	- Input: X dengan shape (batch, seq_len, input_dim)

	<ul style="list-style-type: none"> - Hitung output per time-step menggunakan persamaan LSTM: <ul style="list-style-type: none"> • gate i (input), f (forget), o (output), dan \tilde{c} (candidate cell) • update cell state c dan hidden state h - Output: <ul style="list-style-type: none"> • (batch, seq_len, units) jika return_sequences=True • (batch, units) jika return_sequences=False
--	---

Kemudian, terdapat kelas *Bidirectional* dengan tujuan untuk menggabungkan dua instance LSTM (maju dan mundur) untuk mensimulasikan *bidirectional LSTM* seperti pada Keras. Penjelasan setiap atribut dapat dilihat pada **Tabel 2.1.11.** dan metode pada **Tabel 2.1.12.**

Tabel 2.1.11. Atribut Kelas *Bidirectional*

Atribut	Deskripsi
fw	RNN untuk arah maju
bw	RNN untuk arah mundur
return_sequences	apakah mengembalikan semua output per time-step (True) atau hanya akhir (False)

Tabel 2.1.12. Metode Kelas *Bidirectional*

Metode	Deskripsi
forward(X)	<ul style="list-style-type: none"> - Jalankan forward LSTM di urutan asli - Jalankan backward LSTM di urutan dibalik (X[:, ::-1, :]) - Gabungkan hasil di dimensi fitur (axis 2)

	<ul style="list-style-type: none"> • Jika <code>return_sequences=True</code>: (batch, seq_len, units*2) • Jika <code>False</code>: (batch, units*2)
--	---

Selain itu, terdapat 2 metode global yang didefinisikan pada file ini, yaitu.

- `build_pipeline(keras_model: Model)` dengan tujuan untuk mengonversi model Keras (yang sudah dilatih) menjadi pipeline dari layer-layer custom (Embedding, LSTM, Bidirectional, Layer) untuk diproses secara manual.
- `predict(pipeline: List[Layer], X_tok: np.ndarray) -> np.ndarray` dengan tujuan melakukan forward propagation manual dari input yang sudah di-tokenisasi (`X_tok`) melalui pipeline hasil `build_pipeline`.

2.1.2. Penjelasan Forward Propagation

2.1.2.1. CNN

Untuk implementasi CNN, pembaca dapat lebih memperhatikan file `forward_cnn.py`, yang mana seperti yang telah dijelaskan sebelumnya bahwa file tersebut menyimpan kode implementasi *forward propagation by scratch* secara keseluruhan dari CNN. Objek CNN akan menyimpan seluruh layer dengan bobot dan konfigurasi layer yang ditambahkan pada CNN tersebut.

Oleh karena itu, jika dilihat pada metode konstruktor CNN, pengembang melakukan penanganan layer-layer yang mungkin untuk ditambahkan dalam proses konvolusi, yaitu layer konvolusi, layer pooling, layer flattening, dan layer dense.

Karena bobot diambil dari hasil pelatihan keras, maka dalam implementasi, perlu diidentifikasi unik berupa pemberian nama terhadap setiap layer konvolusi, untuk mengetahui dan mengakses nilai bobot yang dari hasil pelatihan oleh model CNN Keras. Selain itu, untuk layer pooling, terdapat konfigurasi apakah pooling yang dilakukan berupa *max pooling* ataupun *average pooling*. Flatten layer tidak memiliki konfigurasi tambahan. Dan, dense juga sama seperti layer konvolusi menerima bobot hasil pelatihan dari model Keras dan fungsi aktivasi yang digunakan untuk layer tersebut.

Setelah itu, karena CNN menyimpan seluruh layer sebagai objek (tipe: List[Layer]) maka hanya perlu melakukan iterasi terhadap list tersebut, dan memanggil method forward untuk setiap layer tersebut. Oleh karena itu, dimensi data yang di-*passing* untuk forward pada setiap layer harus diperhatikan.

```
class CNN:
    def __init__(self, weights, config):
        self.layers = []
        for spec in config:
            typ = spec[0]
            if typ=='conv':
                name = spec[1]
                W, b = weights[name]
                self.layers.append(Conv2DLayer(W, b))
            elif typ=='pool':
                mode = spec[1]
                self.layers.append(PoolingLayer(mode))
            elif typ=='flatten':
                self.layers.append(FlattenLayer())
            elif typ=='dense':
                name, act = spec[1], spec[2]
                W, b = weights[name]
                layer = Layer(input_size=W.shape[0],
                              output_size=W.shape[1],
                              activation=act,
                              weight_init='zero')
                layer.W = W
                layer.b = b.reshape(1, -1)
                self.layers.append(layer)

    def forward(self, X):
        out = X
        for l in self.layers:
            out = l.forward(out)
        return out

    def predict(self, X):
        logits = self.forward(X)
        return np.argmax(logits, axis=1)
```

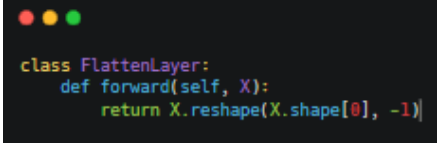
Gambar 2.1.2.1.1. Cuplikan kode implementasi logika penyimpanan dan *forward* pada kelas CNN

Selanjutnya, penjelasan untuk implementasi penanganan untuk layer-layer spesifik yang sebelumnya disebutkan akan dijelaskan dalam tabel berikut.

Tabel 2.1.2.1.1. Penjelasan implementasi setiap layer pada implementasi CNN

Nama Layer / Kelas	Snippet Kode	Penjelasan Langkah
--------------------	--------------	--------------------

Conv2DLayer	<pre> class Conv2DLayer: def __init__(self, W, b): self.W, self.b = W, b def forward(self, X): batch, h, w, c_in = X.shape kh, kw, _, c_out = self.W.shape pad_h, pad_w = kh//2, kw//2 Xp = np.pad(X, ((0,0),(pad_h,pad_h),(pad_w,pad_w),(0,0)), mode='constant') out = np.zeros((batch, h, w, c_out), dtype=X.dtype) for bi in range(batch): for i in range(h): for j in range(w): patch = Xp[bi, i:i+kh, j:j+kw, :] for co in range(c_out): out[bi, i, j, co] = np.sum(patch * self.W[:, :, :, co]) + self.b[co] return np.maximum(0, out) # ReLU </pre>	<ol style="list-style-type: none"> 1. Menyimpan bobot (W) dan bias (b) untuk setiap filter 2. Melakukan zero-padding di sekeliling input agar output memiliki ukuran spatial sama 3. Menggeser window berukuran $kh \times kw$ ke seluruh lokasi: mengalikan elemen patch dengan bobot dan menambahkan bias 4. Menerapkan fungsi aktivasi ReLU pada setiap elemen hasil konvolusi
PoolingLayer	<pre> class PoolingLayer: def __init__(self, mode='max'): self.mode = mode def forward(self, X): batch, h, w, c = X.shape nh, nw = h//2, w//2 out = np.zeros((batch, nh, nw, c), dtype=X.dtype) for bi in range(batch): for i in range(nh): for j in range(nw): region = X[bi, 2*i:2*i+2, 2*j:2*j+2, :] if self.mode == 'max': out[bi, i, j] = np.max(region, axis=(0,1)) else: out[bi, i, j] = np.mean(region, axis=(0,1)) return out </pre>	<ol style="list-style-type: none"> 1. Menggunakan mode 'max' atau 'average' 2. Membagi tiap channel input menjadi blok-blok 2×2 tanpa overlap 3. Untuk setiap blok 2×2, menghitung nilai maksimum (max pooling) atau rata-rata (average pooling) 4. Menghasilkan output dengan ukuran height dan width setengah dari input

FlattenLayer	 <pre>class FlattenLayer: def forward(self, X): return X.reshape(X.shape[0], -1)</pre>	<ol style="list-style-type: none"> 1. Menerima tensor berukuran (batch, h, w, c) 2. Meratakan (flatten) menjadi matriks 2D berukuran (batch, h·w·c) 3. Memudahkan penghubungan ke layer fully-connected selanjutnya
--------------	--	--

2.1.2.2. Simple RNN

Untuk implementasi *forward propagation from scratch* untuk model Simple RNN dapat lebih memperhatikan file `forward_rnn.py`. Objek RNN menyimpan informasi seperti bobot *input to hidden* (W), bobot *hidden to hidden* (U), bias (b_xh), units (banyak neuron *hidden layer*), dan *return_sequences* untuk menandakan apakah seluruh output hidden layer dikeluarkan atau hanya output hidden layer terakhir yang dikeluarkan.

Untuk logika implementasi *forward* adalah dengan melakukan inisialisasi nilai h sebelum masuk *timestep* pertama, yaitu dengan nilai 0 (batch, units). Kemudian, dilakukan iterasi sebanyak panjang sekuens. Kemudian, dilakukan perhitungan h dan update nilai h dengan $\tanh(x_t.W + h.U + b_{xh})$. Nilai h untuk setiap *time step* disimpan ke dalam sebuah variabel.

Terakhir, perhatikan apakah *return_sequences* true atau false. Jika true, maka seluruh h yang disimpan pada seluruh *time step* akan dikeluarkan, sedangkan jika false, maka hanya mengembalikan nilai h terakhir pada variabel yang disimpan. Implementasi kode RNN dapat dilihat pada *code snippet* berikut.

```

class RNN:
    def __init__(self,
                  kernel: np.ndarray,
                  recurrent: np.ndarray,
                  b_xh: np.ndarray,
                  units: int,
                  return_sequences: bool):
        self.W = kernel # input to hidden (input_dims, units)
        self.U = recurrent # hidden to hidden (units, units)
        self.b_xh = b_xh # bias hidden
        self.units = units # number of neuron hidden layer
        self.return_sequences = return_sequences

    def forward(self, X: np.ndarray) -> np.ndarray:
        if X.ndim == 2: #always make sure it is in 3d
            X = X[:, None, :] # (batch, 1, dim)

        # batch: number of independent paralel process
        # seq_len: length of example sequence
        batch, seq_len, _ = X.shape

        h = np.zeros((batch, self.units), dtype=X.dtype)

        outputs = []

        # loop each timestep
        for t in range(seq_len):
            # data for each batch for the specific timestep
            x_t = X[:, t, :] # (batch, in_dim)
            h = tanh(x_t.dot(self.W) + h.dot(self.U) + self.b_xh) # defaultnya keras, tanh
            outputs.append(h)

        if (self.return_sequences):
            return np.stack(outputs, axis = 1)
        else:
            return outputs[-1]

```

Gambar 2.1.2.2.1. Cuplikan kode implementasi logika *forward propagation from scratch* RNN

Selain itu, terdapat implementasi pula untuk menangani kasus RNN *bidirectional* yang mempertimbangkan konteks depan dan belakang dalam mengambil sebuah keputusan atau output dari sebuah *hidden layer*. Oleh karena itu, pengembang menambahkan penanganan tersebut dengan membuat kelas baru bernama *RNNBidirectional*. Implementasi daripada kelas tersebut dapat dilihat pada cuplikan kode berikut.

```

class RNNBidirectional:
    def __init__(self,
                  fw: RNN,
                  bw: RNN,
                  return_sequences: bool):
        self.fw = fw
        self.bw = bw
        self.return_sequences = return_sequences

    def forward(self, X: np.ndarray) -> np.ndarray:
        output_forward = self.fw.forward(X)
        output_backward = self.bw.forward(X[:, ::-1, :])

        if self.return_sequences:
            if output_forward.ndim != 3 or output_backward.ndim != 3:
                raise ValueError(
                    "Cannot return sequences when sublayers do not output sequences"
                )

            output_backward = output_backward[:, ::-1, :]
            return np.concatenate([output_forward, output_backward], axis = 2)
        else:
            if output_forward.ndim == 3:
                output_forward = output_forward[:, -1, :]
            if output_backward.ndim == 3:
                output_backward = output_backward[:, -1, :]
            return np.concatenate([output_forward, output_backward], axis = 1)

```

Gambar 2.1.2.2.2. Cuplikan kode implementasi logika *forward RNN Bidirectional from scratch*

Implementasi pada **Gambar 2.1.2.2.2.** memanfaatkan implementasi yang telah dilakukan pada kelas *RNN unidirectional*. Jadi, konsepnya, adalah dengan melakukan 2 kali *RNN unidirection* forward, dengan sekuens normal dan *reverse* dari sekuens normal tersebut. Hal ini menyebabkan h_1 pada *RNN unidirectional* sekuens normal akan di-*concat* dengan $h_{t_{\max}}$ *RNN unidirectional* sekuens *reversed*. Oleh karena itu, *RNN Bidirectional* akan menghasilkan setiap output h menjadi 2 *units*. Terakhir, sama halnya dengan *RNN unidirectional*, implementasi perlu memperhatikan *return_sequences*. Jika true, maka seluruh output h dikembalikan, dan jika false, maka hanya output *hidden layer terakhir* hasil gabungan yang dikembalikan.

Sebagai tambahan informasi oleh pengembang, bahwa variabel “bw” berarti *RNN* yang menangani sekuens yang di-*reverse*. Jadi, masih tetap menggunakan *RNN unidirectional* yang forward. **Hanya sekuens yang di-*reverse*.**

2.1.2.3. LSTM

LSTM mempunyai implementasi yang mirip dengan Simple RNN, dikarenakan keduanya mempunyai kemungkinan untuk melakukan variasi *bidirectional* dan *unidirectional*. Implementasi *forward propagation from scratch* untuk model LSTM dapat diperhatikan pada file `foward_lstm.py`. Objek LSTM menyimpan informasi bobot *input to hidden* (W), bobot *hidden to hidden* (U), bias (b_{xh}), *units* (banyak neuron *hidden layer*), dan *return_sequences* untuk menandakan apakah seluruh output hidden layer dikeluarkan atau hanya output hidden layer terakhir yang dikeluarkan.

Langkah implementasi forward adalah seperti berikut.

- a. Inisialisasi h dan c dengan dimensi (batch, units) dan variabel penyimpanan “outputs” untuk menyimpan nilai *hidden layer* pada setiap *time step*.
- b. Lakukan iterasi sebanyak *time step* (panjang sekuens dalam kasus ini)
 - i. Kalkulasi nilai z yang merupakan nilai dari $(x_t.W + h.U + b)$ dengan catatan bahwa ini telah termasuk hasil dot dari gate f , i , o , dan candidate cell. Perhatikan bahwa z ini mempunyai dimensi $4 * \text{units}$, sehingga:
 1. $\text{sigmoid}(z[0 \text{ hingga } \text{self.units}])$ merupakan hasil untuk gate input (i_t),
 2. $\text{sigmoid}(z[\text{self.units} \text{ hingga } 2*\text{self.units}])$ merupakan hasil untuk gate forget (f_t),
 3. $\text{tanh}(z[2*\text{self.units} \text{ hingga } 3*\text{self.units}])$ merupakan hasil untuk candidate_cell _{t} .
 4. $\text{sigmoid}(z[3*\text{self.units} \text{ hingga } 4*\text{self.units}])$ merupakan hasil untuk gate output (o_t).
 - ii. Kalkulasi nilai cell state, dengan rumus $c = f*c + i*(\text{candidate_cell}_t)$. Kalkulasi juga nilai output *hidden layer* untuk *time step* tersebut dengan rumus $h = o*\text{tanh}(c)$.
 - iii. Simpan nilai h baru ke variabel “outputs”

- c. Jika *return_sequences* true, maka seluruh *output hidden layer* akan dikembalikan. Jika *return_sequences* false, maka hanya dikembalikan output dari *hidden layer time step* terakhir.

```
class LSTM:
    def __init__(self,
                  kernel: np.ndarray,
                  recurrent: np.ndarray,
                  bias: np.ndarray,
                  units: int,
                  return_sequences: bool):
        self.W = kernel
        self.U = recurrent
        self.b = bias
        self.units = units
        self.return_sequences = return_sequences

    def forward(self, X: np.ndarray) -> np.ndarray:
        # input 2D -> treat as seq_len=1
        if X.ndim == 2:
            X = X[:, None, :] # -> (batch, 1, dim)
        batch, seq_len, _ = X.shape
        h = np.zeros((batch, self.units), dtype=X.dtype)
        c = np.zeros((batch, self.units), dtype=X.dtype)
        outputs = []
        for t in range(seq_len):
            x_t = X[:, t, :] # (batch, in_dim)
            z = x_t.dot(self.W) + h.dot(self.U) + self.b
            i = 1/(1+np.exp(-z[:, :self.units]))
            f = 1/(1+np.exp(-z[:, self.units:2*self.units]))
            c_bar = np.tanh(z[:, 2*self.units:3*self.units])
            o = 1/(1+np.exp(-z[:, 3*self.units:]))
            c = f*c + i*c_bar
            h = o*np.tanh(c)
            outputs.append(h)
        if self.return_sequences:
            return np.stack(outputs, axis=1) # (batch, seq_len, units)
        else:
            return h # (batch, units)
```

Gambar 2.1.2.3.1. Cuplikan kode implementasi logika *forward propagation from scratch* LSTM

Untuk konsep LSTM *bidirectional* juga sama dengan RNN *bidirectional*, namun hanya terdapat perbedaan pada jenis model yang digunakan, yaitu menggunakan model LSTM. Konsepnya masih sama, yaitu menggunakan konkatenasi hasil antara LSTM *reverse sequence* dan *normal sequence*. Cuplikan kode-nya dapat dilihat pada gambar berikut.

```

class Bidirectional:
    def __init__(self,
                  fw: LSTM,
                  bw: LSTM,
                  return_sequences: bool):
        self.fw = fw
        self.bw = bw
        self.return_sequences = return_sequences

    def forward(self, X: np.ndarray) -> np.ndarray:
        # forward pass
        out_f = self.fw.forward(X)
        # backward on reversed time axis
        out_b = self.bw.forward(X[:, ::-1, :])
        if self.return_sequences:
            # both out_f/out_b must be 3D
            out_b = out_b[:, ::-1, :]
            if out_f.ndim != 3 or out_b.ndim != 3:
                raise ValueError('Cannot return sequences when sublayers do not output sequences')
            # concat feature dim
            return np.concatenate([out_f, out_b], axis=2) # (batch, seq_len, units*2)
        else:
            # take last timestep if 3D, else take as is
            if out_f.ndim == 3:
                out_f = out_f[:, -1, :] # (batch, units)
            if out_b.ndim == 3:
                out_b = out_b[:, -1, :]
            return np.concatenate([out_f, out_b], axis=1) # (batch, units*2)

```

Gambar 2.1.2.3.2. Cuplikan kode implementasi logika *forward LSTM Bidirectional from scratch*

2.2. Hasil Pengujian

2.2.1. CNN

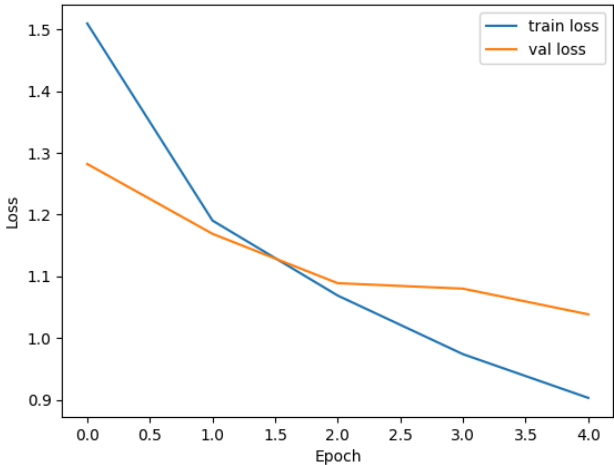
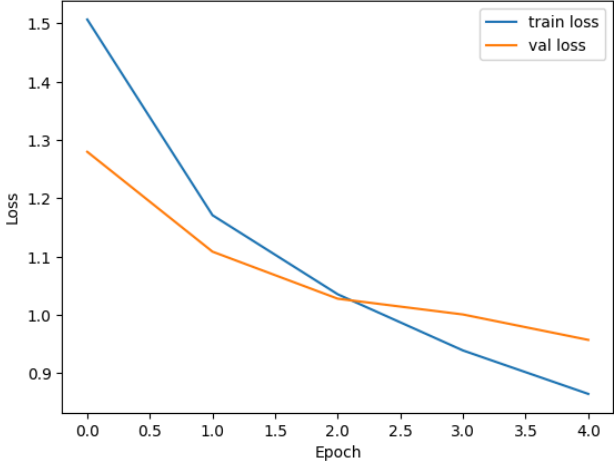
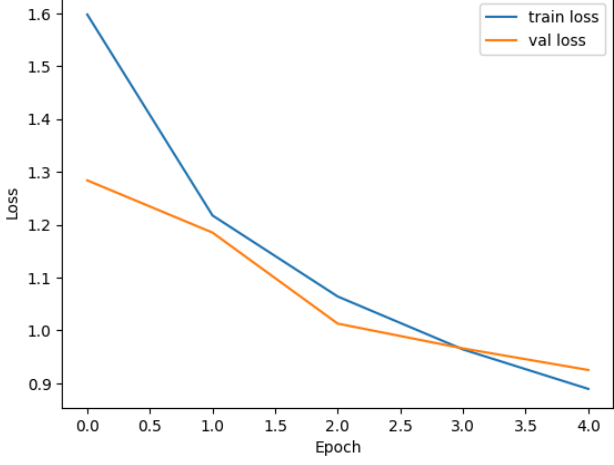
a. Pengaruh jumlah layer konvolusi

Pada eksperimen bagian ini, dilakukan perbandingan variasi jumlah layer konvolusi terhadap arsitektur CNN dengan filter berukuran 32x32x32 per Layer Konvolusi, Layer MaxPooling, 1 Layer Dense dengan 128 neuron dan aktivasi ReLU, serta Output Layer Dense dengan 10 neuron dan aktivasi softmax yakni, sebagai berikut.

1. 1 Layer Conv2d
2. 2 Layer Conv2d
3. 3 Layer Conv2d

Tabel 2.2.1.1. Perbandingan hasil akurasi dan grafik loss setiap eksperimen

Nomor	Nilai F1-score	Grafik Training dan Validation Loss
-------	----------------	-------------------------------------

i	0.6345	<p>1 Conv2D</p>  <table border="1"><thead><tr><th>Epoch</th><th>train loss</th><th>val loss</th></tr></thead><tbody><tr><td>0.0</td><td>1.50</td><td>1.28</td></tr><tr><td>1.0</td><td>1.18</td><td>1.15</td></tr><tr><td>2.0</td><td>1.08</td><td>1.09</td></tr><tr><td>3.0</td><td>0.98</td><td>1.08</td></tr><tr><td>4.0</td><td>0.90</td><td>1.04</td></tr></tbody></table>	Epoch	train loss	val loss	0.0	1.50	1.28	1.0	1.18	1.15	2.0	1.08	1.09	3.0	0.98	1.08	4.0	0.90	1.04
Epoch	train loss	val loss																		
0.0	1.50	1.28																		
1.0	1.18	1.15																		
2.0	1.08	1.09																		
3.0	0.98	1.08																		
4.0	0.90	1.04																		
ii	0.6630	<p>2 Conv2D</p>  <table border="1"><thead><tr><th>Epoch</th><th>train loss</th><th>val loss</th></tr></thead><tbody><tr><td>0.0</td><td>1.50</td><td>1.28</td></tr><tr><td>1.0</td><td>1.15</td><td>1.10</td></tr><tr><td>2.0</td><td>1.02</td><td>1.02</td></tr><tr><td>3.0</td><td>0.92</td><td>1.00</td></tr><tr><td>4.0</td><td>0.85</td><td>0.96</td></tr></tbody></table>	Epoch	train loss	val loss	0.0	1.50	1.28	1.0	1.15	1.10	2.0	1.02	1.02	3.0	0.92	1.00	4.0	0.85	0.96
Epoch	train loss	val loss																		
0.0	1.50	1.28																		
1.0	1.15	1.10																		
2.0	1.02	1.02																		
3.0	0.92	1.00																		
4.0	0.85	0.96																		
iii	0.6698	<p>3 Conv2D</p>  <table border="1"><thead><tr><th>Epoch</th><th>train loss</th><th>val loss</th></tr></thead><tbody><tr><td>0.0</td><td>1.60</td><td>1.28</td></tr><tr><td>1.0</td><td>1.20</td><td>1.18</td></tr><tr><td>2.0</td><td>1.05</td><td>1.01</td></tr><tr><td>3.0</td><td>0.95</td><td>0.95</td></tr><tr><td>4.0</td><td>0.88</td><td>0.92</td></tr></tbody></table>	Epoch	train loss	val loss	0.0	1.60	1.28	1.0	1.20	1.18	2.0	1.05	1.01	3.0	0.95	0.95	4.0	0.88	0.92
Epoch	train loss	val loss																		
0.0	1.60	1.28																		
1.0	1.20	1.18																		
2.0	1.05	1.01																		
3.0	0.95	0.95																		
4.0	0.88	0.92																		

Analisis

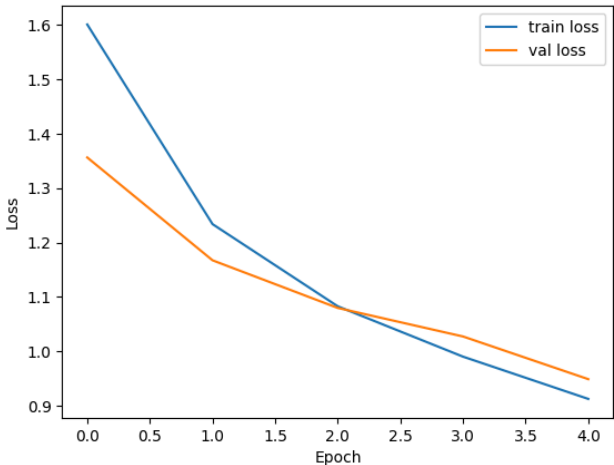
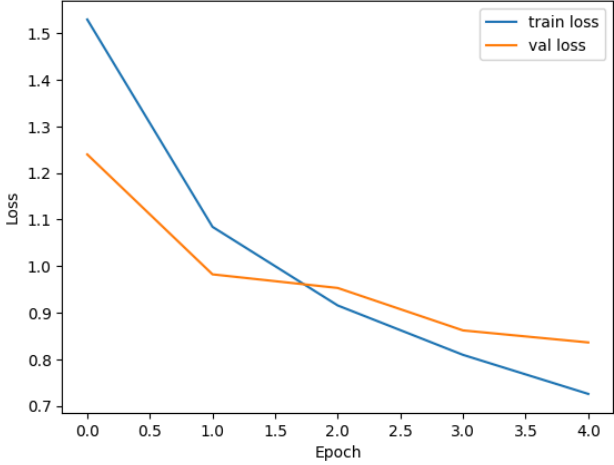
Berdasarkan grafik loss terlihat bahwa ketiga varian arsitektur mampu menurunkan loss baik pada data train maupun validasi secara konsisten selama 5 epoch pertama. Pada model 1 Layer Conv2D, loss awal yang cukup tinggi (sekitar 1.5) turun cukup cepat hingga mencapai sekitar 1.04 untuk validasi pada epoch ke-4, tetapi selisih antara train dan val loss relatif lebih besar, menandakan kapasitas fitur yang masih terbatas. Model 2 Conv2D memperlihatkan penurunan loss yang lebih stabil dan generalisasi sedikit lebih baik, pada epoch ke-4 val loss berada di kisaran 0.96, lebih rendah dibanding model 1 layer. Model 3 Conv2D menunjukkan performa terkuat: val loss terendah (~0.93) dan gap antara train-val loss paling kecil, menandakan representasi fitur yang lebih kaya tanpa overfitting berlebih dalam rentang epoch ini. Berdasarkan hasil F1-score yang diperoleh, peningkatan kedalaman konvolusi juga berbanding lurus dengan kinerja klasifikasi: model 1 layer meraih F1 0.6345, model 2 layer meningkat menjadi 0.6630, dan model 3 layer terbaik dengan F1 0.6698. Hasil ini mengindikasikan bahwa penambahan layer Conv2D mampu menangkap pola spasial lebih mendalam sehingga memperbaiki ketepatan prediksi, meski peningkatan dari 2 ke 3 layer relatif lebih kecil.

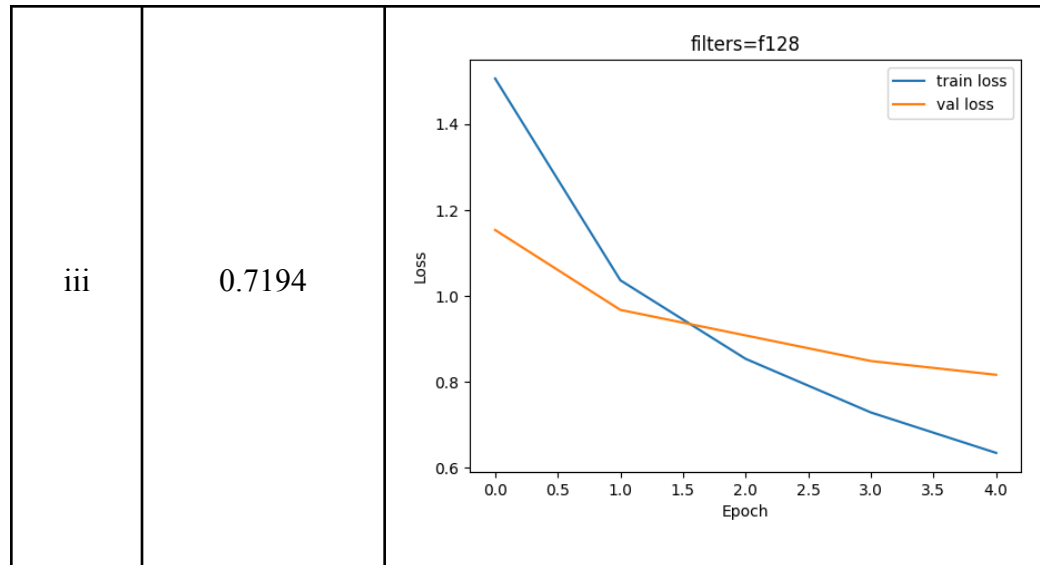
b. Pengaruh banyak filter per layer konvolusi

Pada eksperimen bagian ini, dilakukan perbandingan variasi banyaknya filter per layer konvolusi terhadap konfigurasi arsitektur CNN berupa 1 Layer Konvolusi, 1 Layer MaxPooling, 1 Layer Dense dengan 128 neuron dan aktivasi ReLU, serta Output Layer Dense dengan 10 neuron dan aktivasi softmax, yakni sebagai berikut.

1. 32 Filter berukuran 3x3
2. 64 Filter berukuran 3x3
3. 128 Filter berukuran 3x3

Tabel 2.2.1.2. Perbandingan hasil akurasi dan grafik loss setiap eksperimen

Nomor	Nilai F1-score	Grafik Training dan Validation Loss																		
i	0.6695	<p>filters=f32</p>  <table border="1"><thead><tr><th>Epoch</th><th>train loss</th><th>val loss</th></tr></thead><tbody><tr><td>0.0</td><td>1.60</td><td>1.35</td></tr><tr><td>1.0</td><td>1.23</td><td>1.17</td></tr><tr><td>2.0</td><td>1.08</td><td>1.08</td></tr><tr><td>3.0</td><td>0.98</td><td>1.03</td></tr><tr><td>4.0</td><td>0.92</td><td>0.95</td></tr></tbody></table>	Epoch	train loss	val loss	0.0	1.60	1.35	1.0	1.23	1.17	2.0	1.08	1.08	3.0	0.98	1.03	4.0	0.92	0.95
Epoch	train loss	val loss																		
0.0	1.60	1.35																		
1.0	1.23	1.17																		
2.0	1.08	1.08																		
3.0	0.98	1.03																		
4.0	0.92	0.95																		
ii	0.7099	<p>filters=f64</p>  <table border="1"><thead><tr><th>Epoch</th><th>train loss</th><th>val loss</th></tr></thead><tbody><tr><td>0.0</td><td>1.55</td><td>1.25</td></tr><tr><td>1.0</td><td>1.08</td><td>0.98</td></tr><tr><td>2.0</td><td>0.92</td><td>0.96</td></tr><tr><td>3.0</td><td>0.81</td><td>0.87</td></tr><tr><td>4.0</td><td>0.73</td><td>0.84</td></tr></tbody></table>	Epoch	train loss	val loss	0.0	1.55	1.25	1.0	1.08	0.98	2.0	0.92	0.96	3.0	0.81	0.87	4.0	0.73	0.84
Epoch	train loss	val loss																		
0.0	1.55	1.25																		
1.0	1.08	0.98																		
2.0	0.92	0.96																		
3.0	0.81	0.87																		
4.0	0.73	0.84																		



Analisis

Berdasarkan grafik loss terlihat bahwa ketiga varian jumlah filter per layer konvolusi mampu menurunkan loss baik pada data pelatihan maupun validasi secara konsisten selama 5 epoch pertama. Pada model dengan 32 filter, penurunan loss terjadi cukup cepat, namun val loss masih berada di kisaran 0.95 dengan selisih yang cukup nyata terhadap train loss. Hal ini menunjukkan bahwa kapasitas representasi fitur masih terbatas, tercermin pada F1-score yang hanya mencapai 0.6695. Peningkatan jumlah filter menjadi 64 menghasilkan penurunan loss yang lebih stabil, dengan val loss akhir sekitar 0.84 dan gap yang lebih kecil, mengindikasikan generalisasi yang lebih baik. F1-score pun meningkat signifikan menjadi 0.7099. Model dengan 128 filter menunjukkan performa terbaik: val loss terendah (~0.82) dan gap train–val loss yang tetap terkontrol, menghasilkan F1-score tertinggi 0.7194. Hasil ini mempertegas bahwa penambahan jumlah filter per layer dapat meningkatkan kemampuan model dalam mengekstraksi pola spasial yang kompleks, meskipun peningkatan dari 64 ke 128 filter bersifat marginal dibanding lonjakan dari 32 ke 64.

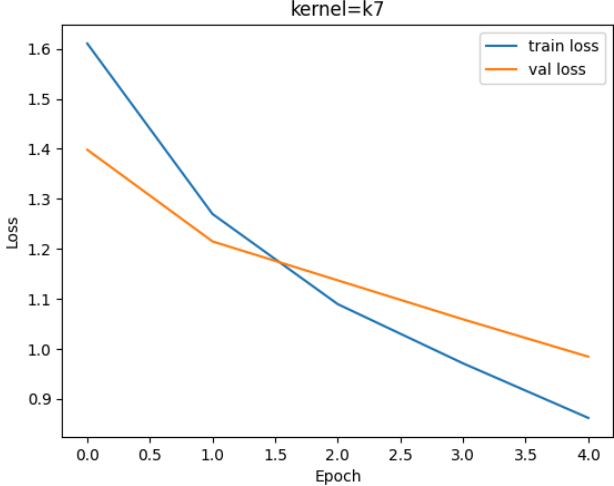
c. Pengaruh ukuran filter per layer

Pada eksperimen bagian ini, dilakukan perbandingan variasi ukuran filter per layer konvolusi terhadap konfigurasi arsitektur CNN berupa 1 Layer Konvolusi, 1 Layer MaxPooling, 1 Layer Dense dengan 128 neuron dan aktivasi ReLU, serta Output Layer Dense dengan 10 neuron dan aktivasi softmax, yakni sebagai berikut.

1. 32 Filter berukuran 3x3
2. 32 Filter berukuran 5x5
3. 32 Filter berukuran 7x7

Tabel 2.2.1.3. Perbandingan hasil akurasi dan grafik loss setiap eksperimen

Nomor	Nilai F1-score	Grafik Training dan Validation Loss																		
i	0.6713	<p>kernel=k3</p> <table border="1"> <caption>Data for kernel=k3</caption> <thead> <tr> <th>Epoch</th> <th>train loss</th> <th>val loss</th> </tr> </thead> <tbody> <tr><td>0.0</td><td>1.58</td><td>1.28</td></tr> <tr><td>1.0</td><td>1.22</td><td>1.22</td></tr> <tr><td>2.0</td><td>1.05</td><td>1.00</td></tr> <tr><td>3.0</td><td>0.95</td><td>1.00</td></tr> <tr><td>4.0</td><td>0.88</td><td>0.95</td></tr> </tbody> </table>	Epoch	train loss	val loss	0.0	1.58	1.28	1.0	1.22	1.22	2.0	1.05	1.00	3.0	0.95	1.00	4.0	0.88	0.95
Epoch	train loss	val loss																		
0.0	1.58	1.28																		
1.0	1.22	1.22																		
2.0	1.05	1.00																		
3.0	0.95	1.00																		
4.0	0.88	0.95																		
ii	0.6498	<p>kernel=k5</p> <table border="1"> <caption>Data for kernel=k5</caption> <thead> <tr> <th>Epoch</th> <th>train loss</th> <th>val loss</th> </tr> </thead> <tbody> <tr><td>0.0</td><td>1.60</td><td>1.35</td></tr> <tr><td>1.0</td><td>1.22</td><td>1.15</td></tr> <tr><td>2.0</td><td>1.05</td><td>1.02</td></tr> <tr><td>3.0</td><td>0.92</td><td>0.98</td></tr> <tr><td>4.0</td><td>0.85</td><td>0.98</td></tr> </tbody> </table>	Epoch	train loss	val loss	0.0	1.60	1.35	1.0	1.22	1.15	2.0	1.05	1.02	3.0	0.92	0.98	4.0	0.85	0.98
Epoch	train loss	val loss																		
0.0	1.60	1.35																		
1.0	1.22	1.15																		
2.0	1.05	1.02																		
3.0	0.92	0.98																		
4.0	0.85	0.98																		

iii	0.6456	 <table border="1"> <caption>Data for kernel=k7 Loss Graph</caption> <thead> <tr> <th>Epoch</th> <th>Train Loss</th> <th>Val Loss</th> </tr> </thead> <tbody> <tr> <td>0.0</td> <td>1.60</td> <td>1.40</td> </tr> <tr> <td>1.0</td> <td>1.25</td> <td>1.20</td> </tr> <tr> <td>2.0</td> <td>1.10</td> <td>1.15</td> </tr> <tr> <td>3.0</td> <td>1.00</td> <td>1.05</td> </tr> <tr> <td>4.0</td> <td>0.85</td> <td>1.00</td> </tr> </tbody> </table>	Epoch	Train Loss	Val Loss	0.0	1.60	1.40	1.0	1.25	1.20	2.0	1.10	1.15	3.0	1.00	1.05	4.0	0.85	1.00
Epoch	Train Loss	Val Loss																		
0.0	1.60	1.40																		
1.0	1.25	1.20																		
2.0	1.10	1.15																		
3.0	1.00	1.05																		
4.0	0.85	1.00																		

Analisis

Berdasarkan grafik loss terlihat bahwa ketiga varian ukuran kernel (kernel size) menunjukkan pola penurunan loss yang serupa, namun dengan perbedaan performa generalisasi yang cukup mencolok. Pada model dengan kernel 3×3 (k3), penurunan loss terjadi secara stabil baik pada data pelatihan maupun validasi, dengan val loss akhir sekitar 0.93 dan gap train-val relatif kecil. Hal ini tercermin pada nilai F1-score tertinggi yaitu 0.6713, menunjukkan bahwa ukuran kernel ini memberikan keseimbangan optimal antara cakupan lokal fitur dan kemampuan generalisasi. Pada kernel 5×5 (k5), meskipun train loss menurun dengan baik, val loss cenderung stagnan dan sedikit meningkat pada epoch akhir. F1-score menurun ke 0.6498, yang mengindikasikan bahwa cakupan fitur yang lebih luas justru dapat menangkap noise atau kehilangan detail spasial yang penting untuk klasifikasi. Model dengan kernel 7×7 (k7) menunjukkan tren serupa: penurunan train loss signifikan, namun val loss turun lebih lambat dan tetap lebih tinggi dibanding k3. F1-score-nya adalah yang terendah, yakni 0.6456. Ini mengindikasikan bahwa ukuran kernel yang terlalu besar dapat menyebabkan model menjadi kurang sensitif terhadap fitur lokal halus, sehingga kinerjanya lebih buruk.

--

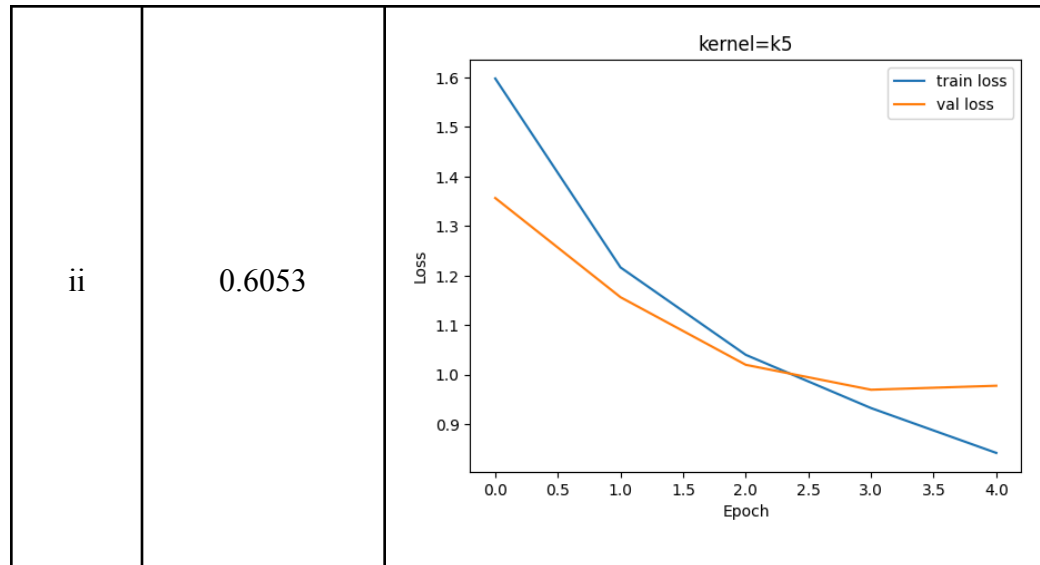
d. Pengaruh jenis pooling layer

Pada eksperimen bagian ini, dilakukan perbandingan variasi jenis pooling layer konvolusi konfigurasi arsitektur CNN berupa 1 Layer Konvolusi dengan 32 kernel berukuran 3x3, 1 Layer Dense dengan 128 neuron dan aktivasi ReLU, serta Output Layer Dense dengan 10 neuron dan aktivasi softmax, yakni sebagai berikut.

1. Max Pooling
2. Average Pooling

Tabel 2.2.1.4. Perbandingan hasil akurasi dan grafik loss setiap eksperimen

Nomor	Nilai F1-score	Grafik Training dan Validation Loss																		
i	0.6664	<table border="1"> <caption>Data for pooling=max graph</caption> <thead> <tr> <th>Epoch</th> <th>train loss</th> <th>val loss</th> </tr> </thead> <tbody> <tr> <td>0.0</td> <td>1.62</td> <td>1.32</td> </tr> <tr> <td>1.0</td> <td>1.25</td> <td>1.15</td> </tr> <tr> <td>2.0</td> <td>1.10</td> <td>1.10</td> </tr> <tr> <td>3.0</td> <td>0.98</td> <td>0.97</td> </tr> <tr> <td>4.0</td> <td>0.90</td> <td>0.97</td> </tr> </tbody> </table>	Epoch	train loss	val loss	0.0	1.62	1.32	1.0	1.25	1.15	2.0	1.10	1.10	3.0	0.98	0.97	4.0	0.90	0.97
Epoch	train loss	val loss																		
0.0	1.62	1.32																		
1.0	1.25	1.15																		
2.0	1.10	1.10																		
3.0	0.98	0.97																		
4.0	0.90	0.97																		



Analisis

Berdasarkan grafik loss, kedua metode pooling, yakni average dan max pooling menunjukkan penurunan loss yang stabil baik pada data pelatihan maupun validasi. Namun, terdapat perbedaan signifikan dalam performa generalisasi. Pada model dengan average pooling, val loss turun secara bertahap hingga ~ 1.08 , dengan gap train-val yang cukup kecil. Meskipun tampak stabil, F1-score yang dihasilkan hanya 0.6053, mengindikasikan bahwa fitur yang diekstrak kurang mampu merepresentasikan pola penting untuk klasifikasi secara efektif. Sebaliknya, model dengan max pooling menunjukkan penurunan loss yang lebih agresif dan konsisten, dengan val loss mencapai ~ 0.96 pada epoch ke-4. F1-score pun meningkat tajam menjadi 0.6664, mencerminkan bahwa max pooling lebih efektif dalam mempertahankan fitur dominan (aktivasi tertinggi) yang bersifat lebih diskriminatif. Hal ini memperkuat temuan umum bahwa max pooling cenderung lebih unggul dalam klasifikasi, karena berfokus pada fitur paling menonjol sehingga dapat membantu model membedakan kelas secara lebih tajam dibanding average pooling yang cenderung meratakan informasi.

e. Perbandingan forward propagation from Scratch dengan Keras

Pada eksperimen ini dilakukan perbandingan forward propagation yang diimplementasikan secara manual dengan Keras terhadap konfigurasi arsitektur CNN berupa 2 Layer Konvolusi dengan 32 filter berukuran 3x3, 2 Layer MaxPooling, 1 Layer Dense dengan 128 neuron dan aktivasi ReLU, serta Output Layer Dense dengan 10 neuron dan aktivasi softmax. Hasil eksperimen menunjukkan bahwa kedua algoritma baik yang diimplementasikan secara manual maupun menggunakan Keras menghasilkan F1-score yang sama, akan tetapi waktu eksekusi algoritma manual jauh lebih lambat dibandingkan dengan Keras. Hal ini terjadi karena implementasi manual menggunakan pendekatan iteratif eksplisit (nested for loop) untuk setiap operasi konvolusi, pooling, dan propagasi neuron, sehingga tidak mampu memanfaatkan optimasi berbasis vectorization maupun eksekusi paralel seperti yang dimiliki oleh backend Keras (misalnya TensorFlow).



```
y_manual = manual_cnn.predict(x_test)
y_keras = np.argmax(keras_cnn.predict(x_test), axis=1)
✓ 14m 46.0s Python

313/313 1s 2ms/step

agreement = np.mean(y_manual == y_keras)
f1_manual = f1_score(y_test, y_manual, average='macro')
f1_keras = f1_score(y_test, y_keras, average='macro')

print(f'Agreement manual vs Keras: {agreement:.2%}')
print(f'Macro F1-score manual: {f1_manual:.4f}')
print(f'Macro F1-score Keras : {f1_keras:.4f}')
✓ 0.0s Python

Agreement manual vs Keras: 100.00%
Macro F1-score manual: 0.6630
Macro F1-score Keras : 0.6630
```

Gambar 2.2.1.1. Cuplikan notebook perbandingan *forward propagation* manual dengan Keras

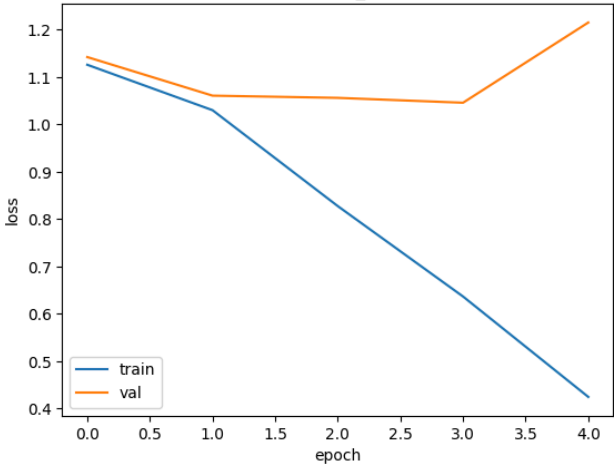
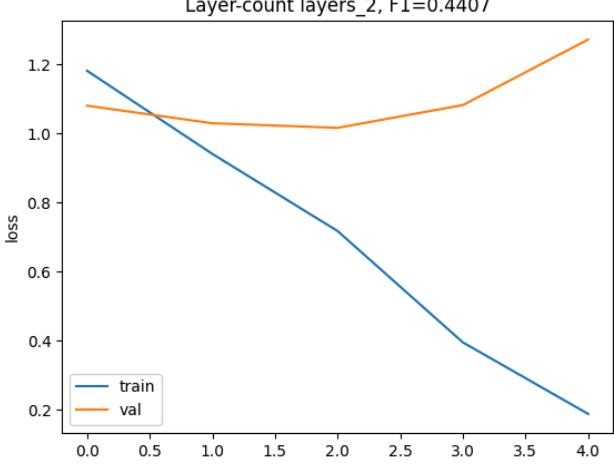
2.2.2. Simple RNN

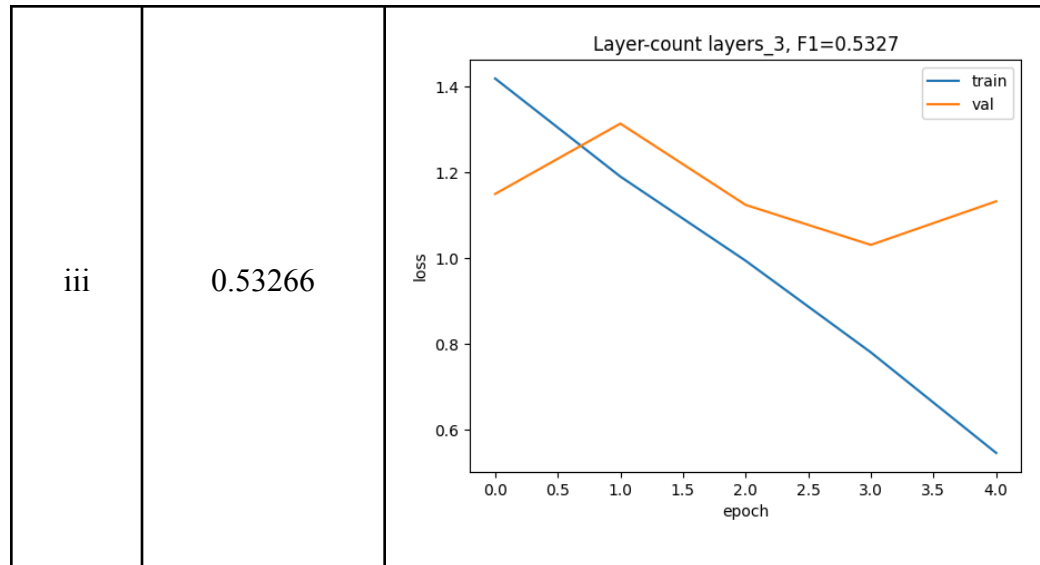
a. Pengaruh jumlah layer RNN

Pada eksperimen ini dilakukan perbandingan variasi jumlah layer RNN terhadap arsitektur RNN berupa n layer SimpleRNN bidirectional dengan 128 neuron dan output layer dengan 3 neuron dan aktivasi softmax, yakni sebagai berikut.

1. 1 Layer SimpleRNN
2. 2 Layer SimpleRNN
3. 3 Layer SimpleRNN

Tabel 2.2.2.1. Perbandingan hasil akurasi dan grafik loss setiap eksperimen

Nomor	Nilai F1-score	Grafik Training dan Validation Loss																		
i	0.37578	<p>Layer-count layers_1, F1=0.3758</p>  <table border="1"> <caption>Data for Layer-count layers_1</caption> <thead> <tr> <th>epoch</th> <th>train</th> <th>val</th> </tr> </thead> <tbody> <tr> <td>0.0</td> <td>1.15</td> <td>1.15</td> </tr> <tr> <td>1.0</td> <td>1.05</td> <td>1.05</td> </tr> <tr> <td>2.0</td> <td>0.85</td> <td>1.05</td> </tr> <tr> <td>3.0</td> <td>0.65</td> <td>1.05</td> </tr> <tr> <td>4.0</td> <td>0.45</td> <td>1.25</td> </tr> </tbody> </table>	epoch	train	val	0.0	1.15	1.15	1.0	1.05	1.05	2.0	0.85	1.05	3.0	0.65	1.05	4.0	0.45	1.25
epoch	train	val																		
0.0	1.15	1.15																		
1.0	1.05	1.05																		
2.0	0.85	1.05																		
3.0	0.65	1.05																		
4.0	0.45	1.25																		
ii	0.44072	<p>Layer-count layers_2, F1=0.4407</p>  <table border="1"> <caption>Data for Layer-count layers_2</caption> <thead> <tr> <th>epoch</th> <th>train</th> <th>val</th> </tr> </thead> <tbody> <tr> <td>0.0</td> <td>1.15</td> <td>1.10</td> </tr> <tr> <td>1.0</td> <td>0.95</td> <td>1.05</td> </tr> <tr> <td>2.0</td> <td>0.75</td> <td>1.05</td> </tr> <tr> <td>3.0</td> <td>0.40</td> <td>1.10</td> </tr> <tr> <td>4.0</td> <td>0.20</td> <td>1.30</td> </tr> </tbody> </table>	epoch	train	val	0.0	1.15	1.10	1.0	0.95	1.05	2.0	0.75	1.05	3.0	0.40	1.10	4.0	0.20	1.30
epoch	train	val																		
0.0	1.15	1.10																		
1.0	0.95	1.05																		
2.0	0.75	1.05																		
3.0	0.40	1.10																		
4.0	0.20	1.30																		



Analisis

Berdasarkan grafik, model dengan 1 layer RNN menunjukkan penurunan train loss yang stabil, namun val loss cenderung stagnan dan meningkat di akhir epoch. Hal ini menandakan gejala overfitting awal serta kapasitas representasi yang terbatas, tercermin dari F1-score yang rendah yaitu 0.3758. Pada model 2 layer RNN, penurunan train loss lebih cepat dan val loss tetap relatif stabil meskipun mulai naik sedikit setelah epoch ke-2. Hal ini mengindikasikan peningkatan kapasitas model untuk memahami data sekuensial, dan hasilnya adalah kenaikan F1-score menjadi 0.4407. Model 3 layer RNN menunjukkan performa terbaik, dengan penurunan loss yang konsisten dan F1-score tertinggi sebesar 0.5327. Meskipun val loss sempat fluktuatif di awal, tren akhirnya menurun, menandakan generalisasi yang membaik. Secara keseluruhan, penambahan layer meningkatkan kemampuan model menangkap pola temporal, meskipun perlu diimbangi dengan strategi pelatihan yang tepat, seperti penyesuaian learning rate dan early stopping.

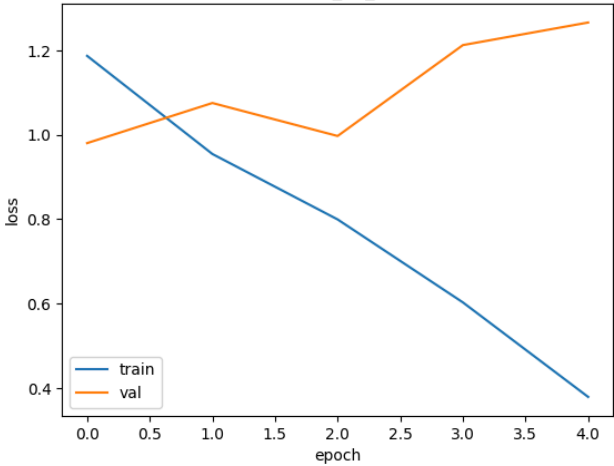
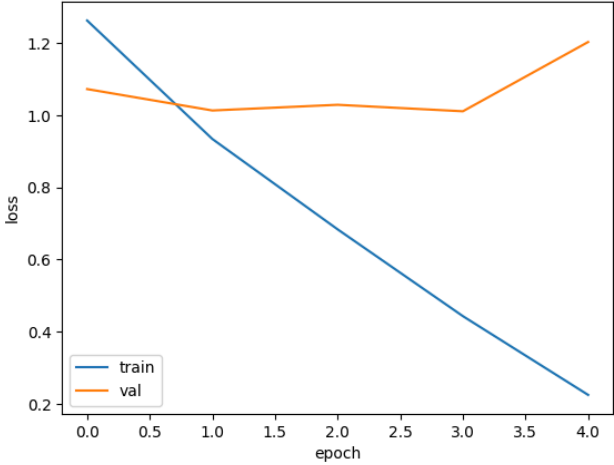
b. Pengaruh banyak cell RNN per layer

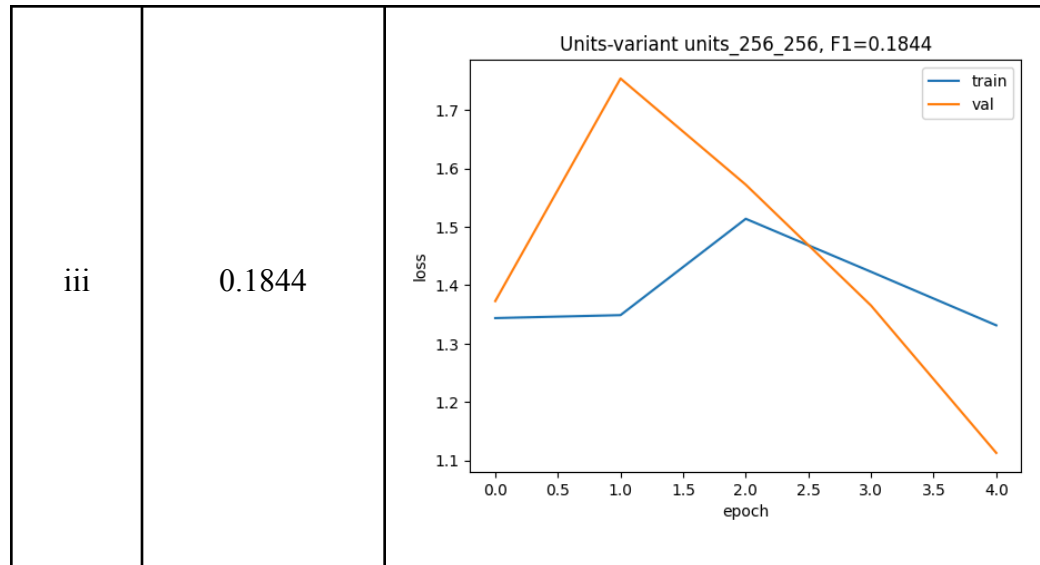
Pada eksperimen ini dilakukan perbandingan variasi banyak cell (neuron) pada tiap layer SimpleRNN dengan arsitektur RNN berupa 3 layer SimpleRNN

bidirectional dengan 128 neuron dan output layer dengan 3 neuron dan aktivasi softmax, yakni sebagai berikut.

- 1. 64 neuron per layer
- 2. 128 neuron per layer
- 3. 256 neuron per layer

Tabel 2.2.2.2. Perbandingan hasil akurasi dan grafik loss setiap eksperimen

Nomor	Nilai F1-score	Grafik Training dan Validation Loss																		
i	0.5280	<div>Units-variant units_64_64, F1=0.5280</div>  <table border="1"><thead><tr><th>epoch</th><th>train</th><th>val</th></tr></thead><tbody><tr><td>0.0</td><td>1.2</td><td>1.0</td></tr><tr><td>1.0</td><td>0.95</td><td>1.1</td></tr><tr><td>2.0</td><td>0.8</td><td>1.0</td></tr><tr><td>3.0</td><td>0.6</td><td>1.2</td></tr><tr><td>4.0</td><td>0.4</td><td>1.3</td></tr></tbody></table>	epoch	train	val	0.0	1.2	1.0	1.0	0.95	1.1	2.0	0.8	1.0	3.0	0.6	1.2	4.0	0.4	1.3
epoch	train	val																		
0.0	1.2	1.0																		
1.0	0.95	1.1																		
2.0	0.8	1.0																		
3.0	0.6	1.2																		
4.0	0.4	1.3																		
ii	0.4623	<div>Units-variant units_128_128, F1=0.4623</div>  <table border="1"><thead><tr><th>epoch</th><th>train</th><th>val</th></tr></thead><tbody><tr><td>0.0</td><td>1.3</td><td>1.1</td></tr><tr><td>1.0</td><td>0.95</td><td>1.0</td></tr><tr><td>2.0</td><td>0.7</td><td>1.0</td></tr><tr><td>3.0</td><td>0.45</td><td>1.0</td></tr><tr><td>4.0</td><td>0.25</td><td>1.2</td></tr></tbody></table>	epoch	train	val	0.0	1.3	1.1	1.0	0.95	1.0	2.0	0.7	1.0	3.0	0.45	1.0	4.0	0.25	1.2
epoch	train	val																		
0.0	1.3	1.1																		
1.0	0.95	1.0																		
2.0	0.7	1.0																		
3.0	0.45	1.0																		
4.0	0.25	1.2																		



Analisis

Berdasarkan grafik dan nilai F1-score, terlihat bahwa jumlah unit RNN (neuron per layer) berpengaruh terhadap performa model. Konfigurasi dengan 64 unit per layer menunjukkan penurunan train loss yang konsisten, dan meskipun val loss fluktuatif, model masih mampu mencapai F1-score tertinggi sebesar 0.5280, menunjukkan keseimbangan antara kapasitas dan generalisasi. Pada konfigurasi 128 unit per layer, performa juga cukup stabil dengan penurunan train loss yang baik. Namun, val loss relatif mendatar dan cenderung meningkat di akhir epoch, menyebabkan penurunan generalisasi dan menghasilkan F1-score yang lebih rendah yakni 0.4623. Sebaliknya, model dengan 256 unit per layer mengalami instabilitas yang jelas, dengan val loss sempat melonjak tajam sebelum akhirnya turun. Ini mengindikasikan overfitting awal yang tidak terkendali, dan performa klasifikasi menjadi sangat buruk dengan F1-score hanya 0.1844. Hasil ini menunjukkan bahwa peningkatan jumlah unit secara berlebihan dapat memperburuk performa jika tidak diimbangi dengan regularisasi atau data yang memadai.

c. Pengaruh jenis layer RNN berdasarkan arah

Pada eksperimen ini dilakukan perbandingan jenis layer RNN dengan arsitektur RNN berupa 2 layer SimpleRNN dengan 128 neuron dan output layer dengan 3 neuron dan aktivasi softmax, yakni sebagai berikut.

1. Unidirectional
2. Bidirectional

Tabel 2.2.2.3. Perbandingan hasil akurasi dan grafik loss setiap eksperimen

Nomor	Nilai F1-score	Grafik Training dan Validation Loss																		
i	0.3447	<p>Direction-variant dir_uni, F1=0.3447</p> <table border="1"> <thead> <tr> <th>epoch</th> <th>train</th> <th>val</th> </tr> </thead> <tbody> <tr><td>0.0</td><td>1.22</td><td>1.08</td></tr> <tr><td>1.0</td><td>1.18</td><td>1.22</td></tr> <tr><td>2.0</td><td>1.05</td><td>1.14</td></tr> <tr><td>3.0</td><td>0.95</td><td>1.52</td></tr> <tr><td>4.0</td><td>0.60</td><td>1.65</td></tr> </tbody> </table>	epoch	train	val	0.0	1.22	1.08	1.0	1.18	1.22	2.0	1.05	1.14	3.0	0.95	1.52	4.0	0.60	1.65
epoch	train	val																		
0.0	1.22	1.08																		
1.0	1.18	1.22																		
2.0	1.05	1.14																		
3.0	0.95	1.52																		
4.0	0.60	1.65																		
ii	0.4785	<p>Direction-variant dir_bi, F1=0.4785</p> <table border="1"> <thead> <tr> <th>epoch</th> <th>train</th> <th>val</th> </tr> </thead> <tbody> <tr><td>0.0</td><td>1.22</td><td>1.22</td></tr> <tr><td>1.0</td><td>0.95</td><td>1.04</td></tr> <tr><td>2.0</td><td>0.75</td><td>1.02</td></tr> <tr><td>3.0</td><td>0.45</td><td>1.12</td></tr> <tr><td>4.0</td><td>0.20</td><td>1.28</td></tr> </tbody> </table>	epoch	train	val	0.0	1.22	1.22	1.0	0.95	1.04	2.0	0.75	1.02	3.0	0.45	1.12	4.0	0.20	1.28
epoch	train	val																		
0.0	1.22	1.22																		
1.0	0.95	1.04																		
2.0	0.75	1.02																		
3.0	0.45	1.12																		
4.0	0.20	1.28																		
Analisis Berdasarkan grafik, terlihat bahwa model unidirectional RNN (dir_uni)																				

mengalami overfitting yang cukup signifikan. Meskipun train loss menurun stabil, val loss justru meningkat tajam setelah epoch ke-2. Hal ini menunjukkan bahwa model kesulitan dalam menangkap konteks sekuensial secara menyeluruh, dengan F1-score yang rendah yaitu 0.3447. Sebaliknya, model bidirectional RNN (dir_bi) menunjukkan performa yang jauh lebih stabil. Baik train loss maupun val loss menurun lebih konsisten, dan gap antara keduanya lebih kecil. F1-score juga meningkat menjadi 0.4785, yang mencerminkan bahwa bidirectional RNN lebih mampu menangkap informasi dari dua arah konteks—baik masa lalu maupun masa depan. Hasil ini mengindikasikan bahwa penggunaan arsitektur bidirectional memberikan keuntungan signifikan dalam memahami konteks sekuensial yang lebih kompleks, sehingga menghasilkan prediksi yang lebih akurat dibanding unidirectional, terutama pada data yang membutuhkan konteks penuh dari keseluruhan urutan.

d. Perbandingan forward propagation from Scratch dengan Keras

Pada eksperimen ini dilakukan perbandingan forward propagation yang diimplementasikan secara manual dengan Keras terhadap konfigurasi arsitektur RNN berupa 2 layer bidirectional RNN, masing-masing terdiri dari 128 neuron, dan output layer dengan 3 neuron dan aktivasi softmax. Hasil eksperimen menunjukkan bahwa kedua algoritma baik yang diimplementasikan secara manual maupun menggunakan Keras menghasilkan F1-score yang sama.

```
probs_keras = model.predict(X_test_tok, verbose=0)
preds_keras = np.argmax(probs_keras, axis=1)

probs_manual = predict(pipeline, X_test_tok)
preds_manual = np.argmax(probs_manual, axis=1)

f1_keras = f1_score(y_test_enc, preds_keras, average='macro')
f1_manual = f1_score(y_test_enc, preds_manual, average='macro')

print(f'F1 macro Keras : {f1_keras:.4f}')
print(f'F1 macro Manual : {f1_manual:.4f}')
print(f'Predictions identical rate: {np.mean(preds_keras == preds_manual):.3f}')
```

F1 macro Keras : 0.4407
F1 macro Manual : 0.4407
Predictions identical rate: 1.000

Gambar 2.2.2.1. Cuplikan notebook perbandingan *forward propagation* manual dengan Keras

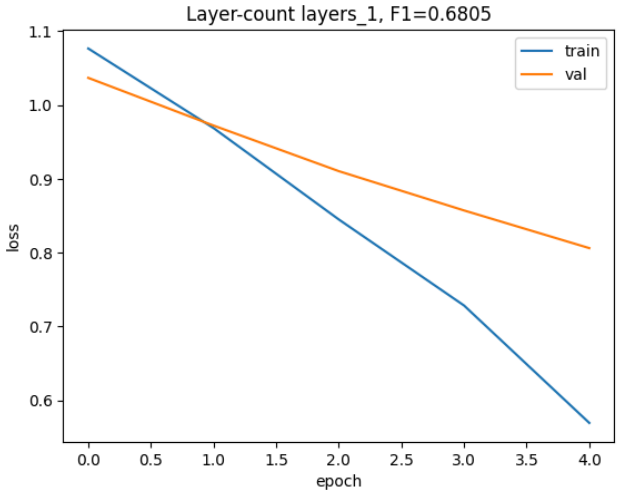
2.2.3. LSTM

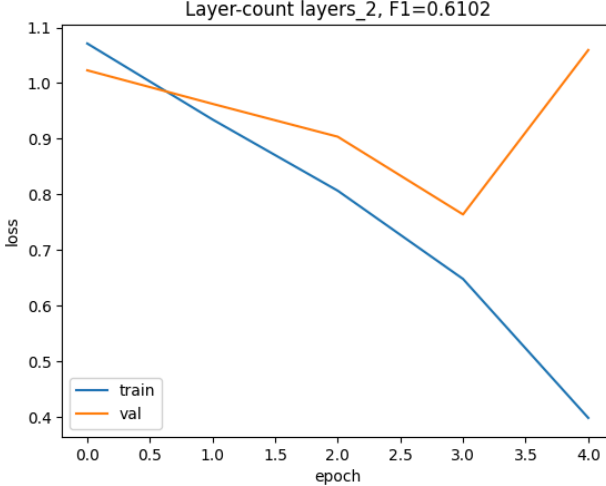
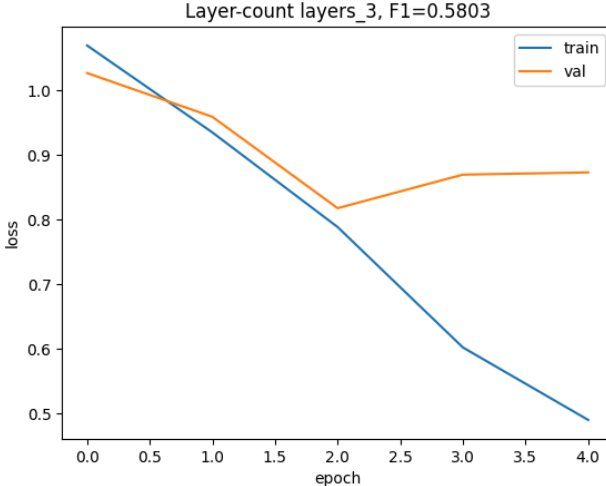
a. Pengaruh jumlah layer LSTM

Pada eksperimen ini dilakukan perbandingan variasi jumlah layer LSTM terhadap arsitektur LSTM berupa n layer LSTM bidirectional dengan 128 neuron dan output layer dengan 3 neuron dan aktivasi softmax, yakni sebagai berikut.

- 1. 1 Layer LSTM
- 2. 2 Layer LSTM
- 3. 3 Layer LSTM

Tabel 2.2.2.1. Perbandingan hasil akurasi dan grafik loss setiap eksperimen

Nomor	Nilai F1-score	Grafik Training dan Validation Loss
i	0.6805	

ii	0.6102	<p>Layer-count layers_2, F1=0.6102</p>  <table border="1"> <caption>Data for Layer-count layers_2</caption> <thead> <tr> <th>epoch</th> <th>train</th> <th>val</th> </tr> </thead> <tbody> <tr><td>0.0</td><td>1.08</td><td>1.02</td></tr> <tr><td>0.5</td><td>1.00</td><td>0.98</td></tr> <tr><td>1.0</td><td>0.92</td><td>0.95</td></tr> <tr><td>1.5</td><td>0.85</td><td>0.92</td></tr> <tr><td>2.0</td><td>0.80</td><td>0.90</td></tr> <tr><td>2.5</td><td>0.75</td><td>0.85</td></tr> <tr><td>3.0</td><td>0.65</td><td>0.78</td></tr> <tr><td>3.5</td><td>0.55</td><td>0.90</td></tr> <tr><td>4.0</td><td>0.40</td><td>1.08</td></tr> </tbody> </table>	epoch	train	val	0.0	1.08	1.02	0.5	1.00	0.98	1.0	0.92	0.95	1.5	0.85	0.92	2.0	0.80	0.90	2.5	0.75	0.85	3.0	0.65	0.78	3.5	0.55	0.90	4.0	0.40	1.08
epoch	train	val																														
0.0	1.08	1.02																														
0.5	1.00	0.98																														
1.0	0.92	0.95																														
1.5	0.85	0.92																														
2.0	0.80	0.90																														
2.5	0.75	0.85																														
3.0	0.65	0.78																														
3.5	0.55	0.90																														
4.0	0.40	1.08																														
iii	0.5803	<p>Layer-count layers_3, F1=0.5803</p>  <table border="1"> <caption>Data for Layer-count layers_3</caption> <thead> <tr> <th>epoch</th> <th>train</th> <th>val</th> </tr> </thead> <tbody> <tr><td>0.0</td><td>1.05</td><td>1.02</td></tr> <tr><td>0.5</td><td>0.98</td><td>0.98</td></tr> <tr><td>1.0</td><td>0.92</td><td>0.95</td></tr> <tr><td>1.5</td><td>0.85</td><td>0.90</td></tr> <tr><td>2.0</td><td>0.78</td><td>0.82</td></tr> <tr><td>2.5</td><td>0.70</td><td>0.85</td></tr> <tr><td>3.0</td><td>0.60</td><td>0.88</td></tr> <tr><td>3.5</td><td>0.55</td><td>0.88</td></tr> <tr><td>4.0</td><td>0.50</td><td>0.88</td></tr> </tbody> </table>	epoch	train	val	0.0	1.05	1.02	0.5	0.98	0.98	1.0	0.92	0.95	1.5	0.85	0.90	2.0	0.78	0.82	2.5	0.70	0.85	3.0	0.60	0.88	3.5	0.55	0.88	4.0	0.50	0.88
epoch	train	val																														
0.0	1.05	1.02																														
0.5	0.98	0.98																														
1.0	0.92	0.95																														
1.5	0.85	0.90																														
2.0	0.78	0.82																														
2.5	0.70	0.85																														
3.0	0.60	0.88																														
3.5	0.55	0.88																														
4.0	0.50	0.88																														
<p>Analisis</p> <p>Berdasarkan hasil eksperimen terhadap variasi jumlah layer pada arsitektur LSTM, terlihat bahwa model dengan 1 layer memberikan performa terbaik. Train loss dan val loss menurun stabil dan paralel, menunjukkan kemampuan generalisasi yang baik, dengan F1-score tertinggi sebesar 0.6805. Penambahan menjadi 2 layer LSTM menghasilkan penurunan loss yang cukup baik di awal, namun val loss menunjukkan fluktuasi cukup besar pada epoch akhir, yang mengindikasikan mulai terjadi overfitting. F1-score-nya menurun menjadi 0.6102, mencerminkan penurunan akurasi klasifikasi. Pada 3 layer LSTM, meskipun train loss terus menurun tajam, val loss cenderung</p>																																

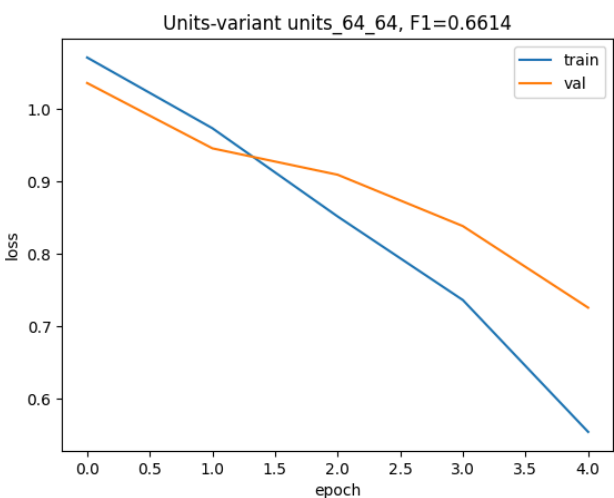
stagnan setelah awal penurunan, dan bahkan sedikit meningkat. F1-score semakin turun menjadi 0.5803, menandakan bahwa kedalaman berlebih pada model ini justru menurunkan performa. Hal ini disebabkan karena model menjadi terlalu kompleks dan mulai mengalami overfitting terhadap data pelatihan, terutama jika tidak disertai peningkatan ukuran dataset atau regularisasi tambahan. Selain itu, penambahan layer LSTM juga dapat menyebabkan hilangnya gradien selama proses backpropagation pada sekuens panjang, yang berdampak negatif terhadap pembelajaran fitur penting.

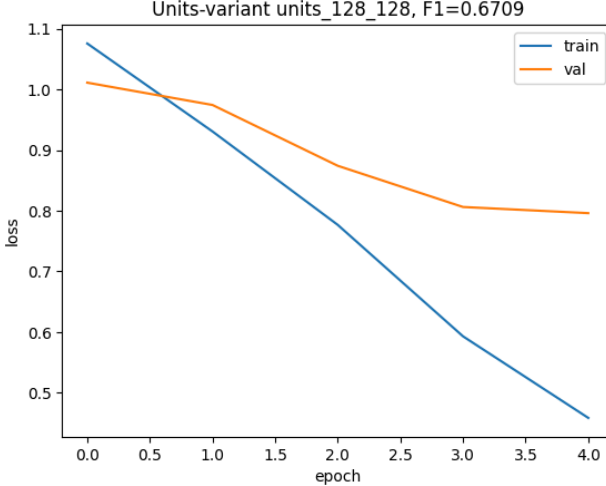
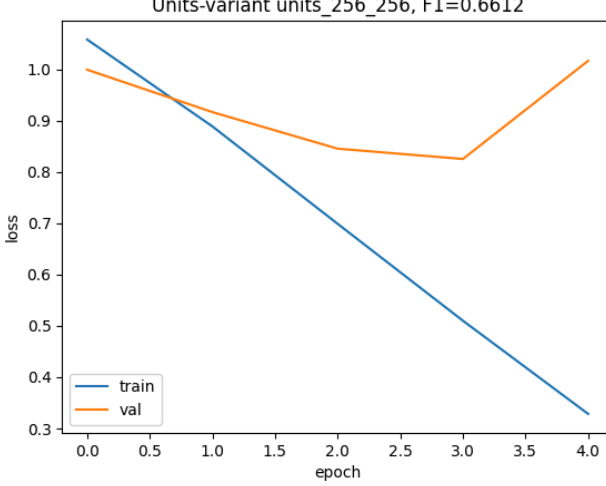
b. Pengaruh banyak cell LSTM per layer

Pada eksperimen ini dilakukan perbandingan variasi banyak cell (neuron) pada tiap layer LSTM dengan arsitektur LSTM berupa 3 layer LSTM bidirectional dengan 128 neuron dan output layer dengan 3 neuron dan aktivasi softmax, yakni sebagai berikut.

1. 64 neuron per layer
2. 128 neuron per layer
3. 256 neuron per layer

Tabel 2.2.2.1. Perbandingan hasil akurasi dan grafik loss setiap eksperimen

Nomor	Nilai F1-score	Grafik Training dan Validation Loss
i	0.6614	 <p>The graph displays the training and validation loss for a model with 64 units per layer. The x-axis represents the number of epochs (0.0 to 4.0), and the y-axis represents the loss (0.6 to 1.0). The training loss (blue line) starts at approximately 1.05 and decreases steadily to about 0.55 by epoch 4.0. The validation loss (orange line) starts at approximately 1.02, decreases to about 0.95 at epoch 1.0, and then continues to decrease more slowly, reaching approximately 0.72 by epoch 4.0. The legend indicates that the blue line represents the training loss and the orange line represents the validation loss.</p>

ii	0.6709	<p>Units-variant units_128_128, F1=0.6709</p>  <table border="1"> <caption>Data for Units-variant units_128_128, F1=0.6709</caption> <thead> <tr> <th>epoch</th> <th>train loss</th> <th>val loss</th> </tr> </thead> <tbody> <tr><td>0.0</td><td>1.08</td><td>1.02</td></tr> <tr><td>0.5</td><td>1.00</td><td>1.00</td></tr> <tr><td>1.0</td><td>0.92</td><td>0.98</td></tr> <tr><td>1.5</td><td>0.84</td><td>0.92</td></tr> <tr><td>2.0</td><td>0.78</td><td>0.88</td></tr> <tr><td>2.5</td><td>0.70</td><td>0.84</td></tr> <tr><td>3.0</td><td>0.60</td><td>0.80</td></tr> <tr><td>3.5</td><td>0.52</td><td>0.80</td></tr> <tr><td>4.0</td><td>0.45</td><td>0.80</td></tr> </tbody> </table>	epoch	train loss	val loss	0.0	1.08	1.02	0.5	1.00	1.00	1.0	0.92	0.98	1.5	0.84	0.92	2.0	0.78	0.88	2.5	0.70	0.84	3.0	0.60	0.80	3.5	0.52	0.80	4.0	0.45	0.80
epoch	train loss	val loss																														
0.0	1.08	1.02																														
0.5	1.00	1.00																														
1.0	0.92	0.98																														
1.5	0.84	0.92																														
2.0	0.78	0.88																														
2.5	0.70	0.84																														
3.0	0.60	0.80																														
3.5	0.52	0.80																														
4.0	0.45	0.80																														
iii	0.6612	<p>Units-variant units_256_256, F1=0.6612</p>  <table border="1"> <caption>Data for Units-variant units_256_256, F1=0.6612</caption> <thead> <tr> <th>epoch</th> <th>train loss</th> <th>val loss</th> </tr> </thead> <tbody> <tr><td>0.0</td><td>1.05</td><td>1.00</td></tr> <tr><td>0.5</td><td>0.98</td><td>0.95</td></tr> <tr><td>1.0</td><td>0.90</td><td>0.90</td></tr> <tr><td>1.5</td><td>0.82</td><td>0.88</td></tr> <tr><td>2.0</td><td>0.72</td><td>0.85</td></tr> <tr><td>2.5</td><td>0.62</td><td>0.84</td></tr> <tr><td>3.0</td><td>0.52</td><td>0.82</td></tr> <tr><td>3.5</td><td>0.42</td><td>0.90</td></tr> <tr><td>4.0</td><td>0.32</td><td>1.02</td></tr> </tbody> </table>	epoch	train loss	val loss	0.0	1.05	1.00	0.5	0.98	0.95	1.0	0.90	0.90	1.5	0.82	0.88	2.0	0.72	0.85	2.5	0.62	0.84	3.0	0.52	0.82	3.5	0.42	0.90	4.0	0.32	1.02
epoch	train loss	val loss																														
0.0	1.05	1.00																														
0.5	0.98	0.95																														
1.0	0.90	0.90																														
1.5	0.82	0.88																														
2.0	0.72	0.85																														
2.5	0.62	0.84																														
3.0	0.52	0.82																														
3.5	0.42	0.90																														
4.0	0.32	1.02																														
<p>Analisis</p> <p>Berdasarkan hasil eksperimen terhadap variasi jumlah unit LSTM per layer, terlihat bahwa model dengan 128 unit per layer memberikan performa terbaik dengan F1-score sebesar 0.6709. Grafik loss menunjukkan penurunan yang stabil baik pada train maupun val loss, serta gap yang terjaga, mengindikasikan keseimbangan yang baik antara kapasitas dan generalisasi model. Model dengan 64 unit menunjukkan hasil yang hampir setara, dengan F1-score 0.6614, namun penurunan val loss cenderung lebih lambat dibanding model 128 unit. Ini menandakan bahwa kapasitas representasinya sedikit lebih terbatas, meskipun masih mampu menjaga performa yang</p>																																

kompetitif. Sementara itu, model dengan 256 unit meskipun berhasil menurunkan train loss secara tajam, justru mengalami lonjakan val loss pada epoch akhir. Hal ini menunjukkan adanya gejala overfitting, yang menyebabkan penurunan generalisasi meski F1-score masih cukup tinggi di angka 0.6612. Hasil ini memperlihatkan bahwa penambahan unit secara berlebihan tidak selalu memberikan keuntungan signifikan dan perlu disesuaikan dengan kompleksitas data.

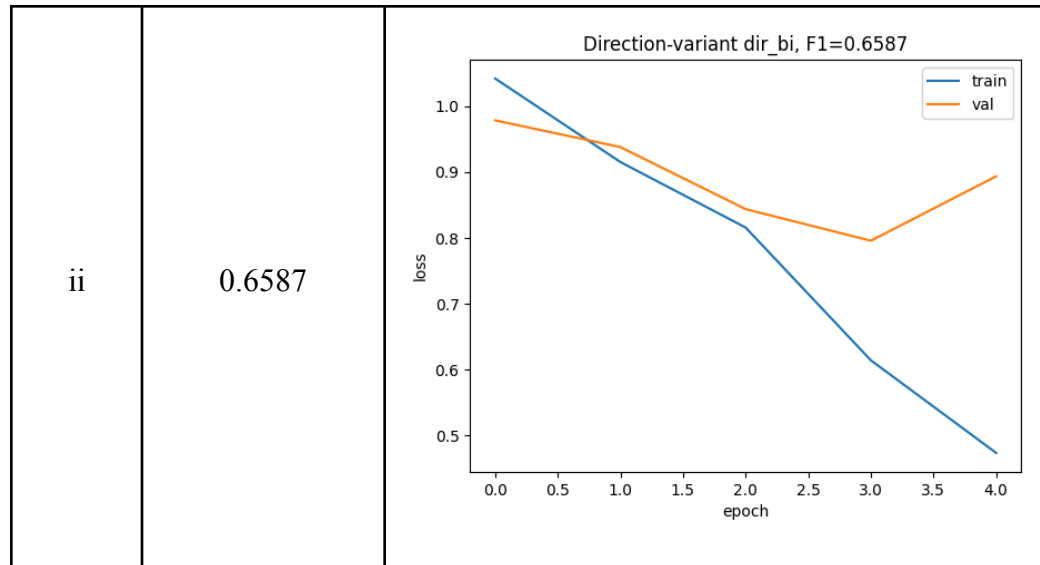
c. Pengaruh jenis layer LSTM berdasarkan arah

Pada eksperimen ini dilakukan perbandingan jenis layer LSTM dengan arsitektur LSTM berupa 2 layer LSTM dengan 128 neuron dan output layer dengan 3 neuron dan aktivasi softmax, yakni sebagai berikut.

1. Unidirectional
2. Bidirectional

Tabel 2.2.2.1. Perbandingan hasil akurasi dan grafik loss setiap eksperimen

Nomor	Nilai F1-score	Grafik Training dan Validation Loss																		
i	0.1844	<p>Direction-variant dir_uni, F1=0.1844</p> <table border="1"> <caption>Data points for Training and Validation Loss</caption> <thead> <tr> <th>epoch</th> <th>train loss</th> <th>val loss</th> </tr> </thead> <tbody> <tr> <td>0.0</td> <td>1.0835</td> <td>1.0795</td> </tr> <tr> <td>1.0</td> <td>1.0865</td> <td>1.0850</td> </tr> <tr> <td>2.0</td> <td>1.0855</td> <td>1.0805</td> </tr> <tr> <td>3.0</td> <td>1.0800</td> <td>1.0780</td> </tr> <tr> <td>4.0</td> <td>1.0835</td> <td>1.0780</td> </tr> </tbody> </table>	epoch	train loss	val loss	0.0	1.0835	1.0795	1.0	1.0865	1.0850	2.0	1.0855	1.0805	3.0	1.0800	1.0780	4.0	1.0835	1.0780
epoch	train loss	val loss																		
0.0	1.0835	1.0795																		
1.0	1.0865	1.0850																		
2.0	1.0855	1.0805																		
3.0	1.0800	1.0780																		
4.0	1.0835	1.0780																		



Analisis

Berdasarkan grafik dan nilai F1-score, terlihat perbedaan signifikan antara penggunaan arsitektur unidirectional dan bidirectional pada model LSTM. Model unidirectional (dir_uni) menunjukkan performa yang sangat buruk, dengan loss yang stagnan dan tidak mengalami penurunan berarti, serta F1-score sangat rendah yaitu 0.1844. Hal ini menunjukkan bahwa model gagal belajar secara efektif dan tidak mampu menangkap konteks urutan secara utuh. Sebaliknya, model bidirectional (dir_bi) menunjukkan penurunan loss yang konsisten baik pada data train maupun validasi, dengan tren yang jelas menuju konvergensi. F1-score yang dihasilkan mencapai 0.6587, menandakan peningkatan substansial dalam kemampuan klasifikasi. Hal ini disebabkan oleh kemampuan bidirectional LSTM dalam menangkap informasi dari dua arah waktu (masa lalu dan masa depan) dalam sekuens, sehingga lebih memahami konteks secara menyeluruh. Dengan demikian, dapat disimpulkan bahwa arsitektur bidirectional memberikan keunggulan yang signifikan dalam pemrosesan data sekuensial, terutama untuk klasifikasi yang membutuhkan pemahaman konteks penuh dari seluruh urutan input.

d. Perbandingan forward propagation from Scratch dengan Keras

Pada eksperimen ini dilakukan perbandingan forward propagation yang diimplementasikan secara manual dengan Keras terhadap konfigurasi arsitektur LSTM berupa 2 layer bidirectional LSTM, masing-masing terdiri dari 128 neuron, dan output layer dengan 3 neuron dan aktivasi softmax. Hasil eksperimen menunjukkan bahwa kedua algoritma baik yang diimplementasikan secara manual maupun menggunakan Keras menghasilkan F1-score yang sama.

```
probs_keras = model.predict(X_test_tok, verbose=0)
preds_keras = np.argmax(probs_keras, axis=1)

probs_manual = predict(pipeline, X_test_tok)
preds_manual = np.argmax(probs_manual, axis=1)

f1_keras = f1_score(y_test_enc, preds_keras, average='macro')
f1_manual = f1_score(y_test_enc, preds_manual, average='macro')

print(f'F1 macro Keras : {f1_keras:.4f}')
print(f'F1 macro Manual : {f1_manual:.4f}')
print(f'Predictions identical rate: {np.mean(preds_keras == preds_manual):.3f}')
```

F1 macro Keras : 0.6352
F1 macro Manual : 0.6352
Predictions identical rate: 1.000

Generate Code Markdown

Gambar 2.2.3.1. Cuplikan notebook perbandingan *forward propagation* manual dengan Keras

BAB 3: Kesimpulan dan Saran

3.1. Kesimpulan

Eksperimen pada arsitektur CNN menunjukkan bahwa peningkatan kedalaman jaringan dan jumlah filter per layer dapat meningkatkan performa model secara bertahap. Model dengan 2–3 layer Conv2D dan jumlah filter menengah (64–128) memberikan hasil terbaik, karena mampu mengekstraksi fitur spasial secara lebih kaya. Namun, penggunaan kernel yang terlalu besar atau filter yang berlebihan justru menurunkan akurasi karena overfitting atau kehilangan detail lokal penting.

Pada arsitektur RNN, penambahan jumlah layer memberikan peningkatan performa awal, tetapi cenderung menurun saat jumlah layer terlalu banyak. Model 2–3 layer memberikan hasil terbaik dalam konteks sederhana, sedangkan penambahan neuron secara berlebihan (misalnya 256 unit) justru menyebabkan overfitting. Penggunaan bidirectional RNN terbukti secara signifikan meningkatkan pemahaman konteks urutan dibandingkan unidirectional.

Untuk arsitektur LSTM, model 1–2 layer dengan jumlah unit sedang (128) menghasilkan F1-score tertinggi. LSTM secara umum lebih stabil dibanding RNN dalam memproses data sekuensial karena kemampuannya mempertahankan informasi jangka panjang. Penambahan unit atau kedalaman yang berlebihan tidak selalu berdampak positif dan bahkan dapat menyebabkan stagnasi val loss atau overfitting jika tidak disertai teknik regularisasi.

3.2. Saran

Sebagai langkah pengembangan model untuk data citra maupun sekuens, disarankan memulai dari arsitektur yang sederhana terlebih dahulu sambil menyesuaikan kompleksitas model dengan kapasitas dan karakteristik data. Penambahan kedalaman, jumlah unit, atau penggunaan bidirectional layer sebaiknya dilakukan secara bertahap dan disertai dengan mekanisme regularisasi seperti dropout. Selain itu, pemantauan melalui grafik loss serta evaluasi menggunakan metrik evaluasi seperti F1-score untuk klasifikasi penting untuk menilai efektivitas dan kemampuan generalisasi model secara menyeluruh.

BAB 4: Pembagian Tugas

Tabel 4.1. Pembagian tugas masing-masing anggota kelompok

Nama	NIM	Deskripsi Tugas
Elbert Chailes	13522045	Pembuatan model Simple RNN
Farel Winalda	13522047	Pembuatan model CNN
Derwin Rustanly	13522115	Pembuatan model CNN dan LSTM

Referensi

- Gomez, Aidan. "Backpropogating an LSTM: A Numerical Example | by Aidan Gomez." *Medium*, 18 April 2016, <https://medium.com/@aidangomez/let-s-do-this-f9b699de31d9>. Accessed 27 May 2025.
- Gupta, Dishashree. "Fundamentals of RNN forward Propagation in Deep Learning." *Analytics Vidhya*, 8 1 2025, <https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/>. Accessed 27 5 2025.
- Lepe, Edwin, and Andres Vazquez. "A Beginner's Tutorial For CNN." *Scribd*, Wong Chi Liek, 21 6 2017, <https://id.scribd.com/document/411788424/A-Beginner-s-Tutorial-for-CNN>. Accessed 27 5 2025.