

TUGAS BESAR 2
IF3270 PEMBELAJARAN MESIN
SEMESTER II TAHUN 2024/2025

“Convolutional Neural Network dan Recurrent Neural Network”



OLEH:

| | |
|--------------------------|----------|
| Irfan Sidiq Permana | 13522007 |
| Bryan Cornelius Lawrence | 13522033 |
| Ahmad Hasan Albana | 13522041 |

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

2025

DAFTAR ISI

| | |
|---|-----------|
| DAFTAR ISI..... | 1 |
| BAB I..... | 3 |
| 1.1. Abstraksi..... | 3 |
| BAB II..... | 4 |
| 3.1 CNN..... | 4 |
| 3.1.1. Deskripsi Kelas..... | 4 |
| 3.1.1.1. Kelas Conv2D..... | 4 |
| 3.1.1.2. Kelas Dense..... | 5 |
| 3.1.1.3. Kelas Flatten..... | 6 |
| 3.1.1.4. Kelas MaxPooling2D..... | 6 |
| 3.1.1.5. Kelas AveragePooling2D..... | 6 |
| 3.1.2 Forward Propagation..... | 7 |
| 3.2 RNN..... | 7 |
| 3.2.1. Deskripsi Kelas..... | 7 |
| 3.2.1.1. Kelas SimpleRNN_c..... | 7 |
| 3.2.1.2. Kelas Bidirectional_c..... | 8 |
| 3.2.1.3. Kelas Dense_c..... | 9 |
| 3.2.1.4. Kelas Embedding_c..... | 10 |
| 3.2.1.5. Kelas Dropout_c..... | 10 |
| 3.2.1.6. Kelas Model_c..... | 10 |
| 3.2.2. Forward Propagation..... | 11 |
| 3.3 LSTM..... | 12 |
| 3.3.1 Deskripsi Kelas..... | 12 |
| 3.3.1.1. Kelas LSTM_c..... | 12 |
| 3.3.1.2. Kelas Bidirectional_c..... | 12 |
| 3.3.1.3. Kelas Dense_c..... | 13 |
| 3.3.1.4. Kelas Embedding_c..... | 14 |
| 3.3.1.5. Kelas Dropout_c..... | 14 |
| 3.3.1.6. Kelas Model_c..... | 15 |
| 3.3.2 Forward Propagation..... | 15 |
| BAB III..... | 17 |
| 4.1 CNN..... | 17 |
| 4.1.1 Pengaruh Jumlah Layer..... | 17 |
| 4.1.2 Pengaruh Banyak Filter per Layer Konvolusi..... | 18 |
| 4.1.3 Pengaruh Ukuran Filter per Layer Konvolusi..... | 19 |
| 4.1.4 Pengaruh Jenis Pooling..... | 20 |

| | | |
|-----------------------------|---|-----------|
| 4.1.5 | Perbandingan dengan Model from Scratch..... | 21 |
| 4.2 | RNN..... | 22 |
| 4.2.1. | Pengaruh Jumlah Layer..... | 22 |
| 4.2.2. | Pengaruh Jumlah Cell..... | 23 |
| 4.2.3. | Pengaruh Jenis Layer..... | 25 |
| 4.2.4. | Perbandingan dengan Model from Scratch..... | 26 |
| 4.3 | LSTM..... | 26 |
| 4.3.1. | Pengaruh Jumlah Layer..... | 26 |
| 4.3.2. | Pengaruh Jumlah Cell..... | 27 |
| 4.3.3. | Pengaruh Jenis Layer..... | 29 |
| 4.3.4. | Perbandingan dengan Model from Scratch..... | 30 |
| BAB IV..... | | 31 |
| 4.1. | Kesimpulan..... | 31 |
| 4.2. | Saran..... | 31 |
| PEMBAGIAN TUGAS..... | | 32 |
| REFERENSI..... | | 33 |
| LAMPIRAN..... | | 34 |
| Repository..... | | 34 |

BAB I

DESKRIPSI MASALAH

1.1. Abstraksi

Peserta kuliah ditugaskan untuk mengimplementasikan modul forward propagation CNN dan RNN from scratch. Lalu dilakukan pelatihan model ML dengan library Keras dengan dataset yang disediakan. Akan dianalisis berbagai pengaruh parameter yang ada pada model tersebut.

BAB II

PEMBAHASAN

3.1 CNN

3.1.1. Deskripsi Kelas

3.1.1.1. Kelas Conv2D

Kelas Conv2D merupakan kelas untuk menginstansiasikan layer konvolusi, yaitu layer terpenting dalam CNN.

| Atribut | Deskripsi |
|--|---|
| <code>num_kernels: int</code> | Berisi jumlah kernel dari layer |
| <code>kernel_size: Tuple[int, int]</code> | Berisi tuple yang menyatakan ukuran kernel |
| <code>input_channels: int</code> | Berisi jumlah channel dari input |
| <code>strides: Tuple[int, int]</code> | Berisi tuple yang menyatakan geser ke bawah berapa langkah, geser ke kanan berapa langkah |
| <code>padding: str</code> | Dapat bernilai 'same' atau 'valid' |
| <code>use_bias: bool</code> | Bila bernilai true, maka bias akan digunakan pada saat training. |
| <code>name: str</code> | Nama dari layer konvolusi. Digunakan untuk membedakan antar layer waktu load bobot. |
| <code>kernel: Tensor</code> | Berisi matriks kernel dari layer |
| <code>bias: Tensor</code> | Berisi matriks bias dari layer |
| <code>_weights_initialized: bool</code> | Bernilai true jika bobot telah diinisialisasi, false jika tidak |
| Metode | Deskripsi |
| <code>_ensure_weights_initialized(self, C_in_from_input: int)</code> | Dipanggil sebelum memulai forwardpropagation untuk memastikan bobot telah terinisialisasi |
| <code>set_weights_from_keras(self, keras_weights: List[np.ndarray])</code> | Fungsi yang digunakan untuk memuat matriks bobot dari Keras |
| <code>forward(self, input_tensor:</code> | Fungsi untuk memulai proses |

| | |
|-------------------|--|
| Tensor) -> Tensor | forwardpropagation dan menghasilkan output |
|-------------------|--|

3.1.1.2. Kelas Dense

Kelas Dense merupakan kelas untuk menginstansiasikan layer Dense (fully connected layer), biasanya sebagai output setelah layer konvolusi.

| Atribut | Deskripsi |
|---|---|
| units: int | Berisi jumlah neuron dari layer Dense |
| activation_class: ActivationFunction | Berisi kelas fungsi aktivasi yang akan digunakan untuk forward() |
| use_bias: bool | Bila bernilai true, maka bias akan digunakan pada saat training. |
| name: str | Nama dari layer Dense. Digunakan untuk membedakan antar layer waktu load bobot. |
| kernel: Tensor | Berisi kernel dari layer |
| bias: Tensor | Bila bernilai true, maka bias akan digunakan pada saat training. |
| weights: List[Tensor] | Berisi bobot dari layer Dense |
| _weights_initialized: bool | Bernilai true jika bobot telah diinisialisasi, false jika tidak |
| Metode | Deskripsi |
| _ensure_weights_initialized(self, C_in_from_input: int) | Dipanggil sebelum memulai forwardpropagation untuk memastikan bobot telah terinisialisasi |
| set_weights_from_keras(self, keras_weights: List[np.ndarray]) | Fungsi yang digunakan untuk memuat matriks bobot dari Keras |
| forward(self, input_tensor: Tensor) -> Tensor | Fungsi untuk memulai proses forwardpropagation dan menghasilkan output |

3.1.1.3. Kelas Flatten

Kelas Flatten merupakan kelas untuk menginstansiasikan layer yang mengubah dimensi hasil konvolusi ke matriks 1D agar dapat diproses oleh layer Dense.

| Atribut | Deskripsi |
|---|---|
| <code>name: str</code> | Nama dari layer Dense. Digunakan untuk membedakan antar layer waktu load bobot. |
| Metode | Deskripsi |
| <code>forward(self, input_tensor: Tensor) -> Tensor</code> | Fungsi untuk memulai proses forwardpropagation dan menghasilkan output |

3.1.1.4. Kelas MaxPooling2D

Kelas Pooling merupakan kelas untuk menginstansiasikan layer pooling yang menggunakan fungsi maksimum.

| Atribut | Deskripsi |
|---|---|
| <code>name: str</code> | Nama dari layer Dense. Digunakan untuk membedakan antar layer waktu load bobot. |
| <code>padding: str</code> | Dapat berupa 'same' atau 'valid' |
| <code>_input_shape_during_forward: Tuple[int, ...]</code> | Berisi bentuk input ketika melakukan forward() |
| <code>_weights_initialized: bool</code> | Bernilai true jika bobot telah diinisialisasi, false jika tidak |
| Metode | Deskripsi |
| <code>forward(self, input_tensor: Tensor) -> Tensor</code> | Fungsi untuk memulai proses forwardpropagation dan menghasilkan output |

3.1.1.5. Kelas AveragePooling2D

Kelas Pooling merupakan kelas untuk menginstansiasikan layer pooling yang menggunakan fungsi mean/rata-rata.

| Atribut | Deskripsi |
|---|---|
| <code>name: str</code> | Nama dari layer Dense. Digunakan untuk membedakan antar layer waktu load bobot. |
| <code>padding: str</code> | Dapat berupa 'same' atau 'valid' |
| <code>_input_shape_during_forward: Tuple[int, ...]</code> | Berisi bentuk input ketika melakukan forward() |
| <code>_weights_initialized: bool</code> | Bernilai true jika bobot telah diinisialisasi, false jika tidak |
| Metode | Deskripsi |
| <code>forward(self, input_tensor: Tensor) -> Tensor</code> | Fungsi untuk memulai proses forwardpropagation dan menghasilkan output |

3.1.2 Forward Propagation

Forward propagation pada CNN dilakukan dengan melakukan konvolusi pada matriks input. Matriks input diasumsikan selalu minimal 2 dimensi dan maksimal 3 dimensi (2D dengan channel). Langkah yang dilakukan yaitu:

- 1) Melakukan konvolusi dengan menggunakan layer Conv2D, dimana layer ini menerima parameter stride dan padding untuk melakukan konvolusi pada matriks input. Sebelum melakukan konvolusi, dihitung dahulu dimensi output hasil konvolusi berdasarkan ukuran input, kernel, stride, dan padding.
- 2) Melakukan pooling menggunakan MaxPooling2D atau AveragePooling2D sehingga dimensi feature map mengecil, agar model dapat menangkap pola yang lebih luas. Pooling juga menerima parameter berupa stride dan padding.
- 3) Melakukan flattening sebelum menyalurkan hasil konvolusi ke layer Dense. Hal ini dilakukan karena layer Dense hanya dapat menerima input 1D, sehingga feature map yang dihasilkan layer konvolusi harus dikonversi dulu menjadi 1D.
- 4) Layer Dense terus melakukan forwardpropagation hingga mencapai layer output, dimana layer Dense output akan menghasilkan probabilitas dari tiap kelas.

3.2 RNN

3.2.1. Deskripsi Kelas

3.2.1.1. Kelas SimpleRNN_c

Kelas SimpleRNN_c merupakan kelas untuk mendefinisikan arsitektur RNN.

| Atribut | Deskripsi |
|---|---|
| <code>return_sequences: bool</code> | Mengontrol apakah RNN mengembalikan output dari semua timestep atau hanya timestep terakhir |
| <code>go_backwards: bool</code> | Menentukan arah inferensi RNN, jika False arahnya dari timestep awal ke akhir |
| <code>units: int</code> | Jumlah neuron output |
| <code>input_size: int</code> | Jumlah neuron input |
| <code>U_t: np.ndarray</code> | Bobot untuk input |
| <code>W_t: np.ndarray</code> | Bobot untuk timestep sebelumnya |
| <code>b_xh: np.ndarray</code> | Bias dari input ke hidden layer |
| Metode | Deskripsi |
| <code>__init__(self, units: int, return_sequences: bool = False, go_backwards: bool = False, input_size: int = 1, activation: str = 'tanh') -> None</code> | Membangun sebuah layer RNN |
| <code>_initialize_weights(self, input_size_runtime=None)</code> | Menginisialisasi bobot pada layer dengan nilai 0 |
| <code>forward(self, input_sequence: np.ndarray) -> np.ndarray</code> | Melakukan inferensi maju pada layer dengan menerima input vektor dan menghasilkan vektor |

3.2.1.2. Kelas Bidirectional_c

Kelas Bidirectional_c merupakan kelas untuk mendefinisikan arsitektur RNN dua arah, yaitu melakukan inferensi dari depan dan belakang.

| Atribut | Deskripsi |
|--|---|
| <code>merge_mode: str</code> | Menentukan metode penggabungan dua layer RNN |
| <code>return_sequences: bool</code> | Mengontrol apakah RNN mengembalikan output dari semua timestep atau hanya timestep terakhir |
| <code>forward_rnn: SimpleRNN_c</code> | Lapisan RNN untuk inferensi maju |
| <code>backward_rnn: SimpleRNN_c</code> | Lapisan RNN untuk inferensi |

| | mundur |
|---|--|
| output_units_bidir: int | Jumlah neuron output |
| Metode | Deskripsi |
| <code>__init__(self, rnn_layer: SimpleRNN_c, merge_mode: str = 'concat')</code> | Membangun sebuah layer Bidirectional RNN |
| <code>forward(self, input_sequence: np.ndarray) -> np.ndarray:</code> | Melakukan inferensi maju pada layer dengan menerima input vektor dan menghasilkan vektor |

3.2.1.3. Kelas Dense_c

Kelas Dense_c merupakan kelas untuk mendefinisikan arsitektur FFNN yang umum digunakan. Kelas ini dibuat untuk menyesuaikan kelas Model_c

| Atribut | Deskripsi |
|---|--|
| units: int | Jumlah neuron output |
| input_size: int | Jumlah neuron input |
| activation_function_str: str | Definisi literal fungsi aktivasi |
| use_bias: bool | Menentukan digunakan atau tidaknya bias |
| W: np.ndarray | Bobot untuk input |
| b: np.ndarray | Bobot bias |
| act_forward | Fungsi aktivasi |
| Metode | Deskripsi |
| <code>__init__(self, units: int, input_size: int = None, activation_function: str = None, use_bias: bool = True)</code> | Membuat sebuah layer Dense |
| <code>_setup_activation(self)</code> | Membuat fungsi aktivasi |
| <code>_initialize_weights_if_needed(self, current_input_size: int)</code> | Mengisi bobot dan bias dengan nilai tertentu |
| <code>_softmax(self, x: np.ndarray) -> np.ndarray</code> | Fungsi aktivasi softmax |
| <code>forward(self, input_data: np.ndarray) -> np.ndarray</code> | Melakukan inferensi maju pada layer dengan menerima input vektor dan menghasilkan vektor |

3.2.1.4. Kelas Embedding_c

Kelas Embedding_c merupakan kelas untuk memetakan *input* menjadi vektor. Dalam kasus ini menjadi vektor 3D.

| Atribut | Deskripsi |
|--|--|
| <code>input_dim: int</code> | Jumlah neuron input |
| <code>output_dim: int</code> | Jumlah neuron output |
| <code>input_length: int</code> | Jumlah timesteps |
| Metode | Deskripsi |
| <code>__init__(self, input_dim: int, output_dim: int, input_length: int = None)</code> | Membangun sebuah layer Embedding |
| <code>forward(self, input_sequence: np.ndarray) -> np.ndarray:</code> | Melakukan inferensi maju pada layer dengan menerima input vektor dan menghasilkan vektor |

3.2.1.5. Kelas Dropout_c

Kelas Dropout_c merupakan kelas untuk mengabaikan beberapa neuron untuk mencegah *overfitting*.

| Atribut | Deskripsi |
|---|--|
| <code>rate: float</code> | Persentase neuron yang diabaikan |
| <code>scale: float</code> | Skala bobot neuron yang diabaikan |
| Metode | Deskripsi |
| <code>__init__(self, rate: float)</code> | Membangun sebuah layer Dropout |
| <code>forward(self, input_data: np.ndarray, training: bool = False) -> np.ndarray</code> | Melakukan inferensi maju pada layer dengan menerima input vektor dan menghasilkan vektor |

3.2.1.6. Kelas Model_c

Kelas Model_c merupakan kelas untuk menyimpan arsitektur model secara keseluruhan.

| Atribut | Deskripsi |
|---------|-----------|
|---------|-----------|

| <code>layers: list</code> | Berisi layer-layer yang tersimpan |
|--|---|
| Metode | Deskripsi |
| <code>__init__(self, layers: list)</code> | Membangun sebuah model |
| <code>predict(self, input_data: np.ndarray, training: bool = None) -> np.ndarray</code> | Melakukan prediksi dari input menggunakan layer yang dimiliki |
| <code>load_weights_from_keras_model(self, keras_model_or_path)</code> | Memuat bobot dari model keras atau file h5 |

3.2.2. Forward Propagation

Forward propagation pada RNN hampir mirip dengan FFNN. Hanya saja, *hidden layer* pada RNN menerima nilai dari *hidden layer* sebelumnya berdasarkan *timestep*. Formula umum RNN sebagai berikut

$$h_t = \tanh(Ux_t + Wh_{t-1} + b_{xh})$$

Jadi, input akan dikalikan dengan matriks U dan ditambahkan dengan nilai *hidden layer* sebelumnya yang dikalikan W lalu ditambahkan bias. Dengan cara tersebut, RNN mampu menyimpan informasi waktu atau keberlanjutan data dari inferensinya. Salah satu bentuk RNN adalah Bidirectional RNN. Bidirectional RNN menggunakan dua layer RNN yang bergerak maju dan bergerak mundur berdasarkan timestepnya. Nantinya setiap hasil inferensi digabungkan dengan metode tertentu.

Arsitektur RNN di tugas ini, terdiri dari Text Vectorization, Embedding layer, Bidirectional RNN, dan Dense layer. Tahapannya sebagai berikut:

- Input berupa teks panjang akan divektorisasi dan diubah menjadi angka sehingga 1 kalimat akan menjadi 1 list berisi angka (vektor).
- Vektor tersebut diubah menjadi vektor beberapa dimensi tergantung dari kalimat terpanjang.
- SimpleRNN atau Bidirectional RNN melakukan inferensi dengan jumlah timesteps dari hasil embedding.
- Keluaran RNN akan digabungkan pada dense layer yang menggunakan softmax untuk klasifikasi.

3.3 LSTM

3.3.1 Deskripsi Kelas

3.3.1.1. Kelas LSTM_c

Kelas LSTM_c merupakan kelas untuk mendefinisikan arsitektur LSTM.

| Atribut | Deskripsi |
|---|--|
| <code>return_sequences: bool</code> | Mengontrol apakah LSTM mengembalikan output dari semua timestep atau hanya timestep terakhir |
| <code>go_backwards: bool</code> | Menentukan arah inferensi LSTM, jika False arahnya dari timestep awal ke akhir |
| <code>units: int</code> | Jumlah neuron output |
| <code>input_size: int</code> | Jumlah neuron input |
| <code>U_t: np.ndarray</code> | Bobot untuk input |
| <code>W_t: np.ndarray</code> | Bobot untuk timestep sebelumnya |
| <code>b_xh: np.ndarray</code> | Bias dari input ke hidden layer |
| Metode | Deskripsi |
| <code>__init__(self, units: int, return_sequences: bool = False, go_backwards: bool = False, input_size: int = 1, activation: str = 'tanh') -> None</code> | Membangun sebuah layer LSTM |
| <code>_initialize_weights(self, input_size_runtime=None)</code> | Menginisialisasi bobot pada layer dengan nilai 0 |
| <code>forward(self, input_sequence: np.ndarray) -> np.ndarray</code> | Melakukan inferensi maju pada layer dengan menerima input vektor dan menghasilkan vektor |

3.3.1.2. Kelas Bidirectional_c

Kelas Bidirectional_c merupakan kelas untuk mendefinisikan arsitektur LSTM dua arah, yaitu melakukan inferensi dari depan dan belakang.

| Atribut | Deskripsi |
|-------------------------------------|---|
| <code>merge_mode: str</code> | Menentukan metode penggabungan dua layer LSTM |
| <code>return_sequences: bool</code> | Mengontrol apakah LSTM |

| | mengembalikan output dari semua timestep atau hanya timestep terakhir |
|---|--|
| <code>forward_lstm: LSTM_c</code> | Lapisan LSTM untuk inferensi maju |
| <code>backward_lstm: LSTM_c</code> | Lapisan LSTM untuk inferensi mundur |
| <code>output_units_bidir: int</code> | Jumlah neuron output |
| Metode | Deskripsi |
| <code>__init__(self, lstm_layer: LSTM_c, merge_mode: str = 'concat')</code> | Membangun sebuah layer Bidirectional LSTM |
| <code>forward(self, input_sequence: np.ndarray) -> np.ndarray:</code> | Melakukan inferensi maju pada layer dengan menerima input vektor dan menghasilkan vektor |

3.3.1.3. Kelas Dense_c

Kelas Dense_c merupakan kelas untuk mendefinisikan arsitektur FFNN yang umum digunakan. Kelas ini dibuat untuk menyesuaikan kelas Model_c

| Atribut | Deskripsi |
|---|--|
| <code>units: int</code> | Jumlah neuron output |
| <code>input_size: int</code> | Jumlah neuron input |
| <code>activation_function_str: str</code> | Definisi literal fungsi aktivasi |
| <code>use_bias: bool</code> | Menentukan digunakan atau tidaknya bias |
| <code>W: np.ndarray</code> | Bobot untuk input |
| <code>b: np.ndarray</code> | Bobot bias |
| <code>act_forward</code> | Fungsi aktivasi |
| Metode | Deskripsi |
| <code>__init__(self, units: int, input_size: int = None, activation_function: str = None, use_bias: bool = True)</code> | Membuat sebuah layer Dense |
| <code>_setup_activation(self)</code> | Membuat fungsi aktivasi |
| <code>_initialize_weights_if_needed(self, current_input_size: int)</code> | Mengisi bobot dan bias dengan nilai tertentu |

| | |
|---|--|
| <code>_softmax(self, x: np.ndarray) -> np.ndarray</code> | Fungsi aktivasi softmax |
| <code>forward(self, input_data: np.ndarray) -> np.ndarray</code> | Melakukan inferensi maju pada layer dengan menerima input vektor dan menghasilkan vektor |

3.3.1.4. Kelas Embedding_c

Kelas Embedding_c merupakan kelas untuk memetakan *input* menjadi vektor. Dalam kasus ini menjadi vektor 3D.

| Atribut | Deskripsi |
|--|--|
| <code>input_dim: int</code> | Jumlah neuron input |
| <code>output_dim: int</code> | Jumlah neuron output |
| <code>input_length: int</code> | Jumlah timesteps |
| Metode | Deskripsi |
| <code>__init__(self, input_dim: int, output_dim: int, input_length: int = None)</code> | Membangun sebuah layer Embedding |
| <code>forward(self, input_sequence: np.ndarray) -> np.ndarray:</code> | Melakukan inferensi maju pada layer dengan menerima input vektor dan menghasilkan vektor |

3.3.1.5. Kelas Dropout_c

Kelas Dropout_c merupakan kelas untuk mengabaikan beberapa neuron untuk mencegah *overfitting*.

| Atribut | Deskripsi |
|---|--|
| <code>rate: float</code> | Persentase neuron yang diabaikan |
| <code>scale: float</code> | Skala bobot neuron yang diabaikan |
| Metode | Deskripsi |
| <code>__init__(self, rate: float)</code> | Membangun sebuah layer Dropout |
| <code>forward(self, input_data: np.ndarray, training: bool = False) -> np.ndarray</code> | Melakukan inferensi maju pada layer dengan menerima input vektor dan menghasilkan vektor |

3.3.1.6. Kelas Model_c

Kelas Model_c merupakan kelas untuk menyimpan arsitektur model secara keseluruhan.

| Atribut | Deskripsi |
|--|---|
| layers: list | Berisi layer-layer yang tersimpan |
| Metode | Deskripsi |
| <code>__init__(self, layers: list)</code> | Membangun sebuah model |
| <code>predict(self, input_data: np.ndarray, training: bool = None) -> np.ndarray</code> | Melakukan prediksi dari input menggunakan layer yang dimiliki |
| <code>load_weights_from_keras_model(self, keras_model_or_path)</code> | Memuat bobot dari model keras atau file h5 |

3.3.2 Forward Propagation

Forward propagation pada LSTM hampir mirip dengan RNN, namun LSTM memiliki mekanisme gerbang (*gates*) yang memungkinkan model untuk mengendalikan aliran informasi, baik yang disimpan, dilupakan, maupun diteruskan. Formula umum untuk setiap *gate* pada LSTM sebagai berikut.

$$f_t = \sigma(U_f x_t + W_f h_{t-1} + b_f) \text{ (Forget Gate)}$$

$$i_t = \sigma(U_i x_t + W_i h_{t-1} + b_i) \text{ (Input Gate)}$$

$$\hat{C}_t = \tanh(U_c x_t + W_c h_{t-1} + b_c) \text{ (Candidate Memory)}$$

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t \text{ (Cell State Update)}$$

$$o_t = \sigma(U_o x_t + W_o h_{t-1} + b_o) \text{ (Output Gate)}$$

$$h_t = o_t * \tanh(C_t) \text{ (Hidden State)}$$

Dengan struktur tersebut, LSTM bisa mengendalikan informasi apa yang disimpan dari masa lalu dan apa yang dilupakan, serta menghindari masalah vanishing gradient yang sering muncul pada RNN biasa. Salah satu bentuk LSTM adalah Bidirectional LSTM. Bidirectional LSTM menggunakan dua layer LSTM yang bergerak maju dan bergerak mundur berdasarkan timesteps-nya. Nantinya setiap hasil inferensi digabungkan dengan metode tertentu.

Arsitektur LSTM di tugas ini, terdiri dari Text Vectorization, Embedding layer, Bidirectional LSTM, dan Dense layer. Tahapannya sebagai berikut:

- e. Input berupa teks panjang akan divektorisasi dan diubah menjadi angka sehingga 1 kalimat akan menjadi 1 list berisi angka (vektor).
- f. Vektor tersebut diubah menjadi vektor beberapa dimensi tergantung dari kalimat terpanjang.
- g. LSTM atau Bidirectional LSTM melakukan inferensi dengan jumlah timesteps dari hasil embedding.
- h. Keluaran LSTM akan digabungkan pada dense layer yang menggunakan softmax untuk klasifikasi.

BAB III

HASIL PENGUJIAN DAN ANALISIS

4.1 CNN

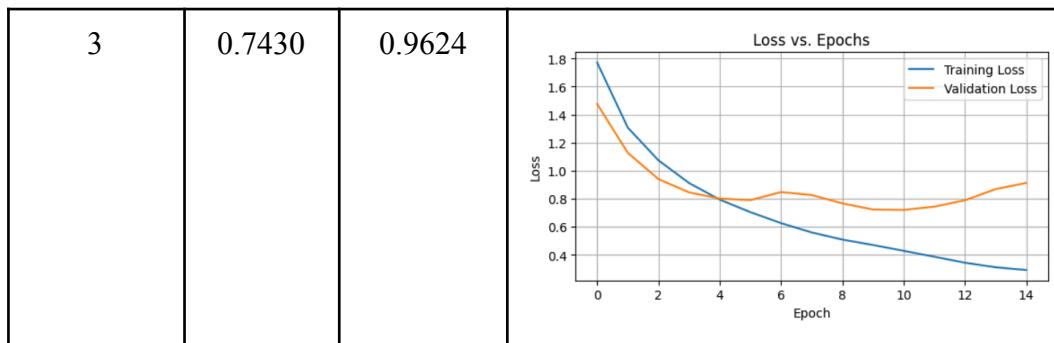
4.1.1 Pengaruh Jumlah Layer

Pengujian model terhadap pengaruh jumlah layer dilakukan dengan nilai-nilai berikut:

| Percobaan | Jumlah Layer |
|-----------|--------------|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |

Dari ketiga percobaan, diperoleh f1-score, loss, serta grafik training loss dan validation loss sebagai berikut:

| Percobaan | F1-Score | Loss | Grafik |
|-----------|----------|--------|--------|
| 1 | 0.6213 | 2.4515 | |
| 2 | 0.6814 | 1.7678 | |



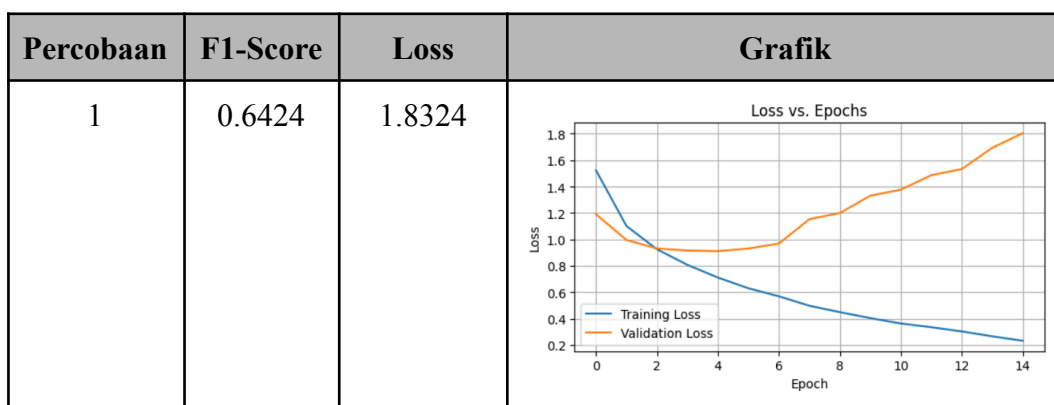
Berdasarkan percobaan tersebut, F1-score tertinggi dimiliki oleh model dengan tiga layer. Hal ini dikarenakan dengan jumlah layer konvolusi yang lebih banyak, model dapat mengenali pola spasial yang lebih kompleks dan detail. Namun terlalu banyak layer konvolusi juga tidak baik, karena dapat menyebabkan model terlalu overfit dengan pola yang ada di training.

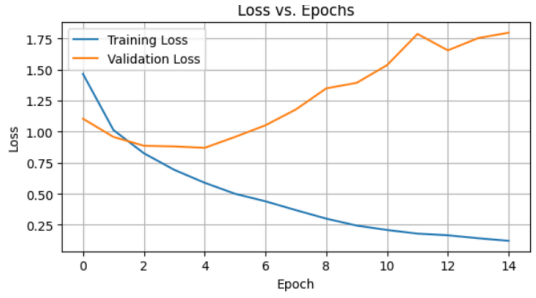
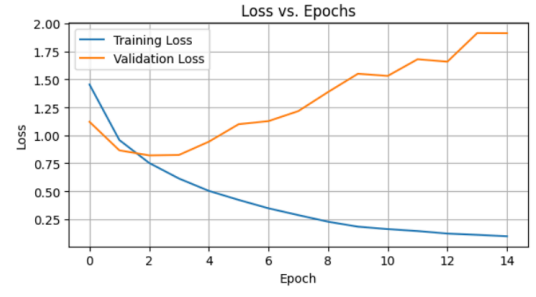
4.1.2 Pengaruh Banyak Filter per Layer Konvolusi

Pengujian model terhadap pengaruh banyak filter dilakukan dengan nilai-nilai berikut:

| Percobaan | Layer Konvolusi 1 | Layer Konvolusi 2 |
|-----------|-------------------|-------------------|
| 1 | 16 filter | 32 filter |
| 2 | 32 filter | 64 filter |
| 3 | 64 filter | 128 filter |

Dari ketiga percobaan, diperoleh f1-score, loss, serta grafik training loss dan validation loss sebagai berikut:



| | | | |
|---|--------|--------|--|
| 2 | 0.6848 | 1.8527 |  |
| 3 | 0.7200 | 1.8504 |  |

Berdasarkan percobaan tersebut, F1-score tertinggi dimiliki oleh model dengan jumlah filter terbanyak, yaitu 64 filter pada blok pertama dan 128 filter pada blok kedua. Hal ini dikarenakan jumlah filter yang lebih besar memungkinkan model untuk mengekstraksi lebih banyak fitur dari data input pada setiap tahap konvolusi. Filter yang lebih banyak dapat menangkap pola-pola visual yang lebih kompleks, sehingga model menjadi lebih mampu membedakan antar kelas dengan lebih akurat.

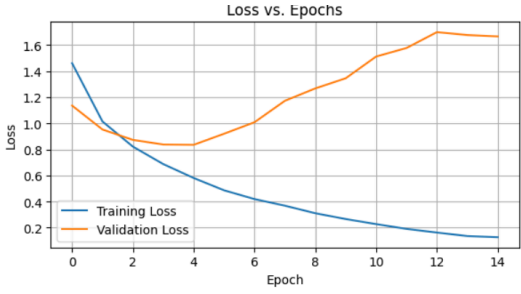
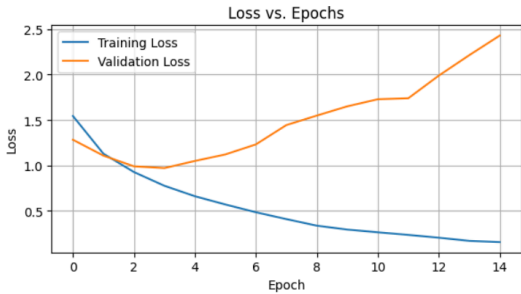
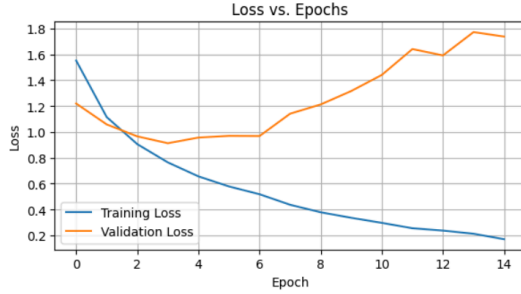
4.1.3 Pengaruh Ukuran Filter per Layer Konvolusi

Pengujian model terhadap pengaruh ukuran filter dilakukan dengan nilai-nilai berikut:

| Percobaan | Ukuran Filter |
|-----------|---------------|
| 1 | 3x3 |
| 2 | 5x3 |
| 3 | 5x5 |

Dari ketiga percobaan, diperoleh f1-score, loss, serta grafik training loss dan validation loss sebagai berikut:

| Percobaan | F1-Score | Loss | Grafik |
|-----------|----------|------|--------|
|-----------|----------|------|--------|

| | | | |
|---|--------|--------|---|
| 1 | 0.7071 | 1.7217 |  |
| 2 | 0.6487 | 2.4249 |  |
| 3 | 0.6778 | 1.7925 |  |

Berdasarkan percobaan tersebut, F1-score tertinggi dimiliki oleh model pertama dengan ukuran filter 3x3. Hal ini dikarenakan ukuran filter yang lebih kecil memungkinkan model untuk mengekstrak pola yang lebih kecil dan lebih detail pada input, dibandingkan kernel berukuran besar yang “memblur”-kan seluruh pola yang ada di input sehingga polanya menjadi abstrak atau tidak jelas. Namun ukuran filter yang terlalu kecil juga akan membuat kernel tidak mendapat gambaran besar dari pola karena terlalu fokus dengan titik kecil, maupun hasil feature map menjadi semakin besar sehingga butuh komputasi yang lebih banyak.

4.1.4 Pengaruh Jenis Pooling

Pengujian model terhadap pengaruh jenis pooling dilakukan dengan nilai-nilai berikut:

| Percobaan | Jenis Pooling |
|-----------|----------------|
| 1 | Maxpooling |
| 2 | AveragePooling |

Dari kedua percobaan, diperoleh f1-score, loss, serta grafik training loss dan validation loss sebagai berikut:

| Percobaan | F1-Score | Loss | Grafik |
|-----------|----------|--------|--------|
| 1 | 0.6962 | 1.9769 | |
| 2 | 0.6745 | 1.8779 | |

Berdasarkan percobaan tersebut, F1-score tertinggi dimiliki oleh model pertama dengan jenis pooling yaitu Maxpooling. Hal ini dikarenakan Maxpooling lebih cocok untuk menangkap fitur yang tajam atau yang menonjol dari input, seperti sisi (edges), tekstur, dan sebagainya yang polanya berbeda antar kelas. Sedangkan AveragePooling meratakan seluruh receptive field menjadi satu, sehingga tepi atau ketajaman pola bisa jadi hilang.

4.1.5 Perbandingan dengan Model from Scratch

Perbandingan model yang kami buat dengan model dari keras sebagai berikut:

| Model | F1-Score |
|-------|----------|
| Keras | 0.3197 |

| | |
|---------|--------|
| Scratch | 0.0364 |
|---------|--------|

Terdapat perbedaan yang signifikan dari model Keras dan model from scratch, dimana model from scratch memiliki nilai yang jauh lebih buruk. Hal ini dikarenakan masih adanya bug dalam model from scratch dimana bobot dari Keras tidak seluruhnya dipasangkan ke layer yang bersesuaian sehingga masih ada layer yang bobotnya 0 di model from scratch. Akibatnya, hasil forwardpropagation pun menjadi bernilai 0.

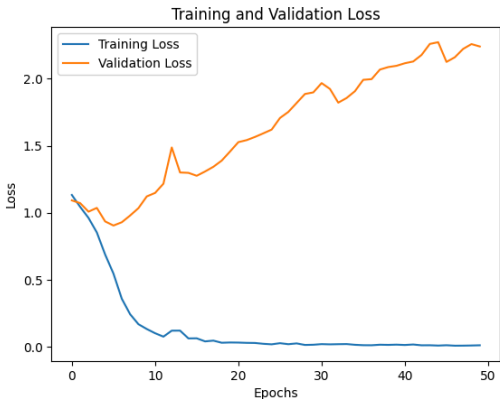
4.2 RNN

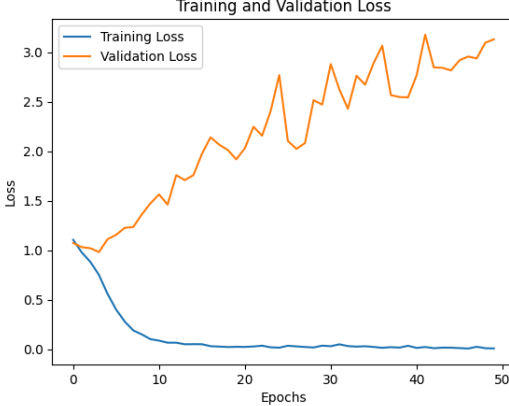
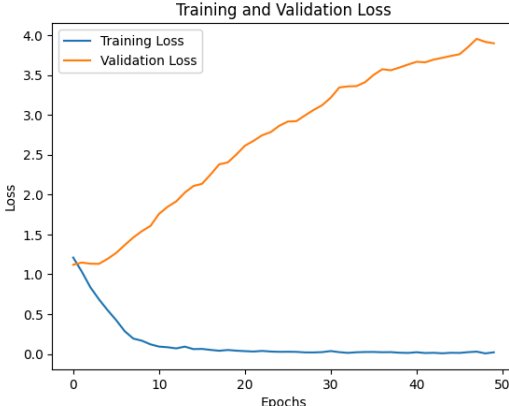
4.2.1. Pengaruh Jumlah Layer

Pengujian model terhadap pengaruh jumlah layer dilakukan dengan nilai-nilai berikut (jumlah cell untuk tiap layer sama, yaitu 32):

| Percobaan | Jumlah Layer |
|-----------|--------------|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |

Dari ketiga percobaan, diperoleh f1-score, loss, serta grafik training loss dan validation loss sebagai berikut:

| Percobaan | F1-Score | Loss | Grafik |
|-----------|----------|--------|--|
| 1 | 0.4773 | 2.1322 |  |

| | | | |
|---|--------|--------|---|
| 2 | 0.4899 | 3.2496 |  |
| 3 | 0.4813 | 3.1361 |  |

Berdasarkan percobaan tersebut, F1-score tertinggi dimiliki oleh model dengan dua layer. Namun, model dengan satu layer memiliki loss yang paling kecil. Artinya, jumlah layer berpengaruh pada model, jumlah layer harus disesuaikan supaya tidak terlalu banyak atau terlalu sedikit. Karena jumlah layer yang banyak akan meningkatkan terjadinya vanishing gradient, tetapi jumlah layer yang terlalu sedikit akan menghilangkan makna dari parameter di layer sebelumnya.

4.2.2. Pengaruh Jumlah Cell

Pengujian model terhadap pengaruh jumlah cell dilakukan dengan nilai-nilai berikut (jumlah layer disamakan, yaitu menggunakan 1 layer):

| Percobaan | Jumlah Layer |
|-----------|--------------|
| 1 | 32 |
| 2 | 64 |
| 3 | 128 |

Dari ketiga percobaan, diperoleh f1-score, loss, serta grafik training loss dan validation loss sebagai berikut:

| Percobaan | F1-Score | Loss | Grafik |
|-----------|----------|--------|--------|
| 1 | 0.49 | 2.065 | |
| 2 | 0.4753 | 2.6544 | |
| 3 | 0.4539 | 2.7788 | |

Berdasarkan percobaan tersebut, F1-score tertinggi dan loss terendah dimiliki oleh model dengan 32 cell. Artinya, jumlah cell tidak perlu terlalu banyak, tetapi

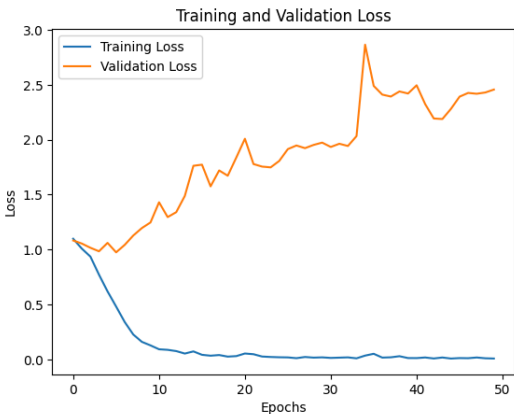
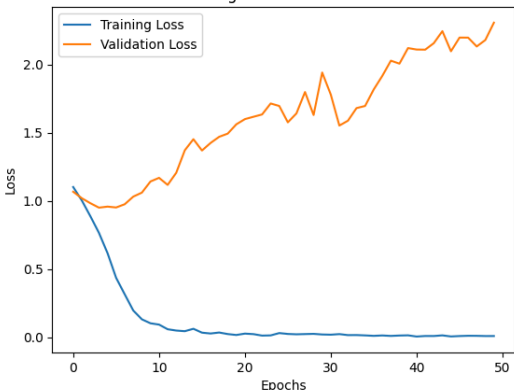
harus disesuaikan dengan input cell serta jumlah cell di layer selanjutnya dan nilainya diusahakan diseimbangkan.

4.2.3. Pengaruh Jenis Layer

Pengujian model terhadap pengaruh jenis layer dilakukan sebagai berikut:

| Percobaan | Jenis |
|-----------|----------------|
| 1 | Unidirectional |
| 2 | Bidirectional |

Dari kedua percobaan, diperoleh f1-score, loss, serta grafik training loss dan validation loss sebagai berikut:

| Percobaan | F1-Score | Loss | Grafik |
|-----------|----------|--------|--|
| 1 | 0.4830 | 2.1189 |  |
| 2 | 0.4845 | 2.4365 |  |

Berdasarkan percobaan tersebut, kedua percobaan memiliki F1-score yang hampir sama, sedangkan nilai loss dari layer unidirectional sedikit lebih kecil. Artinya, dataset yang digunakan tidak terlalu memiliki pengaruh pada variasi layer.

Karena satu data merupakan satu kalimat utuh sehingga nilai yang dihasilkan tidak berbeda jauh.

4.2.4. Perbandingan dengan Model from Scratch

Perbandingan model yang kami buat dengan model dari keras sebagai berikut:

| Model | F1-Score |
|---------|----------|
| Keras | 0.4791 |
| Scratch | 0.3874 |

Terdapat sedikit perbedaan pada hasil F1-Score dari model from scratch dengan model dari keras. Penyebabnya tidak dapat ditemukan secara pasti selain dari nilai floating point yang berbeda antara keras dan numpy. Namun, terdapat dugaan bahwa perbedaan ini disebabkan oleh perbedaan implementasi keras dalam membuat RNN dan keberadaan Dropout layer.

4.3 LSTM

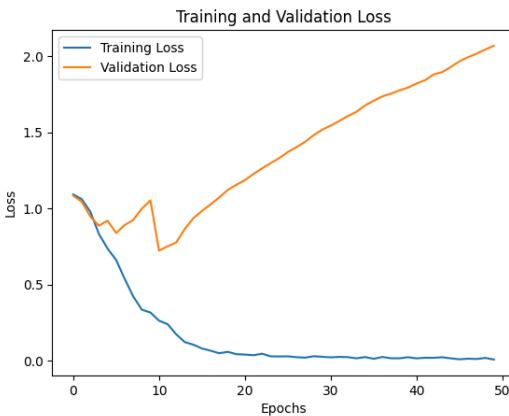
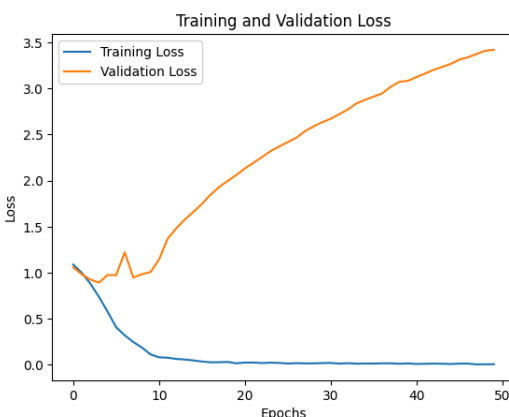
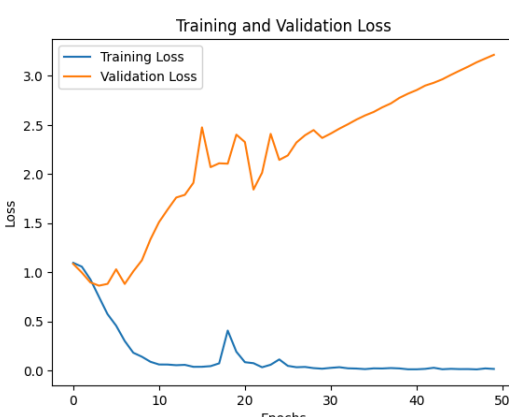
4.3.1. Pengaruh Jumlah Layer

Pengujian model terhadap pengaruh jumlah layer dilakukan dengan nilai-nilai berikut (jumlah cell untuk tiap layer sama, yaitu 32):

| Percobaan | Jumlah Layer |
|-----------|--------------|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |

Dari ketiga percobaan, diperoleh f1-score, loss, serta grafik training loss dan validation loss sebagai berikut:

| Percobaan | F1-Score | Loss | Grafik |
|-----------|----------|------|--------|
|-----------|----------|------|--------|

| | | | |
|---|--------|--------|--|
| 1 | 0.5024 | 2.4740 |  |
| 2 | 0.4988 | 3.4736 |  |
| 3 | 0.4978 | 3.2589 |  |

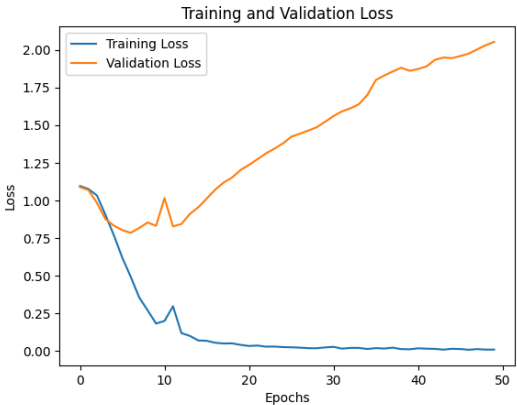

Berdasarkan percobaan tersebut, F1-score tertinggi dimiliki oleh model dengan satu layer yang juga memiliki nilai loss terkecil. Artinya, penambahan jumlah layer berpengaruh pada model sehingga performa dari model tersebut menurun yang diakibatkan kompleksitas yang berlebih dan berkemungkinan untuk overfitting.

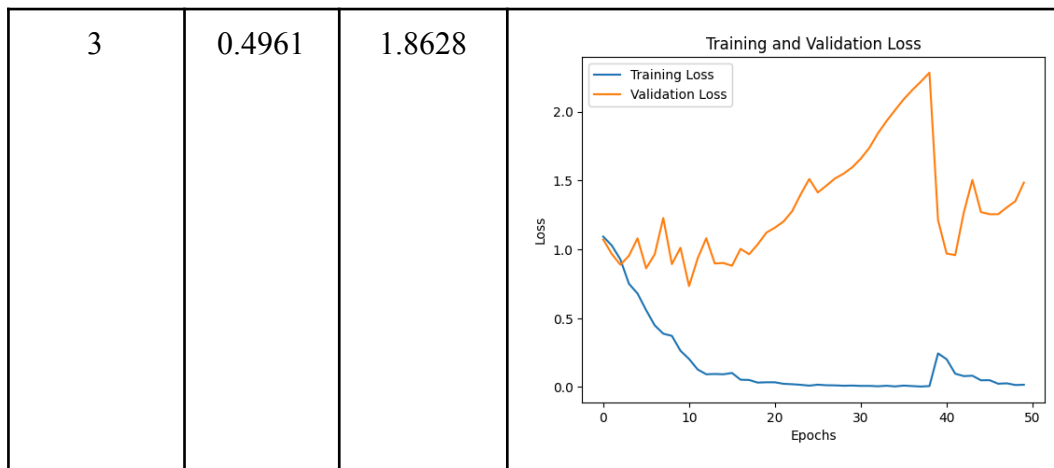
4.3.2. Pengaruh Jumlah Cell

Pengujian model terhadap pengaruh jumlah cell dilakukan dengan nilai-nilai berikut (jumlah layer disamakan, yaitu menggunakan 1 layer):

| Percobaan | Jumlah Layer |
|-----------|--------------|
| 1 | 32 |
| 2 | 64 |
| 3 | 128 |

Dari ketiga percobaan, diperoleh f1-score, loss, serta grafik training loss dan validation loss sebagai berikut:

| Percobaan | F1-Score | Loss | Grafik |
|-----------|----------|--------|--|
| 1 | 0.4967 | 2.3468 |  <p>Training and Validation Loss</p> |
| 2 | 0.4970 | 1.6450 |  <p>Training and Validation Loss</p> |



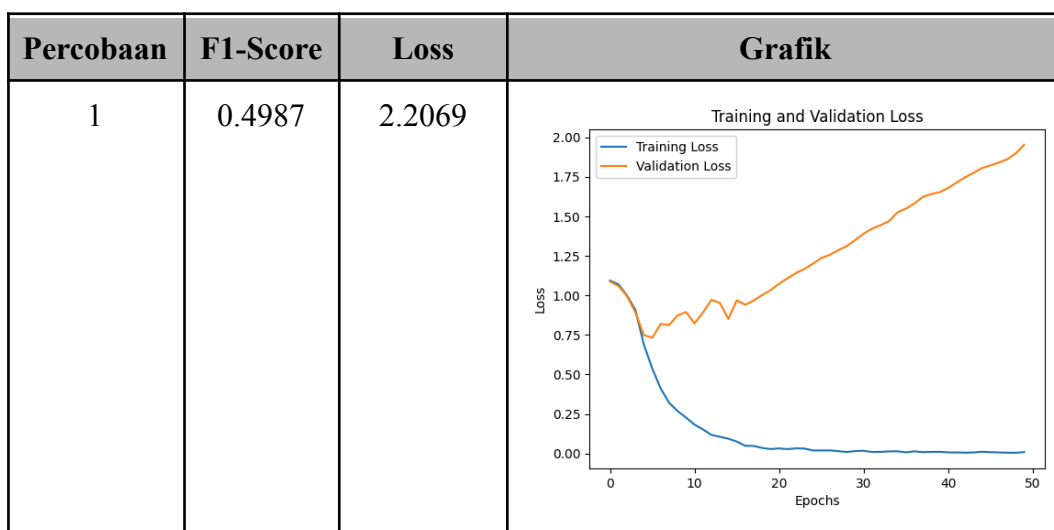
Berdasarkan percobaan tersebut, F1-score tertinggi dan loss terendah dimiliki oleh model dengan 64 cell. Artinya, penambahan jumlah layer dan unit hingga titik tertentu dapat meningkatkan performa model. Namun, menambah terlalu banyak layer/unit tidak selalu meningkatkan kinerja, dan bahkan bisa menyebabkan penurunan F1-score meskipun loss masih cukup rendah.

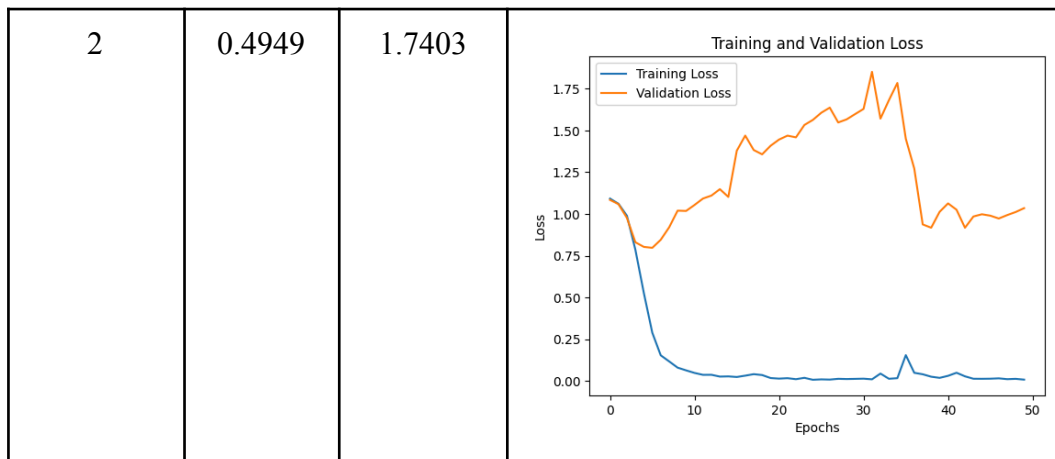
4.3.3. Pengaruh Jenis Layer

Pengujian model terhadap pengaruh jenis layer dilakukan sebagai berikut:

| Percobaan | Jenis |
|-----------|----------------|
| 1 | Unidirectional |
| 2 | Bidirectional |

Dari kedua percobaan, diperoleh f1-score, loss, serta grafik training loss dan validation loss sebagai berikut:





Berdasarkan percobaan tersebut, kedua percobaan memiliki F1-score dari layer unidirectional sedikit lebih kecil, sedangkan nilai loss dari layer bidirectional lebih kecil. Artinya, dataset yang digunakan tidak terlalu memiliki pengaruh pada variasi layer.

4.3.4. Perbandingan dengan Model from Scratch

Perbandingan model yang kami buat dengan model dari keras sebagai berikut:

| Model | F1-Score |
|---------|----------|
| Keras | 0.4791 |
| Scratch | 0.3874 |

Terdapat sedikit perbedaan pada hasil F1-Score dari model from scratch dengan model dari keras. Penyebabnya tidak dapat ditemukan secara pasti selain dari nilai floating point yang berbeda antara keras dan numpy. Namun, terdapat dugaan bahwa perbedaan ini disebabkan oleh perbedaan implementasi keras dalam membuat LSTM dan keberadaan Dropout layer.

BAB IV

KESIMPULAN DAN SARAN

4.1. Kesimpulan

CNN merupakan arsitektur yang paling populer untuk melakukan pengenalan pola dalam gambar, dikarenakan CNN sangat bagus dalam menangkap fitur spasial dari gambar dengan jumlah parameter yang lebih sedikit berkat teknik parameter sharing dan local connectivity. Dalam prakteknya, menentukan arsitektur model CNN yang tepat sangat mempengaruhi hasil dari model CNN. Faktor-faktor dalam arsitektur CNN yang mempengaruhi baik tidaknya hasil output adalah jumlah layer konvolusi, jumlah filter, ukuran filter, dan jenis pooling.

RNN merupakan arsitektur yang bagus untuk memprediksi data yang bersifat *sequence*. Salah satunya adalah pembelajaran kalimat yang merupakan *sequence* dari kata. Setiap satu kata dapat menjadi satu *timestep* sehingga satu kalimat dapat dipelajari oleh RNN tanpa menghilangkan maknanya. Percobaan ini dilakukan untuk mengklasifikasikan konteks suatu kalimat (positif, negatif, atau netral) dan RNN cocok untuk kasus tersebut.

LSTM merupakan arsitektur lanjutan dari RNN yang dirancang untuk mengatasi kelemahan RNN dalam mengingat informasi jangka panjang, seperti vanishing gradient. Arsitektur ini cocok untuk mempelajari urutan kata dalam kalimat karena memiliki mekanisme gerbang yang memungkinkan model untuk menyaring dan menyimpan informasi penting dari setiap *timestep*. Dalam kasus klasifikasi konteks kalimat (positif, negatif, atau netral), LSTM mampu mempertahankan makna keseluruhan kalimat. Dengan demikian, LSTM merupakan pilihan yang tepat untuk memahami konteks teks yang bersifat sekuensial secara mendalam dan stabil.

4.2. Saran

Pemilihan konfigurasi (jumlah layer konvolusi, ukuran filter, dll.) dalam merancang arsitektur CNN menjadi sangat penting dalam menentukan seberapa baik hasil prediksi model kita. Perlu banyak percobaan dan eksplorasi data input untuk dapat menentukan arsitektur mana yang sekiranya lebih cocok untuk suatu dataset.

Penggunaan RNN, khususnya untuk teks, perlu menggunakan tokenisasi atau vektorisasi dan embedding. Teks dijadikan token-token supaya lebih mudah terbaca

oleh komputer. Namun, penggunaan RNN memerlukan percobaan supaya arsitekturnya tidak terlalu sederhana ataupun terlalu kompleks. Selain itu, daripada menggunakan RNN, terdapat alternatif lain yang lebih baik seperti LSTM karena RNN sangat rentan dengan masalah *vanishing gradient*.

Sebagai variasi dari RNN, penggunaan LSTM juga memerlukan tahapan tokenisasi atau vektorisasi dan embedding agar model dapat memahami representasi numerik dari kata-kata. Meskipun LSTM lebih stabil dibanding RNN, pemilihan arsitektur tetap perlu disesuaikan melalui percobaan, seperti jumlah layer dan unit, agar tidak terjadi overfitting atau underfitting.

PEMBAGIAN TUGAS

| Nama | Tugas |
|---------------------------|-------|
| Irfan Sidiq Permana | CNN |
| Bryan Cornelius Lauwrence | RNN |
| Ahmad Hasan Albana | LSTM |

REFERENSI

- https://keras.io/api/layers/core_layers/embedding/
- https://keras.io/api/layers/recurrent_layers/simple_rnn/
- <https://youtu.be/4wuIOcD1LLI?si=vwMUe4xA7lw3CG61>

LAMPIRAN

Repository

https://github.com/IrfanSidiq/IF3270_CNN_RNN_LSTM.git