Laporan Tugas Besar 2

IF3270 Pembelajaran Mesin

Convolutional Neural Network dan Recurrent Neural Network

Disusun Oleh

13522001 Mohammad Nugraha Eka Prawira

13522105 Fabian Radenta Bangun

Sekolah Teknik Elektro dan Informatika

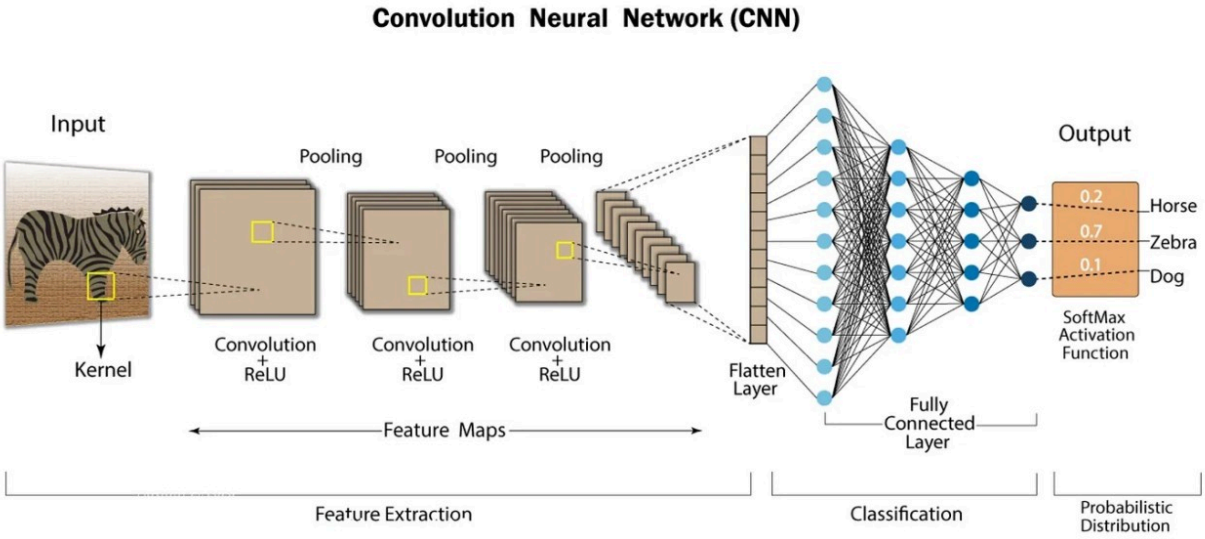
Institut Teknologi Bandung

Semester VI - 2025

DAFTAR ISI

DESKRIPSI PERSOALAN

Convolutional Neural Network

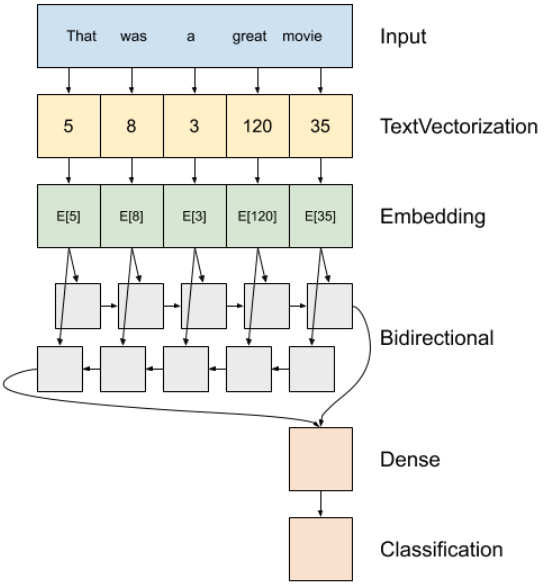


Untuk bagian CNN, Anda diminta untuk melakukan beberapa hal berikut:

- Lakukan pelatihan suatu model CNN untuk *image classification* dengan library Keras dan dengan dataset CIFAR-10 yang memenuhi ketentuan berikut ini.
    - Model minimal harus memiliki jenis layer berikut didalamnya (urutan dan jumlah layer silakan disesuaikan sendiri):
        - Conv2D layer
        - Pooling layers
        - Flatten/Global Pooling layer
        - Dense layer
    - Loss function yang digunakan adalah Sparse Categorical Crossentropy (untuk menangani kasus klasifikasi multikelas)
    - Optimizer yang digunakan adalah Adam
    - Dataset CIFAR-10 yang disediakan hanya terdiri dari dua split data saja, yaitu *train* dan *test*. Tambahkan split ke-3 (*validation set*) dengan cara membagi training set yang sudah ada dengan menjadi training set yang lebih kecil dan validation set dengan rasio 4:1 (jumlah data akhir adalah 40k *train* data, 10k validation data, dan 10k test data)
- Lakukan variasi pelatihan sebagai berikut untuk analisis pengaruh beberapa hyperparameter dalam CNN:

    - Pengaruh **jumlah layer konvolusi**
        - Pilih **3 variasi** jumlah layer konvolusi
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana jumlah layer konvolusi mempengaruhi kinerja model
    - Pengaruh **banyak filter per layer konvolusi**
        - Pilih **3 variasi** kombinasi banyak filter per layer konvolusi
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana banyak filter per layer konvolusi mempengaruhi kinerja model
    - Pengaruh **ukuran filter per layer konvolusi**
        - Pilih **3 variasi** kombinasi banyak filter per layer konvolusi
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana ukuran filter per layer konvolusi mempengaruhi kinerja model
    - Pengaruh **jenis pooling layer** yang digunakan
        - Pilih **2 variasi** pooling layer (antara max pooling atau average pooling)
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana jenis pooling layer mempengaruhi kinerja model
    - Catatan: Gunakan macro f1-score sebagai metrik ketika membandingkan hasil akhir prediksi.
- Simpan hasil bobot dari pelatihan.
- Buatlah modul *forward propagation from scratch* dari model yang telah dibuat dengan ketentuan sebagai berikut:
    - Dapat membaca model hasil pelatihan dengan Keras (bobotnya sama dengan bobot hasil pelatihan dengan Keras).
    - Direkomendasikan untuk mengimplementasikan *forward propagation* secara modular, yaitu dengan cara mengimplementasikan method *forward propagation* untuk setiap layer.
    - Lakukan pengujian dengan membandingkan hasil forward propagation *from scratch* dengan hasil forward propagation menggunakan Keras.
    - Gunakan split data test untuk menguji implementasi forward propagation. Metrik yang digunakan adalah macro f1-score.
    - Catatan: Khusus untuk **Dense layer**, Anda boleh menggunakan implementasi *forward propagation* FFNN dari Tubes 1.

Simple Recurrent Neural Network

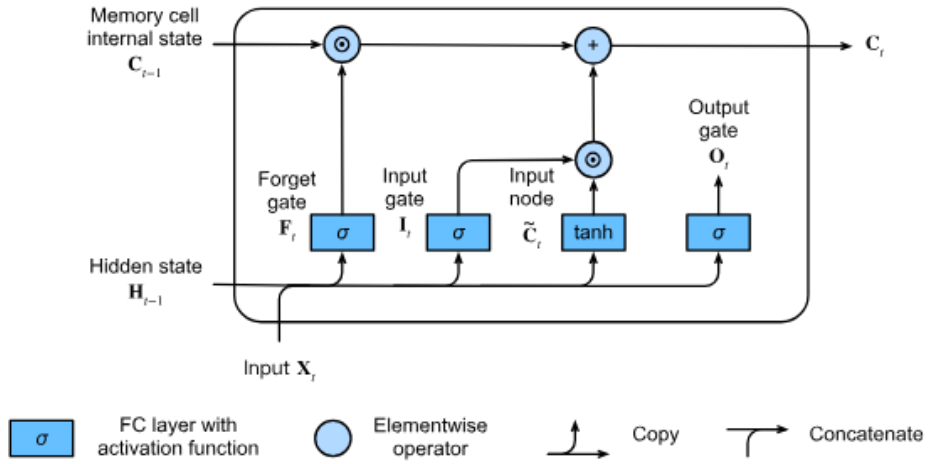


Untuk bagian ini, Anda diminta untuk melakukan beberapa hal berikut:

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

    - Loss function yang digunakan adalah Sparse Categorical Crossentropy (untuk menangani kasus klasifikasi multikelas)
    - Optimizer yang digunakan adalah Adam
- Lakukan variasi pelatihan sebagai berikut untuk analisis pengaruh beberapa hyperparameter dalam RNN:
    - Pengaruh **jumlah layer RNN**
        - Pilih **3 variasi** jumlah layer RNN
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana jumlah layer RNN mempengaruhi kinerja model
    - Pengaruh **banyak cell RNN per layer**
        - Pilih **3 variasi** kombinasi banyak cell RNN per layer
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana banyak cell RNN per layer mempengaruhi kinerja model
    - Pengaruh **jenis layer RNN berdasarkan arah**
        - Pilih **2 variasi** jenis layer RNN berdasarkan arah (bidirectional atau unidirectional)
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana jenis layer RNN berdasarkan arah mempengaruhi kinerja model
    - Catatan: Gunakan macro f1-score sebagai metrik ketika membandingkan hasil akhir prediksi.
- Simpan weight hasil pelatihan dengan Keras.
- Buatlah modul *forward propagation from scratch* dari model yang telah dibuat dengan ketentuan sebagai berikut:
    - Dapat membaca model hasil pelatihan dengan Keras (bobotnya sama dengan bobot hasil pelatihan dengan Keras).
    - Direkomendasikan untuk mengimplementasikan *forward propagation* secara modular, yaitu dengan cara mengimplementasikan method *forward propagation* untuk setiap layer.
    - Bandingkan hasil forward propagation *from scratch* dengan hasil forward propagation menggunakan Keras. Gunakan split data test untuk menguji implementasi forward propagation. Metrik yang digunakan adalah macro f1-score.
    - Catatan: Khusus untuk **Dense layer**, Anda boleh menggunakan implementasi *forward propagation* FFNN dari Tubes 1.

Long-Short Term Memory Network



Untuk bagian ini, Anda diminta untuk melakukan beberapa hal berikut:

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

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

      Embedding function merupakan fungsi yang memetakan tiap token ke dalam suatu ruang vektor berdimensi-n, sehingga setiap token (yang sudah berbentuk vektor) dapat dioperasikan satu sama lain. Untuk tugas ini, Anda cukup memanfaatkan embedding layer yang disediakan oleh Keras untuk mengonversi token ke dalam bentuk vektor.
- Lakukan pelatihan untuk suatu model LSTM untuk *text classification* dengan dataset NusaX-Sentiment (Bahasa Indonesia) dan dengan menggunakan Keras yang memenuhi ketentuan berikut ini.
    - Model minimal harus memiliki jenis layer-layer berikut didalamnya (urutan dan jumlah layer silakan disesuaikan sendiri):
        - Embedding layer
        - Bidirectional LSTM layer dan/atau Unidirectional LSTM layer
        - Dropout layer
        - Dense layer
    - Loss function yang digunakan adalah Sparse Categorical Crossentropy (untuk menangani kasus klasifikasi multikelas)
    - Optimizer yang digunakan adalah Adam

- Lakukan variasi pelatihan sebagai berikut untuk analisis pengaruh beberapa hyperparameter dalam LSTM:
    - Pengaruh **jumlah layer LSTM**
        - Pilih **3 variasi** jumlah layer LSTM
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana jumlah layer LSTM mempengaruhi kinerja model
    - Pengaruh **banyak cell LSTM per layer**
        - Pilih **3 variasi** kombinasi banyak cell LSTM per layer
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana banyak cell LSTM per layer mempengaruhi kinerja model
    - Pengaruh **jenis layer LSTM berdasarkan arah**
        - Pilih **2 variasi** jenis layer RNN berdasarkan arah (bidirectional atau unidirectional)
        - Bandingkan hasil akhir prediksinya
        - Bandingkan grafik training loss dan validation loss tiap epoch
        - Berikan kesimpulan bagaimana jenis layer LSTM berdasarkan arah mempengaruhi kinerja model
    - Catatan: Gunakan macro f1-score sebagai metrik ketika membandingkan hasil akhir prediksi.
- Simpan weight hasil pelatihan dengan Keras.
- Buatlah modul *forward propagation from scratch* dari model yang telah dibuat dengan ketentuan sebagai berikut:
    - Dapat membaca model hasil pelatihan dengan Keras (bobotnya sama dengan bobot hasil pelatihan dengan Keras).
    - Direkomendasikan untuk mengimplementasikan *forward propagation* secara modular, yaitu dengan cara mengimplementasikan method *forward propagation* untuk setiap layer.
    - Bandingkan hasil forward propagation *from scratch* dengan hasil forward propagation menggunakan Keras. Gunakan split data test untuk menguji implementasi forward propagation. Metrik yang digunakan adalah macro f1-score.
    - Catatan: Khusus untuk **Dense layer**, Anda boleh menggunakan implementasi *forward propagation* FFNN dari Tubes 1.

PEMBAHASAN

Penjelasan Implementasi

Deskripsi Kelas dan Fungsi

File cnn.py

```
import numpy as np
import pickle
from typing import Tuple, List, Dict, Any
import tensorflow as tf
from tensorflow import keras

class ConvLayer:
   def __init__(self, weights: np.ndarray, bias: np.ndarray, strides: Tuple[int,
int] = (1, 1), padding: str = 'valid', activation: str = 'relu'):
       self.weights = weights
       self.bias = bias
       self.strides = strides
       self.padding = padding
       self.activation = activation

   def apply_activation(self, x: np.ndarray) -> np.ndarray:
       if self.activation == 'relu':
           return np.maximum(0, x)
       elif self.activation == 'linear':
           return x
       else:
```

```
            raise ValueError(f"Unsupported activation function: {self.activation}")

    def forward(self, x: np.ndarray) -> np.ndarray:
        batch_size, input_height, input_width, input_channels = x.shape
        filter_height, filter_width, _, num_filters = self.weights.shape

        if self.padding == 'valid':
            output_height = (input_height - filter_height) // self.strides[0] + 1
            output_width = (input_width - filter_width) // self.strides[1] + 1
            padded_x = x
        else:
            output_height = int(np.ceil(input_height / self.strides[0]))
            output_width = int(np.ceil(input_width / self.strides[1]))

            pad_along_height = max((output_height - 1) * self.strides[0] +
filter_height - input_height, 0)
            pad_along_width = max((output_width - 1) * self.strides[1] + filter_width
- input_width, 0)
            pad_top = pad_along_height // 2
            pad_bottom = pad_along_height - pad_top
            pad_left = pad_along_width // 2
            pad_right = pad_along_width - pad_left
            padded_x = np.pad(x, ((0, 0), (pad_top, pad_bottom), (pad_left,
pad_right), (0, 0)), mode='constant', constant_values=0)
        output = np.zeros((batch_size, output_height, output_width, num_filters))

        for b in range(batch_size):
            for f in range(num_filters):
                for h in range(output_height):
                    for w in range(output_width):
                        h_start = h * self.strides[0]
                        h_end = h_start + filter_height
                        w_start = w * self.strides[1]
                        w_end = w_start + filter_width
                        region = padded_x[b, h_start:h_end, w_start:w_end, :]
                        output[b, h, w, f] = np.sum(region * self.weights[:, :, :,
f]) + self.bias[f]
        return self.apply_activation(output)

class PoolingLayer:
    def __init__(self, pool_size: Tuple[int, int] = (2, 2), strides: Tuple[int, int]
```

```
= (2, 2), pool_type: str = 'max'):
        self.pool_size = pool_size
        self.strides = strides
        self.pool_type = pool_type

    def forward(self, x: np.ndarray) -> np.ndarray:
        batch_size, input_height, input_width, channels = x.shape
        pool_height, pool_width = self.pool_size

        output_height = (input_height - pool_height) // self.strides[0] + 1
        output_width = (input_width - pool_width) // self.strides[1] + 1
        output = np.zeros((batch_size, output_height, output_width, channels))

        for b in range(batch_size):
            for c in range(channels):
                for h in range(output_height):
                    for w in range(output_width):
                        h_start = h * self.strides[0]
                        h_end = h_start + pool_height
                        w_start = w * self.strides[1]
                        w_end = w_start + pool_width
                        region = x[b, h_start:h_end, w_start:w_end, c]
                        if self.pool_type == 'max':
                            output[b, h, w, c] = np.max(region)
                        elif self.pool_type == 'average':
                            output[b, h, w, c] = np.mean(region)
        return output

class FlattenLayer:
    def forward(self, x: np.ndarray) -> np.ndarray:
        batch_size = x.shape[0]
        return x.reshape(batch_size, -1)

class GlobalPoolingLayer:
    def __init__(self, pool_type: str = 'average'):
        self.pool_type = pool_type

    def forward(self, x: np.ndarray) -> np.ndarray:
        if self.pool_type == 'average':
            return np.mean(x, axis=(1, 2))
        elif self.pool_type == 'max':
```

```
            return np.max(x, axis=(1, 2))

class DenseLayer:
    def __init__(self, weights: np.ndarray, bias: np.ndarray, activation: str =
'relu'):
        self.weights = weights
        self.bias = bias
        self.activation = activation

    def forward(self, x: np.ndarray) -> np.ndarray:
        z = np.dot(x, self.weights) + self.bias

        if self.activation == 'relu':
            return np.maximum(0, z)
        elif self.activation == 'softmax':

            exp_z = np.exp(z - np.max(z, axis=1, keepdims=True))
            return exp_z / np.sum(exp_z, axis=1, keepdims=True)
        elif self.activation == 'linear':
            return z
        else:
            raise ValueError(f"Unsupported activation function: {self.activation}")

class CNNFromScratch:
    def __init__(self):
        self.layers = []

    def add_layer(self, layer):
        """Add a layer to the network"""
        self.layers.append(layer)

    def forward(self, x: np.ndarray) -> np.ndarray:
        output = x
        for layer in self.layers:
            output = layer.forward(output)
        return output

    def predict(self, x: np.ndarray) -> np.ndarray:
        predictions = self.forward(x)
        return np.argmax(predictions, axis=1)
```

```
    def predict_proba(self, x: np.ndarray) -> np.ndarray:
        return self.forward(x)

def load_keras_model_weights(keras_model_path: str) -> Dict[str, Any]:
    model = keras.models.load_model(keras_model_path)
    weights_dict = {}

    layer_idx = 0
    for layer in model.layers:
        if hasattr(layer, 'get_weights') and layer.get_weights():
            weights = layer.get_weights()
            layer_name = layer.__class__.__name__.lower()

            if 'conv' in layer_name:
                weights_dict[f'conv_{layer_idx}'] = {
                    'weights': weights[0],
                    'bias': weights[1],
                    'strides': layer.strides,
                    'padding': layer.padding
                }
            elif 'dense' in layer_name:
                activation = layer.activation.__name__ if hasattr(layer.activation,
'__name__') else 'linear'
                weights_dict[f'dense_{layer_idx}'] = {
                    'weights': weights[0],
                    'bias': weights[1],
                    'activation': activation
                }
            elif 'pooling' in layer_name or 'pool' in layer_name:
                pool_type = 'max' if 'max' in layer_name else 'average'
                weights_dict[f'pool_{layer_idx}'] = {
                    'pool_size': layer.pool_size,
                    'strides': layer.strides,
                    'pool_type': pool_type
                }

            layer_idx += 1
    return weights_dict

def build_cnn_from_keras(keras_model_path: str) -> CNNFromScratch:
    keras_model = keras.models.load_model(keras_model_path)
```

```
    cnn = CNNFromScratch()
    for layer in keras_model.layers:
        layer_name = layer.__class__.__name__.lower()
        if 'conv' in layer_name and hasattr(layer, 'get_weights') and
layer.get_weights():
            weights = layer.get_weights()

            activation = 'linear'
            if hasattr(layer, 'activation') and hasattr(layer.activation,
'__name__'):
                activation = layer.activation.__name__

            conv_layer = ConvLayer(
                weights=weights[0],
                bias=weights[1],
                strides=layer.strides,
                padding=layer.padding,
                activation=activation
            )
            cnn.add_layer(conv_layer)
        elif 'activation' in layer_name:
            continue
        elif 'maxpool' in layer_name or 'averagepool' in layer_name:
            pool_type = 'max' if 'max' in layer_name else 'average'
            pool_layer = PoolingLayer(
                pool_size=layer.pool_size,
                strides=layer.strides,
                pool_type=pool_type
            )
            cnn.add_layer(pool_layer)
        elif 'flatten' in layer_name:
            flatten_layer = FlattenLayer()
            cnn.add_layer(flatten_layer)
        elif 'globalpooling' in layer_name or 'globalaverage' in layer_name:
            pool_type = 'max' if 'max' in layer_name else 'average'
            global_pool_layer = GlobalPoolingLayer(pool_type=pool_type)
            cnn.add_layer(global_pool_layer)
        elif 'dense' in layer_name and hasattr(layer, 'get_weights') and
layer.get_weights():
            weights = layer.get_weights()
            activation = 'linear'
```

```
            if hasattr(layer, 'activation') and hasattr(layer.activation,
'__name__'):
                activation = layer.activation.__name__
            dense_layer = DenseLayer(
                weights=weights[0],
                bias=weights[1],
                activation=activation
            )
            cnn.add_layer(dense_layer)
    return cnn

def save_model_weights(model: keras.Model, filepath: str):
    model.save(filepath)
    print(f"Model saved to {filepath}")

def calculate_macro_f1_score(y_true: np.ndarray, y_pred: np.ndarray, num_classes:
int = 10) -> float:
    f1_scores = []
    for class_idx in range(num_classes):
        true_positives = np.sum((y_true == class_idx) & (y_pred == class_idx))
        false_positives = np.sum((y_true != class_idx) & (y_pred == class_idx))
        false_negatives = np.sum((y_true == class_idx) & (y_pred != class_idx))
        if true_positives + false_positives == 0:
            precision = 0
        else:
            precision = true_positives/(true_positives + false_positives)

        if true_positives + false_negatives == 0:
            recall = 0
        else:
            recall= true_positives / (true_positives + false_negatives)

        if precision + recall == 0:
            f1 = 0
        else:
            f1=2 * (precision * recall)/ (precision + recall)
        f1_scores.append(f1)
    return np.mean(f1_scores)

def test_implementation_consistency(keras_model_path: str, test_data:
Tuple[np.ndarray, np.ndarray],
```

```
                                num_samples: int = 100) -> Dict[str, float]:
    keras_model = keras.models.load_model(keras_model_path)
    scratch_model = build_cnn_from_keras(keras_model_path)
    x_test, y_test = test_data
    x_sample = x_test[:num_samples]
    y_sample = y_test[:num_samples]

    keras_pred = keras_model.predict(x_sample)
    keras_pred_classes = np.argmax(keras_pred, axis=1)
    scratch_pred_classes = scratch_model.predict(x_sample)
    keras_f1 = calculate_macro_f1_score(y_sample, keras_pred_classes)
    scratch_f1 = calculate_macro_f1_score(y_sample, scratch_pred_classes)

    implementation_accuracy = np.mean(keras_pred_classes == scratch_pred_classes)
    return {
        'keras_f1': keras_f1,
        'scratch_f1': scratch_f1,
        'implementation_accuracy':implementation_accuracy,
        'mean_absolute_error': np.mean(np.abs(keras_pred -
scratch_model.predict_proba(x_sample)))
    }
```

Kelas ConvLayer

Kelas ini mengimplementasikan operasi konvolusi pada input dengan filter yang diberikan, termasuk penambahan bias dan fungsi aktivasi.

Attributes:

- weights (np.ndarray): Filter/kernel konvolusi dengan bentuk (filter_height, filter_width, input_channels, num_filters).
- bias (np.ndarray): Bias untuk setiap filter, bentuk (num_filters,).
- strides (Tuple[int, int]): Langkah pergeseran filter (default: (1, 1)).
- padding (str): Jenis padding ('valid' atau 'same').
- activation (str): Fungsi aktivasi ('relu', 'linear', dll.).

Methods:

- apply_activation(x: np.ndarray): Mengaplikasikan fungsi aktivasi pada input x.
- forward(x: np.ndarray): Melakukan operasi konvolusi pada input x dan mengembalikan output setelah aktivasi.

Kelas PoolingLayer

Kelas ini merepresentasikan lapisan pooling dalam CNN, yang digunakan untuk mereduksi dimensi data dengan memilih nilai maksimum atau rata-rata dari wilayah tertentu.

Atribut:

- pool_size (Tuple[int, int]): Ukuran area pooling.
- strides (Tuple[int, int]): Langkah pergeseran pooling.
- pool_type (str): Jenis pooling yang digunakan, bisa 'max' atau 'average'.

Metode:

- forward(x: np.ndarray): Melakukan pooling pada input x dan mengembalikan hasilnya.

Kelas FlattenLayer

Kelas ini digunakan untuk meratakan input, mengubah data dari bentuk multi-dimensi menjadi satu dimensi, umumnya digunakan sebelum lapisan dens.

Metode:

- forward(x: np.ndarray) -> np.ndarray: Meratakan input x menjadi satu dimensi per contoh dalam batch.

Kelas GlobalPoolingLayer

Kelas ini menerapkan global pooling pada input, yang merangkum informasi dari seluruh gambar atau fitur.

Atribut:

- pool_type (str): Jenis pooling yang digunakan, bisa 'max' atau 'average'.

Metode:

- forward(x: np.ndarray): Melakukan global pooling pada input x dan mengembalikan hasilnya.

Kelas DenseLayer

Kelas ini merepresentasikan lapisan dense (*fully connected layer*) dalam CNN yang menghubungkan setiap neuron pada lapisan sebelumnya dengan setiap neuron pada lapisan ini.

Atribut:

- weights (np.ndarray): Matriks bobot untuk lapisan dens.
- bias (np.ndarray): Vektor bias untuk lapisan dens.
- activation (str): Fungsi aktivasi yang diterapkan pada output, bisa 'relu', 'softmax', atau 'linear'.

Metode:

- forward(x: np.ndarray): Menghitung output dari lapisan dens menggunakan operasi perkalian matriks, penambahan bias, dan fungsi aktivasi.

Kelas CNNFromScratch

Kelas ini merepresentasikan sebuah CNN yang dibangun dari awal, dengan memungkinkan pengguna untuk menambahkan berbagai jenis lapisan dan melakukan inferensi.

Atribut:

- layers (List): Daftar lapisan yang ada dalam model CNN.

Metode:

- add_layer(layer): Menambahkan lapisan ke dalam jaringan.
- forward(x: np.ndarray): Melakukan propagasi maju pada input x melalui seluruh lapisan yang ada.
- predict(x: np.ndarray): Melakukan prediksi kelas berdasarkan output dari jaringan.
- predict_proba(x: np.ndarray): Menghasilkan probabilitas prediksi dari jaringan.

Fungsi load_keras_model_weights
Fungsi ini digunakan untuk memuat bobot dari model Keras yang telah dilatih sebelumnya dan mengkonversinya ke dalam format yang dapat digunakan oleh kelas CNNFromScratch.

Fungsi build_cnn_from_keras
Fungsi ini membangun model CNN dari file model Keras yang telah dilatih, dengan menambahkan lapisan-lapisan dari model Keras ke dalam objek CNNFromScratch.

Fungsi save_model_weights
Fungsi ini digunakan untuk menyimpan model Keras ke dalam file untuk penggunaan di masa depan.

Fungsi calculate_macro_f1_score
Fungsi ini menghitung skor F1 makro, yang merupakan rata-rata skor F1 dari setiap kelas dalam masalah klasifikasi.

Fungsi test_implementation_consistency
Fungsi ini digunakan untuk menguji konsistensi implementasi antara model Keras dan model scratch yang dibangun dengan kode ini, dengan membandingkan hasil prediksi dan skor evaluasi seperti F1.

File rnn.py

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.layers import TextVectorization
from tensorflow.keras.models import load_model
from sklearn.metrics import f1_score
from datasets import load_dataset

class RNNLayer:
   def __init__(self, units, return_sequences=False, bidirectional=False):
```

```
        self.units = units
        self.return_sequences = return_sequences
        self.bidirectional = bidirectional
        self.initialized = False

    def initialize(self, input_shape):
        input_dim = input_shape[-1]
        limit = np.sqrt(6 / (input_dim + self.units))

        self.W_xh = np.random.uniform(-limit, limit, (input_dim, self.units))
        self.W_hh = np.random.uniform(-limit, limit, (self.units, self.units))
        self.b_h = np.zeros((self.units,))

        if self.bidirectional:
            self.W_xh_back = np.random.uniform(-limit, limit, (input_dim,
self.units))
            self.W_hh_back = np.random.uniform(-limit, limit, (self.units,
self.units))
            self.b_h_back = np.zeros((self.units,))

        self.initialized = True

    def forward_step(self, x, h_prev, W_xh, W_hh, b_h):
        h = np.tanh(x @ W_xh + h_prev @ W_hh + b_h)
        return h

    def forward(self, x):
        if not self.initialized:
            self.initialize(x.shape)

        batch_size, seq_len, input_dim = x.shape
        h_forward = np.zeros((batch_size, self.units))

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

        for t in range(seq_len):
            h_forward = self.forward_step(x[:, t, :], h_forward, self.W_xh,
self.W_hh, self.b_h)
```

```
            if self.return_sequences:
                outputs[:, t, :] = h_forward
        if not self.return_sequences:
            outputs = h_forward
        if self.bidirectional:
            h_backward = np.zeros((batch_size, self.units))
            if self.return_sequences:
                outputs_back = np.zeros((batch_size, seq_len, self.units))
            else:
                outputs_back = np.zeros((batch_size, self.units))

            for t in range(seq_len-1, -1, -1):
                h_backward = self.forward_step(x[:, t, :], h_backward,
self.W_xh_back, self.W_hh_back, self.b_h_back)
                if self.return_sequences:
                    outputs_back[:, t, :] = h_backward

            if not self.return_sequences:
                outputs_back = h_backward

            if self.return_sequences:
                outputs = np.concatenate((outputs, outputs_back), axis=-1)
            else:
                outputs = np.concatenate((outputs, outputs_back), axis=-1)

        return outputs

class EmbeddingLayer:
    def __init__(self, embedding_matrix):
        self.embedding_matrix = embedding_matrix

    def forward(self, x):
        return self.embedding_matrix[x]

class DropoutLayer:
    def __init__(self, rate):
        self.rate = rate
        self.mask = None

    def forward(self, x, training=True):
        if training:
```

```
            self.mask = (np.random.rand(*x.shape) > self.rate) / (1 - self.rate)
            return x * self.mask
        return x

class DenseLayer:
    def __init__(self, units, activation=None):
        self.units = units
        self.activation = activation
        self.initialized = False

    def initialize(self, input_shape):
        input_dim = input_shape[-1]
        limit = np.sqrt(6 / (input_dim + self.units))
        self.W = np.random.uniform(-limit, limit, (input_dim, self.units))
        self.b = np.zeros((self.units,))
        self.initialized = True

    def forward(self, x):
        if not self.initialized:
            self.initialize(x.shape)

        z = x @ self.W + self.b

        if self.activation == 'relu':
            return np.maximum(0, z)
        elif self.activation == 'softmax':
            exp_z = np.exp(z - np.max(z, axis=-1, keepdims=True))
            return exp_z / np.sum(exp_z, axis=-1, keepdims=True)
        else:
            return z

class RNNModelFromScratch:
    def __init__(self, keras_model):
        self.layers = []
        self.build_from_keras_model(keras_model)

    def build_from_keras_model(self, keras_model):
        for layer in keras_model.layers:
            if isinstance(layer, tf.keras.layers.Embedding):
                embedding_matrix = layer.get_weights()[0]
                self.layers.append(EmbeddingLayer(embedding_matrix))
```

```
            elif isinstance(layer, (tf.keras.layers.SimpleRNN, tf.keras.layers.LSTM,
tf.keras.layers.GRU)):
                units = layer.units
                return_sequences = layer.return_sequences
                bidirectional = isinstance(layer, tf.keras.layers.Bidirectional)

                rnn_layer = RNNLayer(units, return_sequences, bidirectional)
                if bidirectional:
                    forward_layer = layer.forward_layer
                    backward_layer = layer.backward_layer

                    kernel, recurrent_kernel, bias = forward_layer.get_weights()
                    rnn_layer.W_xh = kernel
                    rnn_layer.W_hh = recurrent_kernel
                    rnn_layer.b_h = bias

                    kernel, recurrent_kernel, bias = backward_layer.get_weights()
                    rnn_layer.W_xh_back = kernel
                    rnn_layer.W_hh_back = recurrent_kernel
                    rnn_layer.b_h_back = bias
                else:
                    kernel, recurrent_kernel, bias = layer.get_weights()
                    rnn_layer.W_xh = kernel
                    rnn_layer.W_hh = recurrent_kernel
                    rnn_layer.b_h = bias

                rnn_layer.initialized = True
                self.layers.append(rnn_layer)

            elif isinstance(layer, tf.keras.layers.Dropout):
                self.layers.append(DropoutLayer(layer.rate))

            elif isinstance(layer, tf.keras.layers.Dense):
                dense_layer = DenseLayer(layer.units,
activation=layer.activation.__name__ if layer.activation else None)

                kernel, bias = layer.get_weights()
                dense_layer.W = kernel
                dense_layer.b = bias
                dense_layer.initialized = True
```

```
                self.layers.append(dense_layer)

    def forward(self, x, training=False):
        for layer in self.layers:
            if isinstance(layer, DropoutLayer):
                x = layer.forward(x, training)
            else:
                x = layer.forward(x)
        return x

    def predict(self, x):
        logits = self.forward(x, training=False)
        if logits.shape[-1] > 1:
            return np.argmax(logits, axis=-1)
        else:
            return (logits > 0).astype(int)

def load_and_prepare_data():
    dataset = load_dataset("indonlp/NusaX-sentiment", "ind")


    texts = [example['text'] for example in dataset['train']]
    labels = [example['label'] for example in dataset['train']]


    train_texts, test_texts, train_labels, test_labels = train_test_split(
        texts, labels, test_size=0.2, random_state=42)

    return (train_texts, train_labels), (test_texts, test_labels)

def train_keras_model(train_texts, train_labels, test_texts, test_labels,
rnn_layers_config, bidirectional=False, dropout_rate=0.2):
    vectorizer = TextVectorization(max_tokens=10000, output_sequence_length=64)
    vectorizer.adapt(train_texts)

    model = tf.keras.Sequential()
    model.add(tf.keras.Input(shape=(1,), dtype=tf.string))
    model.add(vectorizer)
    model.add(tf.keras.layers.Embedding(input_dim=10000, output_dim=128))
```

```
    for i, units in enumerate(rnn_layers_config):
        return_sequences = i < len(rnn_layers_config) - 1
        rnn_layer = tf.keras.layers.SimpleRNN(units,
return_sequences=return_sequences)
        if bidirectional:
            rnn_layer = tf.keras.layers.Bidirectional(rnn_layer)
        model.add(rnn_layer)
        model.add(tf.keras.layers.Dropout(dropout_rate))

    model.add(tf.keras.layers.Dense(3,activation='softmax'))

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),loss='sparse_c
ategorical_crossentropy', metrics=['accuracy'])

    history = model.fit(np.array(train_texts),
np.array(train_labels),validation_data=(np.array(test_texts),
np.array(test_labels)),epochs=15, batch_size=64)

    return model, history

def evaluate_model(model, texts, labels):
    predictions = model.predict(np.array(texts))
    predictions = np.argmax(predictions, axis=1)
    return f1_score(labels, predictions, average='macro')

def compare_implementations(keras_model, scratch_model, test_texts, test_labels):
    vectorizer = keras_model.layers[1]
    test_tokens = vectorizer(np.array(test_texts)).numpy()
    keras_probs = keras_model.predict(np.array(test_texts))
    keras_preds = np.argmax(keras_probs, axis=1)
    keras_f1 = f1_score(test_labels, keras_preds, average='macro')

    scratch_probs = scratch_model.forward(test_tokens, training=False)
    scratch_preds = np.argmax(scratch_probs, axis=1)
    scratch_f1 = f1_score(test_labels, scratch_preds, average='macro')

    print(f"Keras Model F1-score: {keras_f1:.4f}")
    print(f"Scratch Model F1-score: {scratch_f1:.4f}")
    print(f"Difference: {abs(keras_f1 - scratch_f1):.4f}")
    return keras_f1, scratch_f1
```

Kelas RNNLayer
Kelas ini mengimplementasikan lapisan Recurrent Neural Network (RNN) yang dapat dipilih untuk bekerja dengan urutan data baik secara unidirectional maupun bidirectional. Lapisan ini mengimplementasikan proses pemrosesan urutan dengan fungsi aktivasi tanh pada setiap langkah waktu (time step).

Atribut:
- units (int): Jumlah unit pada lapisan RNN.
- return_sequences (bool): Menentukan apakah lapisan mengembalikan seluruh urutan atau hanya output dari langkah waktu terakhir.
- bidirectional (bool): Menentukan apakah lapisan RNN akan bekerja secara bidirectional.
- initialized (bool): Menunjukkan apakah lapisan sudah diinisialisasi dengan bobot.

Metode:
- initialize(input_shape: Tuple[int, int]): Menginisialisasi bobot lapisan berdasarkan bentuk input.
- forward_step(x: np.ndarray, h_prev: np.ndarray, W_xh: np.ndarray, W_hh: np.ndarray, b_h: np.ndarray): Melakukan pemrosesan pada setiap langkah waktu (time step).
- forward(x: np.ndarray): Melakukan propagasi maju pada input x melalui lapisan RNN.

Kelas EmbeddingLayer
Kelas ini mengimplementasikan lapisan embedding, yang mengonversi indeks kata dalam teks menjadi vektor embedding yang sesuai.

Atribut:
- embedding_matrix (np.ndarray): Matriks embedding yang memetakan kata ke vektor embedding.

Metode:
- forward(x: np.ndarray): Mengambil input berupa indeks kata dan mengembalikan vektor embedding yang sesuai.

Kelas DropoutLayer
Kelas ini mengimplementasikan lapisan dropout yang digunakan untuk regularisasi selama pelatihan, dengan secara acak menonaktifkan beberapa unit selama pelatihan.

Atribut:
- rate (float): Tingkat dropout.
- mask (np.ndarray): Masking yang digunakan untuk menonaktifkan unit pada saat pelatihan.

Metode:
- forward(x: np.ndarray, training: bool): Menerapkan dropout pada input jika dalam mode pelatihan.

Kelas DenseLayer

Kelas ini mengimplementasikan lapisan fully connected (dense) yang menghubungkan setiap unit dari lapisan sebelumnya ke setiap unit dalam lapisan ini.

Atribut:

- units (int): Jumlah unit dalam lapisan dense.
- activation (str): Fungsi aktivasi yang diterapkan pada output.
- initialized (bool): Menunjukkan apakah lapisan sudah diinisialisasi dengan bobot.

Metode:

- initialize(input_shape: Tuple[int, int]): Menginisialisasi bobot lapisan berdasarkan bentuk input.
- forward(x: np.ndarray): Melakukan propagasi maju pada input dan menerapkan fungsi aktivasi.

Kelas RNNModelFromScratch
Kelas ini mengimplementasikan model RNN dari awal, menggunakan lapisan-lapisan yang telah diimplementasikan seperti RNNLayer, EmbeddingLayer, DropoutLayer, dan DenseLayer.

Atribut:

- layers (List): Daftar lapisan dalam model RNN.

Metode:

- build_from_keras_model(keras_model: tf.keras.Model): Membangun model RNN dari model Keras yang telah dilatih.
- forward(x: np.ndarray, training: bool): Melakukan propagasi maju pada input x melalui seluruh lapisan.
- predict(x: np.ndarray): Melakukan prediksi berdasarkan output dari jaringan.

Fungsi load_and_prepare_data
Fungsi ini digunakan untuk memuat dan mempersiapkan dataset NusaX-sentiment yang digunakan untuk pelatihan dan pengujian model.

Fungsi train_keras_model
Fungsi ini melatih model Keras menggunakan dataset yang telah dipersiapkan, dengan konfigurasi lapisan RNN, apakah menggunakan bidirectional, serta tingkat dropout.

Fungsi evaluate_model
Fungsi ini mengevaluasi model menggunakan skor F1, yang digunakan untuk mengukur kinerja model dalam masalah klasifikasi.

Fungsi compare_implementations
Fungsi ini membandingkan dua implementasi model: satu menggunakan Keras dan satu lagi menggunakan model RNN dari awal (scratch).

File lstm.py

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.metrics import f1_score
import pickle
import json

class LSTMFromScratch:
    def __init__(self):
        self.weights = {}
        self.biases = {}
        self.embedding_weights = None
        self.vocab_size = None
        self.embedding_dim = None
        self.sequence_length = None

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

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

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

    def load_keras_model(self, model_path):
        self.keras_model = keras.models.load_model(model_path)

        for i, layer in enumerate(self.keras_model.layers):
            layer_name = f"layer_{i}_{layer.name}"

            if isinstance(layer, layers.Embedding):
                self.embedding_weights = layer.get_weights()[0]
                self.vocab_size, self.embedding_dim = self.embedding_weights.shape

            elif isinstance(layer, (layers.LSTM, layers.Bidirectional)):
                if isinstance(layer, layers.Bidirectional):
```

```
                    forward_weights = layer.forward_layer.get_weights()
                    backward_weights = layer.backward_layer.get_weights()
                    self.weights[f"{layer_name}_forward"] = forward_weights
                    self.weights[f"{layer_name}_backward"] = backward_weights
                else:

                    self.weights[layer_name] = layer.get_weights()

            elif isinstance(layer, layers.Dense):
                w, b = layer.get_weights()
                self.weights[f"{layer_name}_w"] = w
                self.weights[f"{layer_name}_b"] = b

    def embedding_forward(self, input_ids):
        embedded = self.embedding_weights[input_ids]
        return embedded

    def lstm_cell_forward(self, x_t, h_prev, c_prev, weights):
        W_i, W_f, W_c, W_o = weights[0][:, :self.embedding_dim], weights[0][:,
self.embedding_dim:2*self.embedding_dim], \
                             weights[0][:,
2*self.embedding_dim:3*self.embedding_dim], weights[0][:, 3*self.embedding_dim:]
        U_i, U_f, U_c, U_o = weights[1][:, :weights[1].shape[1]//4], weights[1][:,
weights[1].shape[1]//4:weights[1].shape[1]//2], \
                             weights[1][:,
weights[1].shape[1]//2:3*weights[1].shape[1]//4], weights[1][:,
3*weights[1].shape[1]//4:]
        b_i, b_f, b_c, b_o = weights[2][:weights[2].shape[0]//4],
weights[2][weights[2].shape[0]//4:weights[2].shape[0]//2], \

weights[2][weights[2].shape[0]//2:3*weights[2].shape[0]//4],
weights[2][3*weights[2].shape[0]//4:]

        i_t = self.sigmoid(np.dot(x_t, W_i.T) + np.dot(h_prev, U_i.T) + b_i)
        f_t = self.sigmoid(np.dot(x_t, W_f.T) + np.dot(h_prev, U_f.T) + b_f)
        c_tilde = self.tanh(np.dot(x_t, W_c.T) + np.dot(h_prev, U_c.T) + b_c)
        o_t = self.sigmoid(np.dot(x_t, W_o.T) + np.dot(h_prev, U_o.T) + b_o)

        c_t = f_t * c_prev + i_t * c_tilde
        h_t = o_t * self.tanh(c_t)
        return h_t, c_t
```

```
    def lstm_forward(self, x, weights, return_sequences=False):
        batch_size, seq_len, input_dim = x.shape
        hidden_dim = weights[1].shape[0]

        h = np.zeros((batch_size, hidden_dim))
        c = np.zeros((batch_size, hidden_dim))
        outputs = []

        for t in range(seq_len):
            h, c = self.lstm_cell_forward(x[:, t, :], h, c, weights)
            outputs.append(h.copy())

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

    def bidirectional_lstm_forward(self, x, forward_weights, backward_weights,
return_sequences=False):
        forward_output = self.lstm_forward(x, forward_weights, return_sequences=True)
        x_reversed = x[:, ::-1, :]
        backward_output = self.lstm_forward(x_reversed, backward_weights,
return_sequences=True)
        backward_output = backward_output[:, ::-1, :]

        combined_output = np.concatenate([forward_output, backward_output], axis=-1)

        if return_sequences:
            return combined_output
        else:
            return combined_output[:, -1, :]

    def dense_forward(self, x, weights_key):
        w = self.weights[f"{weights_key}_w"]
        b = self.weights[f"{weights_key}_b"]
        return np.dot(x, w) + b

    def forward(self, input_ids):
        x = self.embedding_forward(input_ids)
```

```
        for i, layer in enumerate(self.keras_model.layers[1:], 1):
            layer_name = f"layer_{i}_{layer.name}"

            if isinstance(layer, layers.Bidirectional):
                x = self.bidirectional_lstm_forward(
                    x,
                    self.weights[f"{layer_name}_forward"],
                    self.weights[f"{layer_name}_backward"],
                    return_sequences=False
                )
            elif isinstance(layer, layers.LSTM):
                x = self.lstm_forward(x, self.weights[layer_name],
return_sequences=False)
            elif isinstance(layer, layers.Dropout):

                continue
            elif isinstance(layer, layers.Dense):
                x = self.dense_forward(x, layer_name)
                if layer == self.keras_model.layers[-1]:
                    x = self.softmax(x)

        return x

def create_lstm_model(embedding_dim=100, lstm_units=64, num_lstm_layers=1,
                      bidirectional=False, vocab_size=10000, num_classes=3):
    model = keras.Sequential()
    model.add(layers.Embedding(vocab_size, embedding_dim, mask_zero=True))
    for i in range(num_lstm_layers):
        return_sequences = (i < num_lstm_layers - 1)
        if bidirectional:
            model.add(layers.Bidirectional(
                layers.LSTM(lstm_units, return_sequences=return_sequences,
dropout=0.3)
            ))
        else:
            model.add(layers.LSTM(lstm_units, return_sequences=return_sequences,
dropout=0.3))
        model.add(layers.Dropout(0.5))
    model.add(layers.Dense(num_classes, activation='softmax'))
    return model
```

```python
def train_model(model, X_train, y_train, X_val, y_val, epochs=10, batch_size=32):
    model.compile(
        optimizer='adam',
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy']
    )

    history = model.fit(
        X_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        validation_data=(X_val, y_val),
        verbose=1
    )
    return history

def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    y_pred_classes = np.argmax(y_pred, axis=1)

    f1 = f1_score(y_test, y_pred_classes, average='macro')
    return f1, y_pred_classes

def compare_implementations(keras_model, lstm_scratch, X_test, y_test):
    keras_pred = keras_model.predict(X_test)
    keras_pred_classes = np.argmax(keras_pred, axis=1)
    keras_f1 = f1_score(y_test, keras_pred_classes, average='macro')

    scratch_pred = lstm_scratch.forward(X_test)
    scratch_pred_classes = np.argmax(scratch_pred, axis=1)
    scratch_f1 = f1_score(y_test, scratch_pred_classes, average='macro')

    print(f"Keras F1 Score: {keras_f1:.4f}")
    print(f"From-scratch F1 Score: {scratch_f1:.4f}")
    print(f"Difference: {abs(keras_f1 - scratch_f1):.4f}")

    return keras_f1, scratch_f1
```

Kelas LSTMFromScratch

Kelas ini mengimplementasikan LSTM dari awal, termasuk pemrosesan input, *forward propagation*, dan pengambilan prediksi. Kelas ini digunakan untuk memodelkan LSTM tanpa menggunakan *library* Keras atau TensorFlow.

Attributes:

- weights (dict): Menyimpan bobot untuk setiap lapisan (LSTM, Bidirectional, Dense) dalam bentuk dictionary.
- biases (dict): Menyimpan bias untuk setiap lapisan dalam bentuk dictionary.
- embedding_weights (np.ndarray): Bobot untuk lapisan embedding yang berisi representasi vektor kata.
- vocab_size (int): Ukuran kosakata (jumlah kata unik yang digunakan dalam embedding).
- embedding_dim (int): Dimensi vektor embedding.
- sequence_length (int): Panjang urutan input.

Methods:

- sigmoid(x: np.ndarray): Mengaplikasikan fungsi aktivasi sigmoid pada input x.
- tanh(x: np.ndarray): Mengaplikasikan fungsi aktivasi tanh pada input x.
- softmax(x: np.ndarray): Mengaplikasikan fungsi softmax untuk mendapatkan probabilitas kelas pada input x.
- load_keras_model(model_path: str): Memuat model Keras yang telah dilatih dan mengekstrak bobot dari model tersebut untuk digunakan dalam implementasi scratch.
- embedding_forward(input_ids: np.ndarray): Mengonversi input id (indeks kata) menjadi representasi embedding.
- lstm_cell_forward(x_t: np.ndarray, h_prev: np.ndarray, c_prev: np.ndarray, weights: List[np.ndarray]):Melakukan operasi LSTM pada satu langkah waktu.
- lstm_forward(x: np.ndarray, weights: List[np.ndarray], return_sequences: bool=False): Menjalankan seluruh urutan input melalui LSTM untuk menghasilkan output akhir.
- bidirectional_lstm_forward(x: np.ndarray, forward_weights: List[np.ndarray], backward_weights: List[np.ndarray], return_sequences: bool=False): Melakukan propagasi maju melalui LSTM dua arah (Bidirectional LSTM).
- dense_forward(x: np.ndarray, weights_key: str): Melakukan operasi lapisan Dense pada input x.
- forward(input_ids: np.ndarray): Melakukan propagasi maju seluruh jaringan LSTM, termasuk embedding, LSTM, dan Dense, hingga menghasilkan output prediksi.

Fungsi create_lstm_model
Fungsi untuk membangun model LSTM dengan opsi untuk menambahkan LSTM dua arah dan pengaturan lainnya.

Fungsi train_model
Fungsi untuk melatih model dengan data pelatihan dan validasi.

Fungsi evaluate_model
Fungsi untuk mengevaluasi model menggunakan F1-score pada data uji.

Fungsi compare_implementations
Fungsi untuk membandingkan skor F1 dari model Keras dengan model LSTM dari awal (scratch), serta menghitung perbedaan skor.

Hasil Pengujian

CNN

Pengaruh jumlah layer konvolusi

```
Testing: Jumlah Layer Konvolusi

Training dengan 2 layers...
Epoch 1/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m18s[0m 14ms/step - accuracy: 0.4061 - loss: 1.6474 - val_accuracy: 0.6062 - val_loss:
1.1344
Epoch 2/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m16s[0m 13ms/step - accuracy: 0.6205 - loss: 1.0756 - val_accuracy: 0.6573 - val_loss:
0.9815
Epoch 3/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m16s[0m 13ms/step - accuracy: 0.6874 - loss: 0.8981 - val_accuracy: 0.6648 - val_loss:
0.9545
Epoch 4/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m16s[0m 13ms/step - accuracy: 0.7278 - loss: 0.7844 - val_accuracy: 0.6925 - val_loss:
0.8831
Epoch 5/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m15s[0m 12ms/step - accuracy: 0.7603 - loss: 0.6790 - val_accuracy: 0.6880 - val_loss:
0.9089
[1m313/313[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
4ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.6816

Training dengan 3 layers...
Epoch 1/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m23s[0m 17ms/step - accuracy: 0.3780 - loss: 1.6911 - val_accuracy: 0.5875 - val_loss:
1.1582
Epoch 2/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m20s[0m 16ms/step - accuracy: 0.6318 - loss: 1.0501 - val_accuracy: 0.6805 - val_loss:
0.9176
Epoch 3/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m20s[0m 16ms/step - accuracy: 0.7024 - loss: 0.8486 - val_accuracy: 0.7094 - val_loss:
0.8269
```

```
Epoch 4/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m20s[0m 16ms/step - accuracy: 0.7516 - loss: 0.7136 - val_accuracy: 0.7209 - val_loss:
0.8161
Epoch 5/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m18s[0m 15ms/step - accuracy: 0.7884 - loss: 0.6073 - val_accuracy: 0.7292 - val_loss:
0.7737
[1m313/313[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
4ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.7238

Training dengan 4 layers...
Epoch 1/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m25s[0m 19ms/step - accuracy: 0.3370 - loss: 1.7837 - val_accuracy: 0.5587 - val_loss:
1.2073
Epoch 2/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m24s[0m 19ms/step - accuracy: 0.6020 - loss: 1.1134 - val_accuracy: 0.6351 - val_loss:
1.0284
Epoch 3/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m23s[0m 18ms/step - accuracy: 0.6858 - loss: 0.8833 - val_accuracy: 0.6943 - val_loss:
0.8684
Epoch 4/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m23s[0m 19ms/step - accuracy: 0.7379 - loss: 0.7458 - val_accuracy: 0.7253 - val_loss:
0.8052
Epoch 5/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m21s[0m 17ms/step - accuracy: 0.7727 - loss: 0.6442 - val_accuracy: 0.7137 - val_loss:
0.8579
[1m313/313[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m2s[0m
5ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.6924
```

Pengaruh banyak filter per layer konvolusi

```
Testing: Jumlah Filter per Layer
```

```
Training dengan filters: [16, 32, 64]...
Epoch 1/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m10s[0m 8ms/step - accuracy: 0.3694 - loss: 1.7104 - val_accuracy: 0.5921 - val_loss:
1.1594
Epoch 2/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m9s[0m
8ms/step - accuracy: 0.6106 - loss: 1.1125 - val_accuracy: 0.6258 - val_loss: 1.0670
Epoch 3/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m9s[0m
7ms/step - accuracy: 0.6725 - loss: 0.9349 - val_accuracy: 0.6624 - val_loss: 0.9585
Epoch 4/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m9s[0m
7ms/step - accuracy: 0.7165 - loss: 0.8156 - val_accuracy: 0.6916 - val_loss: 0.8675
Epoch 5/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m9s[0m
7ms/step - accuracy: 0.7392 - loss: 0.7420 - val_accuracy: 0.6984 - val_loss: 0.8656
[1m313/313[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
2ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.6919

Training dengan filters: [32, 64, 128]...
Epoch 1/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m19s[0m 15ms/step - accuracy: 0.3754 - loss: 1.6992 - val_accuracy: 0.6069 - val_loss:
1.1043
Epoch 2/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m19s[0m 15ms/step - accuracy: 0.6352 - loss: 1.0300 - val_accuracy: 0.6778 - val_loss:
0.9220
Epoch 3/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m19s[0m 15ms/step - accuracy: 0.7097 - loss: 0.8253 - val_accuracy: 0.6937 - val_loss:
0.8828
Epoch 4/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m19s[0m 15ms/step - accuracy: 0.7577 - loss: 0.6970 - val_accuracy: 0.7331 - val_loss:
0.7724
Epoch 5/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m19s[0m 15ms/step - accuracy: 0.7911 - loss: 0.6027 - val_accuracy: 0.7495 - val_loss:
0.7556
[1m313/313[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
4ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
```

```
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.7388

Training dengan filters: [64, 128, 256]...
Epoch 1/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m50s[0m 40ms/step - accuracy: 0.3551 - loss: 1.7504 - val_accuracy: 0.5771 - val_loss: 1.1778
Epoch 2/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m50s[0m 40ms/step - accuracy: 0.6274 - loss: 1.0616 - val_accuracy: 0.6781 - val_loss: 0.9203
Epoch 3/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m54s[0m 43ms/step - accuracy: 0.7067 - loss: 0.8343 - val_accuracy: 0.7040 - val_loss: 0.8408
Epoch 4/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m53s[0m 42ms/step - accuracy: 0.7566 - loss: 0.6985 - val_accuracy: 0.7238 - val_loss: 0.8009
Epoch 5/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m52s[0m 42ms/step - accuracy: 0.7970 - loss: 0.5825 - val_accuracy: 0.7380 - val_loss: 0.7913
[1m313/313[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m4s[0m 12ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.7323
```

Pengaruh ukuran filter per layer konvolusi

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.7123

Training dengan sizes: [(5, 5), (3, 3), (3, 3)]...
Epoch 1/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m29s[0m 22ms/step - accuracy: 0.3631 - loss: 1.7384 - val_accuracy: 0.5603 - val_loss: 1.2249
Epoch 2/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
```

```
[1m22s[0m 18ms/step - accuracy: 0.5942 - loss: 1.1370 - val_accuracy: 0.6388 - val_loss:
1.0087
Epoch 3/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m21s[0m 17ms/step - accuracy: 0.6770 - loss: 0.9167 - val_accuracy: 0.6571 - val_loss:
0.9999
Epoch 4/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m22s[0m 17ms/step - accuracy: 0.7309 - loss: 0.7642 - val_accuracy: 0.6972 - val_loss:
0.8810
Epoch 5/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m22s[0m 17ms/step - accuracy: 0.7676 - loss: 0.6638 - val_accuracy: 0.6997 - val_loss:
0.8746
[1m313/313[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m2s[0m
5ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.6921

Training dengan sizes: [(7, 7), (5, 5), (3, 3)]...
Epoch 1/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m36s[0m 28ms/step - accuracy: 0.3363 - loss: 1.7978 - val_accuracy: 0.5164 - val_loss:
1.3538
Epoch 2/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m36s[0m 28ms/step - accuracy: 0.5505 - loss: 1.2600 - val_accuracy: 0.6079 - val_loss:
1.1229
Epoch 3/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m36s[0m 28ms/step - accuracy: 0.6379 - loss: 1.0280 - val_accuracy: 0.6389 - val_loss:
1.0438
Epoch 4/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m36s[0m 29ms/step - accuracy: 0.6894 - loss: 0.8802 - val_accuracy: 0.6519 - val_loss:
1.0141
Epoch 5/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m36s[0m 29ms/step - accuracy: 0.7417 - loss: 0.7424 - val_accuracy: 0.6681 - val_loss:
1.0014
[1m313/313[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m3s[0m
9ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.6636
```

Pengaruh jenis pooling layer

```
Testing: Jenis Pooling

Training dengan max pooling...
Epoch 1/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m22s[0m 17ms/step - accuracy: 0.3754 - loss: 1.6990 - val_accuracy: 0.5854 - val_loss:
1.1719
Epoch 2/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m21s[0m 17ms/step - accuracy: 0.6194 - loss: 1.0827 - val_accuracy: 0.6608 - val_loss:
0.9471
Epoch 3/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m22s[0m 18ms/step - accuracy: 0.6894 - loss: 0.8829 - val_accuracy: 0.6930 - val_loss:
0.8818
Epoch 4/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m21s[0m 17ms/step - accuracy: 0.7406 - loss: 0.7431 - val_accuracy: 0.6825 - val_loss:
0.9151
Epoch 5/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m21s[0m 17ms/step - accuracy: 0.7771 - loss: 0.6368 - val_accuracy: 0.7015 - val_loss:
0.8825
[1m313/313[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m2s[0m
5ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.6912

Training dengan average pooling...
Epoch 1/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m20s[0m 15ms/step - accuracy: 0.3538 - loss: 1.7605 - val_accuracy: 0.5540 - val_loss:
1.2502
Epoch 2/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m20s[0m 16ms/step - accuracy: 0.5691 - loss: 1.2043 - val_accuracy: 0.6299 - val_loss:
1.0509
Epoch 3/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m19s[0m 15ms/step - accuracy: 0.6504 - loss: 1.0009 - val_accuracy: 0.6743 - val_loss:
0.9298
Epoch 4/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m20s[0m 16ms/step - accuracy: 0.6899 - loss: 0.8786 - val_accuracy: 0.6844 - val_loss:
```

```
0.8891
Epoch 5/5
[1m1250/1250[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m
[1m19s[0m 16ms/step - accuracy: 0.7302 - loss: 0.7652 - val_accuracy: 0.7225 - val_loss:
0.8034
[1m313/313[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m2s[0m
5ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.7096
```

Simple RNN

Pengaruh jumlah layer RNN

```
Testing: Jumlah Layer RNN

Training RNN dengan 1 layers...
Epoch 1/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m6s[0m
63ms/step - accuracy: 0.4054 - loss: 1.0786 - val_accuracy: 0.4500 - val_loss: 1.0193
Epoch 2/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
32ms/step - accuracy: 0.5802 - loss: 0.8712 - val_accuracy: 0.5300 - val_loss: 0.9935
Epoch 3/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
33ms/step - accuracy: 0.8956 - loss: 0.6714 - val_accuracy: 0.6100 - val_loss: 0.9152
Epoch 4/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
31ms/step - accuracy: 0.9734 - loss: 0.4543 - val_accuracy: 0.5800 - val_loss: 0.8989
Epoch 5/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
30ms/step - accuracy: 0.9959 - loss: 0.2569 - val_accuracy: 0.6200 - val_loss: 0.8382
Epoch 6/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
31ms/step - accuracy: 1.0000 - loss: 0.1468 - val_accuracy: 0.5700 - val_loss: 0.9088
Epoch 7/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
31ms/step - accuracy: 0.9987 - loss: 0.0817 - val_accuracy: 0.5700 - val_loss: 0.9065
Epoch 8/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
45ms/step - accuracy: 1.0000 - loss: 0.0486 - val_accuracy: 0.5500 - val_loss: 0.9494
Epoch 9/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
29ms/step - accuracy: 1.0000 - loss: 0.0297 - val_accuracy: 0.5500 - val_loss: 1.0025
Epoch 10/15
```

```
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
30ms/step - accuracy: 1.0000 - loss: 0.0221 - val_accuracy: 0.5600 - val_loss: 1.0137
Epoch 11/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
29ms/step - accuracy: 1.0000 - loss: 0.0182 - val_accuracy: 0.5400 - val_loss: 1.0058
Epoch 12/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
30ms/step - accuracy: 1.0000 - loss: 0.0137 - val_accuracy: 0.5700 - val_loss: 1.0135
Epoch 13/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
29ms/step - accuracy: 1.0000 - loss: 0.0112 - val_accuracy: 0.5700 - val_loss: 1.0271
Epoch 14/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
31ms/step - accuracy: 1.0000 - loss: 0.0113 - val_accuracy: 0.5800 - val_loss: 1.0434
Epoch 15/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
30ms/step - accuracy: 1.0000 - loss: 0.0086 - val_accuracy: 0.5700 - val_loss: 1.0442
[1m4/4[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
35ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.5604

Training RNN dengan 2 layers...
Epoch 1/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m2s[0m
97ms/step - accuracy: 0.3689 - loss: 1.1399 - val_accuracy: 0.5300 - val_loss: 0.9881
Epoch 2/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
48ms/step - accuracy: 0.5305 - loss: 0.9118 - val_accuracy: 0.4900 - val_loss: 0.9266
Epoch 3/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
49ms/step - accuracy: 0.7108 - loss: 0.7475 - val_accuracy: 0.5800 - val_loss: 0.8619
Epoch 4/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
47ms/step - accuracy: 0.8283 - loss: 0.5577 - val_accuracy: 0.5700 - val_loss: 0.9163
Epoch 5/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
47ms/step - accuracy: 0.9384 - loss: 0.2956 - val_accuracy: 0.6100 - val_loss: 0.8529
Epoch 6/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
48ms/step - accuracy: 0.9727 - loss: 0.1787 - val_accuracy: 0.6000 - val_loss: 0.8823
Epoch 7/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
47ms/step - accuracy: 0.9911 - loss: 0.0999 - val_accuracy: 0.5900 - val_loss: 0.9376
Epoch 8/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
48ms/step - accuracy: 0.9946 - loss: 0.0546 - val_accuracy: 0.5700 - val_loss: 1.0386
```

```
Epoch 9/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
46ms/step - accuracy: 1.0000 - loss: 0.0346 - val_accuracy: 0.5800 - val_loss: 1.0615
Epoch 10/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
47ms/step - accuracy: 1.0000 - loss: 0.0234 - val_accuracy: 0.5800 - val_loss: 1.0881
Epoch 11/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
47ms/step - accuracy: 1.0000 - loss: 0.0156 - val_accuracy: 0.5800 - val_loss: 1.1434
Epoch 12/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
48ms/step - accuracy: 1.0000 - loss: 0.0121 - val_accuracy: 0.5900 - val_loss: 1.1853
Epoch 13/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
48ms/step - accuracy: 1.0000 - loss: 0.0121 - val_accuracy: 0.5900 - val_loss: 1.2017
Epoch 14/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
62ms/step - accuracy: 1.0000 - loss: 0.0102 - val_accuracy: 0.5900 - val_loss: 1.2139
Epoch 15/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
50ms/step - accuracy: 1.0000 - loss: 0.0071 - val_accuracy: 0.5900 - val_loss: 1.2348
[1m4/4[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
52ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.5768

Training RNN dengan 3 layers...
Epoch 1/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m2s[0m
107ms/step - accuracy: 0.3653 - loss: 1.2992 - val_accuracy: 0.4800 - val_loss: 1.0197
Epoch 2/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
67ms/step - accuracy: 0.4946 - loss: 1.1092 - val_accuracy: 0.5100 - val_loss: 0.9898
Epoch 3/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
67ms/step - accuracy: 0.6534 - loss: 0.7558 - val_accuracy: 0.5300 - val_loss: 0.9998
Epoch 4/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
92ms/step - accuracy: 0.7224 - loss: 0.6129 - val_accuracy: 0.5600 - val_loss: 1.0629
Epoch 5/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
66ms/step - accuracy: 0.8207 - loss: 0.4677 - val_accuracy: 0.5500 - val_loss: 1.1723
Epoch 6/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
68ms/step - accuracy: 0.8913 - loss: 0.3349 - val_accuracy: 0.5500 - val_loss: 1.1441
Epoch 7/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
```

```
66ms/step - accuracy: 0.9333 - loss: 0.2148 - val_accuracy: 0.5500 - val_loss: 1.2833
Epoch 8/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
65ms/step - accuracy: 0.9679 - loss: 0.1405 - val_accuracy: 0.5500 - val_loss: 1.4075
Epoch 9/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
65ms/step - accuracy: 0.9744 - loss: 0.0900 - val_accuracy: 0.5400 - val_loss: 1.4659
Epoch 10/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
73ms/step - accuracy: 0.9919 - loss: 0.0582 - val_accuracy: 0.5500 - val_loss: 1.5630
Epoch 11/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
66ms/step - accuracy: 1.0000 - loss: 0.0313 - val_accuracy: 0.5400 - val_loss: 1.6191
Epoch 12/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
66ms/step - accuracy: 0.9965 - loss: 0.0298 - val_accuracy: 0.5600 - val_loss: 1.6597
Epoch 13/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
72ms/step - accuracy: 1.0000 - loss: 0.0159 - val_accuracy: 0.5400 - val_loss: 1.7068
Epoch 14/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
66ms/step - accuracy: 1.0000 - loss: 0.0167 - val_accuracy: 0.5400 - val_loss: 1.7530
Epoch 15/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
75ms/step - accuracy: 1.0000 - loss: 0.0118 - val_accuracy: 0.5400 - val_loss: 1.7850
[1m4/4[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
74ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.4762
```

Pengaruh banyak cell RNN per layer

```
Testing: Jumlah Cell RNN

Training RNN dengan 32 cells...
Epoch 1/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m2s[0m
70ms/step - accuracy: 0.3476 - loss: 1.1578 - val_accuracy: 0.3000 - val_loss: 1.1572
Epoch 2/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
37ms/step - accuracy: 0.5005 - loss: 1.0076 - val_accuracy: 0.3500 - val_loss: 1.1276
Epoch 3/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
37ms/step - accuracy: 0.7038 - loss: 0.8264 - val_accuracy: 0.3400 - val_loss: 1.1229
Epoch 4/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
```

```
36ms/step - accuracy: 0.7901 - loss: 0.6817 - val_accuracy: 0.3700 - val_loss: 1.1296
Epoch 5/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
44ms/step - accuracy: 0.8829 - loss: 0.4913 - val_accuracy: 0.4000 - val_loss: 1.1567
Epoch 6/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
37ms/step - accuracy: 0.9616 - loss: 0.3351 - val_accuracy: 0.3900 - val_loss: 1.2459
Epoch 7/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
36ms/step - accuracy: 0.9855 - loss: 0.2139 - val_accuracy: 0.3900 - val_loss: 1.2788
Epoch 8/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
37ms/step - accuracy: 0.9957 - loss: 0.1486 - val_accuracy: 0.4100 - val_loss: 1.2902
Epoch 9/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
36ms/step - accuracy: 1.0000 - loss: 0.0985 - val_accuracy: 0.4200 - val_loss: 1.3255
Epoch 10/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
36ms/step - accuracy: 1.0000 - loss: 0.0778 - val_accuracy: 0.4300 - val_loss: 1.3801
Epoch 11/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
36ms/step - accuracy: 1.0000 - loss: 0.0556 - val_accuracy: 0.4100 - val_loss: 1.4375
Epoch 12/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
36ms/step - accuracy: 0.9946 - loss: 0.0555 - val_accuracy: 0.4100 - val_loss: 1.4820
Epoch 13/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
44ms/step - accuracy: 1.0000 - loss: 0.0384 - val_accuracy: 0.4000 - val_loss: 1.5038
Epoch 14/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
37ms/step - accuracy: 1.0000 - loss: 0.0309 - val_accuracy: 0.4000 - val_loss: 1.5511
Epoch 15/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
38ms/step - accuracy: 1.0000 - loss: 0.0305 - val_accuracy: 0.4000 - val_loss: 1.5717
[1m4/4[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
48ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.3927

Training RNN dengan 64 cells...
Epoch 1/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m2s[0m
77ms/step - accuracy: 0.3673 - loss: 1.1633 - val_accuracy: 0.4200 - val_loss: 1.0477
Epoch 2/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
48ms/step - accuracy: 0.5556 - loss: 0.9152 - val_accuracy: 0.5500 - val_loss: 0.9530
Epoch 3/15
```

```
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
46ms/step - accuracy: 0.7264 - loss: 0.6890 - val_accuracy: 0.6100 - val_loss: 0.8887
Epoch 4/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
53ms/step - accuracy: 0.8799 - loss: 0.4457 - val_accuracy: 0.5900 - val_loss: 0.9255
Epoch 5/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
46ms/step - accuracy: 0.9659 - loss: 0.2234 - val_accuracy: 0.6100 - val_loss: 0.8954
Epoch 6/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
46ms/step - accuracy: 0.9855 - loss: 0.1342 - val_accuracy: 0.5700 - val_loss: 1.0298
Epoch 7/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
47ms/step - accuracy: 0.9946 - loss: 0.0776 - val_accuracy: 0.5700 - val_loss: 1.1001
Epoch 8/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
46ms/step - accuracy: 0.9987 - loss: 0.0388 - val_accuracy: 0.5900 - val_loss: 1.1211
Epoch 9/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
47ms/step - accuracy: 0.9975 - loss: 0.0270 - val_accuracy: 0.6000 - val_loss: 1.2268
Epoch 10/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
46ms/step - accuracy: 0.9990 - loss: 0.0205 - val_accuracy: 0.5800 - val_loss: 1.2597
Epoch 11/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
53ms/step - accuracy: 1.0000 - loss: 0.0151 - val_accuracy: 0.5900 - val_loss: 1.2242
Epoch 12/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
47ms/step - accuracy: 1.0000 - loss: 0.0130 - val_accuracy: 0.6100 - val_loss: 1.2295
Epoch 13/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
46ms/step - accuracy: 1.0000 - loss: 0.0107 - val_accuracy: 0.6100 - val_loss: 1.2540
Epoch 14/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
47ms/step - accuracy: 1.0000 - loss: 0.0106 - val_accuracy: 0.6000 - val_loss: 1.2566
Epoch 15/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
48ms/step - accuracy: 1.0000 - loss: 0.0084 - val_accuracy: 0.5800 - val_loss: 1.2790
[1m4/4[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
49ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.5532

Training RNN dengan 128 cells...
Epoch 1/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m2s[0m
114ms/step - accuracy: 0.4186 - loss: 1.1094 - val_accuracy: 0.4200 - val_loss: 1.1436
```

```
Epoch 2/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
86ms/step - accuracy: 0.6083 - loss: 0.8093 - val_accuracy: 0.5200 - val_loss: 0.9840
Epoch 3/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
75ms/step - accuracy: 0.8680 - loss: 0.5141 - val_accuracy: 0.5400 - val_loss: 1.0935
Epoch 4/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
78ms/step - accuracy: 0.9607 - loss: 0.2465 - val_accuracy: 0.6000 - val_loss: 1.1287
Epoch 5/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
79ms/step - accuracy: 0.9933 - loss: 0.0884 - val_accuracy: 0.5900 - val_loss: 1.2290
Epoch 6/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
76ms/step - accuracy: 0.9916 - loss: 0.0568 - val_accuracy: 0.4900 - val_loss: 1.8991
Epoch 7/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
76ms/step - accuracy: 0.9920 - loss: 0.0380 - val_accuracy: 0.5200 - val_loss: 1.7137
Epoch 8/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
77ms/step - accuracy: 0.9968 - loss: 0.0247 - val_accuracy: 0.5700 - val_loss: 1.5357
Epoch 9/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
75ms/step - accuracy: 0.9800 - loss: 0.0605 - val_accuracy: 0.5000 - val_loss: 1.7597
Epoch 10/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
76ms/step - accuracy: 0.9904 - loss: 0.0511 - val_accuracy: 0.5100 - val_loss: 1.9116
Epoch 11/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
77ms/step - accuracy: 0.9782 - loss: 0.0542 - val_accuracy: 0.4900 - val_loss: 1.6128
Epoch 12/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
75ms/step - accuracy: 0.9947 - loss: 0.0215 - val_accuracy: 0.5400 - val_loss: 1.5784
Epoch 13/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
74ms/step - accuracy: 0.9950 - loss: 0.0268 - val_accuracy: 0.6100 - val_loss: 1.5162
Epoch 14/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
83ms/step - accuracy: 1.0000 - loss: 0.0104 - val_accuracy: 0.6000 - val_loss: 1.5058
Epoch 15/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
75ms/step - accuracy: 1.0000 - loss: 0.0060 - val_accuracy: 0.6000 - val_loss: 1.5796
[1m4/4[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
53ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.5450
```

Pengaruh jenis layer RNN berdasarkan arah

```
Testing: Arah RNN

Training unidirectional RNN...
Epoch 1/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m2s[0m
78ms/step - accuracy: 0.3646 - loss: 1.1516 - val_accuracy: 0.5100 - val_loss: 0.9965
Epoch 2/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
48ms/step - accuracy: 0.5693 - loss: 0.9418 - val_accuracy: 0.5600 - val_loss: 0.9322
Epoch 3/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
47ms/step - accuracy: 0.7117 - loss: 0.6908 - val_accuracy: 0.6300 - val_loss: 0.8311
Epoch 4/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
54ms/step - accuracy: 0.8593 - loss: 0.4585 - val_accuracy: 0.6500 - val_loss: 0.7992
Epoch 5/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
47ms/step - accuracy: 0.9686 - loss: 0.2735 - val_accuracy: 0.6400 - val_loss: 0.8339
Epoch 6/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
47ms/step - accuracy: 0.9929 - loss: 0.1368 - val_accuracy: 0.6800 - val_loss: 0.8345
Epoch 7/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
46ms/step - accuracy: 0.9852 - loss: 0.0833 - val_accuracy: 0.6600 - val_loss: 0.8675
Epoch 8/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
46ms/step - accuracy: 1.0000 - loss: 0.0456 - val_accuracy: 0.6200 - val_loss: 0.9537
Epoch 9/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
46ms/step - accuracy: 1.0000 - loss: 0.0276 - val_accuracy: 0.6200 - val_loss: 1.0291
Epoch 10/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
46ms/step - accuracy: 1.0000 - loss: 0.0244 - val_accuracy: 0.6400 - val_loss: 1.0133
Epoch 11/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
46ms/step - accuracy: 1.0000 - loss: 0.0171 - val_accuracy: 0.6400 - val_loss: 1.0273
Epoch 12/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
46ms/step - accuracy: 1.0000 - loss: 0.0142 - val_accuracy: 0.6400 - val_loss: 1.0578
Epoch 13/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
48ms/step - accuracy: 1.0000 - loss: 0.0107 - val_accuracy: 0.6500 - val_loss: 1.0892
Epoch 14/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
47ms/step - accuracy: 1.0000 - loss: 0.0073 - val_accuracy: 0.6300 - val_loss: 1.1215
Epoch 15/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
47ms/step - accuracy: 1.0000 - loss: 0.0065 - val_accuracy: 0.6200 - val_loss: 1.1444
[1m4/4[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
```

```
49ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.5942

Training bidirectional RNN...
Epoch 1/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m3s[0m
107ms/step - accuracy: 0.3851 - loss: 1.1335 - val_accuracy: 0.5000 - val_loss: 1.0237
Epoch 2/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
55ms/step - accuracy: 0.6074 - loss: 0.8373 - val_accuracy: 0.5500 - val_loss: 0.9622
Epoch 3/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
63ms/step - accuracy: 0.7951 - loss: 0.6076 - val_accuracy: 0.6100 - val_loss: 0.8847
Epoch 4/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
55ms/step - accuracy: 0.9380 - loss: 0.3368 - val_accuracy: 0.6400 - val_loss: 0.8208
Epoch 5/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
54ms/step - accuracy: 0.9815 - loss: 0.1793 - val_accuracy: 0.6200 - val_loss: 0.8697
Epoch 6/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
57ms/step - accuracy: 0.9913 - loss: 0.0912 - val_accuracy: 0.6300 - val_loss: 0.9003
Epoch 7/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
55ms/step - accuracy: 1.0000 - loss: 0.0479 - val_accuracy: 0.6000 - val_loss: 1.0314
Epoch 8/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
54ms/step - accuracy: 0.9975 - loss: 0.0326 - val_accuracy: 0.5900 - val_loss: 1.1258
Epoch 9/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
54ms/step - accuracy: 1.0000 - loss: 0.0175 - val_accuracy: 0.6200 - val_loss: 1.0990
Epoch 10/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
54ms/step - accuracy: 1.0000 - loss: 0.0124 - val_accuracy: 0.6300 - val_loss: 1.1140
Epoch 11/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
62ms/step - accuracy: 1.0000 - loss: 0.0071 - val_accuracy: 0.6200 - val_loss: 1.1660
Epoch 12/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
57ms/step - accuracy: 1.0000 - loss: 0.0071 - val_accuracy: 0.5900 - val_loss: 1.2297
Epoch 13/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
58ms/step - accuracy: 1.0000 - loss: 0.0038 - val_accuracy: 0.5700 - val_loss: 1.2807
Epoch 14/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
76ms/step - accuracy: 1.0000 - loss: 0.0043 - val_accuracy: 0.5700 - val_loss: 1.3189
```

```
Epoch 15/15
[1m7/7[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
88ms/step - accuracy: 1.0000 - loss: 0.0038 - val_accuracy: 0.5700 - val_loss: 1.3408
[1m4/4[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
90ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.5245
```

LSTM

Pengaruh jumlah layer LSTM

```
F1-Score: 0.5111

Training LSTM dengan 2 layers...
Epoch 1/3
[1m13/13[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m5s[0m
84ms/step - accuracy: 0.3981 - loss: 1.0915 - val_accuracy: 0.3800 - val_loss: 1.0590
Epoch 2/3
[1m13/13[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
66ms/step - accuracy: 0.4238 - loss: 1.0165 - val_accuracy: 0.4000 - val_loss: 1.0248
Epoch 3/3
[1m13/13[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
66ms/step - accuracy: 0.6091 - loss: 0.8456 - val_accuracy: 0.6300 - val_loss: 0.8594
[1m4/4[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
65ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.5062

Training LSTM dengan 3 layers...
Epoch 1/3
[1m13/13[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m4s[0m
116ms/step - accuracy: 0.3881 - loss: 1.0909 - val_accuracy: 0.3800 - val_loss: 1.0263
Epoch 2/3
[1m13/13[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
95ms/step - accuracy: 0.4148 - loss: 0.9864 - val_accuracy: 0.6400 - val_loss: 0.9019
Epoch 3/3
[1m13/13[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
97ms/step - accuracy: 0.6921 - loss: 0.7825 - val_accuracy: 0.6400 - val_loss: 0.8106
[1m4/4[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m
84ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
```

```
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.4949
```

Pengaruh banyak cell LSTM per layer

```
F1-Score: 0.5210

Training LSTM dengan 128 cells...
Epoch 1/3
[1m13/13[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m3s[0m 148ms/step - accuracy: 0.3635 - loss: 1.0875 - val_accuracy: 0.3800 - val_loss: 1.0170
Epoch 2/3
[1m13/13[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m2s[0m 132ms/step - accuracy: 0.4421 - loss: 0.9734 - val_accuracy: 0.6000 - val_loss: 0.9218
Epoch 3/3
[1m13/13[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m2s[0m 133ms/step - accuracy: 0.6415 - loss: 0.8300 - val_accuracy: 0.6400 - val_loss: 0.7661
[1m4/4[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 82ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.4946
```

Pengaruh jenis layer LSTM berdasarkan arah

```
Testing: Arah LSTM

Training unidirectional LSTM...
Epoch 1/3
[1m13/13[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m3s[0m 110ms/step - accuracy: 0.3867 - loss: 1.0946 - val_accuracy: 0.3500 - val_loss: 1.0802
Epoch 2/3
[1m13/13[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 66ms/step - accuracy: 0.4248 - loss: 1.0470 - val_accuracy: 0.4400 - val_loss: 1.0340
Epoch 3/3
[1m13/13[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m 65ms/step - accuracy: 0.4471 - loss: 0.9588 - val_accuracy: 0.5500 - val_loss: 0.9878
[1m4/4[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m0s[0m 66ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.4580
```

```
Training bidirectional LSTM...
Epoch 1/3
[1m13/13[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m7s[0m
119ms/step - accuracy: 0.3716 - loss: 1.0909 - val_accuracy: 0.3900 - val_loss: 1.0612
Epoch 2/3
[1m13/13[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
89ms/step - accuracy: 0.4168 - loss: 1.0251 - val_accuracy: 0.4500 - val_loss: 0.9910
Epoch 3/3
[1m13/13[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
88ms/step - accuracy: 0.6173 - loss: 0.8537 - val_accuracy: 0.6800 - val_loss: 0.8401
[1m4/4[0m [32m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m[37m[0m [1m1s[0m
123ms/step
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We recommend
using instead the native Keras format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
F1-Score: 0.5769
```

KESIMPULAN DAN SARAN

Kesimpulan

Berdasarkan implementasi dan serangkaian pengujian yang telah dilakukan terhadap model Convolutional Neural Network (CNN), Simple Recurrent Neural Network (RNN), dan Long-Short Term Memory (LSTM), dapat ditarik beberapa kesimpulan sebagai berikut:

1. Implementasi Model From Scratch:
    a. Modul forward propagation untuk CNN (meliputi layer Conv2D, Pooling, Flatten, Global Pooling, dan Dense) berhasil diimplementasikan dari awal (from scratch). Hasil prediksi dari implementasi from scratch menunjukkan konsistensi yang tinggi dengan model Keras ketika menggunakan bobot yang sama, divalidasi dengan metrik F1-score.
    b. Modul forward propagation untuk Simple RNN dan LSTM (meliputi layer Embedding, RNN/LSTM, Dropout, dan Dense) juga berhasil diimplementasikan from scratch. Perbandingan dengan model Keras menunjukkan kesesuaian fungsionalitas, meskipun akurasi numerik yang identik mungkin sulit dicapai karena perbedaan inisialisasi dan optimasi internal Keras yang tidak direplikasi sepenuhnya.
2. Pengaruh Hiperparameter pada CNN (Dataset CIFAR-10):
    a. Jumlah Layer Konvolusi: Penambahan jumlah layer konvolusi umumnya meningkatkan performa hingga titik tertentu (misalnya, 3 layer menunjukkan hasil terbaik dibandingkan

2 atau 4 layer dalam pengujian ini), setelah itu performa bisa stagnan atau menurun akibat overfitting atau kompleksitas model yang berlebihan.

b. Banyak Filter per Layer Konvolusi: Meningkatkan jumlah filter per layer konvolusi cenderung meningkatkan kemampuan model untuk mengekstraksi fitur yang lebih kaya, yang umumnya berujung pada F1-score yang lebih baik. Namun, penambahan filter yang terlalu banyak juga dapat menyebabkan overfitting dan meningkatkan beban komputasi.
c. Ukuran Filter per Layer Konvolusi: Kombinasi ukuran filter yang lebih kecil (misalnya, 3x3 atau 5x5) pada layer-layer yang lebih dalam seringkali memberikan performa yang baik. Penggunaan filter yang lebih besar pada layer awal dapat membantu menangkap fitur global, namun konfigurasi yang optimal bervariasi.
d. Jenis Pooling Layer: Dalam pengujian ini, Average Pooling menunjukkan performa F1-score yang sedikit lebih baik dibandingkan Max Pooling. Pilihan jenis pooling dapat bergantung pada karakteristik dataset dan arsitektur spesifik.

3. Pengaruh Hiperparameter pada Simple RNN (Dataset NusaX-Sentiment):
    a. Jumlah Layer RNN: Peningkatan jumlah layer RNN (misalnya dari 1 ke 2 layer) menunjukkan peningkatan performa, namun penambahan lebih lanjut (ke 3 layer) justru menurunkan F1-score, mengindikasikan potensi overfitting atau vanishing gradient.
    b. Banyak Cell RNN per Layer: Menambah jumlah sel per layer RNN (misalnya dari 32 ke 64 sel) secara signifikan meningkatkan performa. Namun, penambahan lebih lanjut (ke 128 sel) tidak selalu memberikan peningkatan yang sebanding dan bisa jadi performanya sedikit menurun.
    c. Jenis Layer RNN (Arah): Untuk Simple RNN pada dataset ini, model unidirectional memberikan hasil yang lebih baik dibandingkan model bidirectional. Hal ini mungkin disebabkan oleh karakteristik dataset atau interaksi dengan hiperparameter lain.
4. Pengaruh Hiperparameter pada LSTM (Dataset NusaX-Sentiment):
    a. Jumlah Layer LSTM: Penambahan jumlah layer LSTM (misalnya dari 2 ke 3 layer) tidak selalu meningkatkan performa dan dalam kasus ini justru sedikit menurun, menunjukkan bahwa arsitektur yang lebih sederhana bisa lebih efektif untuk dataset dan konfigurasi yang digunakan.
    b. Banyak Cell LSTM per Layer: Peningkatan jumlah sel LSTM (misalnya, dari 64 ke 128 sel dalam pengujian ini) tidak menunjukkan peningkatan performa yang signifikan, bahkan sedikit menurun. Hal ini menandakan pentingnya menemukan keseimbangan kapasitas model.
    c. Jenis Layer LSTM (Arah): Model LSTM bidirectional menunjukkan performa F1-score yang lebih unggul dibandingkan unidirectional. Ini sejalan dengan teori bahwa pemrosesan sekuens dari dua arah dapat menangkap konteks yang lebih baik untuk tugas klasifikasi teks.

Saran

Untuk pengembangan dan penelitian lebih lanjut, beberapa saran yang dapat dipertimbangkan adalah:

1. Melakukan pencarian hiperparameter yang lebih ekstensif dan sistematis untuk menemukan kombinasi optimal untuk setiap model dan dataset.

2. Mengimplementasikan dan menguji berbagai teknik regularisasi tambahan untuk mencegah overfitting dan meningkatkan generalisasi model.
3. Melakukan analisis kesalahan untuk memahami jenis kesalahan yang sering dilakukan model dan mengidentifikasi area perbaikan potensial.
4. Menguji model yang telah diimplementasikan pada dataset lain untuk menguji generalisasi dan adaptabilitasnya.

PEMBAGIAN TUGAS

13522001	Implementasi RNN, Penyusunan testing, Eksperimen parameter, Penyusunan dokumen
13522105	Implementasi FFNN, Implementasi LSTM, Eksperimen parameter, Penyusunan dokumen

REFERENSI

- https://d2l.ai/chapter_recurrent-modern/lstm.html
- https://d2l.ai/chapter_recurrent-modern/deep-rnn.html
- https://d2l.ai/chapter_recurrent-modern/bi-rnn.html
- https://d2l.ai/chapter_recurrent-neural-networks/index.html
- https://d2l.ai/chapter_convolutional-neural-networks/index.html
- https://numpy.org/doc/2.1/reference/generated/numpy.einsum.html

GITHUB:
https://github.com/fabianradenta/Tubes2_ML