

**Laporan Tugas Besar 2**  
**IF3270 Pembelajaran Mesin**



**Kelompok 39:**

Dzaky Satrio Nugroho	13522059
Rafiki Prawhira Harianto	13522065
Konstan Aftop Anewata Ndruru	12822058

## **Deskripsi Persoalan**

Tugas Besar 2 pada kuliah IF3270 Pembelajaran Mesin agar peserta kuliah mendapatkan wawasan tentang bagaimana cara mengimplementasikan forward propagation CNN, Simple RNN, dan LSTM. Pada tugas ini, peserta kuliah akan ditugaskan untuk mengimplementasikan forward propagation CNN, Simple RNN, dan LSTM *from scratch*.

# Pembahasan

## Penjelasan implementasi

### Deskripsi kelas beserta deskripsi atribut dan methodnya

#### CNN

```
import numpy as np
from tqdm import tqdm

def conv2d_forward(input_data, kernel, bias, strides=(1, 1), padding=0):
    N, H_in, W_in, C_in = input_data.shape
    K_H, K_W, _, C_out = kernel.shape
    S_H, S_W = strides

    if padding == 0:
        P_H, P_W = 0, 0
        H_out = (H_in - K_H) // S_H + 1
        W_out = (W_in - K_W) // S_W + 1
    else:
        raise ValueError("Belum mendukung Padding!")
    output_tensor = np.zeros((N, H_out, W_out, C_out))
    for n in tqdm(range(N)):
        for h_out in range(H_out):
            for w_out in range(W_out):
                h_start = h_out * S_H
                h_end = h_start + K_H
                w_start = w_out * S_W
                w_end = w_start + K_W
                input_patch = input_data[n, h_start:h_end, w_start:w_end, :]
                for c_out in range(C_out):
                    current_kernel = kernel[:, :, :, c_out]
                    current_bias = bias[c_out]
                    conv_sum = np.sum(input_patch * current_kernel)
                    output_tensor[n, h_out, w_out, c_out] = relu(conv_sum + current_bias)

    return output_tensor
```

```

def max_pooling2d_forward(input_data, pool_size=(2, 2), strides=None, padding=0):
    N, H_in, W_in, C_in = input_data.shape
    P_H, P_W = pool_size

    if strides is None:
        S_H, S_W = P_H, P_W
    else:
        S_H, S_W = strides

    if padding == 0:
        H_out = (H_in - P_H) // S_H + 1
        W_out = (W_in - P_W) // S_W + 1
        padded_input = input_data
    else:
        raise ValueError("Belum mendukung Padding!")

    output_tensor = np.zeros((N, H_out, W_out, C_in))

    for n in tqdm(range(N)):
        for c in range(C_in):
            for h_out in range(H_out):
                for w_out in range(W_out):
                    h_start = h_out * S_H
                    h_end = h_start + P_H
                    w_start = w_out * S_W
                    w_end = w_start + P_W
                    input_patch = padded_input[n, h_start:h_end, w_start:w_end, c]
                    max_val = np.max(input_patch)
                    output_tensor[n, h_out, w_out, c] = max_val

    return output_tensor

```

```

def flatten(pooled):
    batch_size = pooled.shape[0]
    output_tensor = pooled.reshape(batch_size, -1)
    return output_tensor

def relu(x):
    return np.maximum(0, x)

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def softmax(x):
    exp_x = np.exp(x)
    return exp_x / np.sum(exp_x, axis=-1, keepdims=True)

def tanh(x):
    return np.tanh(x)

def dense_forward(input_data, kernel, bias, activation_fn=None):
    z = np.dot(input_data, kernel)
    z_plus_bias = z + bias
    if activation_fn is not None:
        output_tensor = activation_fn(z_plus_bias)
    else:
        output_tensor = z_plus_bias
    return output_tensor

```

CNN menggunakan berbagai fungsi-fungsi merepresentasikan layernya.

Berikut penjelasannya:

### **1. conv2d\_forward (input\_data, kernel, bias, stride=(1,1), padding=0)**

Fungsi **conv2d\_forward** melakukan operasi konvolusi 2D dengan menerapkan filter (kernel) pada data masukan, seperti *batch* gambar, digeser dengan *stride* tertentu tanpa *padding*. Untuk setiap posisi *kernel*, ia menghitung perkalian *element-wise* antara *patch* masukan dan *kernel* terkait, menjumlahkannya dengan *bias*, dan menerapkan aktivasi ReLU. Proses ini menghasilkan tensor keluaran yang berisi fitur-fitur yang diekstraksi dari masukan.

## 2. **max\_pooling2d\_forward(input\_data, pool\_size=(2, 2), strides=None, padding=0)**

Fungsi **max\_pooling2d\_forward** melakukan operasi Max Pooling 2D yang bertujuan mengurangi dimensi spasial data masukan, seperti *feature maps*, untuk efisiensi dan robustansi fitur. Dengan menggeser jendela *pooling* (*pool\_size*) sesuai *stride* yang ditentukan, fungsi ini mencari nilai maksimum dari setiap *patch* masukan. Nilai maksimum ini kemudian ditempatkan ke dalam tensor keluaran, menghasilkan *feature maps* yang lebih ringkas.

## 3. **flatten(pooled)**

Meratakan tensor masukan sebelum dimasukkan ke dense layer.

## 4. **relu(x), sigmoid(x), softmax(x), tanh(x)**

Fungsi aktivasi yang bisa digunakan.

## 5. **dense\_forward(input\_data, kernel, bias, activation\_fn=None)**

Fungsi **dense\_forward** melakukan *forward pass* untuk sebuah *layer fully connected* (Dense Layer). Ia menghitung hasil perkalian dot antara **data masukan** dan **bobot** (*kernel*), kemudian menambahkan **bias**. Terakhir, jika ada, fungsi aktivasi yang ditentukan akan diterapkan pada hasil akhir.

## Simple RNN

```
import numpy as np

def embedding_forward(token_indices, embedding_matrix):
    return embedding_matrix[token_indices]

def simple_rnn_forward(inputs, Wx, Wh, b):
    hidden_size = Wh.shape[0]
    h_t = np.zeros((hidden_size,)) # Initial hidden state

    for t in range(inputs.shape[0]): # Iterate over sequence length
```

```

    h_t = np.tanh(np.dot(inputs[t], Wx) + np.dot(h_t, Wh) + b)

    return h_t

def bidirectional_rnn_forward(inputs, Wx_f, Wh_f, b_f, Wx_b, Wh_b, b_b):
    hidden_size = Wh_f.shape[0]
    h_f = np.zeros((hidden_size,))
    h_b = np.zeros((hidden_size,))

    for t in range(inputs.shape[0]):
        h_f = np.tanh(np.dot(inputs[t], Wx_f) + np.dot(h_f, Wh_f) + b_f)
        h_b = np.tanh(np.dot(inputs[-(t+1)], Wx_b) + np.dot(h_b, Wh_b) + b_b)

    return np.concatenate([h_f, h_b], axis=0)

# def dropout_forward(inputs, dropout_rate):
#     mask = np.random.binomial(1, 1 - dropout_rate, size=inputs.shape)
#     return inputs * mask / (1 - dropout_rate)

def dense_forward(inputs, W_dense, b_dense):
    return np.dot(inputs, W_dense) + b_dense

def softmax(logits):
    exp_logits = np.exp(logits - np.max(logits)) # Stability trick
    return exp_logits / np.sum(exp_logits)

```

RNN menggunakan berbagai fungsi-fungsi merepresentasikan layernya. Berikut penjelasannya:

### **1. `embedding_forward(token_indices, embedding_matrix)`**

Mengambil representasi vektor (embedding) dari indeks token berdasarkan matriks embedding.

### **2. `simple_rnn_forward(inputs, Wx, Wh, b)`**

Melakukan forward pass pada Simple RNN untuk satu urutan input.

### **3. `bidirectional_rnn_forward(inputs, Wx_f, Wh_f, b_f, Wx_b, Wh_b, b_b)`**

Melakukan forward pass pada Bidirectional RNN (dua arah). `_f` menyatakan forward, `_b` menyatakan backward.

#### 4. **dense\_forward(inputs, W\_dense, b\_dense)**

Melakukan operasi dense (fully connected layer). Digunakan juga terutama pada forward propagation FFNN.

#### 5. **dropout\_forward(inputs, dropout\_rate)**

Melakukan dropout berdasarkan dropout\_rate. Dropout tidak pakai untuk inference, sehingga di-comment.

#### 6. **softmax(logits)**

Menghitung softmax dari array logits untuk menghasilkan probabilitas.

## LSTM

```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-np.clip(x, -500, 500)))

def tanh(x):
    return np.tanh(np.clip(x, -500, 500))

def softmax(Logits):
    exp_Logits = np.exp(Logits - np.max(Logits))
    return exp_Logits / np.sum(exp_Logits)

def lstm_forward(inputs, Wf, Uf, bf, Wi,Ui, bi, Wo, Uo, bo, Wc, Uc, bc):

    seq_len, input_dim = inputs.shape
    hidden_size = Wf.shape[1]

    # Inisialisasi states
    h_t = np.zeros((hidden_size,))
    c_t = np.zeros((hidden_size,))

    all_hidden_states = []
    all_cell_states = []
```

```

for t in range(seq_len):
    x_t = inputs[t]

    # 1. Forget Gate
    #  $f_t = \sigma(W_f * x_t + U_f * h_{t-1} + b_f)$ 
    f_t = sigmoid(np.dot(x_t, wf) + np.dot(h_t, uf) + bf)

    # 2. Input Gate
    #  $i_t = \sigma(W_i * x_t + U_i * h_{t-1} + b_i)$ 
    i_t = sigmoid(np.dot(x_t, wi) + np.dot(h_t, ui) + bi)

    # 3. Candidate Values
    #  $\tilde{c}_t = \tanh(W_c * x_t + U_c * h_{t-1} + b_c)$ 
    c_tilde = tanh(np.dot(x_t, wc) + np.dot(h_t, uc) + bc)

    # 4. Update Cell State
    #  $c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$ 
    c_t = f_t * c_t + i_t * c_tilde

    # 5. Output Gate
    #  $o_t = \sigma(W_o * x_t + U_o * h_{t-1} + b_o)$ 
    o_t = sigmoid(np.dot(x_t, wo) + np.dot(h_t, uo) + bo)

    # 6. Update Hidden State
    #  $h_t = o_t \circ \tanh(c_t)$ 
    h_t = o_t * tanh(c_t)

    # Store states
    all_hidden_states.append(h_t.copy())
    all_cell_states.append(c_t.copy())

return h_t, np.array(all_hidden_states), np.array(all_cell_states)

```

```

def bidirectional_lstm_forward(inputs, wf_f, uf_f, bf_f, wi_f, ui_f, bi_f,
                               wo_f, uo_f, bo_f, wc_f, uc_f, bc_f, wf_b, uf_b, bf_b, wi_b, ui_b, bi_b, wo_b,
                               uo_b, bo_b, wc_b, uc_b, bc_b):
    # Forward LSTM
    h_f, _, _ = lstm_forward(inputs, wf_f, uf_f, bf_f, wi_f, ui_f, bi_f, wo_f,

```

```

o_f, bo_f, wc_f, uc_f, bc_f)
```

```

# Backward LSTM
inputs_reversed = inputs[::-1]
h_b, _, _ = lstm_forward(inputs_reversed, wf_b, uf_b, bf_b, wi_b, ui_b,
bi_b, wo_b, uo_b, bo_b, wc_b, uc_b, bc_b)

return np.concatenate([h_f, h_b], axis=0)
```

```

def embedding_forward(token_indices, embedding_matrix):
    return embedding_matrix[token_indices]
```

```

def dense_forward(inputs, w_dense, b_dense):
    return np.dot(inputs, w_dense) + b_dense
```

RNN menggunakan berbagai fungsi-fungsi merepresentasikan layernya. Berikut penjelasannya:

### **1. sigmoid(x)**

Fungsi untuk menghitung nilai sigmoid dari suatu nilai x, fungsi sigmoid adalah :

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

### **2. tanh(x)**

Fungsi untuk menghitung nilai sigmoid dari suatu nilai x, fungsi tanh adalah :

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

### **3. softmax(logits)**

Menghitung softmax dari array logits untuk menghasilkan probabilitas.

### **4. lstm\_forward(inputs, Wf, Uf, bf, Wi, Ui, bi, Wo, Uo, bo, Wc, Uc, bc)**

Melakukan forward propagation pada LSTM untuk satu ukuran inputs

**5. `bidirectional_lstm_forward(inputs, Wf_f, Uf_f, bf_f, Wi_f, Ui_f, bi_f, Wo_f, Uo_f, bo_f, Wc_f, Uc_f, bc_f, Wf_b, Uf_b, bf_b, Wi_b, Ui_b, bi_b, Wo_b, Uo_b, bo_b, Wc_b, Uc_b, bc_b)`**

Melakukan forward pass pada Bidirectional LSTM (dua arah). `_f` menyatakan forward, `_b` menyatakan backward.

**6. `embedding_forward(token_indices, embedding_matrix)`**

Mengambil representasi vektor (embedding) dari indeks token berdasarkan matriks embedding.

**7. `dense_forward(inputs, W_dense, b_dense)`**

Melakukan operasi dense (fully connected layer). Digunakan juga terutama pada forward propagation FFNN.

## Penjelasan forward propagation

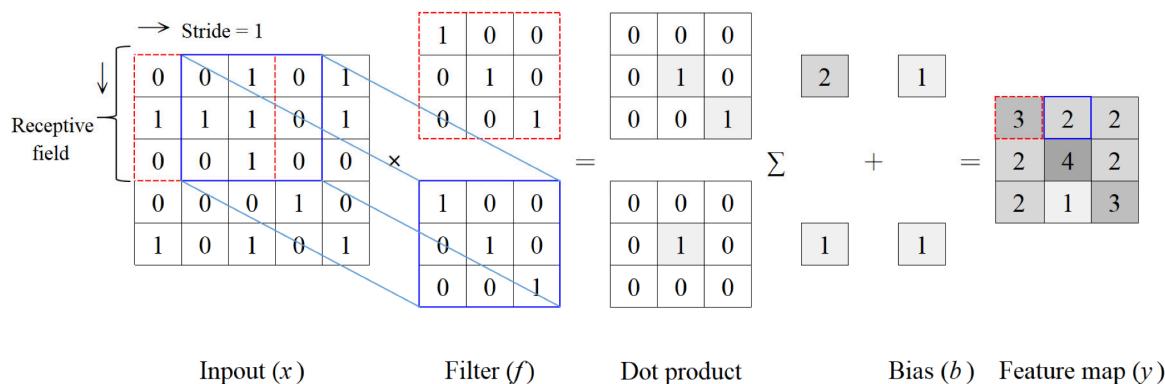
### CNN

**Layer Konvolusi** (`conv2d_forward`) bekerja seperti "detektor pola" yang menggeser sebuah **filter (kernel)** melintasi data masukan. Setiap filter mendeteksi fitur spesifik (misalnya, garis horizontal atau vertikal). Untuk setiap posisi, filter melakukan perkalian *element-wise* dengan *patch* masukan, menjumlahkan hasilnya di semua *channel* masukan, menambahkan satu nilai **bias** unik untuk *output channel* tersebut, lalu menerapkan **fungsi aktivasi** (seperti ReLU). Jumlah **output channel** pada dasarnya adalah jumlah kernel yang digunakan.

Setelah konvolusi, **Layer Pooling** (`max_pooling2d_forward`) sering digunakan untuk mengurangi dimensi spasial (*height* dan *width*) dari *feature map* yang dihasilkan. Ini membantu mengurangi komputasi dan membuat model lebih tangguh terhadap variasi posisi fitur. Max Pooling, misalnya, memilih nilai

maksimum dari setiap jendela yang bergeser, sehingga hanya fitur paling menonjol yang diteruskan.

Terakhir, **Layer Flatten** (*flatten*) mengubah *feature map* multidimensi menjadi vektor satu dimensi. Ini penting agar data dapat dimasukkan ke **Layer Dense** (*dense\_forward*), yang merupakan *layer fully connected* standar. *Layer Dense* kemudian melakukan klasifikasi atau regresi akhir berdasarkan fitur-fitur yang telah diekstraksi dan diratakan tersebut, dengan menerapkan perkalian *dot* antara masukan dan bobot, penambahan *bias*, dan aplikasi fungsi aktivasi akhir.



Sumber : Fan & Chung (2022)

## Simple RNN

Simple RNN adalah modifikasi dari FFNN yang menambahkan dimensi waktu, sehingga dapat memproses data berurutan (sequential data), seperti teks atau data yang berdimensi waktu. Pada Simple RNN, hidden state pada waktu saat ini ( $h_t$ ) dihitung berdasarkan input saat ini ( $x_t$ ) dan hidden state sebelumnya ( $h_{t-1}$ ) menggunakan fungsi aktivasi tanh sebagai berikut.

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

Di mana:

- $x_t$  = input pada waktu ke-t
- $h_{t-1}$  = hidden state pada waktu sebelumnya
- $W_x$  = bobot dari input ke hidden state
- $W_h$  = bobot dari hidden state sebelumnya ke hidden state saat ini
- $b$  = bias
- $\tanh$  = fungsi aktivasi hyperbolic tangent, digunakan untuk menjaga nilai state tetap dalam kisaran  $[-1,1]$

Selanjutnya, terdapat layer Dense yang menghasilkan output  $y_t$  pada setiap t yang dihitung menggunakan aktivasi softmax (bisa juga linear), sebagai berikut.

$$y_t = \text{softmax}(W_{\text{dense}} h_t + b_{\text{dense}})$$

Di mana:

- $W_{\text{dense}}$  = bobot dari hidden state ke output
- $b_{\text{dense}}$  = bias output

(Catatan: dalam materi kuliah,  $W_x = U$ ,  $W_h = W$ ,  $W_{\text{dense}} = V$ ,  $b = b_{xh}$ , dan  $b_{\text{dense}} = b_{hy}$ )

## LSTM

LSTM adalah modifikasi dari RNN yang menutupi kekurangan dari RNN yaitu *Vanishing Gradient Problem* dimana gradien dihitung dari keluaran ke masukan dengan rantai turunan. Jika fungsi aktivasi seperti sigmoid atau tanh digunakan, turunan biasanya berada antara 0 dan 1. Saat gradien ini dikalikan berulang kali dalam urutan waktu yang panjang, hasilnya menjadi semakin kecil (mendekati nol). Akibatnya, Model hanya mampu mengingat informasi jangka pendek. LSTM mengatasi hal tersebut dengan *cell state* dan

3 gerbang yang mengatur aliran informasi. Berikut adalah cara *forward propagation* di LSTM :

- Forget Gate – Tentukan apa yang harus dilupakan:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- Input Gate – Tentukan apa yang akan ditambahkan:

$$i_t = \sigma(W_i \cdot [h_{t-1}, X_t] + b_i)$$

- Candidate Cell State – Hitung informasi baru yang akan ditambahkan ke memori:

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

- Update Cell State:

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

- Output Gate – Tentukan bagian mana dari cell state yang menjadi output:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

- Hitung Hidden State (Output):

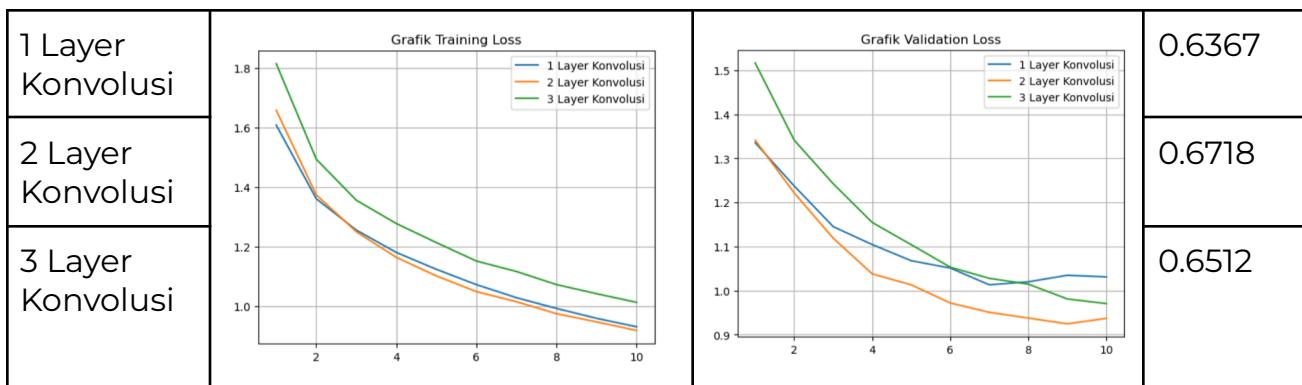
$$h_t = o_t * \tanh(C_t)$$

## Hasil pengujian

### CNN

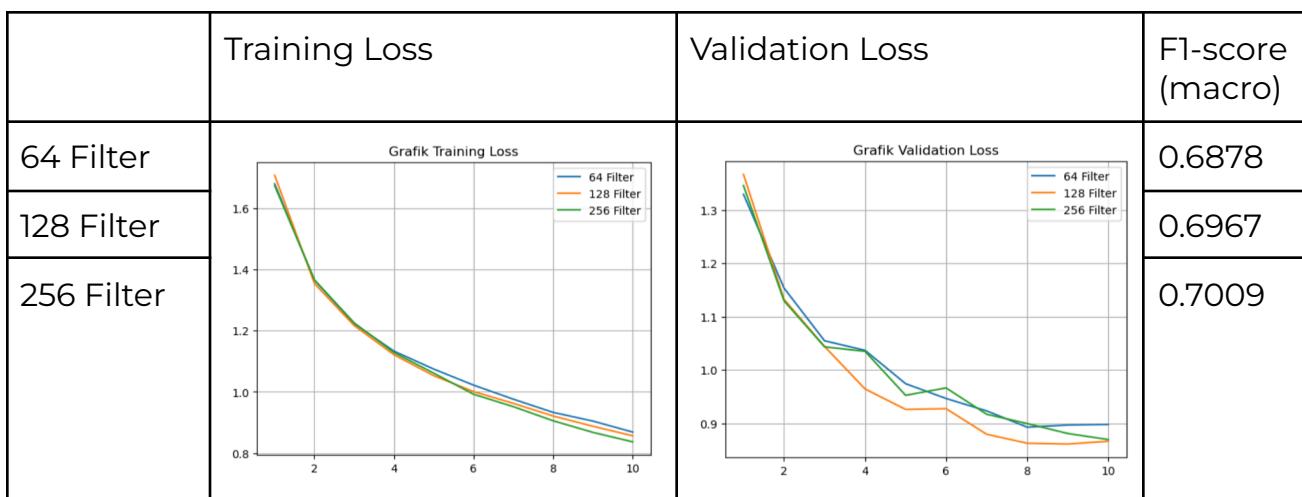
#### Pengaruh jumlah layer konvolusi

	Training Loss	Validation Loss	F1-score (macro)



Dalam implementasi CNN menggunakan TensorFlow, model dengan **2 layer konvolusi** menunjukkan performa terbaik, mencapai Macro F1 Score tertinggi sebesar **0.6718** dan *validation loss* terendah dan paling stabil. Meskipun model dengan 1 *layer* juga performanya baik, ia sedikit kalah, sementara model dengan 3 *layer* konvolusi, meskipun lebih kompleks, cenderung mengalami *overfitting* yang terlihat dari *validation loss* dan Macro F1 Score yang lebih rendah dibandingkan dengan model 2 *layer*.

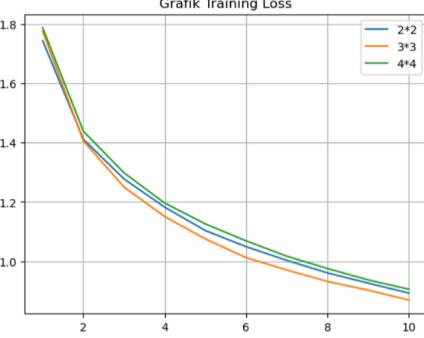
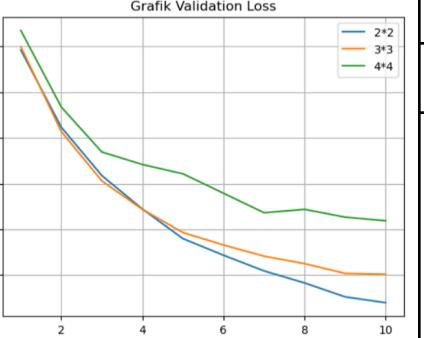
### Pengaruh banyak filter per layer konvolusi



Dalam pengujian model CNN, variasi jumlah filter pada *layer* konvolusi menunjukkan bahwa model dengan 256 filter mencapai kinerja terbaik

dengan Macro *F1 Score* tertinggi 0.7009. Meskipun semua konfigurasi filter menunjukkan penurunan *training* dan *validation loss*, model 256 filter secara konsisten menunjukkan *validation loss* yang paling rendah di epoch akhir, mengungguli konfigurasi 64 filter (0.6878) dan 128 filter (0.6967), yang menunjukkan kemampuan generalisasi yang lebih baik.

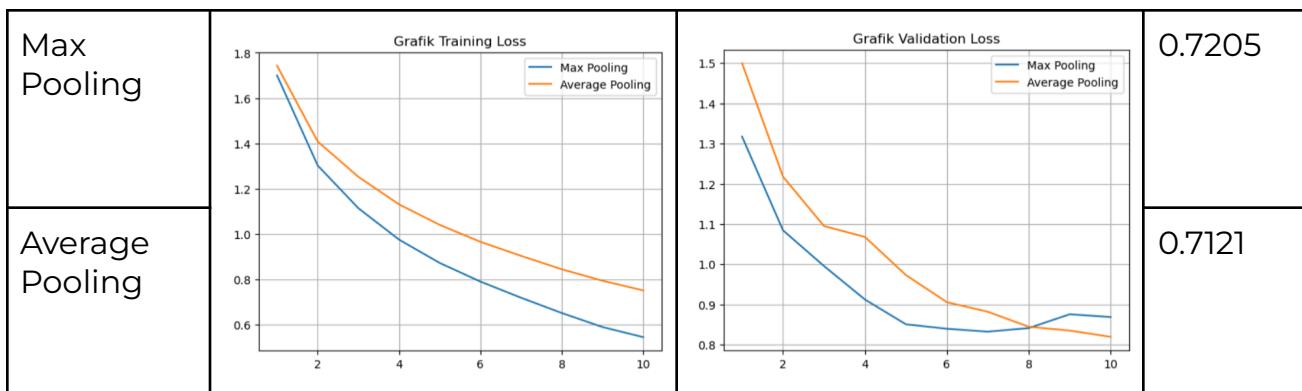
### Pengaruh ukuran filter per layer konvolusi

	Training Loss	Validation Loss	F1-score (macro)
2*2			0.6936
3*3			0.6779
4*4			0.6419

Dalam analisis variasi ukuran *kernel* pada CNN, model dengan **ukuran kernel 2x2** menunjukkan performa paling unggul dengan Macro *F1 Score* tertinggi **0.6936** dan *validation loss* terendah. Meskipun semua konfigurasi *kernel* menunjukkan penurunan *training loss*, ukuran *kernel* 3x3 dan 4x4 menghasilkan Macro *F1 Score* yang lebih rendah (0.6779 dan 0.6419 secara berurutan) serta *validation loss* yang lebih tinggi, mengindikasikan bahwa *kernel* yang lebih kecil (2x2) lebih efektif dalam mengekstraksi fitur dan menggeneralisasi pada dataset ini.

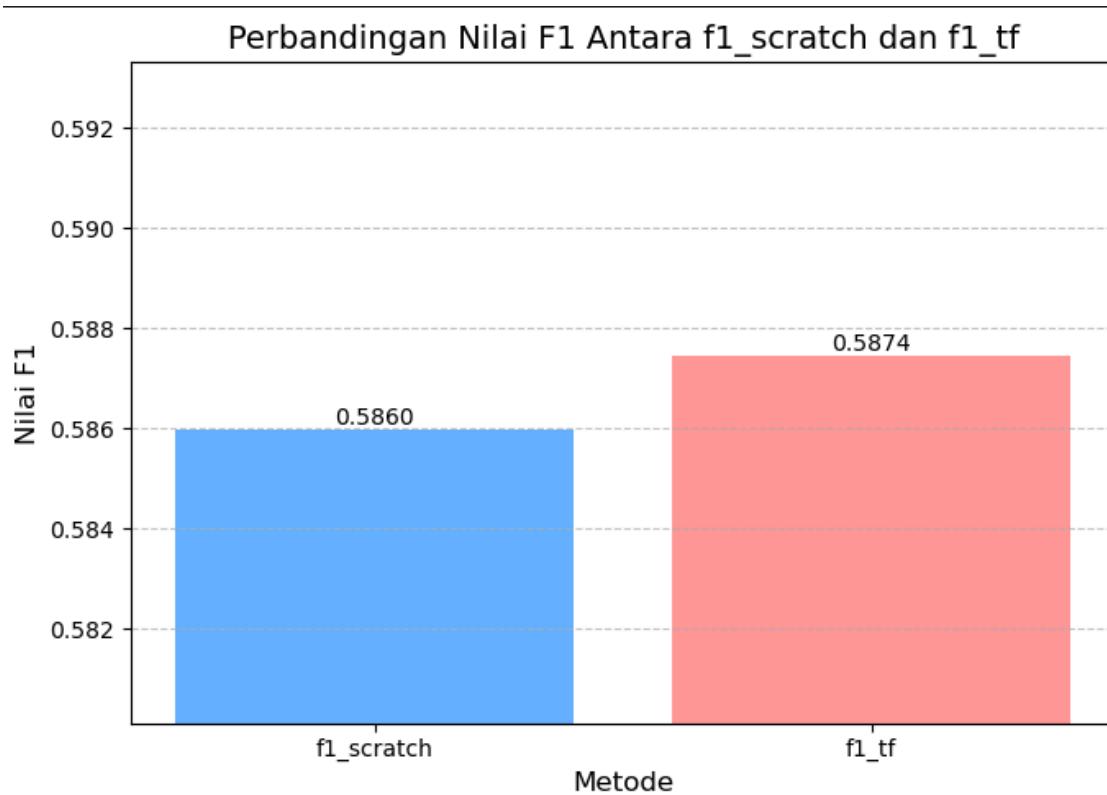
### Pengaruh jenis pooling layer

	Training Loss	Validation Loss	F1-score (macro)



Dalam perbandingan jenis *pooling* pada CNN, **Max Pooling** menunjukkan kinerja yang lebih superior dengan *Macro F1 Score* **0.7205** dan *validation loss* yang lebih rendah serta stabil di epoch akhir, mengungguli *Average Pooling* yang hanya mencapai *Macro F1 Score* 0.7121. Meskipun kedua metode *pooling* berhasil menurunkan *training loss*, efektivitas Max Pooling dalam mempertahankan fitur paling dominan terbukti lebih baik untuk generalisasi pada dataset ini.

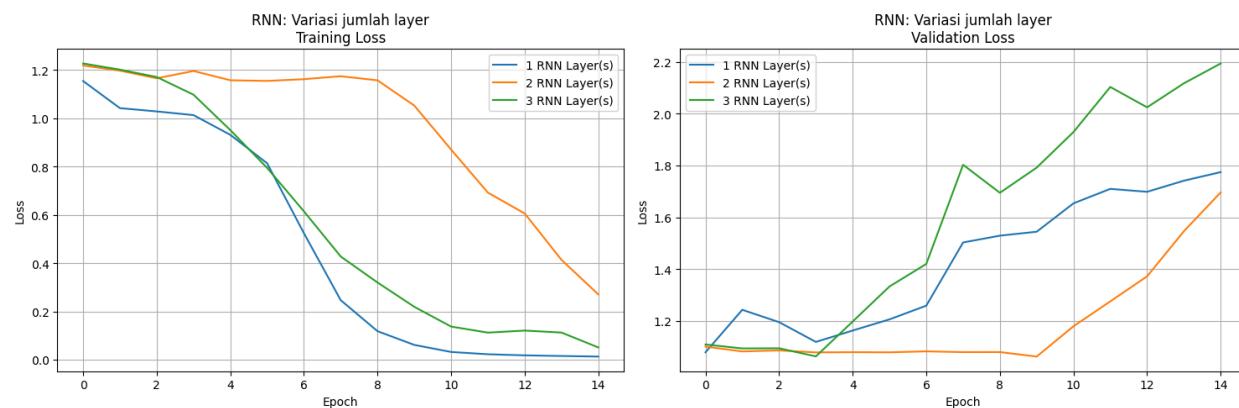
#### **Perbandingan hasil forward propagation from scratch dengan hasil forward propagation menggunakan Keras.**



Berdasarkan gambar di atas, terlihat bahwa f1\_score dari implementasi scratch forward propagation cnn memiliki **performa hampir sama** dengan library tensorflow. Perbedaan ini bisa jadi disebabkan karena, **perbedaan presisi perhitungan** antara Tensorflow dengan Scratch.

## Simple RNN

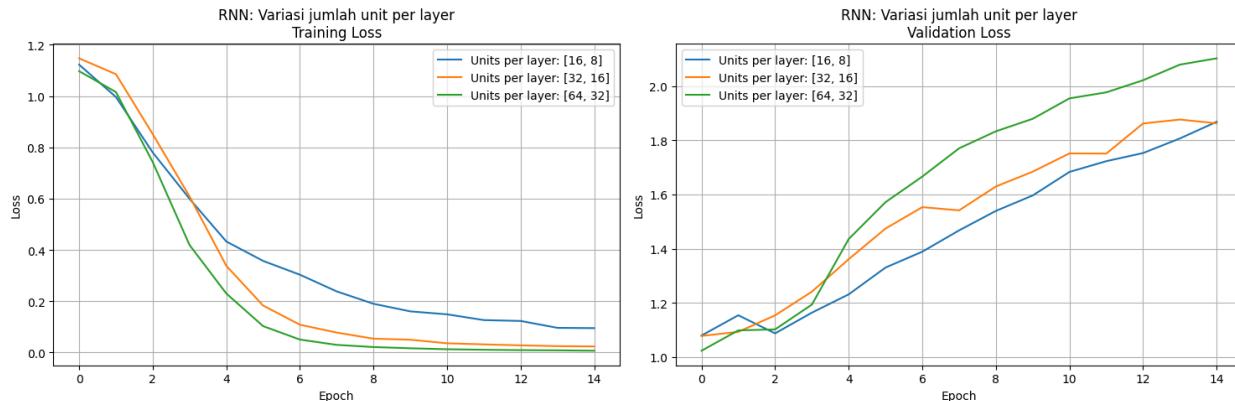
### - Pengaruh jumlah layer RNN



Macro F1-Score (1|2|3 Layer): 0.4413 | 0.3185 | 0.4498

Terlihat bahwa pada hasil testing, loss training menurun dengan loss validation menaik, sehingga terjadi overfitting, kecuali pada 2 layer RNN hingga epoch ke-8. Model terbaik secara score macro F1 dan validation loss berada pada 2 layer RNN, dan dapat lebih optimal lagi jika model diberhentikan di epoch ke-8.

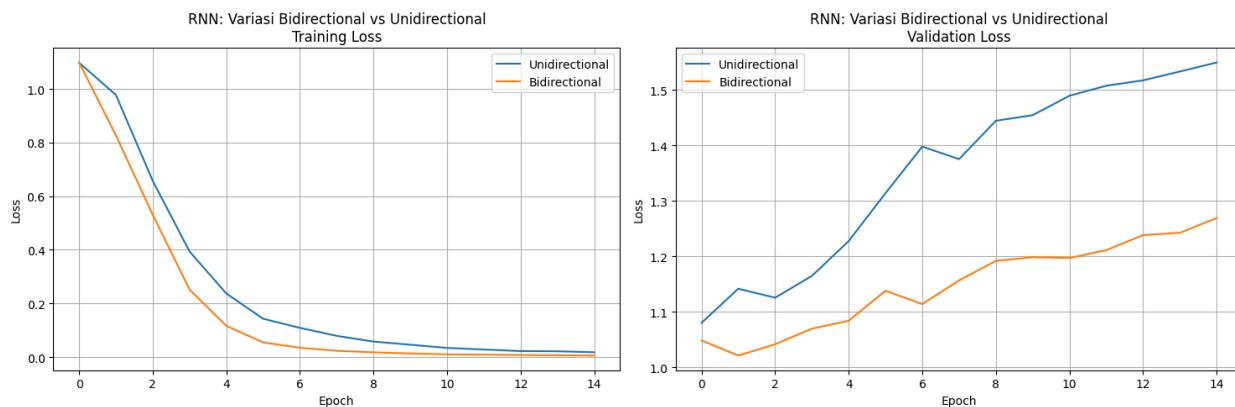
### - Pengaruh banyak cell RNN per layer



Macro F1-Score ([16, 8]||[32, 16]||[64, 32] Cell): 0.3178 | 0.3377 | 0.3561

Loss training menurun dengan loss validation menaik, sehingga terjadi overfitting. Model terbaik secara score macro F1 berada pada [64, 32], dan secara validation loss seri antara [16, 8] dan [32, 16]. ([A, B] menyatakan terdapat A unit pada layer RNN pertama dan B unit pada layer RNN kedua)

#### - Pengaruh jenis layer RNN berdasarkan arah



Macro F1-Score (Unidirectional|Bidirectional): 0.4420 | 0.4683

Loss training menurun dengan loss validation menaik, sehingga terjadi overfitting. Model terbaik secara score macro F1 dan validation loss berada pada Bidirectional.

#### - Perbandingan hasil forward propagation from scratch dengan hasil forward propagation menggunakan Keras

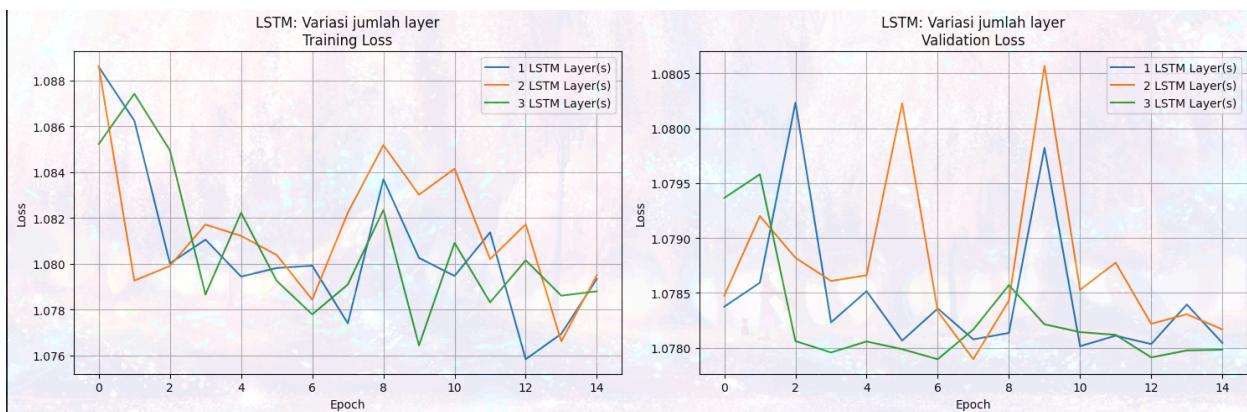
- PS D:\Rafiki-ITB\Kuliah\Semester 6\ML\tubes-2-ml\RNN\keras-reference.py"
   
4/4 0s 46ms/step
   
Macro F1\_Score: 0.44
   
4/4 0s 86ms/step
   
Macro F1\_Score (Bidirectional): 0.56

```
PS D:\Rafiki-ITB\Kuliah\Semester 6\ML\tubes-2-ml\RNN> & C:/Users/Rafik/L/tubes-2-ml/RNN/manual-forward.py"
● Manual Forward Macro F1_score: 0.44203000468823256
  Manual Forward (Bidirectional) Macro F1_score: 0.5555348516218082
```

Terdapat model RNN Unidirectional dan Bidirectional, dengan Manual forward menggunakan weight hasil training keras. Hasil Macro F1 Score model keras dengan manual forward (from scratch) sama, dengan model Unidirectional memiliki score 0.44 dan model Bidirectional memiliki score 0.56.

## LSTM

### - Pengaruh jumlah layer LSTM

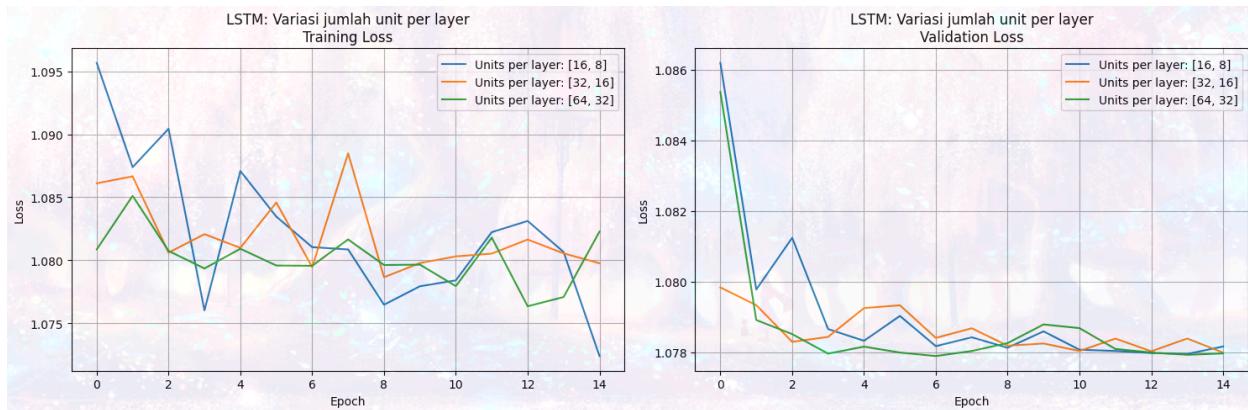


Macro F1-Score (1|2|3 Layer): 0.1836 | 0.1836 | 0.1836

Grafik menunjukkan bahwa penambahan jumlah layer LSTM tidak menurunkan nilai loss pada data pelatihan, namun pada data validasi, model dengan 3 layer LSTM memberikan hasil yang paling stabil dan rendah dibandingkan 1 atau 2 layer. Model dengan 1 layer cukup baik dan stabil,

sementara 2 layer justru menunjukkan fluktuasi yang tinggi dan potensi overfitting.

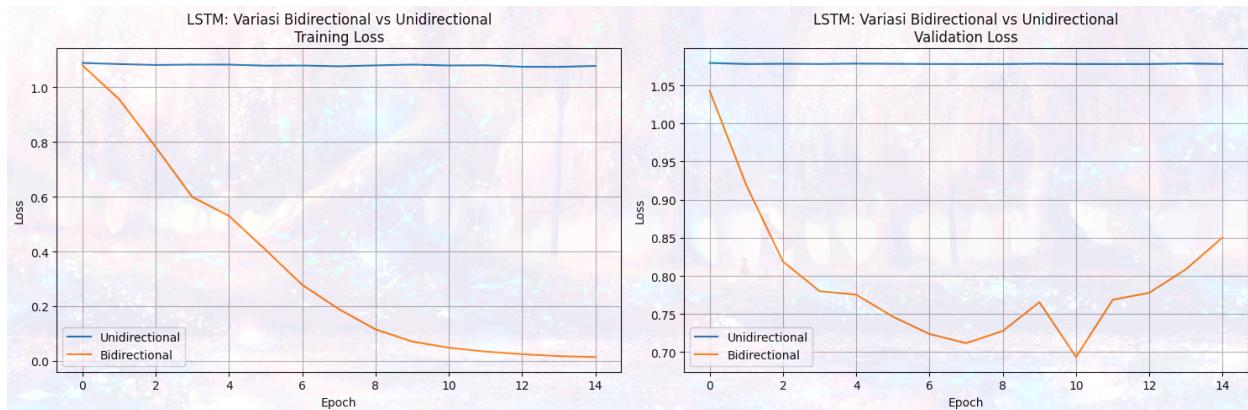
#### - Pengaruh banyak cell LSTM per layer



Macro F1-Score ([16, 8] | [32, 16] | [64, 32] Cell): 0.1836 | 0.1836 | 0.1836

Grafik menunjukkan bahwa peningkatan jumlah unit per layer pada LSTM tidak memberikan perbedaan signifikan terhadap nilai loss, baik pada data pelatihan maupun validasi. Ketiga konfigurasi—[16, 8], [32, 16], dan [64, 32]—memiliki pola penurunan loss yang serupa, dengan [64, 32] sedikit lebih konsisten dan stabil pada training loss, serta memiliki validation loss yang paling rendah dan stabil. Namun, selisih performa antar konfigurasi sangat kecil, sehingga penggunaan unit lebih besar hanya layak dipertimbangkan jika sumber daya komputasi mencukupi dan diperlukan kestabilan tambahan.

#### - Pengaruh jenis layer LSTM berdasarkan arah



Macro F1-Score (Unidirectional|Bidirectional): 0.4420 | 0.7636

Grafik menunjukkan bahwa model LSTM bidirectional memiliki performa yang jauh lebih baik dibandingkan dengan unidirectional, baik pada data pelatihan maupun validasi. Pada training loss, bidirectional LSTM mengalami penurunan loss yang sangat signifikan dan stabil, sedangkan unidirectional tetap hampir konstan dan tinggi. Demikian pula pada validation loss, bidirectional menunjukkan nilai yang jauh lebih rendah dibandingkan unidirectional, meskipun sedikit naik di akhir epoch (indikasi awal overfitting). Secara keseluruhan, penggunaan bidirectional LSTM sangat efektif dalam meningkatkan kualitas pembelajaran dan generalisasi model dibandingkan unidirectional.

- **Perbandingan hasil forward propagation from scratch dengan hasil forward propagation menggunakan Keras**

```
Manual Implementation:  
  Unidirectional F1: 0.2069  
  Bidirectional F1: 0.5573  
  
Keras Implementation:  
  Unidirectional F1: 0.1836  
  Bidirectional F1: 0.5817
```

Didapatkan hasil yang mirip untuk unidirectional maupun bidirectional

## **Kesimpulan dan Saran**

Dalam Tugas Besar2 ini, kami mempelajari cara untuk melakukan implementasi forward propagation dari nol untuk model CNN, RNN, dan LSTM. Selain itu juga dibandingkan hasilnya dengan model milik Keras. Didapatkan hasil yang dapat dilihat diatas.

Saran : sebaiknya mengerjakan bonus

## **Pembagian tugas tiap anggota kelompok**

Berikut adalah pembagian tugas tiap anggota kelompok:

Anggota	Tugas
Dzaky Satrio Nugroho 13522059	LSTM
Rafiki Prawhira Harianto 13522065	Simple RNN
Konstan Aftop Anewata Ndruru 12822058	CNN

## **Referensi**

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