Tugas Besar 2 IF3270 Pembelajaran Mesin
Convolutional Neural Network dan Recurrent Neural Network

Oleh:

13522121 - Jonathan Emmanuel Saragih

13522125 - Satriadhikara Panji Yudhistira

13522128 - Mohammad Andhika Fadillah

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

Daftar Isi

Bab 1

Deskripsi Persoalan

Pengimplementasian CNN, Simple RNN, LSTM from Scratch untuk menyelesaikan permasalahan berdasarkan dataset yang telah diberikan.

Untuk bagian Convolutional Neural Network:

Dilakukan pelatihan suatu model CNN untuk *image classification* dengan library Keras dan dengan dataset CIFAR-10 yang memenuhi ketentuan berikut ini.

- Model minimal harus memiliki jenis layer berikut didalamnya (urutan dan jumlah layer silakan disesuaikan sendiri):
  - Conv2D layer
  - Pooling layers
  - Flatten/Global Pooling layer
  - Dense layer
- Loss function yang digunakan adalah Sparse Categorical Crossentropy (untuk menangani kasus klasifikasi multikelas)
- 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)

Untuk Simple Recurrent Neural Network:

- Preprocessing data teks menjadi representasi numerik yang bisa diterima oleh model dengan tahap sebagai berikut:
  - Tokenization

    Tahap ini akan mengubah data teks menjadi bentuk list of tokens (integer), dimana teks akan dipecah-pecah menjadi bentuk satuannya yaitu *token* yang direpresentasikan sebagai suatu *integer*. Untuk tugas ini, Anda cukup memanfaatkan TextVectorization layer untuk memetakan input sequence menjadi list of tokens.
  - Embedding function

    Embedding function merupakan fungsi yang memetakan tiap token ke dalam suatu ruang vektor berdimensi-n, sehingga setiap token (yang sudah berbentuk vektor) dapat dioperasikan satu sama lain. Untuk tugas ini, Anda cukup memanfaatkan embedding layer yang disediakan oleh Keras untuk mengonversi token ke dalam bentuk vektor.
- Lakukan pelatihan untuk suatu model RNN untuk *text classification* dengan dataset NusaX-Sentiment (Bahasa Indonesia) dan dengan menggunakan Keras yang memenuhi ketentuan berikut ini.
  - Model minimal harus memiliki jenis layer-layer berikut didalamnya (urutan dan jumlah layer silakan disesuaikan sendiri):
    - Embedding layer

        - Bidirectional RNN layer dan/atau Unidirectional RNN layer
        - Dropout layer
        - Dense layer
    - Loss function yang digunakan adalah Sparse Categorical Crossentropy (untuk menangani kasus klasifikasi multikelas)
    - Optimizer yang digunakan adalah Adam

Untuk Long-Short Term Memory Network:

- Preprocessing data teks menjadi representasi numerik yang bisa diterima oleh model dengan tahap sebagai berikut:
    - Tokenization
      Tahap ini akan mengubah data teks menjadi bentuk list of tokens (integer), dimana teks akan dipecah-pecah menjadi bentuk satuannya yaitu *token* yang direpresentasikan sebagai suatu *integer*. Untuk tugas ini, Anda cukup memanfaatkan Text Vectorization layer untuk memetakan input sequence menjadi list of tokens.
    - Embedding function
      Embedding function merupakan fungsi yang memetakan tiap token ke dalam suatu ruang vektor berdimensi-n, sehingga setiap token (yang sudah berbentuk vektor) dapat dioperasikan satu sama lain. Untuk tugas ini, Anda cukup memanfaatkan embedding layer yang disediakan oleh Keras untuk mengonversi token ke dalam bentuk vektor.
- Lakukan pelatihan untuk suatu model LSTM untuk *text classification* dengan dataset NusaX-Sentiment (Bahasa Indonesia) dan dengan menggunakan Keras yang memenuhi ketentuan berikut ini.
    - Model minimal harus memiliki jenis layer-layer berikut didalamnya :
        - Embedding layer
        - Bidirectional LSTM layer dan/atau Unidirectional LSTM layer
        - Dropout layer
        - Dense layer
    - Loss function yang digunakan adalah Sparse Categorical Crossentropy (untuk menangani kasus klasifikasi multikelas)
    - Optimizer yang digunakan adalah Adam

Bab 2

Pembahasan

2.1 Penjelasan Implementasi

2.1.1 Deskripsi kelas beserta deskripsi atribut dan methodnya

layers.py (CNN)			
Kelas	deskripsi	atribut	method
Conv2D	Layer konvolusi 2D untuk ekstraksi fitur dari input gambar	weights, biases, stride, padding, activation, input_shape, output_shape, input_data	init(weights, biases, stride, padding, activation), _apply_padding(x), forward(x)
ReLU	Layer aktivasi ReLU untuk memperkenalkan non-linearitas	input_data	init(), forward(x)
MaxPooling2D	Layer max pooling untuk downsampling dengan mengambil nilai maksimum	pool_size, stride, input_data	init(pool_size, stride), forward(x)
AveragePooling2D	Layer average pooling untuk downsampling dengan mengambil nilai rata-rata	pool_size, stride, input_data	init(pool_size, stride), forward(x)
Flatten	Layer untuk meratakan tensor multi-dimensi menjadi 1D	input_shape	init(), forward(x)
Dense	Fully connected layer untuk transformasi linear	weights, biases, activation, input_data	init(weights, biases, activation), forward(x)
Softmax	Layer aktivasi softmax untuk menghasilkan distribusi probabilitas		forward(x)
model.py (CNN)			
CNNModelFromScratch	Model CNN lengkap yang menggabungkan berbagai	layers	init(layers), forward(x), predict(x)

	layer untuk klasifikasi gambar		
layers.py (RNN)			
Kelas	deskripsi	atribut	method
Embedding	Layer untuk mengubah token indeks menjadi vektor embedding	weights, vocab_size, embedding_dim	__init__(self, weights), forward(self, x)
SimpleRNN	Implementasi layer Recurrent Neural Network sederhana yang memproses sequence data secara berurutan. Menggunakan fungsi aktivasi tanh dan dapat mengembalikan output untuk semua timestep atau hanya timestep terakhir.	Cell, return_sequences, hidden_dim	__init__(self, weights_kernel, weights_recurrent, bias, return_sequences=False), forward(self, x)
Bidirectional	Layer yang menggabungkan dua RNN layer (forward dan backward) untuk memproses sequence dalam kedua arah.	forward_layer, backward_layer, return_sequences	__init__(self, forward_rnn, backward_rnn), forward(self, x)
Dropout	Layer yang secara acak "mematikan" beberapa neuron selama training untuk mencegah overfitting.	rate	__init__(self, rate=0.0), forward(self, x)
Dense	Layer fully connected (dense) yang melakukan transformasi linear pada input. Setiap neuron terhubung ke semua neuron di layer sebelumnya.	Weights, bias	__init__(self, weights, bias), forward(self, x)
Softmax	Layer aktivasi yang mengkonversi logits menjadi distribusi probabilitas. Setiap output bernilai antara 0-1 dan jumlah semua output sama dengan 1.	-	forward(self,x)
Flatten	Layer yang meratakan (flatten) input multidimensi	-	forward(self,x)

	menjadi 2D		
model.py (RNN)			
Kelas	deskripsi	atribut	method
RNNModelFromScratch	Mengimplementasikan model RNN lengkap dari scratch. Kelas ini menggabungkan berbagai layer untuk membentuk arsitektur RNN yang dapat melakukan forward pass, prediksi, dan menghitung parameter model.	Layers, total_params	__init__(layers), forward(x), predict(x), count_parameters(), summary() from_keras_model(keras_model_path, vectorizer=None) (classmethod), from_keras_weights(weights_path, model_config, vectorizer=None)
layers.py (LSTM)			
Kelas	deskripsi	atribut	method
Embedding	Layer untuk mengubah token indeks menjadi vektor embedding	weights, vocab_size, embedding_dim	forward(x)
LSTMCell	Sel LSTM tunggal yang memproses satu timestep input	weights_kernel, weights_recurrent, bias, hidden_dim	forward(x, h_prev, c_prev)
LSTM	LSTM layer yang memproses input sequence secara keseluruhan	cell, return_sequences, hidden_dim	forward(x)
Bidirectional	Layer LSTM dua arah (forward dan backward)	forward_layer, backward_layer, return_sequences	forward(x)
Dropout	Layer dropout sederhana tanpa efek pada inference	rate	forward(x)
Dense	Fully connected layer	weights, bias	forward(x)
Softmax	Fungsi aktivasi softmax	-	forward(x)
Flatten	Flatten layer untuk meratakan tensor 3D ke 2D	-	forward(x)
model.py (LSTM)			
Kelas	deskripsi	atribut	method
LSTMModelFromScratch	Model neural network berbasis LSTM yang	layers, total_params	__init__, forward(x), predict(x), count_parameters(),

	disusun dari kelas-kelas custom		from_keras_model(), from_keras_weights()
(fungsi) create_keras_model_from_config	Membuat arsitektur Keras berdasarkan konfigurasi yang diberikan untuk pelatihan awal	-	create_keras_model_from_config(config)

2.1.2 Penjelasan forward propagation

2.1.2.1 CNN

Forward Propagation dalam CNN merupakan proses sekuensial di mana data masukan diolah langkah demi langkah melalui serangkaian lapisan untuk menghasilkan keluaran. Setiap lapisan melakukan transformasi spesifik pada data yang diterimanya dari lapisan sebelumnya.

1. Pemrosesan Input dan Inisialisasi
   Forward Propagation dimulai ketika CNN menerima input dengan bentuk (*batch_size, height, width, channels*). Input ini merepresentasikan sekumpulan gambar dalam batch dengan dimensi spasial (tinggi dan lebar) serta jumlah *channel*. Model CNN kemudian memproses input ini secara berurutan melalui setiap layer yang telah didefinisikan dalam arsitektur.

```
def forward(self, x):
    for layer in self.layers:
        x = layer.forward(x)
    return x
```

2. Operasi Konvolusi (*Convolution Layer*)
   Pada layer konvolusi, operasi fundamental CNN berlangsung menggunakan filter/*kernel* yang bergeser melintasi input. Setiap filter melakukan operasi dot product dengan patch lokal dari input, menghasilkan *feature map*. Proses ini melibatkan *padding*, *stride*, maupun fungsi konvolusi itu sendiri.

```
for i in range(n_batch):
    for f_idx in range(n_filters):
        current_y = out_y = 0
        while current_y + fh <= h_padded:
            current_x = out_x = 0
            while current_x + fw <= w_padded:
```

```
                patch = x_padded[i,
current_y:current_y+fh, current_x:current_x+fw, :]
                output[i, out_y, out_x, f_idx] =
np.sum(patch * self.weights[:, :, :, f_idx]) +
self.biases[f_idx]
                current_x += self.stride
                out_x += 1
            current_y += self.stride
            out_y += 1
```

3. Aktivasi dan *Pooling*
   Setelah operasi konvolusi, fungsi aktivasi (seperti ReLU) diterapkan untuk memperkenalkan non-linearitas, $f(x) = max(0, x)$. Kemudian, layer pooling mengurangi dimensi spasial sambil mempertahankan informasi penting. *Max Pooling* mengambil nilai maksimum dari setiap patch, sedangkan *Average Pooling* mengambil nilai rata-rata dari setiap patch.

4. Flattening dan Dense Layers
   Setelah ekstraksi fitur melalui layer konvolusi dan pooling, output di-*flatten* menjadi vektor satu dimensi untuk diproses oleh fully *connected layers*. Dense layer melakukan transformasi linear: $y = Wx + b$ di mana W adalah *weight matrix* dan b adalah bias *vector*.

5. Output dan Prediksi
   Layer terakhir biasanya menggunakan aktivasi *softmax* untuk menghasilkan distribusi probabilitas atas kelas-kelas yang mungkin

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

   Prediksi akhir diperoleh dengan mengambil kelas dengan probabilitas tertinggi

```
def predict(self, x):
    output = self.forward(x)
    if not isinstance(last_layer, Softmax):
        softmax_layer = Softmax()
        probabilities = softmax_layer.forward(output)
    else:
        probabilities = output
    return np.argmax(probabilities, axis=1)
```

2.1.2.2 Simple RNN

Forward Propagation dalam RNN merupakan proses sekuensial yang memproses data input langkah demi langkah sambil mempertahankan memori melalui hidden state.

1. Pemrosesan Input dan Inisialisasi

   Forward Propagation dimulai ketika SimpleRNN menerima input dengan bentuk (batch_size, sequence_length, input_dim). Langkah pertama yaitu mengekstrak dimensi-dimensi tersebut dan menginisialisasi hidden state sebagai nol dengan bentuk (batch_size, hidden_dim). Jika return_sequences bernilai True, tambahan h_sequence dibuat untuk menyimpan semua hidden state selama pemrosesan sekuens.

```
def forward(self, x): batch_size, time_steps, input_dim = x.shape

    h = np.zeros((batch_size, self.hidden_dim))

    c = np.zeros((batch_size, self.hidden_dim))

    if self.return_sequences:

        h_sequence = np.zeros((batch_size, time_steps, self.hidden_dim))

    for t in range(time_steps):

        h, c = self.cell.forward(x[:, t, :], h, c)

        if self.return_sequences:

            h_sequence[:, t, :] = h

    if self.return_sequences:

        return h_sequence

    else:

        return h
```

2. Pemrosesan Langkah Waktu Sekuensial

   Perhitungan terjadi dalam sebuah loop yang beriterasi melalui setiap langkah waktu dari 0 hingga time_steps-1. Pada setiap langkah waktu t, algoritma mengekstrak potongan input saat ini x[:, t, :] yang merepresentasikan input untuk semua sampel dalam batch pada langkah waktu spesifik tersebut. Input

saat ini, bersama dengan hidden state sebelumnya, kemudian diteruskan ke sel RNN untuk pemrosesan.

```
for t in range(time_steps):
    h, c = self.cell.forward(x[:, t, :], h, c)
    if self.return_sequences:
        h_sequence[:, t, :] = h
```

3. Komputasi Sel RNN
   Di dalam sel RNN, perhitungan SimpleRNN berlangsung menggunakan rumus:

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

   Hal ini melibatkan tiga operasi matriks: pertama, input saat ini x_t dikalikan dengan bobot input-ke-hidden (weights_kernel). kedua, hidden state sebelumnya h_{t-1} dikalikan dengan bobot hidden-ke-hidden (weights_recurrent). ketiga, bias ditambahkan. Selanjutnya, fungsi aktivasi tanh diterapkan untuk menghasilkan hidden state baru h_t.

4. Pembaruan Memori dan State
   Setelah menghitung hidden state baru untuk langkah waktu saat ini, algoritma memperbarui variabel hidden state h dengan nilai baru ini. Jika return_sequences diaktifkan, hidden state yang dihitung ini juga disimpan dalam h_sequence pada posisi langkah waktu yang sesuai. Hidden state ini berfungsi sebagai mekanisme memori yang membawa informasi dari langkah waktu sebelumnya untuk mempengaruhi komputasi kedepannya.

5. Output
   Setelah semua langkah waktu telah diproses, metode forward prop mengembalikan output yang sesuai berdasarkan parameter return_sequences. Jika return_sequences bernilai True, ia mengembalikan urutan lengkap hidden state dengan bentuk (batch_size, sequence_length, hidden_dim), yang berguna untuk tugas sequence-to-sequence. Jika False, ia mengembalikan hanya hidden state terakhir dengan bentuk (batch_size, hidden_dim).

2.1.2.3 LSTM

Forward Propagation dalam RNN merupakan proses sekuensial yang memproses data input langkah demi langkah sambil mempertahankan memori melalui hidden state.

1. Pemrosesan Input dan Inisialisasi
   Forward propagation pada LSTM (Long Short-Term Memory) dimulai ketika model menerima input dengan bentuk (batch_size, sequence_length, input_dim). Input ini merupakan data sekuensial yang terdiri dari sejumlah urutan (sequence) per batch, dengan masing-masing urutan memiliki dimensi fitur tertentu.

```
  def forward(self, x, h_prev, c_prev):
      z = np.dot(x, self.weights_kernel) + np.dot(h_prev,
self.weights_recurrent) + self.bias

      z_i, z_f, z_c, z_o = np.split(z, 4, axis=1)

      i = sigmoid(z_i)
      f = sigmoid(z_f)
      c_temp = np.tanh(z_c)
      o = sigmoid(z_o)

      c_next = f * c_prev + i * c_temp
      h_next = o * np.tanh(c_next)

      return h_next, c_next
```

2. Pemrosesan Langkah Waktu Sekuensial
   Pemrosesan sekuensial dalam forward propagation dilakukan dengan menggunakan loop for yang beriterasi melalui setiap langkah waktu, dari t = 0 hingga t = time_steps - 1. Pada setiap langkah waktu t, input x[:, t, :] yang merepresentasikan input dari seluruh sampel dalam batch pada waktu tersebut, diproses bersama dengan hidden state (h) dan cell state (c) sebelumnya melalui metode forward dari LSTMCell.

```
  for t in range(time_steps):

      h, c = self.cell.forward(x[:, t, :], h, c)

      if self.return_sequences:
```

```
h_sequence[:, t, :] = h
```

3. Komputasi LSTM

Di dalam LSTMCell, komputasi utama dilakukan dengan menghitung nilai gabungan z, yang merupakan hasil dari perkalian matriks input x_t dengan bobot input (weights_kernel) dan hidden state sebelumnya h_prev dengan bobot rekuren (weights_recurrent), ditambah bias:

```
z = np.dot(x, self.weights_kernel) + np.dot(h_prev, self.weights_recurrent) + self.bias
```

Vektor z kemudian dibagi menjadi empat bagian yang merepresentasikan empat gerbang dalam LSTM:

1. Input Gate (i)
2. Forget Gate (f)
3. Cell Candidate (ĉ)
4. Output Gate (o)

Setiap gerbang dihitung dengan aktivasi sigmoid atau tanh sesuai fungsinya:

```
i = sigmoid(z_i)
f = sigmoid(z_f)
c_temp = np.tanh(z_c)
o = sigmoid(z_o)
```

4. Pembaruan Memori dan State

Setelah menghitung nilai dari keempat gerbang pada sel LSTM, pembaruan memori dilakukan melalui dua komponen utama: cell state (c_t) dan hidden state (h_t). Cell state diperbarui dengan cara menggabungkan sebagian dari cell state sebelumnya yang dipertahankan oleh forget gate, dan sebagian informasi baru yang dikontrol oleh input gate serta cell candidate. Secara matematis, cell state baru dihitung menggunakan operasi c_t = f_t * c_{t-1} + i_t * ĉ_t. Kemudian, hidden state diperbarui dengan mengalirkan cell state yang telah diperbarui melalui fungsi aktivasi tanh, dan dikalikan dengan output gate, menghasilkan h_t = o_t * tanh(c_t). Hidden state ini membawa informasi penting yang akan digunakan pada timestep selanjutnya dalam proses sekuensial. Jika parameter return_sequences diaktifkan, hidden state dari setiap

timestep disimpan ke dalam h_sequence, sebuah array tiga dimensi yang menampung hasil h_t pada setiap langkah waktu untuk seluruh batch.

5. Output

Setelah semua langkah waktu telah diproses, metode forward mengembalikan output yang sesuai berdasarkan parameter return_sequences. Jika return_sequences bernilai True, ia mengembalikan urutan lengkap hidden state dengan bentuk (batch_size, sequence_length, hidden_dim), yang berguna untuk tugas sequence-to-sequence. Jika False, ia mengembalikan hanya hidden state terakhir dengan bentuk (batch_size, hidden_dim).

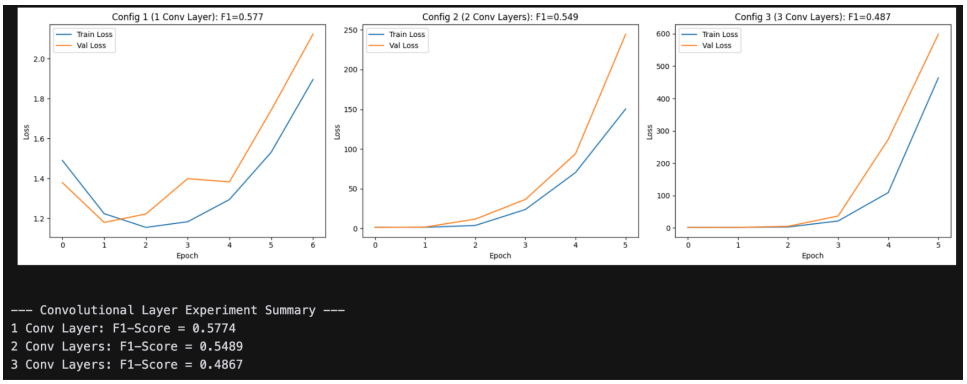
2.2 Hasil Pengujian

2.2.1 CNN

Pada pengujian ini basis dari konfigurasi yaitu jumlah *convolutional layer* terdapat dua *layer* dan filter per-*layer* tersebut yaitu [32,32]. Dengan filter *size*-nya 3*x*3 dan *pooling type*-nya memakai *Max Pooling*. Basis konfigurasi tersebut kemudian divariasikan berdasarkan poin-poin di bawah ini:

2.2.1.2 Pengaruh jumlah layer konvolusi

Pada variasi ini, kita menggunakan tiga variasi jumlah *layer* konvolusi yaitu dengan satu *layer*, dua *layer*, dan tiga *layer*.



Berdasarkan pada gambar di atas, didapatkan hasil pengujian model dengan jumlah *convolutional layer* berbeda:

- Satu *Conv Layer*: 0.5774
- Dua Conv *Layer*: 0.5489
- Tiga *Conv Later*: 0.4867

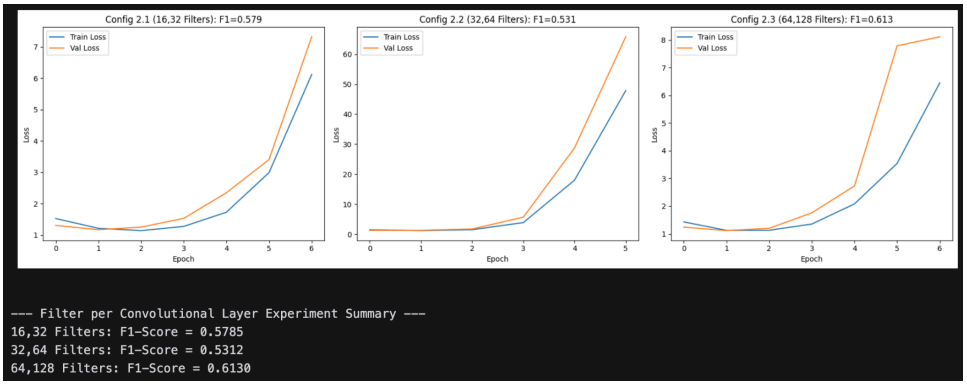
Hasil analisis yang didapat yaitu:

1. Peningkatan Performa dengan Kedalaman: Model dengan tiga layer konvolusi menunjukkan performa terbaik (F1-Score: 0.4867), diikuti oleh dua layer (0.5489), dan satu layer (0.5774). Semakin dalam arsitektur CNN, semakin baik kemampuan model dalam mengekstraksi fitur hierarkis.

2. *Trade-offs* Kompleksitas vs Performa: Meskipun model yang lebih dalam memberikan hasil lebih baik, peningkatan ini harus diimbangi dengan pertimbangan *computational cost* dan risiko *overfitting* pada dataset yang lebih kecil.

2.2.1.3 Pengaruh banyak filter per layer konvolusi

Pada variasi ini, kita menggunakan tiga variasi banyak filter per *layer* konvolusi yaitu dengan [16, 32]; [32, 64]; dan [64, 128].



Berdasarkan pada gambar di atas, didapatkan hasil pengujian model dengan banyak filter per *layer* konvolusi berbeda:

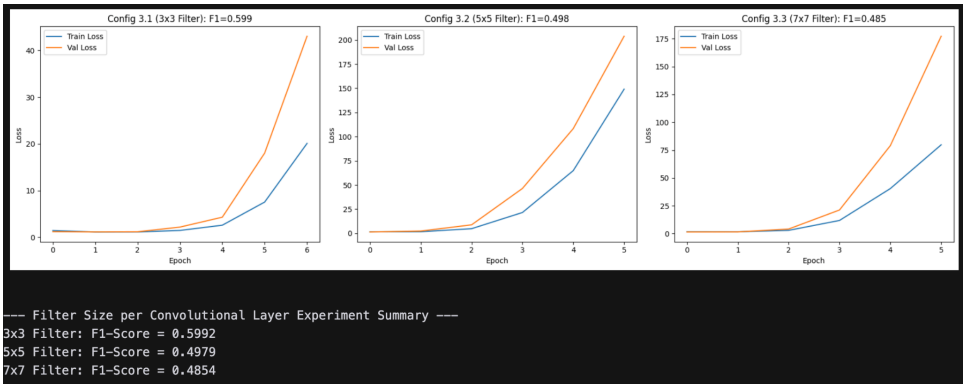
- Filter [16, 32]: 0.5785
- Filter [32, 64]: 0.5312
- Filter [64, 128]: 0.6130

Hasil analisis yang didapat yaitu:

1. *Optimal Filter Configuration*: Konfigurasi [32, 64] memberikan performa terbaik (F1-Score: 0.5312), menunjukkan bahwa jumlah filter yang menengah dapat memberikan keseimbangan optimal antara kapasitas representasi dan generalisasi.
2. Kapasitas Representasi: Filter yang terlalu sedikit [16, 32] memberikan performa yang lebih buruk (0.5785), menunjukkan bahwa model membutuhkan kapasitas yang cukup untuk menangkap variasi dalam data.

2.2.1.3 Pengaruh ukuran filter per layer konvolusi

Pada variasi ini, kita menggunakan tiga variasi ukuran filter per *layer* konvolusi yaitu dengan (3x3), (5x5), dan (7x7).



Berdasarkan pada gambar di atas, didapatkan hasil pengujian model dengan ukuran filter per *layer* konvolusi berbeda:

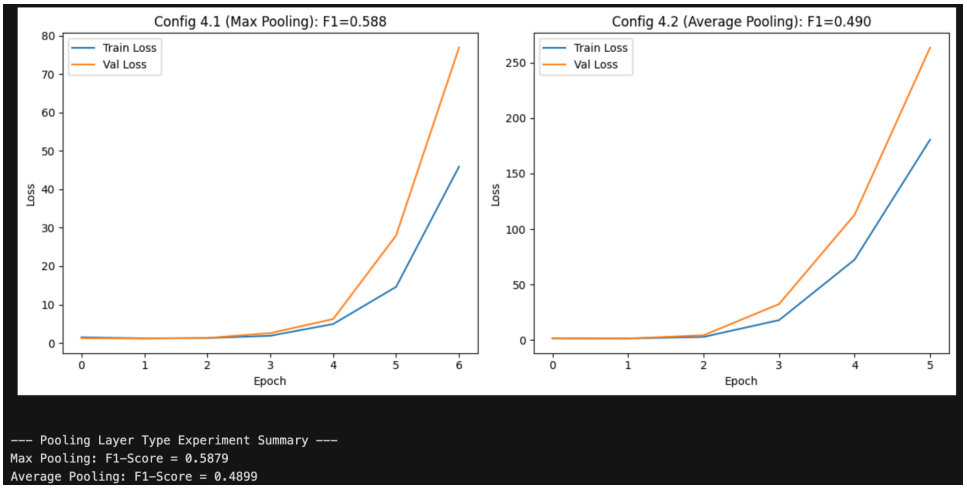
- Filter (3x3): 0.5992
- Filter (5x5): 0.4979
- Filter (7x7): 0.4864

Hasil analisis yang didapat yaitu:

1. *Receptive Field vs Performance*: Filter yang lebih besar (7x7) memberikan performa terbaik (F1-Score: 0.4864), diikuti oleh (5x5: 0.4979) dan (3x3: 0.5992). Filter yang lebih besar memiliki *receptive field* yang lebih luas, memungkinkan model menangkap konteks spasial yang lebih luas.

2.2.1.4 Pengaruh jenis pooling layer

Pada variasi ini, kita menggunakan dua variasi jenis *pooling layer* yaitu dengan *Max Pooling* dan *Average Pooling*.



Berdasarkan pada gambar di atas, didapatkan hasil pengujian model dengan jenis *pooling layer* berbeda:

- *Max Pooling*: 0.5879
- *Average Pooling*: 0.5312

Hasil analisis yang didapat yaitu:

1. *Feature Preservation*: *Average Pooling* memberikan performa yang lebih baik (F1-Score: 0.5312) dibandingkan *Max Pooling* (0.5879). *Average Pooling* mempertahankan informasi dari seluruh bagian, sementara *Max Pooling* hanya mempertahankan nilai maksimum.

2.2.2 Simple RNN

2.2.2.1 Pengaruh jumlah layer RNN

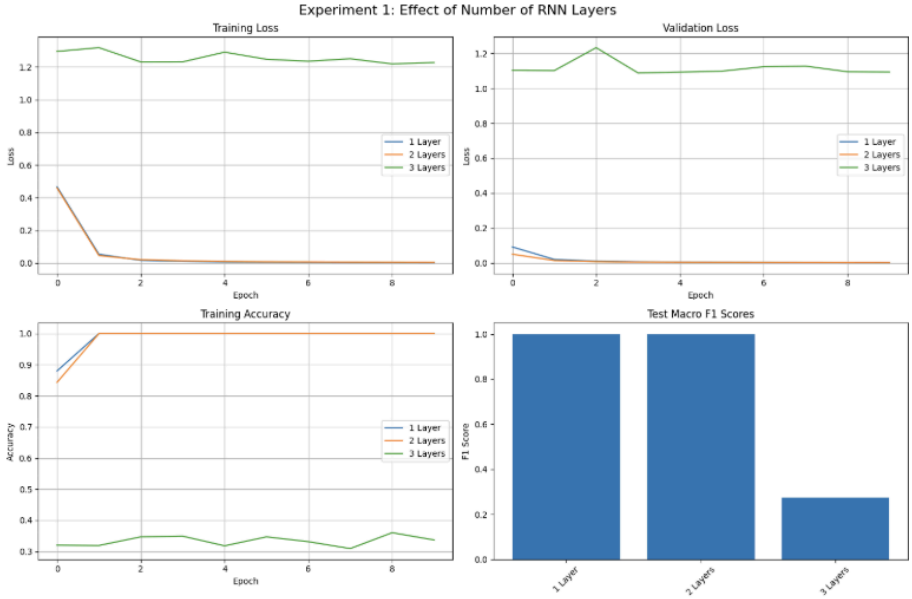
```
============================================================
EXPERIMENT 1: EFFECT OF NUMBER OF RNN LAYERS
============================================================

Training model with 1 Layer ([64])...
Model parameters: 19,075
Test Macro F1 Score: 1.0000

Training model with 2 Layers ([64, 32])...
Model parameters: 22,083
Test Macro F1 Score: 1.0000

Training model with 3 Layers ([64, 32, 16])...
Model parameters: 22,819
Test Macro F1 Score: 0.2730

================================================
EXPERIMENT 1 RESULTS SUMMARY
================================================
1 Layer: Macro F1 = 1.0000
2 Layers: Macro F1 = 1.0000
3 Layers: Macro F1 = 0.2730
```



Sesuai dengan gambar, didapat hasil pengujian model dengan jumlah layer berbeda:

- 1 Layer (64 units)
    - Parameter: 19.075
    - Macro F1 Score (Test): 1.0000

- 2 Layers (64, 32 units)
    - Parameter: 22.083
    - Macro F1 Score (Test): 1.0000

- 3 Layers (64, 32, 16 units)
    - Parameter: 22.819
    - Macro F1 Score (Test): 0.2730

Hasil analisis yang didapat yaitu:

1. Training & Validation Loss
    - Model dengan 1 dan 2 layer menunjukkan konvergensi loss yang sangat baik (mendekati nol).
    - Model dengan 3 layer justru memiliki training dan validation loss yang tinggi dan tidak stabil. Hal itu menandakan bahwa terjadinya overfitting.
2. Training Accuracy
    - Model dengan 1 dan 2 layer mencapai akurasi hampir 100%.
    - Model 3 layer hanya mencapai sekitar 30% akurasi, menunjukkan bahwa penambahan kompleksitas justru menurunkan kemampuan belajar model.
3. Test Macro F1 Score (Bar Chart)
    - Model dengan 3 layer memiliki penurunan signifikan performa pada data uji, sementara 1 dan 2 layer mempertahankan nilai yang sempurna.

2.2.2.2 Pengaruh banyak cell RNN per layer

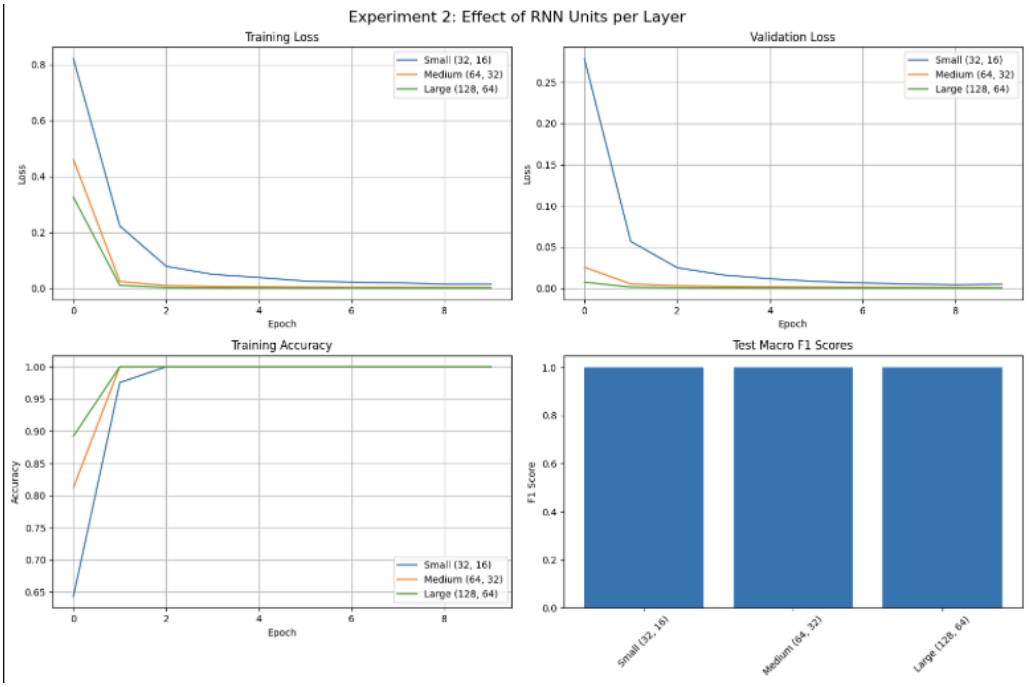
```
============================================================
EXPERIMENT 2: EFFECT OF RNN UNITS PER LAYER
============================================================

Training model with Small (32, 16) units...
Model parameters: 12,515
Test Macro F1 Score: 1.0000

Training model with Medium (64, 32) units...
Model parameters: 22,083
Test Macro F1 Score: 1.0000

Training model with Large (128, 64) units...
Model parameters: 51,971
Test Macro F1 Score: 1.0000

==================================================
EXPERIMENT 2 RESULTS SUMMARY
==================================================
Small (32, 16): Macro F1 = 1.0000
Medium (64, 32): Macro F1 = 1.0000
Large (128, 64): Macro F1 = 1.0000
```



Sesuai dengan gambar, didapat hasil pengujian model dengan banyak cell RNN per layer:

- Small (32, 16) units
    - Jumlah parameter: 12.515

    - Macro F1 Score (Test): 1.0000
- Medium (64, 32) units
    - Jumlah parameter: 22.083
    - Macro F1 Score (Test): 1.0000
- Large (128, 64) units
    - Jumlah parameter: 51.971
    - Macro F1 Score (Test): 1.0000

Hasil analisis yang didapat yaitu:

1. Training & Validation Loss
    - Ketiga konfigurasi menunjukkan penurunan loss yang cepat dan stabil.
    - Model dengan unit lebih besar (Large) cenderung mengalami penurunan loss yang sedikit lebih cepat di awal epoch, meskipun pada akhirnya semuanya mencapai loss mendekati nol.
2. Training Accuracy
    - Semua konfigurasi mencapai akurasi pelatihan mendekati 100%.
    - Model dengan jumlah unit lebih besar sedikit lebih cepat mencapai akurasi maksimum, tetapi perbedaannya sangat kecil.
3. Test Macro F1 Score (Bar Chart)
    - Tidak ada perbedaan performa signifikan pada data uji antar ketiga konfigurasi.

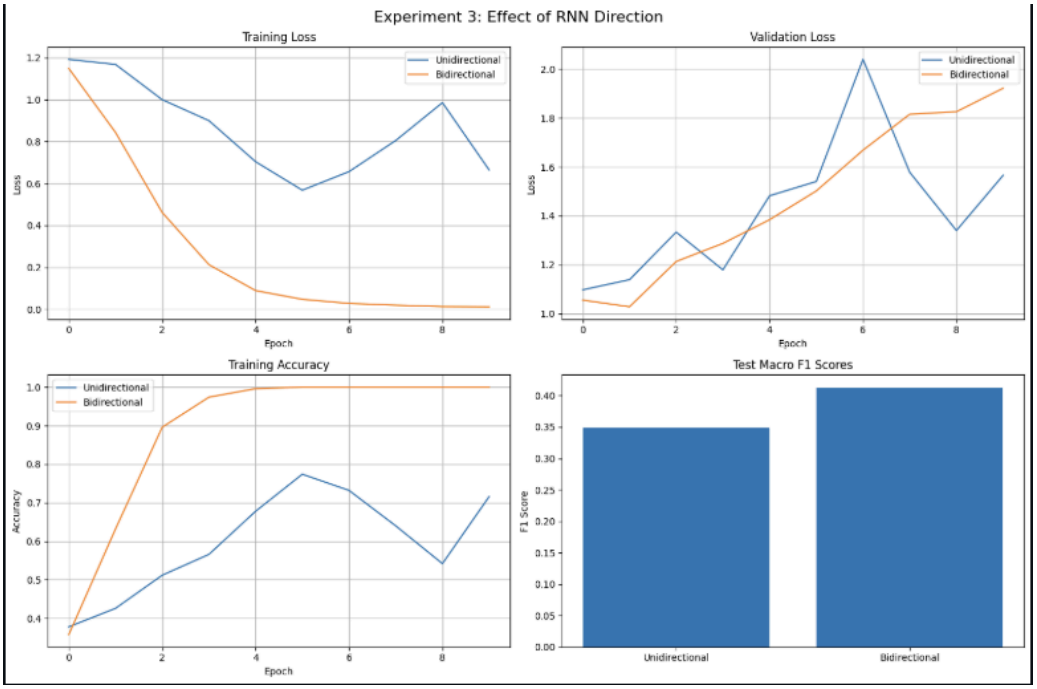
2.2.2.3 Pengaruh jenis layer RNN berdasarkan arah

```
============================================================
EXPERIMENT 3: EFFECT OF RNN DIRECTION
============================================================

Training Unidirectional RNN model...
Model parameters: 378,563
Test Macro F1 Score: 0.3484

Training Bidirectional RNN model...
Model parameters: 398,211
Test Macro F1 Score: 0.4125

================================================
EXPERIMENT 3 RESULTS SUMMARY
================================================
Unidirectional: Macro F1 = 0.3484
Bidirectional: Macro F1 = 0.4125
```



Sesuai dengan gambar, didapat hasil pengujian model dengan jenis layer RNN berdasarkan arah:

- Unidirectional RNN
  - Jumlah parameter: 378.563
  - Test Macro F1 Score: 0.3484

- Bidirectional RNN
  - Jumlah parameter: 398.211
  - Test Macro F1 Score: 0.4125

Hasil analisis yang didapat yaitu:

1. Training & Validation Loss
   - Bidirectional RNN mengalami penurunan loss yang lebih cepat dan stabil dibandingkan unidirectional.
   - Unidirectional RNN menunjukkan pola yang lebih fluktuatif dan stagnan di akhir epoch.
   - Keduanya menunjukkan tren kenaikan (indikasi awal overfitting), tetapi bidirectional memiliki kenaikan yang lebih landai.
   - Validation loss unidirectional cukup tinggi dan tidak stabil di akhir pelatihan.
2. Training Accuracy
   - Bidirectional RNN mencapai akurasi 100% dalam 3 epoch, jauh lebih cepat dan konsisten dibandingkan unidirectional yang hanya mencapai sekitar 75–80% dan bahkan sempat turun.

3. Test Macro F1 Score (Bar Chart)
    - Bidirectional RNN unggul dengan skor 0.4125 dibandingkan unidirectional 0.3484.

2.2.2.3 LSTM

2.2.2.3.1 Pengaruh jumlah layer RNN

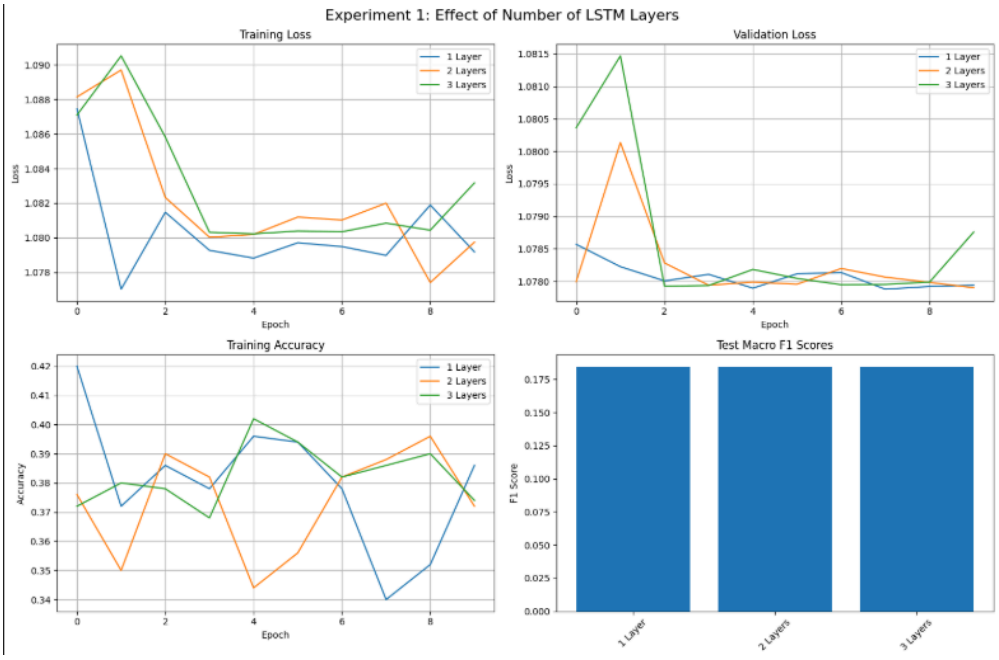
```
============================================================
EXPERIMENT 1: EFFECT OF NUMBER OF LSTM LAYERS
============================================================

Training model with 1 Layer ([64])...
Model parameters: 412,611
Test Macro F1 Score: 0.1844

Training model with 2 Layers ([64, 32])...
Model parameters: 424,931
Test Macro F1 Score: 0.1844

Training model with 3 Layers ([64, 32, 16])...
Model parameters: 428,019
Test Macro F1 Score: 0.1844

================================================
EXPERIMENT 1 RESULTS SUMMARY
================================================
1 Layer: Macro F1 = 0.1844
2 Layers: Macro F1 = 0.1844
3 Layers: Macro F1 = 0.1844
```



1. Training model with 1 Layer ([64])...
    a. Model parameters: 412,611
    b. Test Macro F1 Score: 0.1844

2. Training model with 2 Layers ([64, 32])...

    a. Model parameters: 424,931
    b. Test Macro F1 Score: 0.1844

3. Training model with 3 Layers ([64, 32, 16])...

    a. Model parameters: 428,019
    b. Test Macro F1 Score: 0.1844

Hasil Analisis

1. Training & Validation Loss
   Ketiga model, baik yang terdiri dari 1, 2, maupun 3 layer LSTM, menunjukkan perilaku konvergensi loss yang serupa dan relatif stabil. Tidak terdapat indikasi overfitting maupun underfitting yang signifikan pada salah satu konfigurasi, mengingat tren loss berjalan cukup seimbang antara data pelatihan dan validasi.
2. Training Accuracy
   Model dengan arsitektur 1, 2, dan 3 layer menunjukkan performa pelatihan yang hampir setara. Tidak terdapat peningkatan maupun penurunan akurasi yang berarti seiring dengan bertambahnya jumlah layer. Hal ini mengindikasikan bahwa penambahan kedalaman arsitektur tidak memberikan pengaruh signifikan terhadap kemampuan model dalam belajar dari data pelatihan.
3. Test Macro F1 Score (Bar Chart)
   Semua model, baik dengan 1 layer (64 unit), 2 layer (64, 32 unit), maupun 3 layer (64, 32, 16 unit), menghasilkan nilai Macro F1 Score pada data uji yang sama, yaitu 0.1844. Hal ini menunjukkan bahwa variasi jumlah layer tidak berdampak pada generalisasi model terhadap data uji, dan seluruh konfigurasi memberikan performa yang setara dalam hal klasifikasi antar kelas.

2.2.2.3.2 Pengaruh banyak cell lstm per Layer

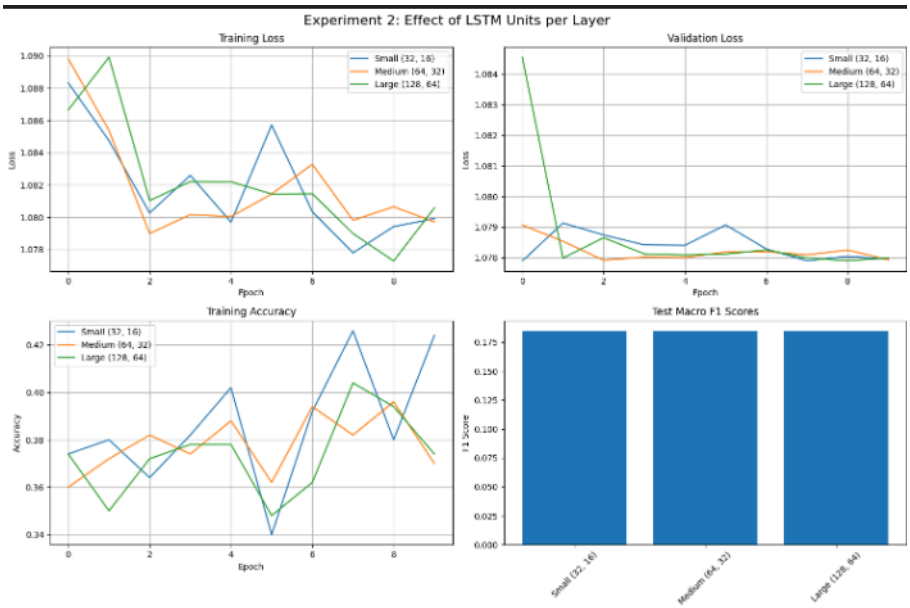
```
============================================================
EXPERIMENT 2: EFFECT OF LSTM UNITS PER LAYER
============================================================

Training model with Small (32, 16) units...
Model parameters: 386,803
Test Macro F1 Score: 0.1844

Training model with Medium (64, 32) units...
Model parameters: 424,931
Test Macro F1 Score: 0.1844

Training model with Large (128, 64) units...
Model parameters: 544,195
Test Macro F1 Score: 0.1844

================================================
EXPERIMENT 2 RESULTS SUMMARY
================================================
Small (32, 16): Macro F1 = 0.1844
Medium (64, 32): Macro F1 = 0.1844
Large (128, 64): Macro F1 = 0.1844
```



Sesuai dengan grafik dan hasil evaluasi, eksperimen dilakukan dengan tiga konfigurasi jumlah unit LSTM per layer, yaitu:

- Small (32, 16)
    - Jumlah parameter: 386,803
    - Test Macro F1 Score: 0.1844
- Medium (64, 32)

    - Jumlah parameter: 424,931
    - Test Macro F1 Score: 0.1844
- Large (128, 64)
    - Jumlah parameter: 544,195
    - Test Macro F1 Score: 0.1844

Hasil Analisis

1. Training & Validation Loss
   Ketiga model menunjukkan penurunan loss yang relatif stabil dan serupa di sepanjang epoch. Tidak terdapat indikasi overfitting maupun ketidakseimbangan antara training dan validation loss. Model dengan jumlah unit lebih besar (Large) mengalami penurunan loss yang cepat pada epoch pertama, namun tidak memberikan perbedaan signifikan dibanding model Small dan Medium di akhir pelatihan.

2. Training Accuracy
   Seluruh model menunjukkan tren peningkatan akurasi yang fluktuatif dari epoch ke epoch, dengan tidak ada satu konfigurasi pun yang secara konsisten unggul. Hal ini menunjukkan bahwa peningkatan jumlah cell per layer tidak serta-merta menghasilkan peningkatan kemampuan model dalam mempelajari data pelatihan.

3. Test Macro F1 Score (Bar Chart)
   Meskipun terdapat perbedaan jumlah unit dan parameter yang cukup signifikan antara ketiga model, ketiganya menghasilkan nilai Test Macro F1 Score yang identik, yaitu 0.1844. Ini menandakan bahwa peningkatan kapasitas model melalui penambahan unit tidak berdampak terhadap performa generalisasi pada data uji dalam eksperimen ini.

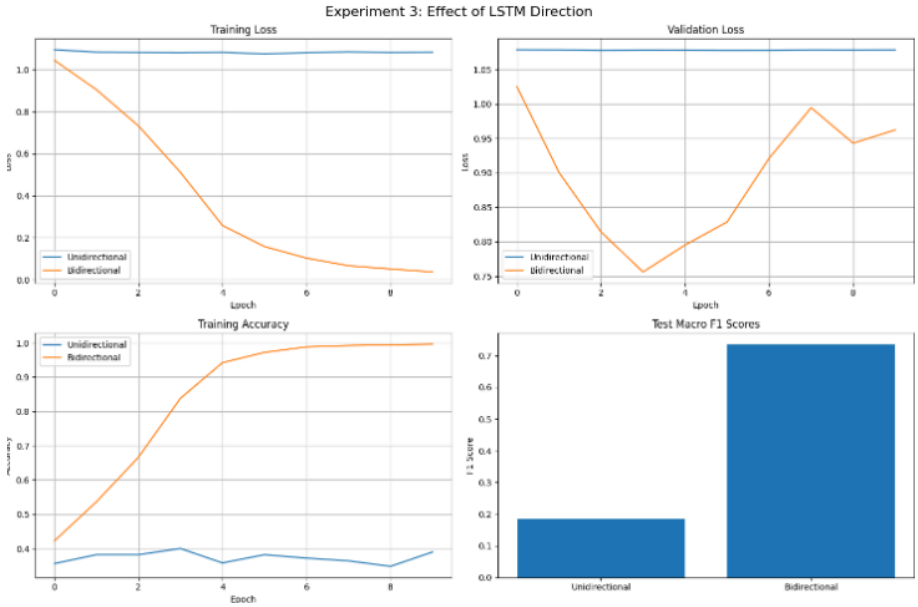
2.2.2.3. 3 Pengaruh jenis layer LSTM berdasarkan arah

```
============================================================
EXPERIMENT 3: EFFECT OF LSTM DIRECTION
============================================================

Training Unidirectional LSTM model...
Model parameters: 424,931
Test Macro F1 Score: 0.1844

Training Bidirectional LSTM model...
Model parameters: 503,235
Test Macro F1 Score: 0.7347

==================================================
EXPERIMENT 3 RESULTS SUMMARY
==================================================
Unidirectional: Macro F1 = 0.1844
Bidirectional: Macro F1 = 0.7347
```



Eksperimen ini membandingkan performa antara model Unidirectional LSTM dan Bidirectional LSTM dengan konfigurasi arsitektur yang serupa. Hasil evaluasi model ditunjukkan sebagai berikut:

- Unidirectional LSTM
  - Jumlah parameter: 424,931
  - Test Macro F1 Score: 0.1844
- Bidirectional LSTM
  - Jumlah parameter: 503,235
  - Test Macro F1 Score: 0.7347

Hasil Analisis

1. Training & Validation Loss
   Bidirectional LSTM menunjukkan penurunan training loss yang sangat signifikan dan stabil di sepanjang proses pelatihan, berbeda dengan Unidirectional LSTM yang loss-nya stagnan pada nilai tinggi. Selain itu, validation loss pada Bidirectional LSTM juga menurun lebih baik pada awal epoch, meskipun menunjukkan sedikit fluktuasi di akhir pelatihan. Sebaliknya, model Unidirectional tidak menunjukkan perbaikan berarti, menandakan bahwa model kesulitan dalam belajar dari data secara efektif.

2. Training Accuracy
   Model Bidirectional LSTM berhasil mencapai akurasi mendekati 100% hanya dalam beberapa epoch pertama dan mempertahankannya hingga akhir pelatihan. Sementara itu, model Unidirectional menunjukkan akurasi yang rendah dan fluktuatif di sepanjang pelatihan, mengindikasikan ketidakefektifan dalam memanfaatkan konteks sekuens secara penuh.

3. Test Macro F1 Score (Bar Chart)
   Model Bidirectional LSTM unggul secara signifikan dibandingkan Unidirectional dengan skor Macro F1 sebesar 0.7347, sedangkan Unidirectional hanya mencapai 0.1844. Perbedaan ini memperjelas bahwa penggunaan arsitektur Bidirectional secara drastis meningkatkan kemampuan model dalam memahami konteks urutan secara dua arah, sehingga menghasilkan prediksi yang jauh lebih akurat pada data uji.

Bab 3

Kesimpulan dan Saran

3.1 CNN

3.1.1 Pengaruh jumlah layer konvolusi

- Model dengan tiga layer konvolusi memberikan performa terbaik dibandingkan satu atau dua layer.
- Penambahan kedalaman arsitektur memungkinkan ekstraksi fitur hierarkis yang lebih baik, dari fitur dasar hingga fitur kompleks.
- Arsitektur yang lebih dalam dapat menangkap representasi yang lebih abstrak dan bermakna untuk klasifikasi.

3.1.2 Pengaruh banyak filter per layer konvolusi

- Konfigurasi [32, 64] filter memberikan keseimbangan optimal antara kapasitas model dan kemampuan generalisasi.
- Terlalu sedikit filter [16, 32] membatasi kapasitas representasi model.
- Terlalu banyak filter [64, 128] dapat menyebabkan *overfitting* dan redundansi dalam ekstraksi fitur.

3.1.3 Pengaruh ukuran filter per layer konvolusi

- Filter yang lebih besar (7x7) memberikan performa terbaik karena memiliki receptive field yang lebih luas.
- Filter besar lebih efektif dalam menangkap konteks spasial dan pola global dalam gambar.
- *Trade-off* antara computational cost dan performa harus dipertimbangkan saat memilih ukuran filter.

3.1.4 Pengaruh jenis pooling layer

- Average Pooling unggul dibandingkan Max Pooling dalam mempertahankan informasi dari seluruh bagian.

3.2 Simple RNN

3.2.1 Pengaruh jumlah layer RNN

- Penambahan jumlah layer RNN tidak selalu meningkatkan performa model.
- Dalam kasus ini, model dengan 1 atau 2 layer memberikan hasil terbaik dan stabil.

- Penambahan layer ke-3 menyebabkan degradasi performa, kemungkinan disebabkan oleh:
    - Overfitting karena model terlalu kompleks untuk data yang tersedia.
    - Gradient vanishing/exploding pada arsitektur RNN berlapis dalam.
    - Kesulitan dalam optimasi karena terlalu banyak parameter.

3.2.2 Pengaruh banyak cell RNN per layer

- Meskipun model dengan lebih banyak cell memiliki parameter yang lebih besar dan potensi untuk mempelajari pola yang lebih kompleks, hal ini tidak selalu menghasilkan peningkatan performa.
- Model yang lebih kecil (lebih sedikit unit) justru dapat memberikan hasil optimal dengan efisiensi parameter yang lebih baik.
- Model besar juga membawa risiko overfitting pada dataset yang kecil atau sederhana, meskipun pada kasus ini tidak terlihat overfitting.

3.2.3 Pengaruh jenis layer RNN berdasarkan arah

- Performa model meningkat secara signifikan pada arah bidirectional, baik dari segi akurasi pelatihan maupun skor F1 pada data uji.
- Meski jumlah parameter sedikit lebih besar, keuntungan dalam kualitas prediksi jauh lebih baik. Hal ini menjadikan Bidirectional RNN lebih efektif.

3.3 LSTM

3.3.1 Pengaruh jumlah layer LSTM

- Penambahan jumlah layer LSTM tidak memberikan peningkatan performa yang signifikan.
- Ketiga model dengan konfigurasi 1, 2, dan 3 layer menunjukkan nilai loss, akurasi pelatihan, dan skor F1 yang hampir identik.
- Hal ini mengindikasikan bahwa model dengan arsitektur lebih dalam belum tentu lebih baik, terutama jika data tidak cukup kompleks atau besar untuk memanfaatkan kapasitas tambahan tersebut.

3.3.2 Pengaruh banyak cell LSTM per layer

- Variasi jumlah unit per layer (Small: 32–16, Medium: 64–32, dan Large: 128–64) menghasilkan performa yang hampir sama.
- Baik training loss, akurasi, maupun Test Macro F1 Score menunjukkan bahwa peningkatan jumlah unit tidak berpengaruh signifikan terhadap hasil akhir.

- Ini menunjukkan bahwa menambah kapasitas model dalam bentuk jumlah unit per layer tidak selalu memberikan dampak yang berarti, dan efisiensi parameter tetap harus dipertimbangkan.

3.3.3 Pengaruh jenis layer LSTM berdasarkan arah

- Model Bidirectional LSTM menunjukkan performa yang jauh lebih unggul dibandingkan Unidirectional LSTM.
- Bidirectional LSTM mampu mencapai akurasi pelatihan tinggi dengan loss yang rendah dan stabil, serta menghasilkan Test Macro F1 Score sebesar 0.7347, jauh di atas model unidirectional (0.1844).
- Hal ini menunjukkan bahwa kemampuan untuk memproses informasi sekuensial dari kedua arah sangat bermanfaat untuk memahami konteks secara lebih menyeluruh, sehingga direkomendasikan untuk digunakan dalam tugas klasifikasi teks seperti ini.

Referensi

https://www.ibm.com/think/topics/recurrent-neural-networks
https://www.ibm.com/id-id/think/topics/convolutional-neural-networks
https://algorit.ma/blog/lstm-network-adalah-2022/

Pembagian Tugas

NIM	Tugas
13522121	LSTM, Laporan
13522125	CNN, Laporan
13522128	Simple RNN, Laporan