

LAPORAN TUGAS BESAR II
IF3270 PEMBELAJARAN MESIN
CONVOLUTIONAL NEURAL NETWORK & RECURRENT NEURAL NETWORK

Disusun untuk memenuhi tugas mata kuliah Pembelajaran Mesin
pada Semester II Tahun Akademik 2024/2025



Oleh Kelompok 2:

Thea Josephine Halim	13522012
Raffael Boymian Siahaan	13522046
Novelya Putri Ramadhani	13522096

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2025

DAFTAR ISI

DAFTAR ISI	2
BAB I	
DESKRIPSI MASALAH	4
1.1. Model CNN	4
1.2. Model SimpleRNN	4
1.3. Model LSTM	5
BAB II	
LANDASAN TEORI	6
1.4. Dasar Teori	6
2.4.1. Natural Language Processing (NLP)	6
2.4.2. Representasi Data Teks	6
2.4.2.1. Tokenisasi	6
2.4.2.2. Embedding	7
2.4.3. Convolutional Neural Network (CNN)	7
2.4.4. Recurrent Neural Network (RNN)	9
2.4.5. Long-Short Term Memory (LSTM)	11
BAB III	
IMPLEMENTASI DAN PENGUJIAN	13
3.1. Implementasi Program	13
3.1.1. Implementasi Utils	13
3.1.2. Implementasi CNN	21
3.1.3. Implementasi SimpleRNN	31
3.1.4. Implementasi LSTM	40
3.2. Hasil Pengujian	51
3.2.1. Pengujian CNN	52
3.2.1.1. Eksperimen Variasi Hyperparameter CNN	52
3.2.1.1.1 Eksperimen jumlah layer konvolusi	52
3.2.1.1.2 Eksperimen banyak filter per layer konvolusi	53
3.2.1.1.3 Eksperimen ukuran filter per layer konvolusi	53
3.2.1.1.4 Eksperimen jenis pooling layer konvolusi	54
3.2.1.1.4 Perbandingan dengan Implementasi From Scratch	55
3.2.2. Pengujian SimpleRNN	56
3.2.2.1. Eksperimen Variasi Hyperparameter SimpleRNN	56
3.2.2.2. Perbandingan dengan Implementasi From Scratch	59
3.2.3. Pengujian LSTM	60
3.2.3.1. Eksperimen Variasi Hyperparameter LSTM	60
3.2.3.2. Perbandingan dengan Implementasi From Scratch	62
BAB IV	

ANALISIS	63
4.1. CNN	63
4.1.1. Pengaruh jumlah layer konvolusi	63
4.1.2. Pengaruh banyak filter per layer konvolusi	64
4.1.3. Pengaruh ukuran filter per layer konvolusi	65
4.1.4. Pengaruh jenis pooling layer yang digunakan	67
4.1.5. Perbandingan dengan implementasi from scratch	68
4.2. RNN	69
4.2.1. Pengaruh jumlah layer RNN	69
4.2.2. Pengaruh banyak cell RNN per layer	70
4.2.3. Pengaruh jenis layer RNN berdasarkan arah	71
4.2.4. Perbandingan dengan implementasi from scratch	72
4.3. LSTM	73
4.3.1. Pengaruh jumlah layer LSTM	73
4.3.2. Pengaruh banyak cell LSTM per layer	73
4.3.3. Pengaruh jenis layer LSTM berdasarkan arah	74
4.3.4. Perbandingan dengan implementasi from scratch	74
BAB V	76
KESIMPULAN DAN SARAN	76
5.1. Kesimpulan	76
5.2. Saran	76
PEMBAGIAN TUGAS	77
LAMPIRAN	77
Repository	77
DAFTAR PUSTAKA	77

BAB I

DESKRIPSI MASALAH

Perkembangan teknologi kecerdasan buatan *deep learning*, telah membawa kemajuan signifikan dalam berbagai domain seperti computer vision dan natural language processing. Convolutional Neural Networks (CNN), Simple Recurrent Neural Networks (RNN), dan Long Short-Term Memory Networks (LSTM) adalah beberapa contoh arsitektur populer yang mampu mengekstraksi fitur kompleks dari data.

Pada tugas besar ini akan dilakukan implementasi dan analisis model-model dengan berbagai parameter sebagai berikut:

1.1. Model CNN

Ada beberapa parameter pilihan penting yang akan diuji dalam pengaruhnya terhadap performansi dari model CNN:

- Pengaruh Jumlah Layer Konvolusi: Tiga variasi jumlah layer konvolusi akan diuji untuk menilai sejauh mana kedalaman jaringan memengaruhi kemampuan model dalam mengekstraksi fitur
- Pengaruh Banyak Filter per Layer: Tiga kombinasi jumlah filter akan diuji untuk mengamati hubungan antara kapasitas representasi fitur dan risiko overfitting.
- Pengaruh Ukuran Filter per Layer: Tiga variasi ukuran filter akan diuji untuk mengevaluasi bagaimana ukuran reseptif mempengaruhi deteksi fitur spasial.
- Pengaruh Jenis Pooling Layer: Dua jenis pooling akan dibandingkan untuk menentukan metode pooling yang paling efektif dalam mereduksi dimensi tanpa kehilangan informasi penting.

1.2. Model SimpleRNN

Ada beberapa parameter pilihan penting yang akan diuji dalam pengaruhnya terhadap performansi dari model RNN:

- Pengaruh Jumlah Layer RNN: Model diuji dengan 1, 2, dan 3 layer RNN untuk melihat dampak kedalaman terhadap kemampuan mempelajari dependensi waktu.
- Pengaruh Banyak Cell RNN per Layer: Tiga konfigurasi jumlah unit untuk mengukur sejauh mana kapasitas memori jaringan memengaruhi akurasi prediksi
- Pengaruh Arah RNN (Unidirectional vs Bidirectional): Model unidirectional dan bidirectional akan dibandingkan untuk melihat dampak pemrosesan dua arah terhadap konteks informasi.

1.3. Model LSTM

Ada beberapa parameter pilihan penting yang akan diuji dalam pengaruhnya terhadap performansi dari model LSTM:

- Pengaruh Jumlah Layer LSTM: Model diuji dengan 1, 2, dan 3 layer RNN untuk melihat dampak kedalaman terhadap kemampuan menyimpan informasi jangka panjang.
- Pengaruh Banyak Cell LSTM per Layer: Tiga kombinasi jumlah cell per layer akan dibandingkan untuk memahami hubungan antara kapasitas memori dan hasil prediksi.
- Pengaruh Arah Layer (Bidirectional vs Unidirectional): Model unidirectional dan bidirectional akan dibandingkan untuk melihat dampak pemrosesan dua arah terhadap konteks informasi.

BAB II

LANDASAN TEORI

1.4. Dasar Teori

2.4.1. Natural Language Processing (NLP)

Natural Language Processing (NLP) adalah cabang dari kecerdasan buatan yang berfokus pada interaksi antara komputer dan bahasa manusia (*natural language*). Tujuan utama NLP adalah memungkinkan komputer untuk memahami, menginterpretasikan, dan menghasilkan bahasa yang digunakan oleh manusia secara alami. NLP mencakup berbagai tugas seperti klasifikasi teks, ekstraksi informasi, penerjemahan otomatis, dan analisis sentimen.

Adanya NLP ini membantu peningkatan automasi task yang repetitif, meningkatkan analisis data, meningkatkan kualitas *content generation*, dan lainnya. Salah satu contoh NLP adalah translasi bahasa, di mana mesin akan mengubah dari satu bahasa ke bahasa yang lain dengan tetap memperhatikan konteks dan maknanya. Fungsi NLP untuk meningkatkan analisis data dapat dilihat dari *insight extraction* data tidak terstruktur seperti review pelanggan. Dari melihat pola dan tren yang muncul, mesin dapat mengidentifikasi perasaan dan emosi pelanggan. Dalam tugas besar ini, NLP akan kita gunakan untuk memproses data teks dalam bahasa alami dan melakukan klasifikasi sentimen berdasarkan isi teks tersebut.

2.4.2. Representasi Data Teks

Dalam konteks pemrosesan bahasa alami (Natural Language Processing), data teks perlu diubah terlebih dahulu ke dalam bentuk numerik agar dapat diproses oleh model pembelajaran mesin atau jaringan saraf. Proses ini dikenal sebagai representasi data teks, dan merupakan tahap awal yang sangat penting karena model hanya dapat memproses input dalam bentuk angka. Dua tahapan utama dalam representasi data teks yang digunakan dalam proyek ini adalah tokenisasi dan embedding.

2.4.2.1. Tokenisasi

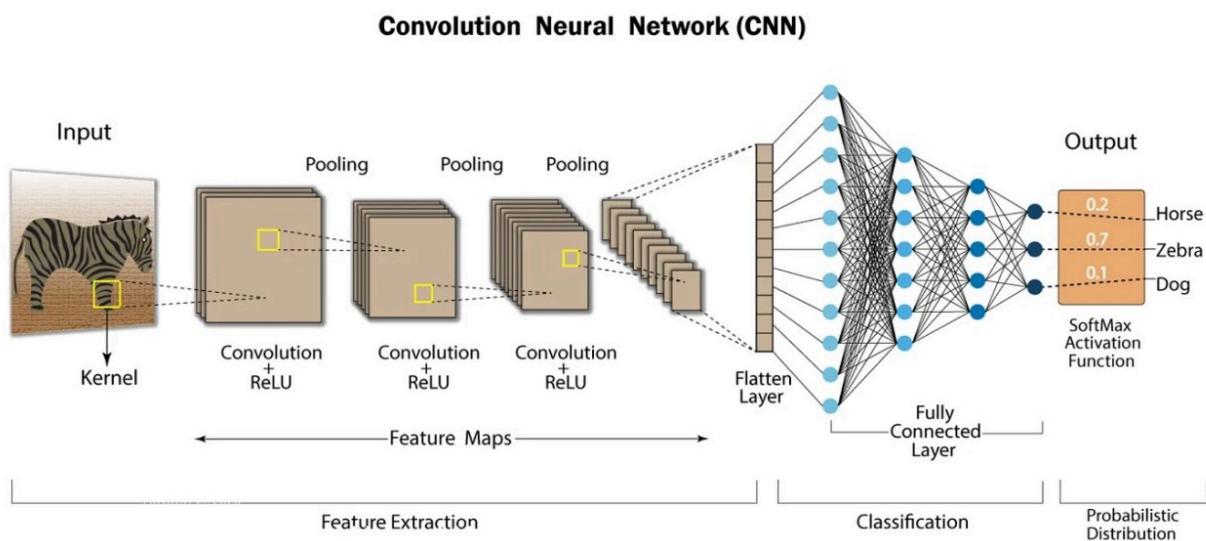
Tokenisasi merupakan proses memecah data teks menjadi bagian-bagian kecil yang disebut sebagai token. Token bisa berupa kata, frasa, atau bahkan karakter, tergantung dari tingkat tokenisasi yang digunakan. Dalam tugas ini, tokenisasi

dilakukan pada level kata, di mana setiap kata dari input teks dipetakan ke dalam sebuah bilangan bulat yang unik. Proses ini dilakukan menggunakan layer TextVectorization dari Keras, yang secara otomatis membangun *vocabulary* dari data pelatihan, lalu memetakan setiap kata yang dikenali ke indeks numeriknya. Hasil dari tahap ini adalah *sequence* berupa daftar integer, yang mewakili setiap kata dalam teks sesuai urutan kemunculannya.

2.4.2.2. Embedding

Setelah teks dikonversi menjadi urutan integer melalui proses tokenisasi, tahap selanjutnya adalah *embedding*. Embedding adalah teknik untuk memetakan token-token tersebut ke dalam ruang vektor berdimensi tetap, yang disebut sebagai *embedding space*. Dalam ruang ini, setiap kata diwakili sebagai vektor real berdimensi-n, di mana dimensi ini dapat diatur sesuai kebutuhan (misalnya 64 dimensi). Proses embedding dilakukan menggunakan Embedding layer dari Keras, yang belajar selama proses pelatihan untuk menempatkan kata-kata dengan makna atau fungsi yang mirip ke dalam posisi vektor yang berdekatan. Dengan representasi vektor ini, model dapat mempelajari hubungan semantik antar kata secara lebih efektif dibandingkan representasi numerik biasa.

2.4.3. Convolutional Neural Network (CNN)



Convolutional Neural Network (CNN) adalah salah satu arsitektur jaringan saraf tiruan yang secara khusus dirancang untuk menangani data berdimensi spasial, terutama citra digital. CNN memanfaatkan sifat lokalitas dan translasi invarian dari gambar untuk mengekstraksi fitur-fitur penting melalui serangkaian lapisan konvolusi dan subsampling. Tidak seperti jaringan saraf fully connected biasa yang menghubungkan setiap neuron dengan seluruh input, CNN memanfaatkan koneksi lokal (local receptive fields) untuk mengurangi jumlah parameter dan meningkatkan efisiensi pelatihan.

Lapisan atau layer CNN yang akan diimplementasikan pada tugas besar ini adalah sebagai berikut:

- Conv2D layer

Lapisan ini merupakan inti dari CNN yang bertugas mengekstraksi fitur dari gambar input. Conv2D layer menerapkan filter (kernel) berukuran kecil yang melakukan operasi konvolusi pada bagian-bagian lokal dari citra, menghasilkan feature maps. Setiap filter dapat mengenali pola tertentu, seperti garis, sudut, atau tekstur. Parameter dalam filter dipelajari selama proses pelatihan agar dapat mengenali fitur penting dari data.

- Pooling layers

Pooling layer berfungsi untuk mengurangi dimensi spasial (tinggi dan lebar) dari feature maps yang dihasilkan oleh Conv2D, sehingga menurunkan kompleksitas komputasi dan membantu mencegah overfitting.

- Flatten/Global Pooling layer

Layer ini mengubah data dari bentuk dua dimensi (hasil konvolusi dan pooling) menjadi vektor satu dimensi agar bisa diproses oleh Dense layer

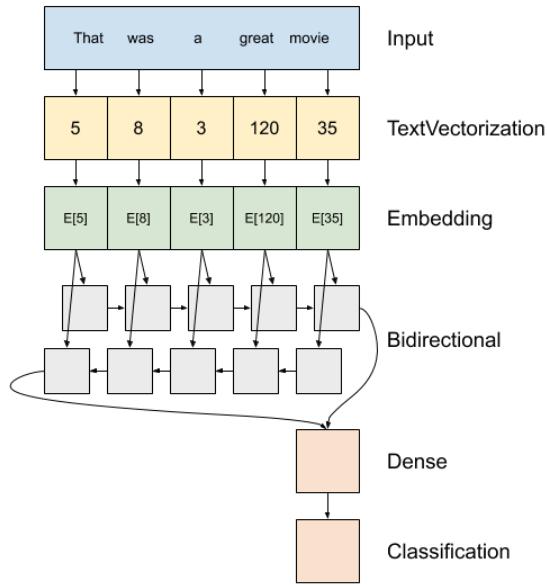
- Dense layer

Lapisan ini menghubungkan semua neuron dari lapisan sebelumnya ke setiap neuron di lapisan ini. Dense layer digunakan untuk melakukan klasifikasi berdasarkan fitur yang telah diekstraksi oleh layer-layer sebelumnya. Di layer terakhir (output layer), biasanya digunakan fungsi aktivasi softmax (untuk klasifikasi multikelas) agar hasilnya berupa probabilitas dari masing-masing kelas.

CNN bersifat hierarkis dalam hal pembelajaran fitur. Lapisan awal biasanya mendeteksi fitur sederhana seperti garis dan warna, sementara lapisan lebih dalam menangkap

fitur kompleks seperti bentuk objek atau pola khas tertentu. Keunggulan CNN terletak pada kemampuannya untuk belajar secara otomatis dari data mentah tanpa memerlukan ekstraksi fitur manual, serta efisiensinya dalam menangani data berdimensi tinggi seperti gambar.

2.4.4. Recurrent Neural Network (RNN)



Recurrent Neural Network (RNN) adalah jenis arsitektur jaringan saraf yang dirancang khusus untuk memproses data sekuensial, seperti teks, ucapan, atau deret waktu. Tidak seperti jaringan saraf feedforward biasa, RNN memiliki loop internal yang memungkinkan informasi dari langkah sebelumnya dibawa ke langkah berikutnya. Ini membuat RNN sangat cocok untuk menangkap dependensi urutan atau konteks dalam data yang memiliki struktur berurutan.

RNN bekerja dengan memproses input satu per satu dalam urutan waktu, sambil mempertahankan hidden state yang diperbarui di setiap langkah. Hidden state ini berfungsi sebagai “memori” dari informasi sebelumnya, sehingga RNN dapat membuat keputusan berdasarkan urutan, bukan hanya pada input saat ini. Namun, RNN standar memiliki keterbatasan dalam mengingat informasi jangka panjang karena masalah vanishing gradient, yang sering diatasi dengan arsitektur seperti LSTM (Long Short-Term Memory) atau GRU (Gated Recurrent Unit).

Lapisan atau layer CNN yang akan diimplementasikan pada tugas besar ini adalah sebagai berikut:

- Embedding layer

Layer ini mengubah token (integer) hasil tokenisasi menjadi vektor berdimensi tetap. Embedding layer memungkinkan representasi kata dalam bentuk vektor yang dapat belajar makna semantik selama pelatihan. Kata-kata dengan makna mirip akan memiliki vektor yang relatif dekat dalam ruang embedding. Ini merupakan tahap penting untuk menjembatani data kategorikal (teks) ke bentuk numerik yang dapat diproses oleh lapisan RNN.

- Unidirectional RNN Layer

Lapisan ini membaca input sekuensial hanya dari satu arah, biasanya dari awal ke akhir. Pada tiap langkah waktu, layer ini menerima input saat ini dan hidden state dari langkah sebelumnya untuk menghasilkan output. Unidirectional RNN cocok digunakan jika urutan input memiliki arah waktu yang jelas dan hanya informasi masa lalu yang relevan untuk prediksi.

- Bidirectional RNN Layer

Berbeda dari versi unidirectional, bidirectional RNN memproses data dari dua arah: satu dari awal ke akhir, dan satu dari akhir ke awal. Output dari kedua arah ini kemudian digabung (concatenation atau penjumlahan) untuk memperkaya konteks informasi.

- Dropout layer

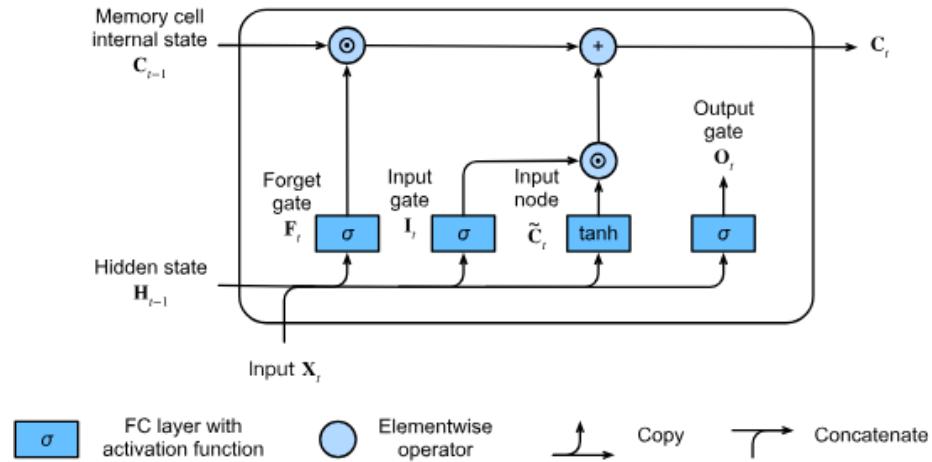
Lapisan ini digunakan untuk mencegah overfitting dengan cara “menonaktifkan” sejumlah unit secara acak selama pelatihan. Dengan membuat jaringan tidak terlalu bergantung pada neuron tertentu, dropout membantu generalisasi model. Dalam konteks RNN, dropout bisa diterapkan baik pada input maupun antar time step, tergantung konfigurasi.

- Dense layer

Sama seperti pada CNN, Dense layer adalah lapisan *fully connected* yang menerima output akhir dari RNN (baik satu langkah terakhir atau seluruh urutan yang diringkas) dan melakukan klasifikasi. Fungsi aktivasi seperti softmax biasa digunakan di layer ini untuk menghasilkan distribusi probabilitas dari kelas-kelas yang tersedia.

Dense layer berfungsi sebagai pengambil keputusan akhir dari seluruh rangkaian proses.

2.4.5. Long-Short Term Memory (LSTM)



LSTM merupakan pengembangan dari Recurrent Neural Network (RNN) yang dirancang untuk lebih efektif menangani dependensi jangka panjang pada data sekuensial. Secara konsep, LSTM hampir sama seperti RNN dalam hal memproses input secara berurutan, tetapi LSTM memiliki mekanisme internal berupa cell state dan gerbang-gerbang (input, forget, output gate) yang memungkinkannya untuk menyimpan atau membuang informasi secara selektif. Hal ini membuat LSTM lebih stabil dalam pelatihan dan lebih baik dalam mengingat informasi penting dari bagian awal urutan input.

Keunikan utama LSTM adalah struktur internalnya yang terdiri dari cell state dan tiga gerbang utama:

- Forget gate: Menentukan informasi mana dari cell state sebelumnya yang akan dilupakan.
- Input gate: Menentukan informasi baru mana yang akan ditambahkan ke cell state.
- Output gate: Menentukan apa yang akan dihasilkan dari cell state.

Lapisan atau layer CNN yang akan diimplementasikan pada tugas besar ini adalah sebagai berikut:

- Embedding layer

Layer ini sama seperti pada model RNN, berfungsi untuk mengubah token hasil tokenisasi menjadi representasi vektor berdimensi tetap yang dapat dipelajari.

- LSTM Layer (Bidirectional/Unidirectional)

Secara umum fungsinya sama seperti RNN layer, yaitu memproses input sekuensial. Namun, perbedaannya terletak pada kemampuan LSTM untuk mengelola informasi jangka panjang dengan lebih stabil melalui struktur gerbang internal.

- Unidirectional LSTM hanya melihat urutan satu arah (dari awal ke akhir).
- Bidirectional LSTM membaca urutan dari dua arah, sehingga dapat menangkap konteks dari masa lalu dan masa depan sekaligus.

- Dropout layer

Fungsinya identik seperti pada model RNN, yaitu untuk mencegah overfitting dengan menonaktifkan sebagian neuron secara acak selama pelatihan.

- Dense layer

Sama seperti pada model-model lainnya, digunakan untuk memproyeksikan output akhir menjadi prediksi kelas. Biasanya dilengkapi dengan aktivasi softmax untuk klasifikasi multikelas.

BAB III

IMPLEMENTASI DAN PENGUJIAN

3.1. Implementasi Program

Program dibagi dalam 4 folder utama: cnn, rnn, lstm, dan utils. Folder utils akan berisi modul-modul pendukung yang digunakan secara bersama oleh ketiga model, meliputi data_loader.py untuk memuat dan memproses dataset, layers.py yang mengimplementasikan layer-layer neural network kustom, metrics.py untuk menghitung berbagai metrik evaluasi model, tokenizer.py untuk preprocessing teks, dan other.py yang berisi fungsi-fungsi utilitas tambahan.

Folder rnn mengandung implementasi Recurrent Neural Network dengan file utama rnn_model.py yang mendefinisikan arsitektur model RNN, rnn_layer.py yang berisi implementasi layer RNN dari scratch, train_rnn.py untuk proses pelatihan model, dan init.py sebagai inisialisasi modul. Terdapat juga subfolder experiments yang menyimpan hasil eksperimen berupa laporan klasifikasi, bobot model Keras , dan visualisasi plot hasil pelatihan dalam png. Struktur serupa diterapkan pada folder cnn dan lstm, dimana masing-masing memiliki file model, layer, dan training script yang spesifik untuk arsitektur CNN dan LSTM.

3.1.1. Implementasi Utils

data_loader.py

Fungsi load data akan load data dari csv NusaX ataupun CIFAR-10 dan mengoutputkan dalam variabel train, test, dan val. Fungsi ds_to_numpy mengubah menjadi array numpy untuk CIFAR-10

```
● ● ●
1 def load_data(self):
2     if self.dataset == 'cifar10':
3         ds, _ = tfds.load(self.dataset, with_info=True, as_supervised=True, split=['train[:80%]', 'train[80%:]', 'test'], shuffle_files=True)
4         train_data, val_data, test_data = ds
5
6         x_train, y_train = self.ds_to_numpy(train_data)
7         x_val, y_val = self.ds_to_numpy(val_data)
8         x_test, y_test = self.ds_to_numpy(test_data)
9
10        return x_train, x_val, x_test, y_train, y_val, y_test
11
12    elif self.dataset == 'NusaX':
13        BASE_DIR = os.path.dirname(os.path.abspath(__file__))
14        base_path = os.path.join(BASE_DIR, 'NusaX')
15        train_df = pd.read_csv(os.path.join(base_path, 'train.csv'))
16        val_df = pd.read_csv(os.path.join(base_path, 'valid.csv'))
17        test_df = pd.read_csv(os.path.join(base_path, 'test.csv'))
18
19        x_train = train_df.drop(columns=['label']).values
20        y_train = train_df['label'].values
21        x_val = val_df.drop(columns=['label']).values
22        y_val = val_df['label'].values
23        x_test = test_df.drop(columns=['label']).values
24        y_test = test_df['label'].values
25
26        return np.array(x_train), np.array(x_val), np.array(x_test), np.array(y_train), np.array(y_val), np.array(y_test)
27
```

```
● ● ●
1 def ds_to_numpy(self, ds):
2     x, y = [], []
3
4     for img, label in ds:
5         x.append(np.array(img))
6         y.append(np.array(label))
7
8     x = np.array(x)
9     y = np.array(y)
10
11    return x, y
```

other.py

Memuat fungsi aktivasi, loss function, dan turunan tiap fungsi aktivasi

```
● ● ●
1  class ActivationFunction:
2
3
4      @staticmethod
5      def linear(x):
6          return x
7
8      @staticmethod
9      def relu(x):
10         return np.maximum(0, x)
11
12     @staticmethod
13     def sigmoid(x):
14         return 1 / (1 + np.exp(-x))
15
16     @staticmethod
17     def tanh(x):
18         return (np.exp(x) - np.exp(-x)) / (np.exp(x) + np.exp(-x))
19
20     @staticmethod
21     def softmax(x):
22         exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))
23         return exp_x / np.sum(exp_x, axis=1, keepdims=True)
24
25     # Bonus: Swish, softplus, and ELU
26     @staticmethod
27     def swish(x):
28         return x * ActivationFunction.sigmoid(x)
29
30     @staticmethod
31     def softplus(x):
32         return np.log(1 + np.exp(x))
33
34     @staticmethod
35     def elu(x, alpha=1.0):
36         return np.where(x > 0, x, alpha * (np.exp(x) - 1))
```

```
● ● ●
1  class Derivative:
2
3
4      @staticmethod
5      def linear(x):
6          return np.ones_like(x)
7
8      @staticmethod
9      def relu(x):
10         return np.where(x > 0, 1, 0)
11
12     @staticmethod
13     def sigmoid(x):
14         s = ActivationFunction.sigmoid(x)
15         return s * (1 - s)
16
17     @staticmethod
18     def tanh(x):
19         return 1 - np.tanh(x)**2
20
21     @staticmethod
22     def softmax(x):
23         s = ActivationFunction.softmax(x)
24         batch_size, num_classes = s.shape
25
26         # Buat matriks turunan untuk setiap sampel dalam batch
27         jacobian = np.zeros((batch_size, num_classes, num_classes))
28
29         for i in range(batch_size):
30             s_i = s[i].reshape(-1, 1)
31             jacobian[i] = np.diagflat(s_i) - np.dot(s_i, s_i.T)
32
33     return jacobian
34
35     # Bonus: Swish, softplus, and ELU
36     @staticmethod
37     def swish(x):
38         s = ActivationFunction.sigmoid(x)
39         return s + x * s * (1 - s)
40
41     @staticmethod
42     def softplus(x):
43         return 1 / (1 + np.exp(-x))
44
45     @staticmethod
46     def elu(x, alpha=1.0):
47         return np.where(x > 0, 1, alpha * np.exp(x))
```

```
● ● ●
1 class LossFunction:
2
3
4     @staticmethod
5     def mse(yi, y_hat):
6         return np.mean((yi - y_hat) ** 2)
7
8     @staticmethod
9     def binCrossEntropy(yi, y_hat):
10        return -np.mean(yi * np.log(y_hat) + (1 - yi) * np.log(1 - y_hat))
11
12    @staticmethod
13    def catCrossEntropy(yi, y_hat):
14        eps = 1e-15
15        y_hat = np.clip(y_hat, eps, 1 - eps)
16
17        n = yi.shape[0]
18
19        loss = -np.sum(yi * np.log(y_hat)) / n
20
21    return loss
```

tokenizer.py

Kelas TextPreprocessor memproses teks menjadi format yang dapat digunakan oleh model *machine learning*.

Kelas ini mendukung dua metode tokenisasi teks: menggunakan TextVectorization (fitur bawaan Keras untuk mengubah teks menjadi urutan angka) dan Tokenizer (fitur klasik Keras untuk membuat kamus kata dan konversi ke urutan indeks). Fungsi-fungsi utama di dalamnya meliputi pembuatan dan adaptasi layer vektorisasi, konversi teks ke dalam bentuk numerik, padding untuk keseragaman panjang input, serta akses ke informasi seperti ukuran kosakata dan indeks kata.

```

● ● ●
1 import tensorflow as tf
2 from tensorflow.keras.preprocessing.text import Tokenizer
3 from tensorflow.keras.preprocessing.sequence import pad_sequences
4 import numpy as np
5 import pandas as pd
6 import pickle
7
8 class TextPreprocessor:
9     def __init__(self, max_features=10000, max_length=100):
10         self.max_features = max_features
11         self.max_length = max_length
12         self.tokenizer = None
13         self.text_vectorization = None
14
15     def create_text_vectorization(self, texts):
16         # Create TextVectorization layer for tokenization
17         self.text_vectorization = tf.keras.layers.TextVectorization(
18             max_tokens=self.max_features,
19             output_sequence_length=self.max_length,
20             output_mode="int"
21         )
22
23         # Adapt to the texts
24         self.text_vectorization.adapt(texts)
25
26         return self.text_vectorization
27
28     def preprocess_with_keras(self, texts):
29         # Preprocess texts using Keras TextVectorization
30         if self.text_vectorization is None:
31             self.create_text_vectorization(texts)
32
33         # Convert texts to vector sequences
34         sequences = self.text_vectorization(texts)
35
36         return sequences.numpy()
37
38     def get_vocab_size(self):
39         if self.text_vectorization is not None:
40             return self.text_vectorization.vocabulary_size()
41         elif self.tokenizer is not None:
42             return len(self.tokenizer.word_index) + 1
43         else:
44             return self.max_features
45
46     def preprocess_with_tokenizer(self, texts):
47         # Preprocess texts using keras Tokenizer
48         if self.tokenizer is None:
49             self.tokenizer = Tokenizer(num_words=self.max_features, oov_token="")
50             self.tokenizer.fit_on_texts(texts)
51
52         # Convert texts to sequences
53         sequences = self.tokenizer.texts_to_sequences(texts)
54
55         # Pad sequences to ensure uniform input size
56         padded_sequences = pad_sequences(sequences, maxlen=self.max_length, padding="post", truncating="post")
57
58         return padded_sequences
59
60     def get_word_index(self):
61         if self.tokenizer is not None:
62             return self.tokenizer.word_index
63         else:
64             return {}
65
66     def save_tokenizer(self, filepath):
67         # Save the tokenizer to a file
68         if self.tokenizer is not None:
69             with open(filepath, 'wb') as handle:
70                 pickle.dump(self.tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
71         else:
72             raise ValueError("Tokenizer is not initialized.")
73
74     def load_tokenizer(self, filepath):
75         # Load the tokenizer from a file
76         with open(filepath, 'rb') as handle:
77             self.tokenizer = pickle.load(handle)

```

layers.py

Mendefinisikan berbagai lapisan (layer) dan unit yang mencakup EmbeddingLayer, DenseLayer, RNNCell, LSTMCell, dan DropoutLayer. EmbeddingLayer memetakan indeks kata ke vektor representasi berdimensi tetap; DenseLayer adalah lapisan fully connected dengan fungsi aktivasi yang dapat dikustomisasi; RNNCell dan LSTMCell mengimplementasikan recurrent neural network, dengan LSTMCell menangani informasi jangka panjang menggunakan gate input, forget, dan output; sedangkan DropoutLayer melakukan regularisasi dengan menonaktifkan sebagian neuron selama

pelatihan untuk mencegah overfitting. Seluruh lapisan ini mendukung berbagai fungsi aktivasi yang diambil dari modul ActivationFunction.

```
● ● ●
1  from utils.other import ActivationFunction
2  import numpy as np
3
4  class EmbeddingLayer:
5      def __init__(self, input_dim, output_dim, weights=None):
6          self.input_dim = input_dim
7          self.output_dim = output_dim
8          if weights is not None:
9              self.embedding_matrix = weights
10         else:
11             # just use random if cannot read
12             self.embedding_matrix = np.random.randn(input_dim, output_dim) * 0.01
13
14     def forward(self, x):
15         # x = (batch_size, sequence_length)
16         return np.take(self.embedding_matrix, x, axis=0)
17
18 class Denselayer:
19     def __init__(self, weight, bias, activation='linear'):
20         self.weight = weight
21         self.bias = bias
22         self.activation = activation
23
24     def forward(self, input):
25         sigma = np.dot(input, self.weight) + self.bias
26         return self._activate(sigma)
27
28     def _activate(self, sigma):
29         if self.activation == 'linear':
30             return ActivationFunction.linear(sigma)
31         elif self.activation == 'relu':
32             return ActivationFunction.relu(sigma)
33         elif self.activation == 'sigmoid':
34             return ActivationFunction.sigmoid(sigma)
35         elif self.activation == 'tanh':
36             return ActivationFunction.tanh(sigma)
37         elif self.activation == 'softmax':
38             return ActivationFunction.softmax(sigma)
39         elif self.activation == 'swish':
40             return ActivationFunction.swish(sigma)
41         elif self.activation == 'softplus':
42             return ActivationFunction.softplus(sigma)
43         elif self.activation == 'elu':
44             return ActivationFunction.elu(sigma)
45         else:
46             raise ValueError("Unknown activation function")
47
```

```

47
48     class RNNCell:
49         def __init__(self, Wx, Wh, b, activation='tanh'):
50             self.Wx = Wx
51             self.Wh = Wh
52             self.b = b
53             self.activation = activation
54
55         def _activate(self, sigma):
56             if self.activation == 'linear':
57                 return ActivationFunction.linear(sigma)
58             elif self.activation == 'relu':
59                 return ActivationFunction.relu(sigma)
60             elif self.activation == 'sigmoid':
61                 return ActivationFunction.sigmoid(sigma)
62             elif self.activation == 'tanh':
63                 return ActivationFunction.tanh(sigma)
64             elif self.activation == 'softmax':
65                 return ActivationFunction.softmax(sigma)
66             elif self.activation == 'swish':
67                 return ActivationFunction.swish(sigma)
68             elif self.activation == 'softplus':
69                 return ActivationFunction.softplus(sigma)
70             elif self.activation == 'elu':
71                 return ActivationFunction.elu(sigma)
72             else:
73                 raise ValueError("Unknown activation function")
74
75         def forward(self, x_t, h_prev):
76             return self._activate(np.dot(x_t, self.Wx) + np.dot(h_prev, self.Wh) + self.b)

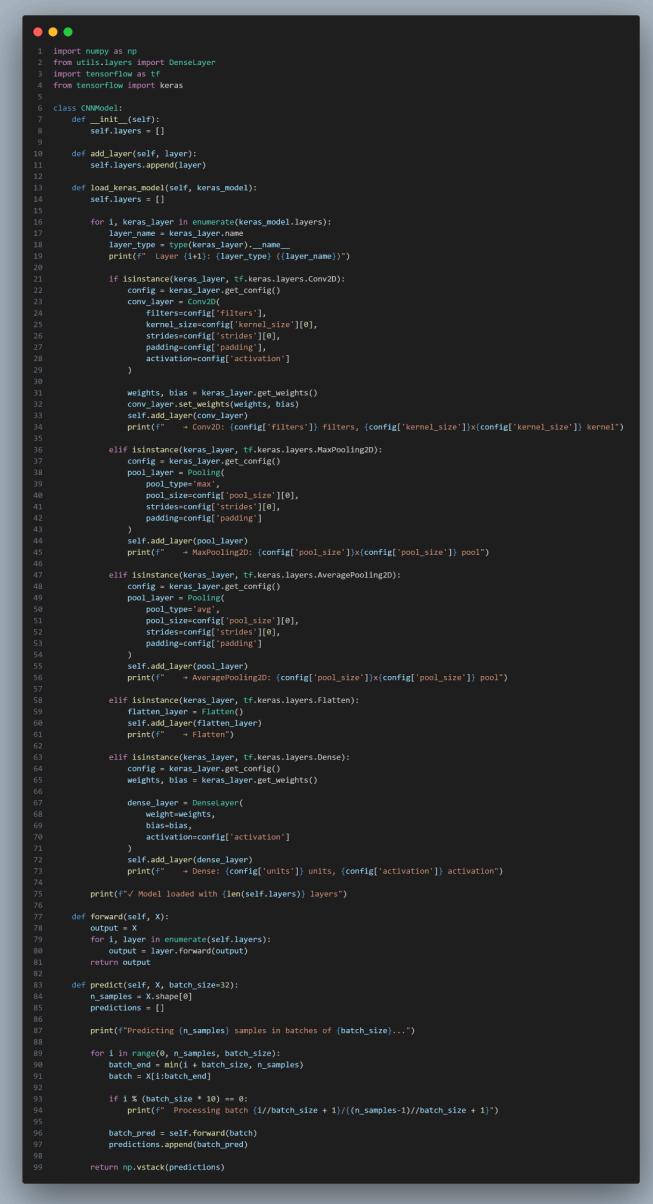
```

```

● ● ●
1  class LSTMCell:
2      """
3          Represents a single LSTM cell.
4          Takes input x_t, h_prev, and c_prev,
5          and outputs h_t and c_t.
6      """
7      def __init__(self, Wi, WF, Wo, Wc,
8                   Uf, Uo, Uc,
9                   bi, bf, bo, bc):
10         # init weights and biases
11         self.Wi, self.WF, self.Wo, self.Wc = Wi, WF, Wo, Wc
12         self.Ui, self.UF, self.Uo, self.Uc = Ui, UF, Uo, Uc
13         self.bi, self.bf, self.bo, self.bc = bi, bf, bo, bc
14
15     def forward(self, x_t, h_prev, c_prev):
16         # input gate
17         i_t = ActivationFunction.sigmoid(
18             np.dot(x_t, self.Wi) + np.dot(h_prev, self.Ui) + self.bi
19         )
20
21         # forget gate
22         f_t = ActivationFunction.sigmoid(
23             np.dot(x_t, self.WF) + np.dot(h_prev, self.UF) + self.bf
24         )
25
26         # output gate
27         o_t = ActivationFunction.sigmoid(
28             np.dot(x_t, self.Wo) + np.dot(h_prev, self.Uo) + self.bo
29         )
30
31         # candidate cell state
32         c_hat_t = ActivationFunction.tanh(
33             np.dot(x_t, self.Wc) + np.dot(h_prev, self.Uc) + self.bc
34         )
35
36         # new cell state
37         c_t = f_t * c_prev + i_t * c_hat_t
38
39         # new hidden state
40         h_t = o_t * ActivationFunction.tanh(c_t)
41
42         return h_t, c_t
43
44     class Dropoutlayer:
45         def __init__(self, rate=0):
46             self.rate = rate
47             self.mask = None
48             self.is_training = True
49
50         def setrate(self, rate):
51             self.rate = rate
52
53         def setis_training(self, is_training):
54             self.is_training = is_training
55
56         def forward(self, x):
57             if self.is_training and self.rate > 0:
58                 self.mask = (np.random.rand(*x.shape) > self.rate).astype(np.float32)
59                 return (x * self.mask) / (1.0 - self.rate)
60             return x

```

3.1.2. Implementasi CNN

cnn_model.py	
<p>File ini berisi kelas-kelas yang mendefinisikan model CNN <i>from scratch</i> (hanya menggunakan perhitungan numpy), yang mencakup implementasi beberapa layer utama, yaitu layer konvolusi, pooling, dan flatten dengan perhitungan matematis, serta kelas untuk implementasi model CNN menggunakan library keras.</p>	
Kelas CNNModel mendefinisikan fungsi-fungsi serta atribut yang dapat me-load bobot hasil training model CNN dengan keras.	 <pre> 1 import numpy as np 2 from utils.layers import DenseLayer 3 import tensorflow as tf 4 from tensorflow import keras 5 6 class CNNModel: 7 def __init__(self): 8 self.layers = [] 9 10 def add_layer(self, layer): 11 self.layers.append(layer) 12 13 def load_keras_model(self, keras_model): 14 self.layers = [] 15 16 for i, keras_layer in enumerate(keras_model.layers): 17 layer_name = keras_layer.name 18 layer_type = type(keras_layer).__name__ 19 print(f" - Layer {i+1}: {layer_type} ({layer_name})") 20 21 if isinstance(keras_layer, tf.keras.layers.Conv2D): 22 config = keras_layer.get_config() 23 filters=config['filters'], 24 kernel_size=config['kernel_size'][0], 25 strides=config['strides'][0], 26 padding=config['padding'], 27 activation=config['activation'] 28) 29 30 weights, bias = keras_layer.get_weights() 31 conv_layer.set_weights(weights, bias) 32 self.add_layer(conv_layer) 33 print(f" - Conv2D: {config['filters']} filters, {config['kernel_size']}x{config['kernel_size']} kernel") 34 35 36 elif isinstance(keras_layer, tf.keras.layers.MaxPooling2D): 37 config = keras_layer.get_config() 38 pool_layer = pooling(39 pool_type='max', 40 pool_size=config['pool_size'][0], 41 strides=config['strides'][0], 42 padding=config['padding'] 43) 44 45 self.add_layer(pool_layer) 46 print(f" - MaxPooling2D: {config['pool_size']}x{config['pool_size']} pool") 47 48 elif isinstance(keras_layer, tf.keras.layers.AveragePooling2D): 49 config = keras_layer.get_config() 50 pool_layer = pooling(51 pool_type='avg', 52 pool_size=config['pool_size'][0], 53 strides=config['strides'][0], 54 padding=config['padding'] 55) 56 57 self.add_layer(pool_layer) 58 print(f" - AveragePooling2D: {config['pool_size']}x{config['pool_size']} pool") 59 60 elif isinstance(keras_layer, tf.keras.layers.Flatten): 61 flatten_layer = flatten() 62 self.add_layer(flatten_layer) 63 print(f" - flatten") 64 65 elif isinstance(keras_layer, tf.keras.layers.Dense): 66 config = keras_layer.get_config() 67 weights, bias = keras_layer.get_weights() 68 69 dense_layer = DenseLayer(70 weight=weights, 71 bias=bias, 72 activation=config['activation'] 73) 74 75 self.add_layer(dense_layer) 76 print(f" - Dense: {config['units']} units, {config['activation']} activation") 77 78 print(f"~ Model loaded with {len(self.layers)} layers") 79 80 def forward(self, X): 81 output = X 82 for i, layer in enumerate(self.layers): 83 output = layer.forward(output) 84 return output 85 86 def predict(self, X, batch_size=32): 87 n_samples = X.shape[0] 88 predictions = [] 89 90 print(f"Predicting {n_samples} samples in batches of {batch_size}...") 91 92 for i in range(0, n_samples, batch_size): 93 batch_end = min(i + batch_size, n_samples) 94 batch = X[i:batch_end] 95 96 if i % (batch_size * 10) == 0: 97 print(f" - Processing batch {i//batch_size + 1}/({n_samples-i}//batch_size + 1)") 98 99 batch_pred = self.forward(batch) 100 predictions.append(batch_pred) 101 102 return np.vstack(predictions)</pre>

Kelas Conv2D mendefinisikan layer yang berisi proses perhitungan lengkap pada tahap konvolusi, mulai dari tahapan ekstraksi fitur pada data input menggunakan kernel (filter), hingga tahap detektor menggunakan fungsi aktivasi relu.

```

1 class Conv2D:
2     def __init__(self, filters, kernel_size, strides, padding, activation='relu'):
3         self.filters = filters
4         self.kernel_size = kernel_size
5         self.strides = strides
6         self.padding = padding
7         self.activation = activation
8         self.kernel = None
9         self.bias = None
10
11    def set_weights(self, weights, bias):
12        self.weights = weights
13        self.bias = bias
14
15    def _pad_input(self, X):
16        if self.padding == 'same':
17            pad_h = (self.kernel_size - 1) // 2
18            pad_w = (self.kernel_size - 1) // 2
19            return np.pad(X, ((0, 0), (pad_h, pad_h), (pad_w, pad_w), (0, 0)), mode='constant')
20        return X
21
22    def _activation(self, x):
23        if self.activation == 'relu':
24            return np.maximum(0, x)
25        return x
26
27    def forward(self, X):
28        X_padded = self._pad_input(X)
29        batch_size, H, W, C_in = X_padded.shape
30        K_h, K_w, C_in_W, C_out = self.weights.shape
31
32        out_H = (H - K_h) // self.strides + 1
33        out_W = (W - K_w) // self.strides + 1
34
35        output = np.zeros((batch_size, out_H, out_W, C_out))
36
37        for i in range(out_H):
38            for j in range(out_W):
39                h_start = i * self.strides
40                h_end = h_start + K_h
41                w_start = j * self.strides
42                w_end = w_start + K_w
43
44                patch = X_padded[:, h_start:h_end, w_start:w_end, :]
45
46                for f in range(C_out):
47                    output[:, i, j, f] = np.sum(patch * self.weights[:, :, :, f], axis=(1, 2, 3)) + self.bias[f]
48
49        return self._activation(output)

```

Kelas Pooling mendefinisikan layer yang berisi proses pada tahap pooling, yang terbagi atas dua jenis pooling, yaitu max pooling dan avg (average) pooling.

```

1 class Pooling:
2     def __init__(self, pool_type='max', pool_size=2, strides=2, padding='valid'):
3         self.pool_type = pool_type
4         self.pool_size = pool_size
5         self.strides = strides
6         self.padding = padding
7
8     def _pad_input(self, X):
9         if self.padding == 'same':
10            pad_h = (self.pool_size - 1) // 2
11            pad_w = (self.pool_size - 1) // 2
12            return np.pad(X, ((0, 0), (pad_h, pad_h), (pad_w, pad_w), (0, 0)), mode='constant')
13        return X
14
15    def forward(self, X):
16        X_padded = self._pad_input(X)
17        batch_size, H, W, C = X_padded.shape
18
19        out_H = (H - self.pool_size) // self.strides + 1
20        out_W = (W - self.pool_size) // self.strides + 1
21
22        output = np.zeros((batch_size, out_H, out_W, C))
23
24        for i in range(out_H):
25            for j in range(out_W):
26                h_start = i * self.strides
27                h_end = h_start + self.pool_size
28                w_start = j * self.strides
29                w_end = w_start + self.pool_size
30
31                patch = X_padded[:, h_start:h_end, w_start:w_end, :]
32
33                if self.pool_type == 'max':
34                    output[:, i, j, :] = np.max(patch, axis=(1, 2))
35                else:
36                    output[:, i, j, :] = np.mean(patch, axis=(1, 2))
37
38        return output
39
40    class Flatten:
41        def __init__(self):
42            pass
43
44        def forward(self, X):
45            batch_size = X.shape[0]
46            return X.reshape(batch_size, -1)

```

experiment_cnn.py

File ini berisi beberapa jenis pelatihan model CNN untuk menganalisis pengaruh berbagai hyperparameter dalam CNN, seperti jumlah layer konvolusi, jumlah filter per layer konvolusi, ukuran filter per layer konvolusi, serta jenis pooling layer yang digunakan. Setiap analisis pengaruh tersebut mencakup 2-3 variasi.

Kelas CNNKeras mendefinisikan fungsi-fungsi yang berkaitan dengan eksperimen variasi hyperparameter pada model keras.

```
 1  class CNNKeras:
 2      def __init__(self):
 3          # load cifar10 data
 4          self.data_loader = DataLoader('cifar10')
 5          self.x_train, self.x_val, self.x_test, self.y_train, self.y_val, self.y_test = self.data_loader.load_data()
 6
 7          # normalize data
 8          self.x_train = self.x_train.astype('float32') / 255.0
 9          self.x_val = self.x_val.astype('float32') / 255.0
10          self.x_test = self.x_test.astype('float32') / 255.0
11
12          print(f'Dataset loaded:')
13          print(f' Training: {self.x_train.shape} - {len(self.y_train)} samples')
14          print(f' Validation: {self.x_val.shape} - {len(self.y_val)} samples')
15          print(f' Test: {self.x_test.shape} - {len(self.y_test)} samples')
16
17      def base_model(self, num_conv_layers=2, filters=[32, 64], kernel_sizes=[3, 3], pooling_type='max'):
18          model = keras.Sequential()
19
20          # conv layer pertama
21          model.add(layers.Conv2D(filters[0], kernel_sizes[0], activation='relu',
22                                 input_shape=(32, 32, 3), padding='same', name='conv2d_1'))
23
24          if pooling_type == 'max':
25              model.add(layers.MaxPooling2D((2, 2), name='max_pooling2d_1'))
26          else:
27              model.add(layers.AveragePooling2D((2, 2), name='average_pooling2d_1'))
28
29          # handle kalau ada lebih dari satu layer konvolusi
30          for i in range(1, num_conv_layers):
31              filter_idx = min(i, len(filters) - 1)
32              kernel_idx = min(1, len(kernel_sizes) - 1)
33
34              model.add(layers.Conv2D(filters[filter_idx], kernel_sizes[kernel_idx],
35                                     activation='relu', padding='same', name=f'conv2d_{i+1}'))
36
37          if pooling_type == 'max':
38              model.add(layers.MaxPooling2D((2, 2), name=f'max_pooling2d_{i+1}'))
39          else:
40              model.add(layers.AveragePooling2D((2, 2), name=f'average_pooling2d_{i+1}'))
41
42          # Classifier
43          model.add(layers.Flatten(name='flatten'))
44          model.add(layers.Dense(64, activation='relu', name='dense_1'))
45          model.add(layers.Dense(10, activation='softmax', name='dense_2'))
46
47      return model
```

Fungsi
train_and_evaluate() yang
melakukan train
terhadap data
train dan evaluasi
terhadap data val
dan test.

```
1 def train_and_evaluate(self, model, model_name, epochs=20):
2     print(f"\n{'='*60}")
3     print(f"Training: {model_name}")
4     print(f"{'='*60}")
5
6     model.compile(
7         optimizer='adam',
8         loss='sparse_categorical_crossentropy',
9         metrics=['accuracy']
10    )
11
12     history = model.fit(
13         self.x_train, self.y_train,
14         batch_size=32,
15         epochs=epochs,
16         validation_data=(self.x_val, self.y_val),
17         verbose=1
18    )
19
20     test_loss, test_accuracy = model.evaluate(self.x_test, self.y_test, verbose=0)
21
22     y_pred_proba = model.predict(self.x_test, verbose=0)
23     y_pred = np.argmax(y_pred_proba, axis=1)
24     f1_macro = f1_score(self.y_test, y_pred, average='macro')
25
26     # simpan model beserta si bobot
27     model.save(f'{model_name}.h5')
28
29     print(f"Results for {model_name}:")
30     print(f" Test Accuracy: {test_accuracy:.4f}")
31     print(f" Test F1-Score (macro): {f1_macro:.4f}")
32
33     return {
34         'model': model,
35         'history': history,
36         'test_accuracy': test_accuracy,
37         'f1_score': f1_macro,
38         'predictions': y_pred,
39         'model_name': model_name
40     }
```

Fungsi untuk
melakukan
eksperimen layer
konvolusi.

```
1 # eksperimen layer konvolusi
2 def experiment_conv_layers(self, epochs=20):
3     print(f"\n{'='*80}")
4     print("EXPERIMENT 1: PENGARUH JUMLAH LAYER KONVOLUSI")
5     print(f"{'='*80}")
6
7     configs = [
8         (1, [32], [3], "1_conv_layer"),
9         (2, [32, 64], [3, 3], "2_conv_layers"),
10        (3, [32, 64, 128], [3, 3, 3], "3_conv_layers")
11    ]
12
13     results = {}
14     for num_layers, filters, kernels, name in configs:
15         model = self.create_base_model(num_layers, filters, kernels, 'max')
16         results[name] = self.train_and_evaluate(model, name, epochs)
17
18     self.plot_comparison(results, "Number of Convolutional Layers")
19     self.analyze_conv_layers(results)
20     return results
```

Fungsi untuk melakukan eksperimen jumlah filters.

```
● ● ●
1 def experiment_filters(self, epochs=20):
2     print(f"\n{'#'*80}")
3     print("EXPERIMENT 2: PENGARUH BANYAK FILTER PER LAYER")
4     print(f"{'#'*80}")
5
6     configs = [
7         ([16, 32], "filters_16_32"),
8         ([32, 64], "filters_32_64"),
9         ([64, 128], "filters_64_128")
10    ]
11
12     results = {}
13     for filters, name in configs:
14         model = self.create_base_model(2, filters, [3, 3], 'max')
15         results[name] = self.train_and_evaluate(model, name, epochs)
16
17     self.plot_comparison(results, "Number of Filters")
18     self.analyze_filters(results)
19
return results
```

Fungsi untuk melakukan eksperimen ukuran filter.

```
● ● ●
1 def experiment_kernel_sizes(self, epochs=20):
2     print(f"\n{'#'*80}")
3     print("EXPERIMENT 3: PENGARUH UKURAN FILTER (KERNEL)")
4     print(f"{'#'*80}")
5
6     configs = [
7         ([3, 3], "kernel_3x3"),
8         ([5, 5], "kernel_5x5"),
9         ([3, 5], "kernel_3x5_mixed")
10    ]
11
12     results = {}
13     for kernels, name in configs:
14         model = self.create_base_model(2, [32, 64], kernels, 'max')
15         results[name] = self.train_and_evaluate(model, name, epochs)
16
17     self.plot_comparison(results, "Kernel Sizes")
18     self.analyze_kernels(results)
19
return results
```

Fungsi untuk melakukan eksperimen terhadap dua jenis pooling yang berbeda, yaitu max pooling dan avg pooling.

```
● ● ●
1 def experiment_pooling(self, epochs=20):
2     print(f"\n{'#'*80}")
3     print("EXPERIMENT 4: PENGARUH JENIS POOLING LAYER")
4     print(f"{'#'*80}")
5
6     configs = [
7         ('max', "max_pooling"),
8         ('avg', "avg_pooling")
9     ]
10
11     results = {}
12     for pooling, name in configs:
13         model = self.create_base_model(2, [32, 64], [3, 3], pooling)
14         results[name] = self.train_and_evaluate(model, name, epochs)
15
16     self.plot_comparison(results, "Pooling Types")
17     self.analyze_pooling(results)
18
return results
```

Fungsi untuk melakukan plot comparison terhadap variasi hyperparameter.

```
1 def plot_comparison(self, results, experiment_name):
2     plt.figure(figsize=(15, 5))
3
4     # Training & Validation Loss
5     plt.subplot(1, 3, 1)
6     for name, result in results.items():
7         history = result['history']
8         plt.plot(history.history['loss'], label=f'{name} (train)', linewidth=2)
9         plt.plot(history.history['val_loss'], label=f'{name} (val)', linestyle='--', linewidth=2)
10    plt.title(f'{experiment_name} - Loss')
11    plt.xlabel('Epoch')
12    plt.ylabel('Loss')
13    plt.legend()
14    plt.grid(True, alpha=0.3)
15
16    # Training & Validation Accuracy
17    plt.subplot(1, 3, 2)
18    for name, result in results.items():
19        history = result['history']
20        plt.plot(history.history['accuracy'], label=f'{name} (train)', linewidth=2)
21        plt.plot(history.history['val_accuracy'], label=f'{name} (val)', linestyle='--', linewidth=2)
22    plt.title(f'{experiment_name} - Accuracy')
23    plt.xlabel('Epoch')
24    plt.ylabel('Accuracy')
25    plt.legend()
26    plt.grid(True, alpha=0.3)
27
28    # F1-Score Comparison
29    plt.subplot(1, 3, 3)
30    names = list(results.keys())
31    f1_scores = [results[name]['f1_score'] for name in names]
32    bars = plt.bar(names, f1_scores, alpha=0.7)
33    plt.title(f'{experiment_name} - F1-Score')
34    plt.ylabel('F1-Score (macro)')
35    plt.xticks(rotation=45)
36
37    # Add value labels on bars
38    for bar, f1 in zip(bars, f1_scores):
39        plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.005,
40                 f'{f1:.3f}', ha='center', va='bottom')
41
42    plt.tight_layout()
43    plt.savefig(f'{experiment_name.lower().replace(" ", "_")}_comparison.png', dpi=300, bbox_inches='tight')
44    plt.show()
```

Fungsi untuk menganalisis hasil yang diperoleh berdasarkan eksperimen terhadap hyperparameter yang berbeda, termasuk menentukan jenis hyperparameter yang memiliki performa terbaik ataupun terburuk.

```
1 def analyze_conv_layers(self, results):
2     print(f"\n{'='*60}")
3     print("ANALISIS: PENGARUH JUMLAH LAYER KONVOLUSI")
4     print(f"{'='*60}")
5
6     for name, result in results.items():
7         layers = name.split('_')[0]
8         print(f'{layers} layer(s): F1-Score = {result['f1_score']:.4f}")
9
10    best = max(results.items(), key=lambda x: x[1]['f1_score'])
11    worst = min(results.items(), key=lambda x: x[1]['f1_score'])
12
13    print(f"\nKESIMPULAN:")
14    print(f"- Performa terbaik: {best[0]} (F1-Score: {best[1]['f1_score']:.4f})")
15    print(f"- Performa terburuk: {worst[0]} (F1-Score: {worst[1]['f1_score']:.4f})")
16
17 def analyze_filters(self, results):
18     print(f"\n{'='*60}")
19     print("ANALISIS: PENGARUH BANYAK FILTER PER LAYER")
20     print(f"{'='*60}")
21
22     for name, result in results.items():
23         filters = name.replace('filters_', '').replace('_', '-')
24         print(f'Filter {filters}: F1-Score = {result['f1_score']:.4f}")
25
26     best = max(results.items(), key=lambda x: x[1]['f1_score'])
27     worst = min(results.items(), key=lambda x: x[1]['f1_score'])
28
29     print(f"\nKESIMPULAN:")
30     print(f"- Performa terbaik: {best[0]} (F1-Score: {best[1]['f1_score']:.4f})")
31     print(f"- Performa terburuk: {worst[0]} (F1-Score: {worst[1]['f1_score']:.4f})")
32
33 def analyze_kernels(self, results):
34     print(f"\n{'='*60}")
35     print("ANALISIS: PENGARUH UKURAN FILTER (KERNEL)")
36     print(f"{'='*60}")
37
38     for name, result in results.items():
39         kernel = name.replace('kernel_', '').replace('_', '(mixed)')
40         print(f'Kernel {kernel}: F1-Score = {result['f1_score']:.4f}")
41
42     best = max(results.items(), key=lambda x: x[1]['f1_score'])
43     worst = min(results.items(), key=lambda x: x[1]['f1_score'])
44
45     print(f"\nKESIMPULAN:")
46     print(f"- Performa terbaik: {best[0]} (F1-Score: {best[1]['f1_score']:.4f})")
47     print(f"- Performa terburuk: {worst[0]} (F1-Score: {worst[1]['f1_score']:.4f})")
48
49 def analyze_pooling(self, results):
50     print(f"\n{'='*60}")
51     print("ANALISIS: PENGARUH JENIS POOLING LAYER")
52     print(f"{'='*60}")
53
54     for name, result in results.items():
55         pooling = name.replace('_pooling', ' pooling')
56         print(f'(pooling.capitalize()): F1-Score = {result['f1_score']:.4f}")
57
58     best = max(results.items(), key=lambda x: x[1]['f1_score'])
59     worst = min(results.items(), key=lambda x: x[1]['f1_score'])
60
61     print(f"\nKESIMPULAN:")
62     print(f"- Performa terbaik: {best[0]} (F1-Score: {best[1]['f1_score']:.4f})")
63     print(f"- Performa terburuk: {worst[0]} (F1-Score: {worst[1]['f1_score']:.4f})")
```

Fungsi main untuk menjalankan semua eksperimen terhadap variasi hyperparameter.

```
1 def main():
2     print("*80")
3     print("CNN HYPERPARAMETER EXPERIMENTS - CIFAR-10")
4     print("*80")
5
6     np.random.seed(42)
7     tf.random.set_seed(42)
8
9     experiments = CNNKeras()
10
11    results = experiments.run_all_experiments(epochs=20)
12
13 if __name__ == "__main__":
14     main()
```

test_cnn.py

File ini berisi kelas dan fungsi-fungsi untuk memastikan implementasi CNN from scratch sudah tepat. Kelas dan fungsi-fungsi tersebut termasuk fungsi untuk me-load bobot yang telah dipelajari di model keras, serta perbandingan F1-Score antar dua model.

Kelas

CNNTester yang berisi fungsi-fungsi untuk me-load bobot yang telah dipelajari di modul keras ke model CNN from scratch yang telah dibuat, hingga validasi F1-score antar dua model yang berbeda.

F1-score antar dua model yang berbeda

berbeda

```

1 from tensorflow import Conv2D, Flatten, Pooling, CNNModel
2 import tensorflow as tf
3 from tensorflow import keras
4 from tensorflow import layers
5 from utilsls_data_loader import DataLoader
6 from sklearn.metrics import f1_score
7 import numpy as np
8
9 class ONTestError:
10     def __init__(self):
11         self.model_loader = DataLoader('mnist10')
12         self.x_train, self.y_train = self.model_loader.load_data()
13         self.x_test, self.y_test = self.model_loader.load_data()
14         self.y_test = self.y_test.astype('float32') / 255.0
15         print("test data loaded: (self.y_test.shape)")
16
17     def validate_model(self, model_path, test_samples=500):
18         """Validate from-scratch implementation against Keras model"""
19         keras_model = keras.models.load_model(model_path)
20         print("Keras model loaded: (model_path)")
21         print("VALIDATING MODEL: (model_path)")
22         print("(-1980)")
23
24     @load_keras_model
25     def validate(self, model_path, test_samples=500):
26         """Validate from-scratch implementation against Keras model"""
27         keras_model = keras.models.load_model(model_path)
28         print("Keras model loaded: (model_path)")
29         except IOError as e:
30             print("Failed to load keras model: [e]")
31         return None
32
33     def load_keras_model():
34         try:
35             keras_model = keras.models.load_model(model_path)
36             print("Keras model loaded: (model_path)")
37         except IOError as e:
38             print("Failed to load keras model: [e]")
39         return None
40
41     def print_model_summary():
42         keras.model.summary()
43
44     def create_fromscratch_model():
45         scratch_model = CNNModel()
46         scratch_model.load_keras_model(keras_model)
47
48     # Use subset for testing
49     x_test_subset = self.x_test[:test_samples]
50     y_test_subset = self.y_test[:test_samples]
51
52     print("Working with (test_samples) samples...")
53
54     # Get Keras predictions
55     print("Getting Keras predictions...")
56     keras_pred = keras_model.predict(x_test_subset, batch_size=32, verbose=0)
57     keras_classes = np.argmax(keras_pred, axis=1)
58     keras_f1 = f1_score(y_test_subset, keras_classes, average='macro')
59
60     # Get from-scratch predictions
61     print("Predicting from-scratch predictions...")
62     scratch_pred = scratch_model.predict(x_test_subset, batch_size=32, verbose=0)
63     scratch_classes = np.argmax(scratch_pred, axis=1)
64     scratch_f1 = f1_score(y_test_subset, scratch_classes, average='macro')
65
66     # Compare results
67     max_diff = np.max(np.abs(keras_pred - scratch_pred)) ** 2
68     max_err = np.max(np.abs(keras_pred - scratch_pred))
69     identical_predictions = np.array_equal(keras_classes, scratch_classes)
70
71     # Print detailed comparison for first few samples
72     print("Detailed comparison for first few samples")
73     print("Keras vs scratch predictions comparison (first 5 samples):")
74     print("preds = keras pred | scratch pred | keras class | scratch class | true label")
75     print("-" * 75)
76     for i in range(5, len(keras_pred)):
77         k_pred = keras_pred[i]
78         s_pred = scratch_pred[i]
79         k_class = keras_classes[i]
80         s_class = scratch_classes[i]
81         true_label = y_test_subset[i]
82         print("%s | %s | %s | %s | %s" % (k_pred[0:k], k_pred[1:5], k_class, s_class, true_label))
83
84     # Print status
85     total = len(keras_pred)
86     total -= 5
87
88     if max_diff < tolerance and identical_predictions:
89         status = "PASSED"
90     elif max_diff < tolerance + 0.01:
91         status = "PASSED with small differences - Implementation is mostly correct"
92     else:
93         status = "WARNING FAILED - Implementation needs debugging!"
94
95     print(status)
96
97     return {
98         'model_name': model_path,
99         'mse_diff': mse_diff,
100        'max_diff': max_diff,
101        'max_err': max_err,
102        'scratch_f1': scratch_f1,
103        'identical': identical_predictions,
104        'status': status
105    }
106
107     def validate_all_models(self, test_samples=500):
108         """Validate all models in the current directory"""
109         print("Validating all TRAINING MODELS")
110         print("(-1980)")
111
112     import os
113     model_files = [f for f in os.listdir('.') if f.endswith('.h5')]
114
115     if not model_files:
116         print("No .h5 model files found!")
117         print("Please run the training script first: python complete_cnn_implementation.py")
118         return {}
119
120     print("Found (%d) model files: %s" % (len(model_files), model_files))
121     for f in model_files:
122         print("(-%s)" % f)
123
124     results = {}
125     for model_file in model_files:
126         try:
127             result = self.validate_model(model_file, test_samples)
128         except Exception as e:
129             print("Error validating (%s): (%s)" % (model_file, e))
130             print("Traceback")
131             traceback.print_exc()
132
133     self.print_validation_summary(results)
134
135     return results
136
137
138     def print_validation_summary(self, results):
139         """Print summary of all validations"""
140         print("(-1980)")
141         print("Total successful validation (%d)" % len(results))
142         print("Passed validation (%d)" % len([r for r in results.values() if r['status'] == 'PASSED']))
143         print("Failed validation (%d)" % len([r for r in results.values() if r['status'] != 'PASSED']))
144         print("(-1980)")
145
146     if not results:
147         print("No successful validations.")
148         return
149
150     passed = sum([r for r in results.values() if r['status'] == 'PASSED'])
151     total = len(results)
152
153     print("Total successful validation (%d)" % passed)
154     print("Passed validation (%d)" % passed)
155     print("Failed validation (%d)" % (total - passed))
156     print("Success rate (%.2f)" % (passed/total*100.0))
157
158     print("OVERALL RESULTS")
159     for model_name, result in results.items():
160         print("(-%s)" % model_name)
161         print("MSE difference: (%s)" % result['mse_diff'])
162         print("Max difference: (%s)" % result['max_diff'])
163         print("(-1 differences (%s))" % abs(result['keras_f1']) - result['scratch_f1'])
164         print("Identical predictions (%s)" % identical_predictions)
165
166     if passed == total:
167         print("All models passed validation!")
168         print("ONN from-scratch implementation is CORRECT!")
169     else:
170         print("No new models failed validation!")
171         print("Please check the implementation (or debugging).")
172

```

Fungsi quick_test sebagai test cepat terhadap modul yang berbeda, untuk memastikan bahwa model from scratch terimplementasi dengan baik pada kuantitas data test yang sedikit.

```

1 def quick_test():
2     print("=-*60")
3     print("QUICK TEST - Creating and testing a simple model")
4     print("=-*60")
5
6     # Create simple test model
7     model = keras.Sequential([
8         keras.layers.Conv2D(8, 3, activation='relu', input_shape=(32, 32, 3), padding='same'),
9         keras.layers.MaxPooling2D(2),
10        keras.layers.Flatten(),
11        keras.layers.Dense(10, activation='softmax')
12    ])
13
14     model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
15
16     # Create dummy data
17     x_dummy = np.random.randint(0, 10, 10)
18     y_dummy = np.random.randint(0, 10, 10)
19
20     # Quick training
21     print("Training simple model...")
22     model.fit(x_dummy, y_dummy, epochs=1, verbose=0)
23
24     # Test implementation
25     print("Testing implementation...")
26     validator = CNNTester()
27
28     # Use small subset for quick test
29     x_test_small = validator.x_test[:50]
30
31     # Keras prediction
32     keras_pred = model.predict(x_test_small, verbose=0)
33
34     # From-scratch prediction
35     scratch_model = CNNModel()
36     scratch_model.load_keras_model(model)
37     scratch_pred = scratch_model.predict(x_test_small, batch_size=10)
38
39     # Compare
40     mse_diff = np.mean((keras_pred - scratch_pred) ** 2)
41     max_diff = np.max(np.abs(keras_pred - scratch_pred))
42
43     print(f"MSE difference: {mse_diff:.10f}")
44     print(f"Max difference: {max_diff:.10f}")
45
46     if mse_diff < 1e-6:
47         print("Quick test PASSED!")
48         return True
49     else:
50         print("Quick test FAILED!")
51         return False

```

Fungsi main untuk menjalankan validasi terhadap setiap modul yang berbeda.

```

1 def main():
2     print("=-*80")
3     print("CNN FROM SCRATCH - FINAL VALIDATION")
4     print("=-*80")
5
6     # Quick test first
7     print("Running quick test...")
8     if not quick_test():
9         print("Quick test failed. Please check implementation.")
10        return
11
12     print("\n" + "=-*80")
13     print("Quick test passed! Proceeding with full validation...")
14     print("=-*80")
15
16     # Full validation
17     validator = CNNTester()
18     results = validator.validate_all_models(test_samples=1000)
19
20     print(f"\n{'=-*80}'")
21     print("VALIDATION COMPLETED!")
22     print(f"{'=-*80}'")
23
24 if __name__ == "__main__":
25     main()

```

3.1.3. Implementasi SimpleRNN

rnn_model.py	
Kelas RNNModel	
Kelas RNNModel merupakan abstraksi utama yang digunakan untuk melatih model RNN menggunakan Keras, menyimpan dan mengekstrak bobot dari model yang telah dilatih, membangun dan menjalankan implementasi RNN dari nol (from scratch), serta membandingkan kinerja model Keras dan from-scratch menggunakan metrik seperti F1-score.	
Inisialisasi kelas RNNModel dengan atribut terkait.	<pre>● ● ● 1 class RNNModel: 2 def __init__(self, vocab_size, embedding_dim, max_length, num_classes): 3 self.vocab_size = vocab_size 4 self.embedding_dim = embedding_dim 5 self.max_length = max_length 6 self.num_classes = num_classes 7 self.keras_model = None 8 self.scratch_model = None 9 self.weights = {}</pre>

Membuat dan melatih model RNN menggunakan library Keras.

Proses dimulai dengan membuat arsitektur layer-layer yang terdiri dari Embedding, SimpleRNN (yang bisa unidirectional atau bidirectional), Dropout, dan Dense.

```

1  def train_keras_model(self, x_train, y_train, x_val, y_val,
2                         rnn_layers=2, rnn_units=[64, 32],
3                         bidirectional=True, dropout_rate=0.2,
4                         epochs=5, batch_size=32):
5
6     y_train_cat = to_categorical(y_train, num_classes=self.num_classes)
7     y_val_cat = to_categorical(y_val, num_classes=self.num_classes)
8
9     model = Sequential()
10
11    model.add(Embedding(input_dim=self.vocab_size,
12                          output_dim=self.embedding_dim,
13                          input_length=self.max_length))
14
15    # RNN layers
16    for i in range(rnn_layers):
17        return_sequences = (i < rnn_layers - 1) # Only last layer doesn't return sequences
18
19        if bidirectional:
20            model.add(Bidirectional(
21                SimpleRNN(rnn_units[i],
22                           return_sequences=return_sequences,
23                           activation='tanh'),
24                merge_mode='concat'
25            ))
26        else:
27            model.add(SimpleRNN(rnn_units[i],
28                               return_sequences=return_sequences,
29                               activation='tanh'))
30
31        if dropout_rate > 0:
32            model.add(Dropout(dropout_rate))
33
34    # Output layer
35    model.add(Dense(self.num_classes, activation='softmax'))
36
37    # Compile model
38    model.compile(optimizer=Adam(learning_rate=0.001),
39                   loss='categorical_crossentropy',
40                   metrics=['accuracy'])
41
42    print("Keras Model Architecture:")
43    model.summary()
44
45    # Train model
46    history = model.fit(x_train, y_train_cat,
47                          validation_data=(x_val, y_val_cat),
48                          epochs=epochs,
49                          batch_size=batch_size,
50                          verbose=1)
51
52    self.keras_model = model
53    return history
54

```

Menyimpan bobot model Keras ke dalam file eksternal dengan format .h5.

```

1  def save_model_weights(self, filepath):
2      if self.keras_model is not None:
3          self.keras_model.save_weights(filepath)
4          print(f"Model weights saved to {filepath}")
5      else:
6          raise ValueError("Keras model is not trained yet.")

```

Mengambil semua bobot layer dari model Keras dan menyimpannya ke dalam dictionary

```
● ● ●
1 def extract_weights_from_keras(self, rnn_layers=2,
2     rnn_units=[64, 32], bidirectional=True):
3     if self.keras_model is None:
4         raise ValueError("Keras model is not trained yet.")
5
6     weights = {}
7     layer_idx = 0
8
9     # Extract embedding weights
10    embedding_layer = self.keras_model.layers[layer_idx]
11    weights['embedding'] = embedding_layer.get_weights()[0]
12    layer_idx += 1
13
14    # Extract RNN weights
15    for i in range(rnn_layers):
16        if bidirectional:
17            # Bidirectional layer
18            bi_layer = self.keras_model.layers[layer_idx]
19            forward_weights = bi_layer.forward_layer.get_weights()
20            backward_weights = bi_layer.backward_layer.get_weights()
21
22            weights[f'rnn_{i}_forward'] = {
23                'kernel': forward_weights[0], # Wx
24                'recurrent_kernel': forward_weights[1], # Wh
25                'bias': forward_weights[2] # b
26            }
27            weights[f'rnn_{i}_backward'] = {
28                'kernel': backward_weights[0],
29                'recurrent_kernel': backward_weights[1],
30                'bias': backward_weights[2]
31            }
32        else:
33            # Regular RNN layer
34            rnn_layer = self.keras_model.layers[layer_idx]
35            rnn_weights = rnn_layer.get_weights()
36            weights[f'rnn_{i}'] = {
37                'kernel': rnn_weights[0],
38                'recurrent_kernel': rnn_weights[1],
39                'bias': rnn_weights[2]
40            }
41
42        layer_idx += 1
43
44        # Skip dropout layer if present
45        if layer_idx < len(self.keras_model.layers) and
46        'dropout' in self.keras_model.layers[layer_idx].name.lower():
47            layer_idx += 1
48
49    # Extract dense layer weights
50    dense_layer = self.keras_model.layers[-1]
51    dense_weights = dense_layer.get_weights()
52    weights['dense'] = {
53        'kernel': dense_weights[0],
54        'bias': dense_weights[1]
55    }
56
57    self.weights = weights
58    return weights
```

Membangun ulang model RNN dari scratch tanpa menggunakan Keras, dengan memanfaatkan bobot hasil pelatihan model Keras

```
1 def rnn_scratch(self, rnn_layers=2, rnn_units=[64, 32], bidirectional=True):
2
3     self.extract_weights_from_keras(rnn_layers, rnn_units, bidirectional)
4
5     self.scratch_model = RNNScratchModel(
6         weights=self.weights,
7         rnn_layers=rnn_layers,
8         rnn_units=rnn_units,
9         bidirectional=bidirectional,
10        max_length=self.max_length,
11        num_classes=self.num_classes
12    )
13
14    print("RNN from scratch model created successfully!")
```

Menilai performa model baik Keras maupun scratch berdasarkan prediksi terhadap data uji.

```
1 def evaluate_model(self, x_test, y_test, model_type='keras'):
2     """Evaluate model and return F1 score"""
3
4     if model_type == 'keras':
5         if self.keras_model is None:
6             raise ValueError("Keras model is not trained yet.")
7
8         # Predict with Keras model
9         y_pred_proba = self.keras_model.predict(x_test, verbose=0)
10        y_pred = np.argmax(y_pred_proba, axis=1)
11
12    elif model_type == 'scratch':
13        if self.scratch_model is None:
14            raise ValueError("Scratch model is not created yet.")
15
16        # Predict with scratch model
17        y_pred = self.scratch_model.predict(x_test)
18
19    else:
20        raise ValueError("model_type must be 'keras' or 'scratch'")
21
22    # Calculate F1 score
23    f1 = f1_score(y_test, y_pred, average='macro')
24    accuracy = accuracy_score(y_test, y_pred)
25
26    print(f'{model_type.capitalize()} Model - F1 Score: {f1:.4f}, Accuracy: {accuracy:.4f}')
27
28    return f1, y_pred
```

Kelas RNNScratchModel

Kelas RNNScratchModel adalah implementasi RNN dari nol tanpa bantuan Keras, yang dibangun berdasarkan bobot hasil pelatihan model Keras untuk memungkinkan perbandingan performa dan pemahaman mendalam tentang cara kerja RNN.

Inisialisasi kelas RNNModelScratch dengan atribut terkait.

```

1 class RNNScratchModel:
2     def __init__(self, weights, rnn_layers, rnn_units, bidirectional, max_length, num_classes):
3         self.weights = weights
4         self.rnn_layers = rnn_layers
5         self.rnn_units = rnn_units
6         self.bidirectional = bidirectional
7         self.max_length = max_length
8         self.num_classes = num_classes
9
10        # Initialize layers
11        self.embedding_layer = EmbeddingLayer(
12            input_dim=weights['embedding'].shape[0],
13            output_dim=weights['embedding'].shape[1],
14            weights=weights['embedding']
15        )
16
17        self.rnn_cells = self._create_rnn_cells()
18
19        # Initialize dropout layers
20        self.dropout_layers = []
21        for _ in range(rnn_layers):
22            self.dropout_layers.append(DropoutLayer(rate=0.2)) # Using same dropout rate as Keras model
23
24        # Initialize dense layer
25        self.dense_layer = DenseLayer(
26            weight=weights['dense'][0]['kernel'],
27            bias=weights['dense'][0]['bias'],
28            activation='softmax'
29        )

```

Membuat daftar sel RNN (RNNCell) dari bobot model yang telah diekstrak. Jika parameter bidirectional=True, maka untuk setiap layer RNN akan dibuat sepasang sel yaitu forward_cell dan backward_cell untuk memproses urutan input dalam dua arah

```

1 def _create_rnn_cells(self):
2     """Create RNN cells from weights"""
3     cells = []
4
5     for i in range(self.rnn_layers):
6         if self.bidirectional:
7             forward_cell = RNNCell(
8                 Wx=self.weights[f'rnn_{i}_forward'][0]['kernel'],
9                 Wh=self.weights[f'rnn_{i}_forward'][0]['recurrent_kernel'],
10                b=self.weights[f'rnn_{i}_forward'][0]['bias'],
11                activation='tanh'
12            )
13             backward_cell = RNNCell(
14                 Wx=self.weights[f'rnn_{i}_backward'][0]['kernel'],
15                 Wh=self.weights[f'rnn_{i}_backward'][0]['recurrent_kernel'],
16                 b=self.weights[f'rnn_{i}_backward'][0]['bias'],
17                 activation='tanh'
18            )
19             cells.append((forward_cell, backward_cell))
20         else:
21             cell = RNNCell(
22                 Wx=self.weights[f'rnn_{i}'][0]['kernel'],
23                 Wh=self.weights[f'rnn_{i}'][0]['recurrent_kernel'],
24                 b=self.weights[f'rnn_{i}'][0]['bias'],
25                 activation='tanh'
26            )
27             cells.append(cell)
28
29     return cells

```

Mengubah input token menjadi vektor melalui layer Embedding, kemudian melewati semua RNN layers secara berurutan. Output kemudian melewati layer Dropout dan dikirim ke layer Dense dengan aktivasi Softmax

```

1  def predict(self, x):
2      """Forward pass through the scratch RNN"""
3      batch_size = x.shape[0]
4
5      # Set dropout layers to evaluation mode
6      for dropout_layer in self.dropout_layers:
7          dropout_layer.setis_training(False)
8
9      # Embedding layer
10     embedded = self.embedding_layer.forward(x) # (batch_size, seq_len, embedding_dim)
11
12     # RNN layers
13     current_input = embedded
14
15     for layer_idx in range(self.rnn_layers):
16         if self.bidirectional:
17             forward_cell, backward_cell = self.rnn_cells[layer_idx]
18
19             # Forward pass
20             forward_outputs = self._rnn_forward_pass(current_input, forward_cell)
21
22             # Backward pass
23             backward_outputs = self._rnn_backward_pass(current_input, backward_cell)
24
25             # Concatenate forward and backward outputs
26             if layer_idx == self.rnn_layers - 1:
27                 # Last layer: only take final outputs
28                 layer_output = np.concatenate([
29                     forward_outputs[:, -1, :],
30                     backward_outputs[:, 0, :]
31                 ], axis=1)
32             else:
33                 # Intermediate layer: concatenate all timesteps
34                 layer_output = np.concatenate([forward_outputs, backward_outputs], axis=2)
35         else:
36             cell = self.rnn_cells[layer_idx]
37             outputs = self._rnn_forward_pass(current_input, cell)
38
39             if layer_idx == self.rnn_layers - 1:
40                 # Last layer: only take final output
41                 layer_output = outputs[:, -1, :]
42             else:
43                 # Intermediate layer: use all outputs
44                 layer_output = outputs
45
46             # dropout layer
47             current_input = self.dropout_layers[layer_idx].forward(layer_output)
48
49             # Dense layer
50             logits = self.dense_layer.forward(current_input)
51             return np.argmax(logits, axis=1)
52
53

```

Melakukan forward pass RNN satu arah untuk seluruh timestep dalam urutan input.

```

1  def _rnn_forward_pass(self, inputs, cell):
2      # Forward pass through RNN cell
3      batch_size, seq_len, input_dim = inputs.shape
4      hidden_dim = cell.Wh.shape[0]
5
6      h = np.zeros((batch_size, hidden_dim))
7      outputs = []
8
9      for t in range(seq_len):
10          x_t = inputs[:, t, :]
11          h = cell.forward(x_t, h)
12          outputs.append(h)
13
14      return np.stack(outputs, axis=1)

```

Seperti
_rnn_forward_pas
s, namun
memproses urutan
input dalam arah
yang berlawanan
untuk
implementasi
backward RNN

```
● ● ●  
1 def _rnn_backward_pass(self, inputs, cell):  
2     # Backward pass through RNN cell (for bidirectional)  
3     batch_size, seq_len, input_dim = inputs.shape  
4     hidden_dim = cell.Wh.shape[0]  
5  
6     h = np.zeros((batch_size, hidden_dim))  
7     outputs = []  
8  
9     # Process sequence in reverse order  
10    for t in range(seq_len - 1, -1, -1):  
11        x_t = inputs[:, t, :]  
12        h = cell.forward(x_t, h)  
13        outputs.append(h)  
14  
15    # Reverse outputs to match forward direction  
16    outputs.reverse()  
17    return np.stack(outputs, axis=1)
```

Konfigurasi
dengan parameter
yang akan diubah
ketika pengujian

```
● ● ●  
1 # --- Configuration ---  
2 DATASET = "NusaX"  
3 EMBEDDING_DIM = 100  
4 MAX_LENGTH = 100  
5 RNN_LAYERS = 2  
6 RNN_UNITS = [64, 32]  
7 BIDIRECTIONAL = True  
8 DROPOUT_RATE = 0.2  
9 EPOCHS = 5  
10 BATCH_SIZE = 32  
11  
12 print("Loading NusaX dataset...")  
13 data_loader = DataLoader(DATASET)  
14 x_train_raw, x_val_raw, x_test_raw, y_train_raw, y_val_raw, y_test_raw = data_loader.load_data()  
15  
16 print(f"Data loaded successfully!")  
17 print(f"Train set: {x_train_raw.shape}, {y_train_raw.shape}")  
18 print(f"Validation set: {x_val_raw.shape}, {y_val_raw.shape}")  
19 print(f"Test set: {x_test_raw.shape}, {y_test_raw.shape}")  
20  
21 # --- Extract text data ---  
22 x_train_texts = x_train_raw[:, 1].astype(str)  
23 x_val_texts = x_val_raw[:, 1].astype(str)  
24 x_test_texts = x_test_raw[:, 1].astype(str)  
25
```

Encode target
class menjadi
angka numerik.

Negative -> 0
Neutral -> 1
Positive -> 2

```
● ● ●
1 # Create label encoder to convert string labels to integers
2 label_encoder = LabelEncoder()
3
4 # Fit the encoder on all labels (train + val + test) to ensure consistency
5 all_labels = np.concatenate([y_train_raw, y_val_raw, y_test_raw])
6 label_encoder.fit(all_labels)
7
8 # Transform labels to integers
9 y_train = label_encoder.transform(y_train_raw)
10 y_val = label_encoder.transform(y_val_raw)
11 y_test = label_encoder.transform(y_test_raw)
12
13 # Print label mapping
14 print("\nLabel mapping:")
15 for i, label in enumerate(label_encoder.classes_):
16     print(f" {label} -> {i}")
17
18 # Get number of classes
19 NUM_CLASSES = len(label_encoder.classes_)
20 print(f"Number of unique classes: {NUM_CLASSES}")
21
22 # Print label distribution
23 print("\nLabel distribution in training set:")
24 unique_labels, counts = np.unique(y_train, return_counts=True)
25 for label_idx, count in zip(unique_labels, counts):
26     label_name = label_encoder.classes_[label_idx]
27     print(f" {label_name}: {count} samples ({count/len(y_train)*100:.1f}%)")
28
```

Preprocessing teks
menjadi vektor
dengan
TextPreprocessor

```
● ● ●
1 # --- Preprocess Text ---
2 print("\nPreprocessing text data...")
3 tokenizer = TextPreprocessor(max_features=10000, max_length=MAX_LENGTH)
4 tokenizer.create_text_vectorization(x_train_texts)
5
6 x_train = tokenizer.preprocess_with_keras(x_train_texts)
7 x_val = tokenizer.preprocess_with_keras(x_val_texts)
8 x_test = tokenizer.preprocess_with_keras(x_test_texts)
9 vocab_size = tokenizer.get_vocab_size()
10
11 print(f"Vocabulary size: {vocab_size}")
12 print(f"Processed train shape: {x_train.shape}")
13 print(f"Processed validation shape: {x_val.shape}")
14 print(f"Processed test shape: {x_test.shape}")
15
16
```

Membuat model
RNN dengan
parameter pilihan,
mengambil hasil
weight, dan
menerapkannya
ke scratch

```
● ● ●
1 # --- Model Training ---
2 print("\nInitializing and training model...")
3 model = RNNModel(vocab_size, EMBEDDING_DIM, MAX_LENGTH, NUM_CLASSES)
4 history = model.train_keras_model(
5     x_train, y_train,
6     x_val, y_val,
7     rnn_layers=RNN_LAYERS,
8     rnn_units=RNN_UNITS,
9     bidirectional=BIDIRECTIONAL,
10    dropout_rate=DROPOUT_RATE,
11    epochs=EPOCHS,
12    batch_size=BATCH_SIZE
13 )
14
15 # --- Extract Weights and Predict with Scratch RNN ---
16 print("\nExtracting weights and building from-scratch model...")
17 model.save_model_weights('src/rnn/experiments/keras_weights_rnn.weights.h5')
18 model.rnn_scratch(rnn_layers=RNN_LAYERS, rnn_units=RNN_UNITS, bidirectional=BIDIRECTIONAL)
19
```

Evaluate model dan menyimpan hasil evaluasi ke dalam folder

```
1 # --- Evaluation ---
2 print("\nEvaluating Keras model...")
3 f1_keras, y_pred_keras = model.evaluate_model(x_test, y_test, model_type='keras')
4
5 print("\nEvaluating from-scratch model...")
6 f1_scratch, y_pred_scratch = model.evaluate_model(x_test, y_test, model_type='scratch')
7
8 # --- Additional Analysis ---
9 print("\nAdditional Analysis:")
10 print(f"Keras model F1-score: {f1_keras:.4f}")
11 print(f"From-scratch model F1-score: {f1_scratch:.4f}")
12 print(f"Difference: {abs(f1_keras - f1_scratch):.4f}")
13
14 # Check prediction agreement
15 match_percentage = np.mean(y_pred_keras == y_pred_scratch) * 100
16 print(f"Prediction match percentage: {match_percentage:.2f}%")
17
18 # --- Save report ---
19 from sklearn.metrics import classification_report
20
21 report_path = "src/rnn/experiments/classification_report_rnn.txt"
22 with open(report_path, "w") as f:
23     f.write("== RNN Model Evaluation Report ==\n")
24     f.write(f"\nF1 Score (Keras): {f1_keras:.4f}\n")
25     f.write(f"\nF1 Score (Scratch): {f1_scratch:.4f}\n")
26     f.write(f"\nPrediction Match: {match_percentage:.2f}%\n\n")
27     f.write("== Classification Report (Keras) ==\n")
28     f.write(classification_report(y_test, y_pred_keras))
29     f.write("\n\n== Classification Report (Scratch) ==\n")
30     f.write(classification_report(y_test, y_pred_scratch))
31
32 print(f"Saved classification report to: {report_path}")
33
```

Plot hasil grafik validation loss, accuracy, dan F1 scores perbandingan implementasi scratch dan keras

```
1 # --- Plotting ---
2 plt.figure(figsize=(16, 5))
3
4 # Training and Validation Loss
5 plt.subplot(1, 3, 1)
6 plt.plot(history.history['loss'], label='Training Loss', marker='o')
7 plt.plot(history.history['val_loss'], label='Validation Loss', marker='x')
8 plt.title("Training & Validation Loss")
9 plt.xlabel("Epochs")
10 plt.ylabel("Loss")
11 plt.legend()
12 plt.grid(True, alpha=0.3)
13
14 # Training and Validation Accuracy
15 plt.subplot(1, 3, 2)
16 plt.plot(history.history['accuracy'], label='Training Accuracy', marker='o')
17 plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marker='x')
18 plt.title("Training & Validation Accuracy")
19 plt.xlabel("Epochs")
20 plt.ylabel("Accuracy")
21 plt.legend()
22 plt.grid(True, alpha=0.3)
23
24 # F1 Scores
25 plt.subplot(1, 3, 3)
26 plt.bar(['Keras Model', 'Scratch RNN'], [f1_keras, f1_scratch], color=['blue', 'orange'])
27 plt.title("Macro F1 Score Comparison")
28 plt.ylabel("F1 Score")
29 plt.grid(True, alpha=0.3, axis='y')
30
31 for i, v in enumerate([f1_keras, f1_scratch]):
32     plt.text(i, v + 0.01, f'{v:.3f}', ha='center', va='bottom')
33
34 plt.tight_layout()
35 plot_path = "src/rnn/experiments/rnn_plot.png"
36 plt.savefig(plot_path, dpi=300, bbox_inches='tight')
37 print(f"Saved training plot to: {plot_path}")
38 plt.show()
```

3.1.4. Implementasi LSTM

train_lstm.py

Program untuk melatih model klasifikasi teks menggunakan arsitektur LSTM (Long Short-Term Memory), termasuk embedding, LSTM layer, dense layer, dan evaluasi model pada dataset NusaX.

Import dan konfigurasi awal

```
● ● ●
1 import os
2 import sys
3 sys.path.insert(0, os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
4
5 import numpy as np
6 import tensorflow as tf
7 from tensorflow import keras
8 from sklearn.metrics import classification_report
9 from sklearn.preprocessing import LabelEncoder
10 from sklearn.utils.class_weight import compute_class_weight
11 from utils.data_loader import DataLoader
12 from utils.tokenizer import TextPreprocessor
13 from utils.metrics import f1_score_macro, print_classification_report, plot_history
14 import random
15
16 # Set seeds for reproducibility
17 SEED = 42
18 np.random.seed(SEED)
19 tf.random.set_seed(SEED)
20 random.seed(SEED)
21
22 # Hyperparameter configuration
23 DATASET      = "NusaX"
24 EMBEDDING_DIM = 64
25 MAX_LENGTH   = 100
26 LSTM_LAYERS  = 2
27 LSTM_UNITS   = [32, 16]
28 BIDIRECTIONAL = False
29 DROPOUT_RATE = 0.5
30 EPOCHS       = 15
31 BATCH_SIZE    = 32
32 LEARNING_RATE = 0.001
```

Membangun model LSTM dengan embedding, (bi)LSTM, dropout, dense layer, dan output softmax.

Konfigurasi seperti bidirectional, jumlah unit, dan dropout diatur melalui parameter.

```
1 def build_lstm_model(vocab_size, num_classes):
2     """
3         Builds a Keras model with the following architecture:
4             Embedding -> (Bi)LSTM -> Dropout -> Dense (softmax).
5     """
6     inputs = keras.Input(shape=(MAX_LENGTH,), name="input_text")
7
8     # embedding layer
9     x = keras.layers.Embedding(
10         input_dim=vocab_size,
11         output_dim=EMBEDDING_DIM,
12         mask_zero=True,
13         name="embedding",
14         embeddings_initializer=keras.initializers.RandomUniform(seed=SEED)
15     )(inputs)
16
17     x = keras.layers.Dropout(0.3, seed=SEED)(x)
18
19     # Lstm layers
20     for i, units in enumerate(LSTM_UNITS):
21         return_seq = (i < LSTM_LAYERS - 1)
22         if BIDIRECTIONAL:
23             x = keras.layers.Bidirectional(
24                 keras.layers.LSTM(
25                     units,
26                     return_sequences=return_seq,
27                     dropout=0.3,
28                     recurrent_dropout=0.3,
29                     kernel_regularizer=keras.regularizers.l2(0.01),
30                     kernel_initializer=keras.initializers.GlorotUniform(seed=SEED),
31                     recurrent_initializer=keras.initializers.Orthogonal(seed=SEED)
32                 ),
33                 name=f"bidi_lstm_{i+1}"
34             )(x)
35         else:
36             x = keras.layers.LSTM(
37                 units,
38                 return_sequences=return_seq,
39                 dropout=0.3,
40                 recurrent_dropout=0.3,
41                 kernel_regularizer=keras.regularizers.l2(0.01),
42                 name=f"lstm_{i+1}",
43                 kernel_initializer=keras.initializers.GlorotUniform(seed=SEED),
44                 recurrent_initializer=keras.initializers.Orthogonal(seed=SEED)
45             )(x)
46
47     x = keras.layers.Dropout(0.4, seed=SEED)(x)
48
49     x = keras.layers.Dense(32, activation='relu',
50                           kernel_regularizer=keras.regularizers.l2(0.01),
51                           kernel_initializer=keras.initializers.GlorotUniform(seed=SEED))(x)
52     x = keras.layers.Dropout(DROPOUT_RATE, seed=SEED)(x)
53
54     # output layer
55     outputs = keras.layers.Dense(
56         num_classes,
57         activation="softmax",
58         name="output_softmax",
59         kernel_initializer=keras.initializers.GlorotUniform(seed=SEED)
60     )(x)
61
62     optimizer = keras.optimizers.Adam(learning_rate=LEARNING_RATE)
63
64     model = keras.Model(inputs, outputs, name="LSTM_Classifier")
65     model.compile(
66         optimizer=optimizer,
67         loss="sparse_categorical_crossentropy",
68         metrics=["accuracy"]
69     )
70
71 return model
```

Load dan Preprocessing data. Melakukan encoding label dan vektorisasi teks menjadi input numerik, serta menghitung class weight untuk menangani ketidakseimbangan data.

```
1 def main():
2     # Load dataset
3     print(f"Loading dataset: {DATASET}...")
4     loader = DataLoader(DATASET)
5     x_tr_raw, x_val_raw, x_te_raw, y_tr, y_val, y_te = loader.load_data()
6
7     # extract text data
8     x_tr_txt = x_tr_raw[:, 1].astype(str)
9     x_val_txt= x_val_raw[:, 1].astype(str)
10    x_te_txt = x_te_raw[:, 1].astype(str)
11
12    # convert labels to numeric if they are strings
13    if y_tr.dtype == 'object' or y_tr.dtype.kind in ('U', 'S'):
14        print("Converting string labels to numeric...")
15        label_encoder = LabelEncoder()
16        y_tr = label_encoder.fit_transform(y_tr)
17        y_val = label_encoder.transform(y_val)
18        y_te = label_encoder.transform(y_te)
19
20    y_tr = y_tr.astype(np.int32)
21    y_val = y_val.astype(np.int32)
22    y_te = y_te.astype(np.int32)
23
24    print(f"Label range: {np.min(y_tr)} to {np.max(y_tr)}")
25    print(f"Number of classes: {len(np.unique(y_tr))}")
26
27    # check class distribution
28    unique, counts = np.unique(y_tr, return_counts=True)
29    print("Class distribution:", dict(zip(unique, counts)))
30
31    class_weights = compute_class_weight(
32        'balanced',
33        classes=np.unique(y_tr),
34        y=y_tr
35    )
36    class_weight_dict = dict(enumerate(class_weights))
37    print("Class weights:", class_weight_dict)
38
39    tp = TextPreprocessor(max_features=5000, max_length=MAX_LENGTH)
40    tp.create_text_vectorization(x_tr_txt)
41    x_tr = tp.preprocess_with_keras(x_tr_txt)
42    x_val = tp.preprocess_with_keras(x_val_txt)
43    x_te = tp.preprocess_with_keras(x_te_txt)
44    vocab_size = tp.get_vocab_size()
```

Melatih model menggunakan model.fit dengan callback EarlyStopping dan ReduceLROnPlateau untuk menghindari overfitting.

```

1  model = build_lstm_model(vocab_size, num_classes=len(np.unique(y_tr)))
2
3  print("\nModel summary:")
4  model.summary()
5
6  callbacks = [
7      keras.callbacks.EarlyStopping(
8          monitor='val_loss',
9          patience=3,
10         restore_best_weights=True,
11         verbose=1
12     ),
13     keras.callbacks.ReduceLROnPlateau(
14         monitor='val_loss',
15         factor=0.2,
16         patience=2,
17         min_lr=1e-6,
18         verbose=1
19     )
20 ]
21
22 history = model.fit(
23     x_tr, y_tr,
24     validation_data=(x_val, y_val),
25     epochs=EPOCHS,
26     batch_size=BATCH_SIZE,
27     class_weight=class_weight_dict,
28     callbacks=callbacks,
29     verbose=1,
30     validation_freq=1
31 )

```

Evaluasi dan penyimpanan model

```

1  plot_history(history, save_path="src/lstm/lstm_training_history.png")
2  print("Training curves saved to src/lstm/lstm_training_history.png")
3
4  os.makedirs("src/lstm", exist_ok=True)
5  model.save("src/lstm/lstm_model.keras")
6  print("Keras LSTM model saved at src/lstm/lstm_model.keras")
7
8  loss, acc = model.evaluate(x_te, y_te, batch_size=BATCH_SIZE, verbose=0)
9  y_pred = np.argmax(model.predict(x_te, batch_size=BATCH_SIZE, verbose=0), axis=1)
10 f1 = f1_score_macro(y_te, y_pred)
11
12 print("\n[Keras LSTM] Test Results:")
13 print(f"Test loss: {loss:.4f}")
14 print(f"Test accuracy: {acc:.4f}")
15 print(f"Macro F1-Score: {f1:.4f}")
16
17 print("\n[Keras LSTM] Classification Report:")
18 print_classification_report(y_te, y_pred)
19
20 report_txt = classification_report(y_te, y_pred, zero_division=0)
21 with open("src/lstm/classification_report_keras_lstm.txt", "w") as f:
22     f.write(f"Model Configuration:\n")
23     f.write(f"\tEMBEDDING_DIM: {EMBEDDING_DIM}\n")
24     f.write(f"\tMAX_LENGTH: {MAX_LENGTH}\n")
25     f.write(f"\tLSTM_UNITS: {LSTM_UNITS}\n")
26     f.write(f"\tBIDIRECTIONAL: {BIDIRECTIONAL}\n")
27     f.write(f"\tDROPOUT_RATE: {DROPOUT_RATE}\n")
28     f.write(f"\tLEARNING RATE: {LEARNING_RATE}\n")
29     f.write(f"\tEPOCHS_TRAINED: {len(history.history['loss'])}\n")
30     f.write(f"\tVOCAB_SIZE: {vocab_size}\n")
31     f.write(f"\tFinal Results:\n")
32     f.write(f"\t\tTest Loss: {loss:.4f}\n")
33     f.write(f"\t\tTest Accuracy: {acc:.4f}\n")
34     f.write(f"\t\tMacro F1-Score: {f1:.4f}\n")
35     f.write(f"\t\tClassification Report:\n")
36     f.write(report_txt)
37
38 print("Classification report saved to src/lstm/classification_report_keras_lstm.txt")
39 if __name__ == "__main__":
40     main()

```

forward_lstm.py

Melakukan forward propagation manual (dari awal) untuk arsitektur LSTM hasil training Keras, lalu membandingkan hasilnya dengan output Keras untuk mengevaluasi kesesuaian implementasi dari awal (scratch).

Load model dan struktur layer.

Memuat model .keras hasil pelatihan sebelumnya, kemudian mengekstrak bobot dari layer embedding,

LSTM, dan Dense ke dalam bentuk kelas buatan (EmbeddingLayer, LSTMCell, dll).

```
1  def load_keras_weights(model_path: str):
2      """
3          Load a Keras model and return all its weights as a list.
4      """
5      model = tf.keras.models.load_model(model_path)
6      return model
7
8  def inspect_model_architecture(keras_model):
9      """
10         Inspect the actual architecture of the loaded model.
11     """
12     print("\n==== Model Architecture Inspection ===")
13     for i, layer in enumerate(keras_model.layers):
14         print(f"Layer {i}: {layer.name} - {type(layer).__name__}")
15         if hasattr(layer, 'output_shape'):
16             print(f" Output shape: {layer.output_shape}")
17         if hasattr(layer, 'get_weights') and layer.get_weights():
18             weights = layer.get_weights()
19             print(f" Weights shapes: {[w.shape for w in weights]}")
20     print("-" * 50)
```

```
1  # Load and inspect model
2  print("Loading Keras model...")
3  keras_model = load_keras_weights("src/lstm/lstm_model.keras")
4
5  emb, lstm_cells, intermediate_dense, dense = build_scratch_layers(keras_model)
```

Melakukan perhitungan manual untuk setiap tahap: embedding → LSTM (dengan masking dan dropout) → dense layer pertama → output softmax.

```

119         outputs.append(h.copy())
120
121     if i == len(lstm_cells) - 1:
122         current_output = outputs[-1]
123         print(f" Final output shape: {current_output.shape}")
124     else:
125         current_output = np.stack(outputs, axis=1)
126         print(f" Intermediate output shape: {current_output.shape}")
127
128     dropout_layer = DropoutLayer(rate=0.4)
129     dropout_layer.sets_training(False)
130     if i == len(lstm_cells) - 1:
131         current_output = dropout_layer.forward(current_output)
132     else:
133         pass
134
135     dropout_layer = DropoutLayer(rate=0.5)
136     dropout_layer.sets_training(False)
137     current_output = dropout_layer.forward(current_output)
138
139     print(f"Before intermediate dense layer: {current_output.shape}")
140     current_output = intermediate_dense.forward(current_output)
141     print(f"After intermediate dense layer: {current_output.shape}")
142
143     print(f"Before output dense layer: {current_output.shape}")
144     y_prob = dense.forward(current_output)
145     print(f"After output dense layer: {y_prob.shape}")
146
147     # temperature = 1.1
148     # y_prob = y_prob ** (1/temperature)
149     # y_prob = y_prob / np.sum(y_prob, axis=1, keepdims=True)
150
151     # class_boost = np.array([1.05, 1.0, 1.0])
152     # y_prob = y_prob * class_boost
153     # y_prob = y_prob / np.sum(y_prob, axis=1, keepdims=True)
154
155 return y_prob

```

Preprocessing data test.
Melakukan encoding label dan vectorisasi teks dari data test dengan preprocessing yang sama seperti training.

```

● ○ ●
1 def main():
2     # Load dataset
3     print(f"[Scratch] Loading dataset: {DATASET}...")
4     loader = Dataloader(DATASET)
5     x_tr_raw, x_val_raw, x_te_raw, y_tr, y_val, y_te = loader.load_data()
6
7     x_tr_txt = x_tr_raw[:, 1].astype(str)
8     x_te_txt = x_te_raw[:, 1].astype(str)
9
10    # Label conversion
11    if y_te.dtype == 'object' or y_te.dtype.kind in {'U', 'S'}:
12        print("Converting string labels to numeric...")
13        label_encoder = LabelEncoder()
14        if y_tr.dtype == 'object' or y_tr.dtype.kind in {'U', 'S'}:
15            y_tr = label_encoder.fit_transform(y_tr)
16            y_te = label_encoder.transform(y_te)
17        else:
18            y_te = label_encoder.fit_transform(y_te)
19
20    y_te = y_te.astype(np.int32)
21
22    # text preprocessing
23    tp = TextPreprocessor(max_features=5000, max_length=100)
24    tp.create_text_vectorization(x_tr_txt)
25    x_te = tp.preprocess_with_keras(x_te_txt)
26
27    print(f"Test data shape: {x_te.shape}")

```

Evaluasi dan simpan hasil.

```
● ● ●
1 print("Performing scratch forward propagation...")
2 y_prob = scratch_forward(emb, lstm_cells, intermediate_dense, dense, x_te)
3 y_pred = np.argmax(y_prob, axis=1)
4
5 f1 = f1_score_macro(y_te, y_pred)
6 print(f"\n[Scratch LSTM] Macro-F1: {f1:.4f}")
7
8 print("\n[Scratch LSTM] Classification Report:")
9 print_classification_report(y_te, y_pred)
10
11 report_txt = classification_report(y_te, y_pred, zero_division=0)
12 with open("src/lstm/classification_report_scratch_lstm.txt", "w") as f:
13     f.write(f"Scratch LSTM Forward Propagation Results:\n")
14     f.write(f"MAX_LENGTH: {MAX_LENGTH}\n")
15     f.write(f"EMBEDDING_DIM: {EMBEDDING_DIM}\n")
16     f.write(f"LSTM_UNITS: {LSTM_UNITS}\n")
17     f.write(f"BIDIRECTIONAL: {BIDIRECTIONAL}\n\n")
18     f.write(f"Macro F1-Score: {f1:.4f}\n\n")
19     f.write("Classification Report:\n")
20     f.write(report_txt)
21
22 print("Classification report saved to src/lstm/classification_report_scratch_lstm.txt")
```

experiment_lstm.py

Melakukan eksperimen grid search terhadap berbagai konfigurasi model LSTM (jumlah layer, jumlah unit, bidirectional atau unidirectional), mencatat hasil performa, menyimpan model, grafik, dan laporan klasifikasi.

Setup dan konfigurasi

```
1 import os
2 import sys
3 sys.path.insert(0, os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
4
5 import numpy as np
6 import pandas as pd
7 import tensorflow as tf
8 from sklearn.metrics import classification_report
9 from sklearn.preprocessing import LabelEncoder
10 from sklearn.utils.class_weight import compute_class_weight
11 from utils.data_loader import DataLoader
12 from utils.tokenizer import TextPreprocessor
13 from utils.metrics import f1_score_macro, print_classification_report, plot_history
14 import train_lstm
15 import random
16
17 # Set seeds
18 SEED = 42
19 np.random.seed(SEED)
20 tf.random.set_seed(SEED)
21 random.seed(SEED)
22
23 print("Starting LSTM experiments...")
24
25 # --- Global configuration ---
26 DATASET = "NusaX"
27 MAX_LENGTH = train_lstm.MAX_LENGTH
28 BATCH_SIZE = train_lstm.BATCH_SIZE
29 EPOCHS = train_lstm.EPOCHS
30
31 # --- Hyperparameter grid ---
32 param_grid = [
33     {"layers": 1, "units": [64], "bidirectional": False},
34     {"layers": 2, "units": [64, 32], "bidirectional": False},
35     {"layers": 2, "units": [32, 16], "bidirectional": False},
36     {"layers": 3, "units": [64, 32, 16], "bidirectional": False},
37
38     {"layers": 2, "units": [32, 32], "bidirectional": False},
39     {"layers": 2, "units": [64, 64], "bidirectional": False},
40     {"layers": 2, "units": [128, 64], "bidirectional": False},
41
42     {"layers": 1, "units": [64], "bidirectional": True},
43     {"layers": 2, "units": [32, 16], "bidirectional": True}
44 ]
45
46 # --- Directories setup ---
47 save_dir = os.path.join("src", "lstm", "experiments")
48 os.makedirs(save_dir, exist_ok=True)
```

Meload data teks,
melakukan
encoding label,
dan mengubah
teks ke dalam
bentuk vektor
menggunakan
TextPreprocessor.

```
1 # --- Load and preprocess data ---
2 print("Loading NusA dataset...")
3 loader = DataLoader(DATASET)
4 x_tr_raw, x_val_raw, x_te_raw, y_tr, y_val, y_te = loader.load_data()
5
6 x_tr_txt = x_tr_raw[:, 1].astype(str)
7 x_val_txt = x_val_raw[:, 1].astype(str)
8 x_te_txt = x_te_raw[:, 1].astype(str)
9
10 if y_tr.dtype == 'object' or y_tr.dtype.kind in {'U', 'S'}:
11     print("Converting string labels to numeric...")
12     label_encoder = LabelEncoder()
13     y_tr = label_encoder.fit_transform(y_tr)
14     y_val = label_encoder.transform(y_val)
15     y_te = label_encoder.transform(y_te)
16
17 y_tr = y_tr.astype(np.int32)
18 y_val = y_val.astype(np.int32)
19 y_te = y_te.astype(np.int32)
20
21 print(f"Data loaded successfully!")
22 print(f"Label range: {np.min(y_tr)} to {np.max(y_tr)}")
23 print(f"Number of classes: {len(np.unique(y_tr))}")
24
25 tp = TextPreprocessor(max_features=5000, max_length=MAX_LENGTH)
26 tp.create_text_vectorization(x_tr_txt)
27 X_train = tp.preprocess_with_keras(x_tr_txt)
28 X_val = tp.preprocess_with_keras(x_val_txt)
29 X_test = tp.preprocess_with_keras(x_te_txt)
30 vocab_size = tp.get_vocab_size()
31 num_classes = len(np.unique(y_tr))
32
33 print(f"Vocabulary size: {vocab_size}")
34 print(f"Training data shape: {X_train.shape}")
35
36 class_weights = compute_class_weight(
37     'balanced',
38     classes=np.unique(y_tr),
39     y=y_tr
40 )
41 class_weight_dict = dict(enumerate(class_weights))
42 print("Class weights:", class_weight_dict)
```

Melakukan iterasi terhadap semua konfigurasi model dan melatih model dengan parameter tersebut. Model dan hasilnya disimpan.

```
1 # --- Experiment Loop ---
2 results = []
3 for experiment_idx, cfg in enumerate(param_grid):
4     # reset seeds for each experiment
5     experiment_seed = SEED + experiment_idx
6     np.random.seed(experiment_seed)
7     tf.random.set_seed(experiment_seed)
8     random.seed(experiment_seed)
9
10    layers = cfg["layers"]
11    units = cfg["units"]
12    bidi_flag = cfg["bidirectional"]
13    tag = f'L{layers}_U{",".join(map(str,units))}_B{int(bidi_flag)}'
14    print(f"\n==== Running experiment: {tag} (seed: {experiment_seed}) ====")
15
16    # Override hyperparams in train_lstm
17    train_lstm.LSTM_LAYERS = layers
18    train_lstm.LSTM_UNITS = units
19    train_lstm.BIDIRECTIONAL = bidi_flag
20    train_lstm.SEED = experiment_seed
21
22    # Build & train model
23    model = train_lstm.build_lstm_model(vocab_size, num_classes)
24
25    callbacks = [
26        train_lstm.keras.callbacks.EarlyStopping(
27            monitor='val_loss',
28            patience=3,
29            restore_best_weights=True,
30            verbose=1
31        ),
32        train_lstm.keras.callbacks.ReduceLROnPlateau(
33            monitor='val_loss',
34            factor=0.2,
35            patience=2,
36            min_lr=1e-6,
37            verbose=1
38        )
39    ]
40
41    history = model.fit(
42        X_train, y_tr,
43        validation_data=(X_val, y_val),
44        epochs=EPOCHS,
45        batch_size=BATCH_SIZE,
46        class_weight=class_weight_dict,
47        callbacks=callbacks,
48        verbose=2
49    )
50
51    model_path = os.path.join(save_dir, f"model_{tag}.keras")
52    model.save(model_path)
53    print(f"Model saved: {model_path}")
```

Model yang sudah dilatih dievaluasi pada data test.

Disimpan metrik evaluasi dan classification report

```
● ● ●
1 # Save training curves
2 curve_path = os.path.join(save_dir, f"history_{tag}.png")
3 plot_history(history, save_path=curve_path)
4 print(f"Training curves saved to {curve_path}")
5
6 # Evaluate on test set
7 loss, acc = model.evaluate(X_test, y_te, batch_size=BATCH_SIZE, verbose=0)
8 y_pred = np.argmax(model.predict(X_test, batch_size=BATCH_SIZE, verbose=0), axis=1)
9 f1 = f1_score_macro(y_te, y_pred)
10 print(f"Result (tag): loss={loss:.4f}, acc={acc:.4f}, macro-F1={f1:.4f}")
11
12 # Classification report
13 print(f"\n[Classification Report for {tag}]")
14 print_classification_report(y_te, y_pred)
15
16 # Save classification report to file
17 report_txt = classification_report(y_te, y_pred, zero_division=0)
18 cr_path = os.path.join(save_dir, f"class_report_{tag}.txt")
19 with open(cr_path, "w") as f:
20     f.write(f"Experiment Configuration: {tag}\n")
21     f.write(f"Seed: {experiment_seed}\n")
22     f.write(f"Layers: {layers}\n")
23     f.write(f"Units: {units}\n")
24     f.write(f"Bidirectional: {bidi_flag}\n")
25     f.write(f"Epochs trained: {len(history.history['loss'])}\n")
26     f.write(f"Final Results:\n")
27     f.write(f"Loss: {loss:.4f}\n")
28     f.write(f"Accuracy: {acc:.4f}\n")
29     f.write(f"Macro F1: {f1:.4f}\n\n")
30     f.write("Classification Report:\n")
31     f.write(report_txt)
32 print(f"Classification report saved to {cr_path}")
33
34 # Collect results
35 results.append({
36     "config": tag,
37     "seed": experiment_seed,
38     "layers": layers,
39     "units": "-".join(map(str, units)),
40     "bidirectional": int(bidi_flag),
41     "epochs_trained": len(history.history['loss']),
42     "loss": float(loss),
43     "accuracy": float(acc),
44     "macro_f1": float(f1),
45     "model_path": model_path,
46     "curve_path": curve_path,
47     "report_path": cr_path
48 })

```

Hasil dari semua eksperimen disimpan ke dalam file CSV dan ditampilkan tiga hasil terbaik berdasarkan macro F1.

```
● ● ●
1 # --- Save summary to CSV ---
2 csv_path = os.path.join(save_dir, "results_lstm.csv")
3 df = pd.DataFrame(results)
4 df = df.sort_values('macro_f1', ascending=False)
5 df.to_csv(csv_path, index=False)
6 print(f"\nAll experiments completed. Summary saved to {csv_path}")
7 print("\nTop 3 configurations by Macro F1:")
8 print(df[['config', 'seed', 'accuracy', 'macro_f1']].head(3))
9
```

3.2. Hasil Pengujian

Hasil Pengujian dilakukan pada dataset [NusaX](#) untuk model SimpleRNN dan LSTM, dan dataset [CIFAR-10](#) untuk model CNN.

3.2.1. Pengujian CNN

Pengujian pada model CNN dilakukan dalam dua pengujian besar, yaitu pengujian dengan variasi pelatihan untuk menganalisis pengaruh hyperparameter pada CNN, serta pengujian dengan membandingkan hasil forward propagation from scratch dengan hasil forward propagation menggunakan keras.

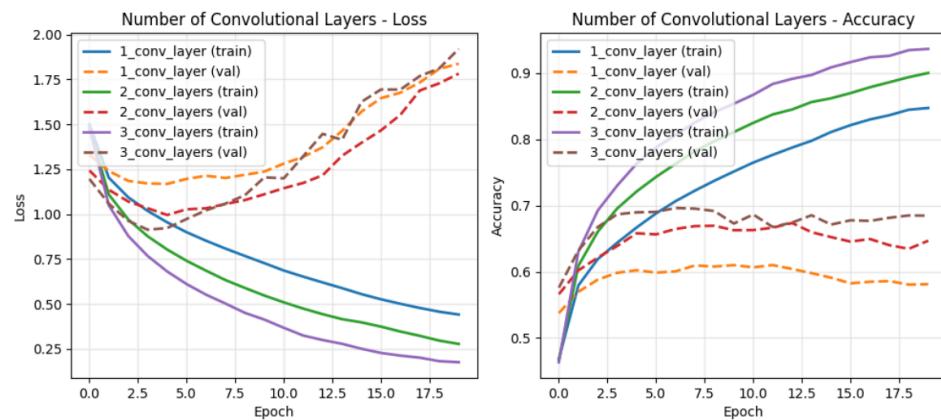
3.2.1.1. Eksperimen Variasi Hyperparameter CNN

3.2.1.1.1. Eksperimen jumlah layer konvolusi

Eksperimen dilakukan dalam tiga variasi jumlah layer yang berbeda, dengan jumlah epoch 20, ukuran setiap kernel adalah 3x3, dan menggunakan *max pooling*.

n-layers	Filter per layer	Hasil
1	[32]	Test Accuracy: 0.5689 Test F1-Score (macro): 0.5658
2	[32, 64]	Test Accuracy: 0.6470 Test F1-Score (macro): 0.6457
3	[32, 64, 128]	Test Accuracy: 0.6784 Test F1-Score (macro): 0.6729

Dengan perbandingan Loss dan Accuracy antar ketiga variasi jumlah layer yang direpresentasikan dengan grafik adalah sebagai berikut:

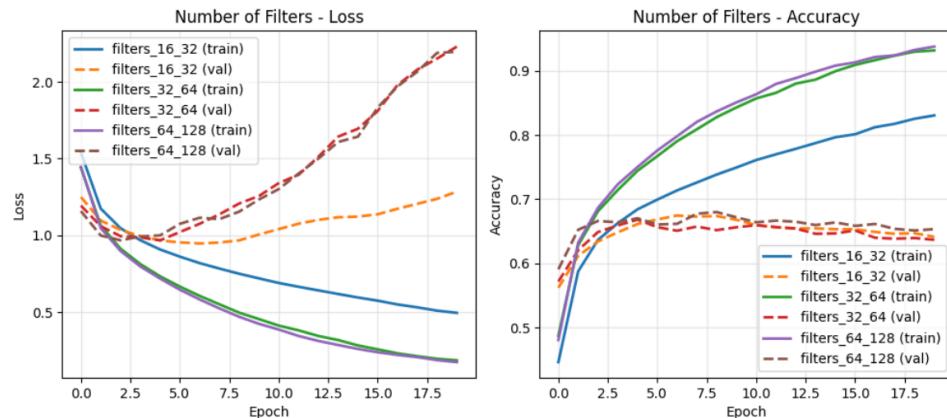


3.2.1.1.2 Eksperimen banyak filter per layer konvolusi

Eksperimen dilakukan dalam tiga variasi banyak filter per layer konvolusi yang berbeda, dengan jumlah epoch 20, jumlah layer adalah 2, ukuran setiap kernel adalah 3x3, dan menggunakan *max pooling*.

Filter per layer	Hasil
[16, 32]	Test Accuracy: 0.6497 Test F1-Score (macro): 0.6456
[32, 64]	Test Accuracy: 0.6393 Test F1-Score (macro): 0.6357
[64, 128]	Test Accuracy: 0.6515 Test F1-Score (macro): 0.6506

Dengan perbandingan Loss dan Accuracy antar ketiga variasi banyak filter per layer yang direpresentasikan dengan grafik adalah sebagai berikut:



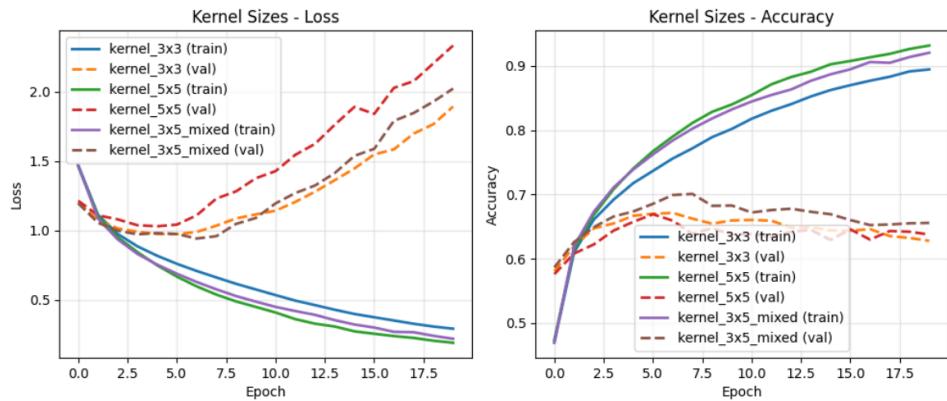
3.2.1.1.3 Eksperimen ukuran filter per layer konvolusi

Eksperimen dilakukan dalam tiga variasi ukuran filter per layer konvolusi yang berbeda, dengan jumlah epoch 20, jumlah layer adalah 2, banyak kernel (filter) per layer adalah [32, 64], dan menggunakan *max pooling*.

Ukuran filter per layer	Hasil
[3, 3]	Test Accuracy: 0.6238 Test F1-Score (macro): 0.6240

[5, 5]	Test Accuracy: 0.6306 Test F1-Score (macro): 0.6307
[3, 5]	Test Accuracy: 0.6547 Test F1-Score (macro): 0.6591

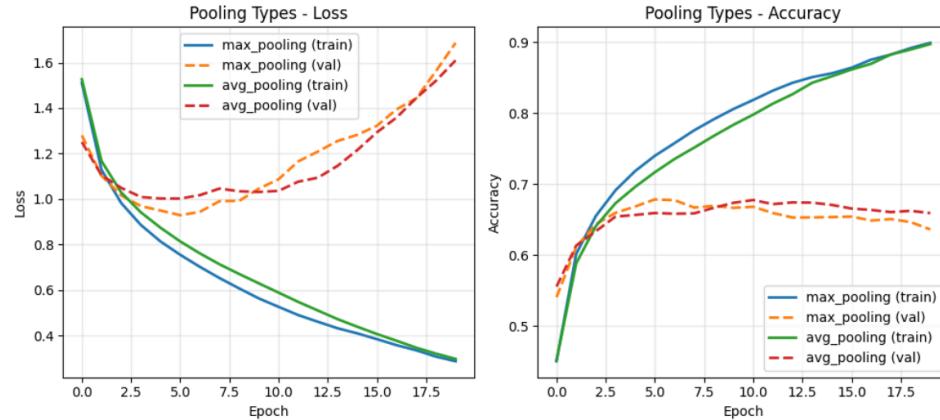
Dengan perbandingan Loss dan Accuracy antar ketiga variasi ukuran filter per layer yang direpresentasikan dengan grafik adalah sebagai berikut:



3.2.1.1.4 Eksperimen jenis pooling layer konvolusi

Eksperimen dilakukan dalam dua variasi jenis pooling konvolusi yang berbeda, dengan jumlah epoch 20, jumlah layer adalah 2, banyak kernel (filter) per layer adalah [32, 64], dan ukuran kernel (filter) adalah 3x3.

Jenis Pooling	Hasil
Max Pooling	Test Accuracy: 0.6315 Test F1-Score (macro): 0.6278
Average Pooling	Test Accuracy: 0.6615 Test F1-Score (macro): 0.6581



3.2.1.1.4 Perbandingan dengan Implementasi From Scratch

Untuk menilai apakah model CNN from scratch yang telah dibuat diimplementasikan dengan baik, dilakukan transfer bobot dari model CNN yang dilatih menggunakan Keras ke model CNN from scratch. Kemudian, F1 score yang dihasilkan oleh model from scratch dibandingkan dengan F1 score dari model Keras. Sebagai indikator tambahan, digunakan MSE (Mean Squared Error) sebagai metrik evaluasi tambahan untuk melihat perbedaan secara lebih presisi dalam kinerja kedua model tersebut.

Jenis Variasi	Variasi	Hasil
Jumlah Layer	1 layer	MSE difference: 8.73e-15 Max difference: 1.24e-06 F1 difference: 0.00000000 Identical predictions: True
	2 layer	MSE difference: 2.20e-14 Max difference: 2.45e-06 F1 difference: 0.00000000 Identical predictions: True
	3 layer	MSE difference: 2.41e-14 Max difference: 2.88e-06 F1 difference: 0.00000000 Identical predictions: True
Banyak Filter per Layer	Layer 1 : 16 filter Layer 2 : 32 filter	MSE difference: 1.28e-14 Max difference: 1.75e-06 F1 difference: 0.00000000

		Identical predictions: True
	Layer 1 : 32 filter Layer 2 : 64 filter	MSE difference: 1.60e-14 Max difference: 1.74e-06 F1 difference: 0.00000000 Identical predictions: True
	Layer 1 : 64 filter Layer 2 : 128 filter	MSE difference: 2.58e-14 Max difference: 2.89e-06 F1 difference: 0.00000000 Identical predictions: True
Ukuran Filter per Layer	Layer 1 : 3 x 3 Layer 2 : 3 x 3	MSE difference: 1.72e-14 Max difference: 2.16e-06 F1 difference: 0.00000000 Identical predictions: True
	Layer 1 : 5 x 5 Layer 2 : 5 x 5	MSE difference: 3.44e-14 Max difference: 5.10e-06 F1 difference: 0.00000000 Identical predictions: True
	Layer 1 : 3 x 3 Layer 2 : 5 x 5	MSE difference: 2.30e-14 Max difference: 2.20e-06 F1 difference: 0.00000000 Identical predictions: True
	Max Pooling	MSE difference: 1.83e-14 Max difference: 2.37e-06 F1 difference: 0.00000000 Identical predictions: True
Jenis Pooling	Avg Pooling	MSE difference: 7.28e-15 Max difference: 1.36e-06 F1 difference: 0.00000000 Identical predictions: True

3.2.2. Pengujian SimpleRNN

Pengujian dilakukan dengan menggunakan dimensi embedding sebesar 100, panjang maksimum urutan 100, tingkat dropout sebesar 0.2, sebanyak 10 epoch, dan ukuran batch 32.

3.2.2.1. Eksperimen Variasi Hyperparameter SimpleRNN

Layers	Units	Bidirectional	Macro F1-Score	Graph

1	[64]	True	0.5534	<p>Direction_Effect - Bidirectional Training History</p> <ul style="list-style-type: none"> Training Loss: Blue line with circles. Starts at ~1.0 and decreases steadily to ~0.05. Validation Loss: Red line with diamonds. Starts at ~0.95, fluctuates between 0.95 and 1.25, and ends at ~1.20. Training Accuracy: Green line with circles. Starts at ~0.35 and increases to ~0.98. Validation Accuracy: Orange line with diamonds. Starts at ~0.50, peaks at ~0.58 at epoch 3, and fluctuates between 0.52 and 0.58.
1	[64]	False	0.5052	<p>Direction_Effect - Unidirectional Training History</p> <ul style="list-style-type: none"> Training Loss: Blue line with circles. Starts at ~1.0 and decreases steadily to ~0.15. Validation Loss: Red line with diamonds. Starts at ~1.05, fluctuates between 0.85 and 1.05, and ends at ~1.05. Training Accuracy: Green line with circles. Starts at ~0.40 and increases to ~0.95. Validation Accuracy: Orange line with diamonds. Starts at ~0.47, peaks at ~0.58 at epoch 5, and fluctuates between 0.47 and 0.58.
1	[64]	True	0.5453	<p>Layer_Effect - Single_Layer Training History</p> <ul style="list-style-type: none"> Training Loss: Blue line with circles. Starts at ~1.0 and decreases steadily to ~0.05. Validation Loss: Red line with diamonds. Starts at ~1.00, fluctuates between 0.90 and 1.05, and ends at ~1.05. Training Accuracy: Green line with circles. Starts at ~0.40 and increases to ~0.98. Validation Accuracy: Orange line with diamonds. Starts at ~0.50, peaks at ~0.58 at epoch 3, and fluctuates between 0.50 and 0.58.
2	[64, 32]	True	0.4888	<p>Layer_Effect - Two_Layers Training History</p> <ul style="list-style-type: none"> Training Loss: Blue line with circles. Starts at ~1.0 and decreases steadily to ~0.05. Validation Loss: Red line with diamonds. Starts at ~1.00, fluctuates between 0.85 and 1.15, and ends at ~1.10. Training Accuracy: Green line with circles. Starts at ~0.40 and increases to ~0.98. Validation Accuracy: Orange line with diamonds. Starts at ~0.44, peaks at ~0.58 at epoch 3, and fluctuates between 0.44 and 0.58.

3	[64, 32, 16]	True	0.4591	<p>The figure contains four subplots arranged in a 2x2 grid. The top-left plot shows 'Training Loss' (blue line with circles) decreasing from approximately 1.0 at epoch 0 to near 0.0 at epoch 8. The top-right plot shows 'Validation Loss' (red line with diamonds) starting at 1.0, fluctuating between 1.1 and 1.4, and ending at approximately 1.4. The bottom-left plot shows 'Training Accuracy' (green line with squares) increasing from 0.4 at epoch 0 to nearly 1.0 by epoch 4, then remaining flat. The bottom-right plot shows 'Validation Accuracy' (orange line with triangles) starting at 0.48, peaking at 0.50 around epoch 4, and then fluctuating between 0.44 and 0.48.</p>
2	[32, 16]	True	0.5580	<p>The figure contains four subplots arranged in a 2x2 grid. The top-left plot shows 'Training Loss' (blue line with circles) decreasing from approximately 1.0 at epoch 0 to near 0.0 at epoch 8. The top-right plot shows 'Validation Loss' (red line with diamonds) starting at 1.04, fluctuating between 0.90 and 1.04, and ending at approximately 1.04. The bottom-left plot shows 'Training Accuracy' (green line with squares) increasing from 0.4 at epoch 0 to nearly 1.0 by epoch 4, then remaining flat. The bottom-right plot shows 'Validation Accuracy' (orange line with triangles) starting at 0.55, peaking at 0.62 around epoch 4, and then fluctuating between 0.55 and 0.62.</p>
2	[64, 32]	True	0.4642	<p>The figure contains four subplots arranged in a 2x2 grid. The top-left plot shows 'Training Loss' (blue line with circles) decreasing from approximately 1.0 at epoch 0 to near 0.0 at epoch 8. The top-right plot shows 'Validation Loss' (red line with diamonds) starting at 1.0, fluctuating between 0.90 and 1.15, and ending at approximately 1.15. The bottom-left plot shows 'Training Accuracy' (green line with squares) increasing from 0.4 at epoch 0 to nearly 1.0 by epoch 4, then remaining flat. The bottom-right plot shows 'Validation Accuracy' (orange line with triangles) starting at 0.55, peaking at 0.62 around epoch 4, and then fluctuating between 0.54 and 0.62.</p>
2	[128, 64]	True	0.4068	<p>The figure contains four subplots arranged in a 2x2 grid. The top-left plot shows 'Training Loss' (blue line with circles) decreasing from approximately 1.0 at epoch 0 to near 0.0 at epoch 8. The top-right plot shows 'Validation Loss' (red line with diamonds) starting at 1.1, fluctuating between 1.1 and 1.6, and ending at approximately 1.6. The bottom-left plot shows 'Training Accuracy' (green line with squares) increasing from 0.4 at epoch 0 to nearly 1.0 by epoch 4, then remaining flat. The bottom-right plot shows 'Validation Accuracy' (orange line with triangles) starting at 0.38, peaking at 0.46 around epoch 4, and then fluctuating between 0.38 and 0.46.</p>

Hasil pengujian lebih lengkap dapat dilihat pada [Hasil Eksperimen SimpleRNN](#)

3.2.2.2. Perbandingan dengan Implementasi From Scratch

Pengujian dilakukan dengan menggunakan 2 lapisan SimpleRNN dengan jumlah unit masing-masing 64 dan 32 secara berurutan, serta unidirectional SimpleRNN.

Hasil dengan Keras:

```
F1 Score (Keras): 0.3785

==== Classification Report (Keras) ====
precision    recall   f1-score  support
          0       0.41     0.31     0.36      153
          1       0.27     0.33     0.30      96
          2       0.46     0.50     0.48     151

   accuracy                           0.39      400
  macro avg       0.38     0.38     0.38      400
weighted avg     0.40     0.39     0.39      400
```

Hasil from scratch:

```
F1 Score (Scratch): 0.3785

==== Classification Report (Scratch) ====
precision    recall   f1-score  support
          0       0.41     0.31     0.36      153
          1       0.27     0.33     0.30      96
          2       0.46     0.50     0.48     151

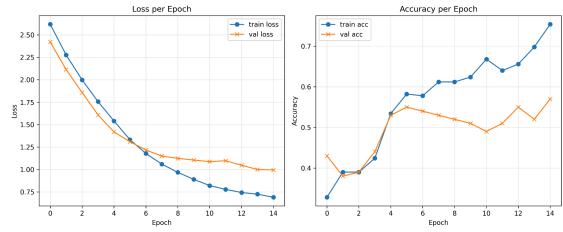
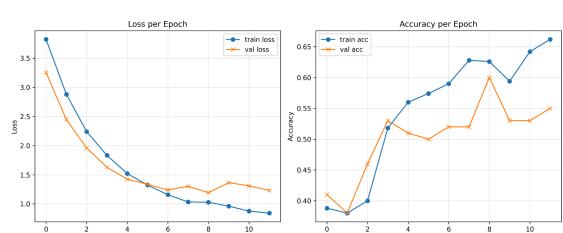
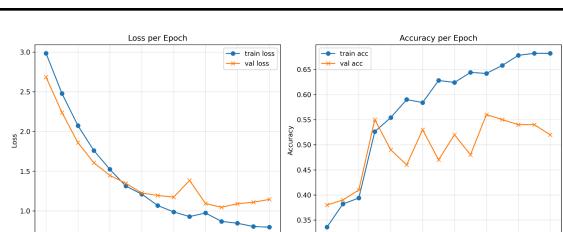
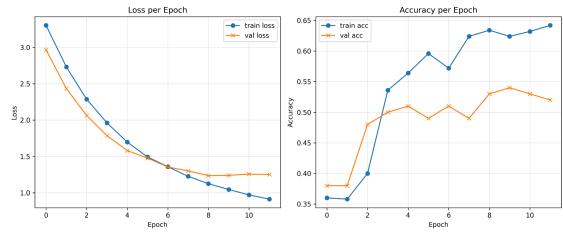
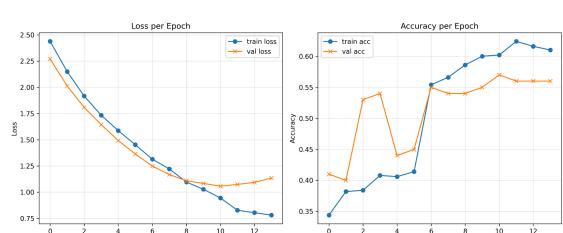
   accuracy                           0.39      400
  macro avg       0.38     0.38     0.38      400
weighted avg     0.40     0.39     0.39      400
```

3.2.3. Pengujian LSTM

Pengujian dilakukan dengan menggunakan dimensi embedding sebesar 64, panjang maksimum urutan 100, tingkat dropout sebesar 0.5, sebanyak 15 epoch, ukuran batch 32, dan *learning rate* sebesar 0.001.

3.2.3.1. Eksperimen Variasi Hyperparameter LSTM

Layers	Units	Bidirectional	Macro F1-Score	Graph
1	[64]	True	0.6862809901 985515	
2	[32, 16]	True	0.6242006938 245189	
2	[64, 64]	False	0.5998425805 205466	
1	[64]	False	0.5723244815 364682	

2	[32, 32]	False	0.5556420043 061264	
2	[128, 64]	False	0.5150677337 284764	
2	[64, 32]	False	0.5055911244 424305	
3	[64, 32, 16]	False	0.5003191867 826909	
2	[32, 16]	False	0.4778791122 79138	

Hasil pengujian lebih lengkap dapat dilihat pada [Hasil Eksperimen LSTM](#)

3.2.3.2. Perbandingan dengan Implementasi From Scratch

Pengujian dilakukan dengan menggunakan 2 lapisan LSTM dengan jumlah unit masing-masing 32 dan 16 secara berurutan, serta unidirectional LSTM.

Hasil dengan Keras:

```
Test Loss: 1.0046  
Test Accuracy: 0.5750  
Macro F1-Score: 0.5126
```

```
Classification Report:  
precision recall f1-score support  
  
0      0.58    0.10    0.17    153  
1      0.42    0.85    0.56     96  
2      0.75    0.88    0.81    151  
  
accuracy                           0.57    400  
macro avg       0.58    0.61    0.51    400  
weighted avg     0.60    0.57    0.50    400
```

Hasil from scratch:

```
Macro F1-Score: 0.5126  
  
Classification Report:  
precision recall f1-score support  
  
0      0.58    0.10    0.17    153  
1      0.42    0.85    0.56     96  
2      0.75    0.88    0.81    151  
  
accuracy                           0.57    400  
macro avg       0.58    0.61    0.51    400  
weighted avg     0.60    0.57    0.50    400
```

BAB IV

ANALISIS

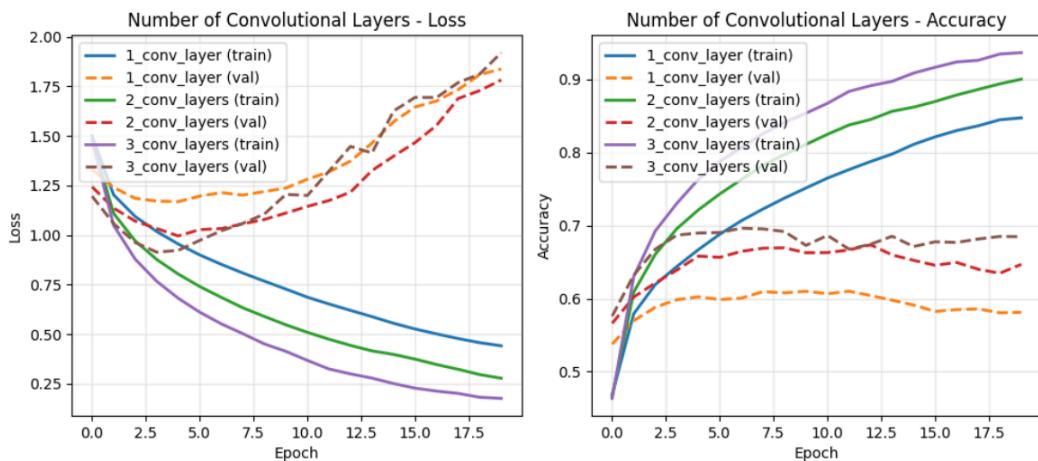
4.1. CNN

4.1.1. Pengaruh jumlah layer konvolusi

Berdasarkan pengujian yang telah dilakukan terhadap jumlah layer konvolusi, diperoleh hasil sebagai berikut:

n-layers	Filter per layer	Hasil
1	[32]	Test Accuracy: 0.5689 Test F1-Score (macro): 0.5658
2	[32, 64]	Test Accuracy: 0.6470 Test F1-Score (macro): 0.6457
3	[32, 64, 128]	Test Accuracy: 0.6784 Test F1-Score (macro): 0.6729

Dapat dilihat bahwa peningkatan jumlah layer konvolusi sebanding dengan peningkatan hasil accuracy dan F1-score terhadap data test.



Berdasarkan grafik Loss dan Accuracy pada gambar di atas, dapat dilihat bahwa semakin banyak jumlah layer konvolusi, semakin cepat nilai loss menurun selama epoch pelatihan, disertai dengan peningkatan akurasi yang lebih tinggi dan stabil jika dibandingkan dengan model satu dan dua layer.

Model dengan tiga layer konvolusi memiliki penurunan loss yang lebih tajam dibandingkan dengan model lainnya yang hanya memiliki satu atau dua layer. Hal ini berbanding lurus dengan teori, di mana semakin banyak layer, model dapat mengekstrak fitur yang lebih kompleks. Misalnya, dari yang hanya sekadar mempelajari fitur tepi dan tekstur pada layer awal, layer yang lebih dalam memungkinkan model untuk mengenali fitur spasial yang lebih komprehensif, seperti bentuk, pola, atau objek.

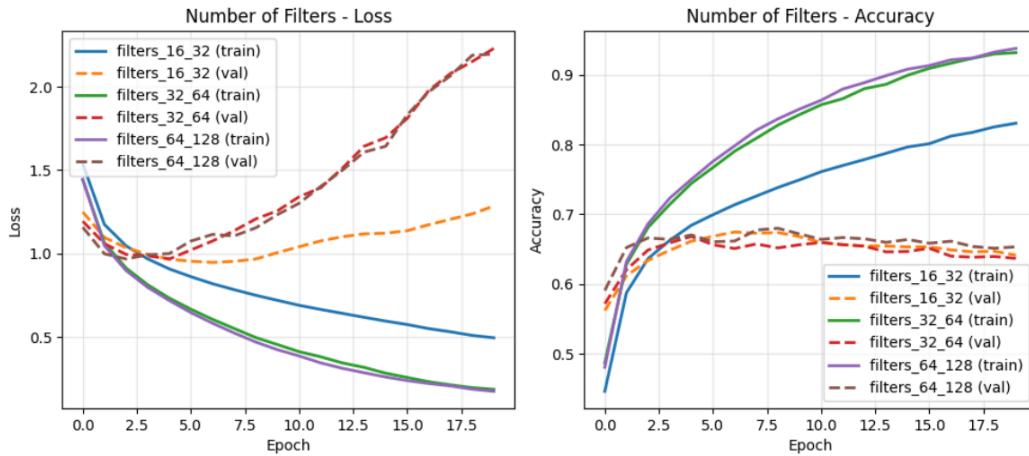
Namun, dapat juga dilihat melalui grafik bahwa semakin dalam model, semakin besar risiko overfitting, yaitu kondisi ketika seiring peningkatan jumlah epoch, loss cenderung stagnan atau bahkan meningkat, sementara akurasi cenderung stagnan atau menurun pada data validasi karena model terlalu belajar hingga menangkap *noise* dari input gambar.

4.1.2. Pengaruh banyak filter per layer konvolusi

Berdasarkan pengujian yang telah dilakukan terhadap banyak filter per layer konvolusi, diperoleh hasil sebagai berikut:

Filter per layer	Hasil
[16, 32]	Test Accuracy: 0.6497 Test F1-Score (macro): 0.6456
[32, 64]	Test Accuracy: 0.6393 Test F1-Score (macro): 0.6357
[64, 128]	Test Accuracy: 0.6515 Test F1-Score (macro): 0.6506

Dapat dilihat bahwa peningkatan banyak filter per layer konvolusi “cenderung” sebanding dengan peningkatan hasil accuracy dan F1-score terhadap data test.



Berdasarkan grafik Loss dan Accuracy pada gambar di atas, dapat dilihat bahwa semakin banyak jumlah filter pada layer konvolusi, semakin cepat nilai loss menurun selama epoch pelatihan, disertai dengan peningkatan akurasi yang lebih tinggi dan stabil jika dibandingkan dengan variasi jumlah filter lainnya.

Model dengan jumlah filter yang lebih banyak memiliki penurunan loss yang lebih tajam dibandingkan dengan model lainnya. Hal ini berbanding lurus dengan teori, dimana semakin banyak jumlah filter, semakin banyak parameter yang dipelajari oleh model, yang memungkinkan model untuk mengekstrak fitur yang lebih kompleks dan komprehensif dari data train.

Namun, dapat juga dilihat melalui grafik bahwa semakin banyak jumlah filter, semakin besar risiko overfitting, yaitu kondisi ketika seiring peningkatan jumlah epoch, loss cenderung stagnan atau bahkan meningkat, sementara akurasi cenderung stagnan atau menurun pada data validasi karena model terlalu belajar hingga menangkap *noise* dari input gambar.

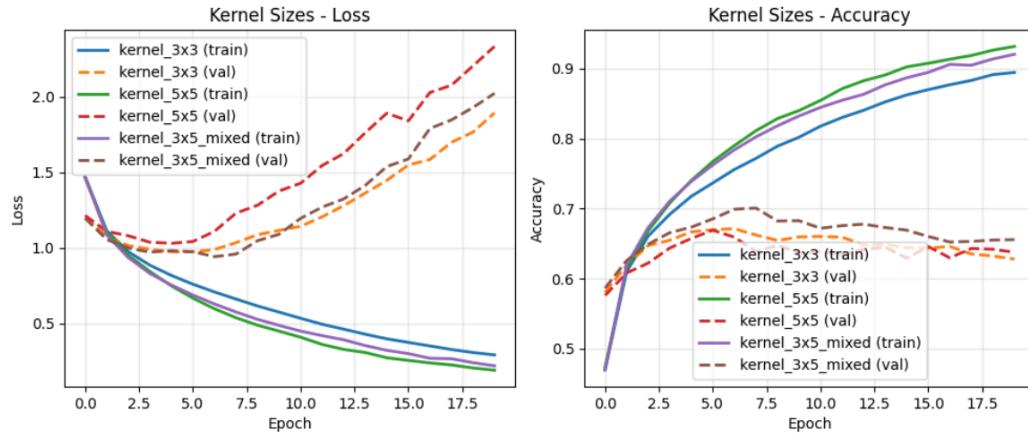
4.1.3. Pengaruh ukuran filter per layer konvolusi

Berdasarkan pengujian yang telah dilakukan terhadap ukuran filter per layer konvolusi, diperoleh hasil sebagai berikut:

Ukuran filter per layer	Hasil
[3, 3]	Test Accuracy: 0.6238 Test F1-Score (macro): 0.6240
[5, 5]	Test Accuracy: 0.6306

	Test F1-Score (macro): 0.6307
[3, 5]	Test Accuracy: 0.6547 Test F1-Score (macro): 0.6591

Dapat dilihat bahwa peningkatan ukuran filter per layer konvolusi sebanding dengan peningkatan hasil accuracy dan F1-score terhadap data test.



Berdasarkan grafik Loss dan Accuracy pada gambar di atas, dapat dilihat bahwa semakin besar ukuran filter pada layer konvolusi, semakin cepat nilai loss menurun selama epoch pelatihan, disertai dengan peningkatan akurasi yang lebih tinggi dan stabil jika dibandingkan dengan variasi jumlah filter lainnya. Namun, kombinasi ukuran filter pada layer konvolusi yang berbeda jauh lebih baik dibanding ukuran filter yang konstan untuk setiap setiap layer konvolusi.

Semakin besar ukuran filter pada layer konvolusi memungkinkan model untuk menangkap lebih banyak fitur spasial dalam setiap step konvolusi. Filter yang lebih besar mampu menangkap pola yang lebih luas dalam data, yang dapat memberikan gambaran secara komprehensif mengenai fitur pada input gambar. Hal ini mengarah pada penurunan loss yang lebih cepat dan peningkatan akurasi yang lebih stabil karena model mampu mengidentifikasi fitur yang lebih kompleks dalam data.

Namun, ada juga keuntungan menggunakan kombinasi ukuran filter yang berbeda, seperti kombinasi filter 3x3 dan 5x5. Kombinasi ini memungkinkan model untuk menggunakan filter yang lebih kecil untuk mendeteksi detail halus (seperti tepi atau tekstur), dan filter yang lebih besar untuk menangkap informasi

spasial yang lebih besar dan pola yang lebih kompleks. Dengan demikian, kombinasi filter ini dapat menambah kemampuan model untuk menggabungkan kelebihan dari kedua jenis filter, mempercepat proses pelatihan, dan mencegah overfitting, karena model dapat lebih baik menggeneralisasi pola dari berbagai tingkat kompleksitas dalam data.

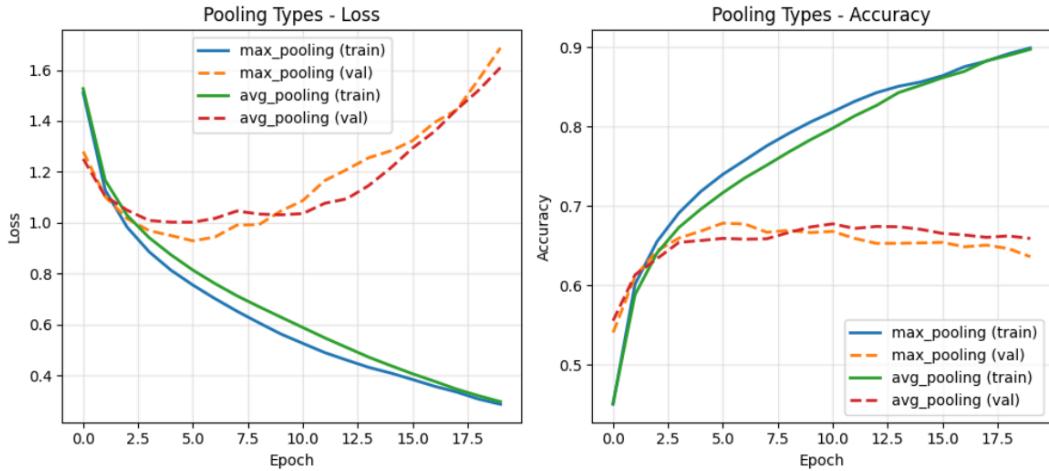
Namun, dapat juga dilihat melalui grafik bahwa semakin banyak jumlah filter, semakin besar risiko overfitting, yaitu kondisi ketika seiring peningkatan jumlah epoch, loss cenderung stagnan atau bahkan meningkat, sementara akurasi cenderung stagnan atau menurun pada data validasi karena kehilangan kemampuan untuk menggeneralisasi pada *unseen data*.

4.1.4. Pengaruh jenis pooling layer yang digunakan

Berdasarkan pengujian yang telah dilakukan terhadap jenis pooling pada tahap konvolusi, diperoleh hasil sebagai berikut:

Jenis Pooling	Hasil
Max Pooling	Test Accuracy: 0.6315 Test F1-Score (macro): 0.6278
Average Pooling	Test Accuracy: 0.6615 Test F1-Score (macro): 0.6581

Dapat dilihat bahwa *average pooling* memiliki nilai test accuracy serta F1-Score yang lebih baik dibandingkan *max pooling*.



Berdasarkan grafik Loss dan Accuracy di atas, dapat dilihat bahwa pada Max Pooling, penurunan loss terjadi lebih tajam dan stabil pada data pelatihan (train) serta data validasi (val) di epoch awal, namun model cenderung rentan mengalami overfitting pada data validasi seiring bertambahnya epoch. Sementara itu, pada Average Pooling, meskipun penurunan loss pada data pelatihan lebih halus dan lambat dibandingkan Max Pooling di awal, model menunjukkan kinerja yang lebih stabil dan tidak terlalu sensitif terhadap fluktuasi kecil pada data pelatihan, sehingga memperlambat kemungkinan terjadinya overfitting dan membantu model untuk dapat menggeneralisasi dengan lebih baik.

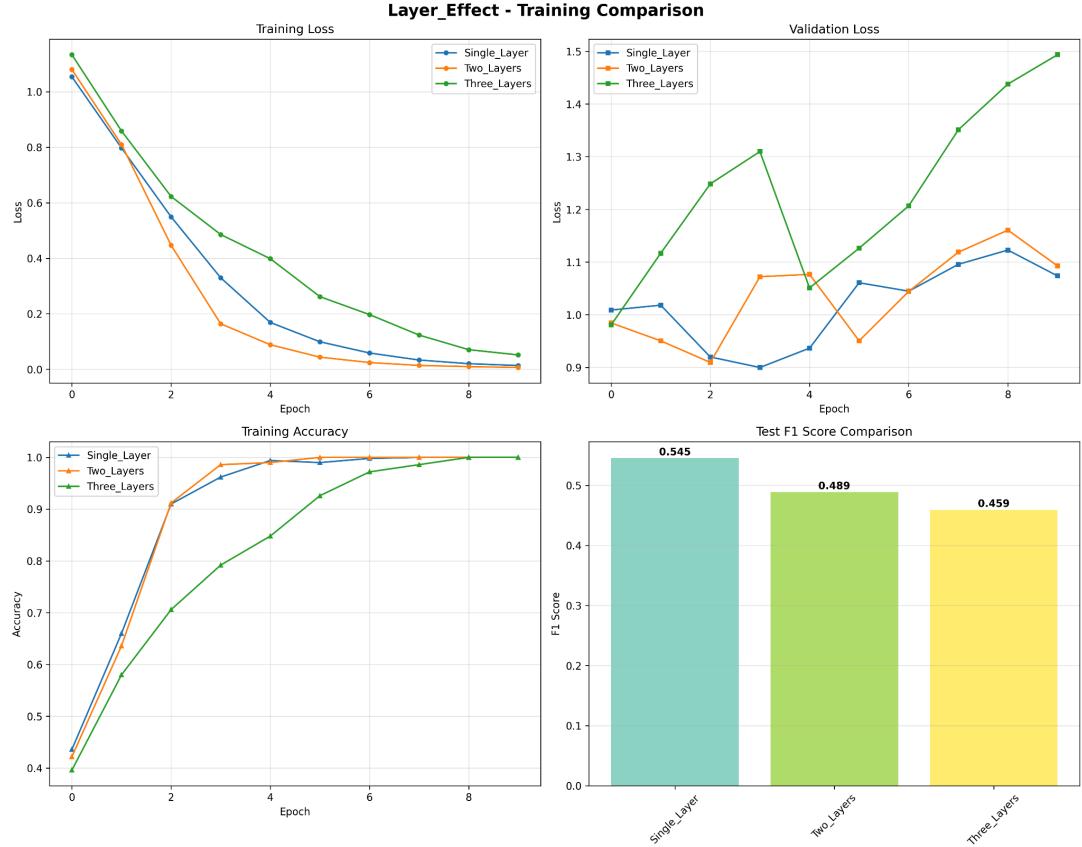
4.1.5. Perbandingan dengan implementasi *from scratch*

Berdasarkan tabel yang menunjukkan perbandingan antara model CNN Keras dan model *from scratch*, dapat dilihat bahwa pada setiap eksperimen yang dijalankan (baik dalam variasi jumlah layer, jumlah filter per layer, ukuran filter, maupun jenis pooling), hasil prediksi yang diperoleh oleh kedua model tersebut adalah identik. Selain itu, F1 difference yang mencapai nilai 0,000x serta nilai MSE dan *max difference* yang sangat kecil menunjukkan bahwa meskipun kedua model tersebut memiliki beberapa perbedaan arsitektur atau proses training, kualitas prediksi yang dihasilkan sangat mirip. Dengan demikian, dapat disimpulkan bahwa model *from scratch* yang dibuat berhasil meniru model Keras karena memiliki kemampuan untuk menghasilkan prediksi yang serupa, meskipun mungkin saja menggunakan implementasi dan pendekatan yang berbeda dalam

arsitektur dan pelatihan (dimana model keras tentunya sudah melewati berbagai proses optimasi yang dilakukan).

4.2. RNN

4.2.1. Pengaruh jumlah layer RNN



Eksperimen dilakukan dengan memvariasikan jumlah layer RNN sebanyak 1, 2, dan 3, menggunakan konfigurasi unit [64], [64, 32], dan [64, 32, 16] secara berurutan. Semua model bersifat bidirectional agar hasil tidak dipengaruhi oleh arah.

Jumlah Layer	Konfigurasi Unit	Macro F1-Score
1	[64]	0.5453
2	[64, 32]	0.4888
3	[64, 32, 16]	0.4591

Hasil evaluasi menunjukkan bahwa model dengan satu layer RNN menghasilkan performa terbaik dengan macro F1-score sebesar 0.5453. Sebaliknya, penambahan jumlah layer justru menurunkan performa model, di mana konfigurasi dua layer mendapat F1-score sebesar 0.4888, dan tiga layer hanya mencapai 0.4591. Hal ini menunjukkan bahwa penambahan layer tidak selalu sejalan dengan peningkatan kualitas klasifikasi.

Fakta ini dapat dijelaskan melalui beberapa kemungkinan. Pertama, penambahan layer secara langsung meningkatkan jumlah parameter dalam model, yang dapat menyebabkan overfitting, terutama jika data pelatihan tidak cukup besar atau tidak cukup beragam. Kedua, jaringan RNN dalam bentuk sederhana (SimpleRNN) diketahui memiliki keterbatasan dalam menangani urutan panjang, dan model dengan banyak layer dapat mengalami masalah *vanishing gradient* yang semakin parah. Hal ini membuat proses pelatihan menjadi lebih sulit dan konvergensi model tidak optimal.

Secara keseluruhan, hasil eksperimen ini menunjukkan bahwa model yang lebih dalam belum tentu lebih baik, dan dalam kasus tertentu seperti klasifikasi teks menggunakan SimpleRNN, struktur yang tidak dalam justru dapat memberikan hasil yang lebih stabil dan akurat.

4.2.2. Pengaruh banyak cell RNN per layer

Eksperimen ini bertujuan untuk mengevaluasi pengaruh jumlah unit atau cell pada setiap layer SimpleRNN terhadap performa model. Seluruh model dikonfigurasi menggunakan dua layer RNN dan bersifat bidirectional agar fokus eksperimen hanya pada jumlah unit, bukan pada arah propagasi. Tiga kombinasi jumlah unit yang diuji adalah [32, 16], [64, 32], dan [128, 64].

Konfigurasi Unit	Macro F1-Score
[32, 16]	0.5580
[64, 32]	0.4642

[128, 64]	0.4068
-----------	--------

Hasil evaluasi menunjukkan bahwa konfigurasi dengan jumlah unit paling kecil yaitu [32, 16] justru memberikan performa terbaik dengan macro F1-score sebesar 0.5580. Sebaliknya, konfigurasi dengan unit lebih besar seperti [64, 32] dan [128, 64] menunjukkan penurunan performa berturut-turut menjadi 0.4642 dan 0.4068.

Hasil pengujian ini mengindikasikan bahwa penambahan jumlah unit per layer tidak selalu memberikan keuntungan, bahkan bisa berdampak negatif. Jumlah unit yang terlalu besar dapat menyebabkan model menjadi terlalu kompleks, sehingga rentan mengalami overfitting, terutama pada dataset yang berukuran terbatas atau memiliki variasi distribusi kelas yang tidak merata. Selain itu, semakin besar ukuran hidden state, semakin besar pula risiko *vanishing/exploding gradient*, yang dapat menghambat pembelajaran efektif selama training. Dengan demikian, eksperimen ini menunjukkan bahwa jumlah unit yang moderat cenderung lebih optimal dalam menghasilkan generalisasi yang baik pada klasifikasi teks dengan SimpleRNN.

4.2.3. Pengaruh jenis layer RNN berdasarkan arah

Eksperimen ini dilakukan untuk menganalisis pengaruh arah propagasi informasi dalam arsitektur Recurrent Neural Network, dengan membandingkan performa antara bidirectional dan unidirectional RNN. Kedua model dikonfigurasi dengan jumlah layer dan unit yang sama, yaitu dua layer dengan unit [64]. Tujuan eksperimen ini adalah untuk mengisolasi variabel arah (directionality) dan melihat dampaknya terhadap performa model dalam tugas klasifikasi teks.

Jumlah Layer	Unit	Bidirectional	Macro F1-Score
1	[64]	True	0.5534

1	[64]	False	0.5052
---	------	-------	--------

Hasil pengujian menunjukkan bahwa model bidirectional RNN memberikan performa yang lebih baik dengan macro F1-score sebesar 0.5534, dibandingkan dengan model unidirectional RNN yang memperoleh 0.5052. Perbedaan ini mengindikasikan bahwa penggunaan arah bidirectional dalam RNN berhasil meningkatkan akurasi prediksi. Bidirectional RNN memungkinkan model untuk memproses informasi baik dari awal maupun akhir urutan teks, sehingga dapat menangkap konteks yang lebih lengkap, terutama pada kalimat yang struktur informasinya tidak hanya linier dari kiri ke kanan.

Kelebihan ini menjadi signifikan pada model dengan arsitektur yang relatif ringan (satu layer), di mana setiap tambahan informasi kontekstual dapat meningkatkan kualitas representasi. Sebaliknya, model unidirectional hanya mengandalkan informasi dari arah maju, yang justru dapat menyebabkan kehilangan konteks penting pada bagian akhir kalimat.

4.2.4. Perbandingan dengan implementasi from scratch

Eksperimen dilakukan dengan membandingkan performa model SimpleRNN yang diimplementasikan menggunakan framework Keras dengan model yang dibangun from scratch menggunakan layer manual untuk Embedding, RNNCell, Dense, dan Dropout.

Model	Macro F1-Score
Keras	0.3785
From Scratch	0.3785

Kedua model diuji menggunakan konfigurasi arsitektur dan bobot yang sama, dengan bobot hasil pelatihan Keras diekstraksi dan disisipkan ke dalam model scratch. Hasil pengujian menunjukkan bahwa performa kedua model

identik, dengan macro F1-score sebesar 0.3785 untuk masing-masing. Hal ini menunjukkan bahwa implementasi from scratch yang dilakukan sudah benar secara fungsional dan berhasil mereplikasi perilaku model Keras secara tepat selama proses inferensi.

4.3. LSTM

4.3.1. Pengaruh jumlah layer LSTM

Eksperimen dilakukan dengan memvariasikan jumlah layer LSTM sebanyak 1, 2, dan 3, menggunakan konfigurasi unit [64], [64, 32], dan [64, 32, 16] secara berurutan. Semua model bersifat unidirectional agar hasil tidak dipengaruhi oleh arah.

Jumlah Layer	Konfigurasi Unit	Macro F1-Score
1	[64]	0.5723
2	[64, 32]	0.5056
3	[64, 32, 16]	0.5003

Penambahan jumlah layer LSTM tidak selalu meningkatkan performa. Model dengan satu layer justru menghasilkan macro F1-score terbaik. Kemungkinan, model dengan terlalu banyak layer mengalami overfitting atau kesulitan dalam konvergensi karena jumlah parameter yang meningkat.

4.3.2. Pengaruh banyak cell LSTM per layer

Eksperimen ini mengevaluasi kombinasi jumlah unit (cell) LSTM dengan tetap menggunakan dua layer dan konfigurasi unidirectional.

Konfigurasi Unit	Macro F1-Score
[32, 16]	0.4779
[64, 64]	0.5998

[128, 64]	0.5151
-----------	--------

Model dengan kombinasi [64, 64] menghasilkan performa terbaik. Jumlah cell yang terlalu kecil seperti [32, 16] kurang mampu menangkap kompleksitas data, sedangkan jumlah yang terlalu besar seperti [128, 64] mungkin menimbulkan overfitting atau memerlukan regularisasi lebih kuat.

4.3.3. Pengaruh jenis layer LSTM berdasarkan arah

Eksperimen ini membandingkan performa antara LSTM unidirectional dan bidirectional dengan konfigurasi jumlah layer dan unit yang sama.

Jumlah Layer	Unit	Bidirectional	Macro F1-Score
1	[64]	False	0.5723
1	[64]	True	0.6863
2	[32, 16]	False	0.4779
2	[32, 16]	True	0.6242

Bidirectional LSTM secara konsisten menghasilkan performa yang lebih tinggi dibandingkan unidirectional. Hal ini wajar karena bidirectional LSTM dapat memproses informasi dari dua arah, yang sangat berguna dalam konteks data sekuensial seperti teks.

4.3.4. Perbandingan dengan implementasi from scratch

Eksperimen dilakukan dengan membandingkan performa model LSTM yang diimplementasikan menggunakan framework Keras dengan model yang dibangun from scratch menggunakan layer manual untuk Embedding, LSTMCell, Dense, dan Dropout.

Model	Macro F1-Score
-------	----------------

Keras	0.5126
From Scratch	0.5126

Hasil pengujian menunjukkan bahwa implementasi from scratch berhasil mereplikasi perilaku model Keras secara identik, baik dari segi nilai macro F1-score maupun classification report untuk masing-masing kelas. Ini membuktikan bahwa proses forward propagation, struktur LSTM, serta penanganan masking dan padding dalam implementasi manual telah dilakukan secara benar dan sesuai dengan standar yang digunakan Keras.

BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Peningkatan jumlah layer, filter, dan ukuran filter pada model CNN secara umum berbanding lurus dengan peningkatan akurasi dan F1-score pada *data train*. Namun, penambahan elemen-elemen ini juga meningkatkan potensi overfitting, terutama pada epoch yang lebih dalam. Penggunaan *average pooling* juga telah terbukti lebih *robust* terhadap *noise* dan membantu model dalam menggeneralisasi dengan lebih baik pada data validasi dan *unseen data*. Model CNN yang diimplementasikan from scratch berhasil meniru performa model Keras, yang menunjukkan bahwa arsitektur dan pelatihan dilakukan dengan benar.

Pada model SimpleRNN, penambahan layer tidak selalu meningkatkan performa, dengan model satu layer memberikan macro F1-score terbaik. Selain itu, bidirectional RNN memberikan hasil lebih baik dibandingkan unidirectional. Penggunaan model LSTM juga menunjukkan bahwa penambahan jumlah layer tidak selalu meningkatkan performa, dan konfigurasi bidirectional LSTM lebih efektif dalam menangkap informasi dari dua arah, terutama pada data sekuensial. LSTM mengatasi masalah *vanishing gradient* yang sering terjadi pada RNN tradisional, dan eksperimen menunjukkan bahwa implementasi from scratch untuk LSTM berhasil meniru model Keras dengan hasil macro F1-score identik.

5.2. Saran

Selama penggerjaan tugas besar ini kami memiliki beberapa saran untuk penggerjaan tugas besar berikutnya, di antaranya:

- Mencari referensi lebih banyak lagi sebagai bahan literasi penambah wawasan.
- Membuat sheets berisi link-link terintegrasi yang dibutuhkan selama penggerjaan tugas serta jadwal yang jelas terkait waktu pertemuan maupun *soft deadline* untuk setiap task.

PEMBAGIAN TUGAS

NIM	Nama	Kontribusi
13522012	Thea Josephine Halim	Simple RNN
13522046	Raffael Boymian Siahaan	CNN
13522096	Novelya Putri Ramadhani	LSTM

LAMPIRAN

Repository

https://github.com/pandaandsushi/IF3270_Tubes2_CNN_RNN

DAFTAR PUSTAKA

- Asisten, S. A. (2025). Pembelajaran Mesin Convolutional Neural Network dan Recurrent Neural Network. Diakses dari Google Dokumen: <https://docs.google.com/document>
- Holdsworth, J. What is NLP (natural language processing)? Diakses pada tanggal 15 Mei 2025, dari <https://www.ibm.com/think/topics/natural-language-processing>
- Raghav, P. (2018, 8 Maret). Understanding of Convolutional Neural Network (CNN) — Deep Learning. Diakses pada tanggal 20 Mei 2025, dari <https://medium.com/@RaghavPrabhu/>
- StatQuest with Josh Starmer. (2022, 11 Juli). Recurrent Neural Networks (RNNs), Clearly Explained!!! [Video]. <https://www.youtube.com/watch?v=AsNTP8Kwu80>
- StatQuest with Josh Starmer. (2022, 7 November). Long Short-Term Memory (LSTM), Clearly Explained [Video]. <https://www.youtube.com/watch?v=YCzL96nL7j0>
- TechwithJulles. (2024, 13 Oktober). Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) — Creating an LSTM Model in Python Using TensorFlow and Keras. Diakses pada tanggal 20 Mei 2025, dari <https://medium.com/@techwithjulles>