LAPORAN TUGAS BESAR II

IF3270 Machine Learning

"Convolutional Neural Network dan Recurrent Neural Network"

Dosen:

Dr. Fariska Zakhralativa Ruskanda, S.T., M.T.

Kelompok :

13522130 Justin Aditya Putra P.

13522155 Axel Santadi Warih

13522163 Atqiya Haydar Luqman

PROGRAM STUDI TEKNIK INFORMATIKA

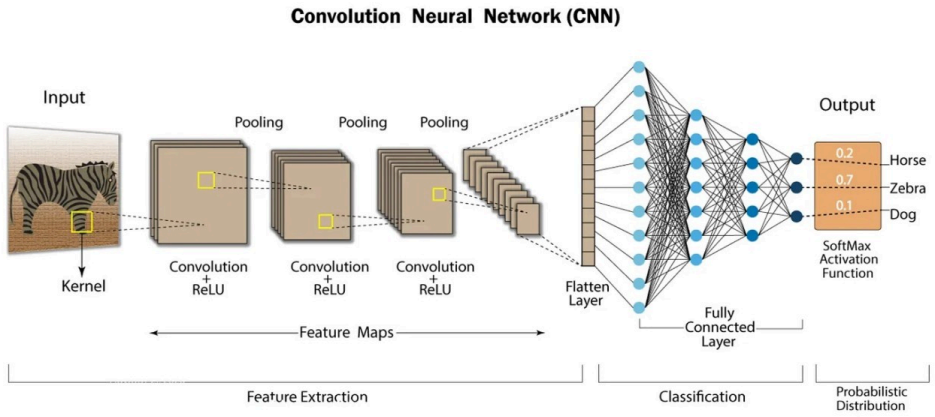
INSTITUT TEKNOLOGI BANDUNG

SEMESTER II TAHUN 2023/2024

Daftar Isi

1. Deskripsi Persoalan

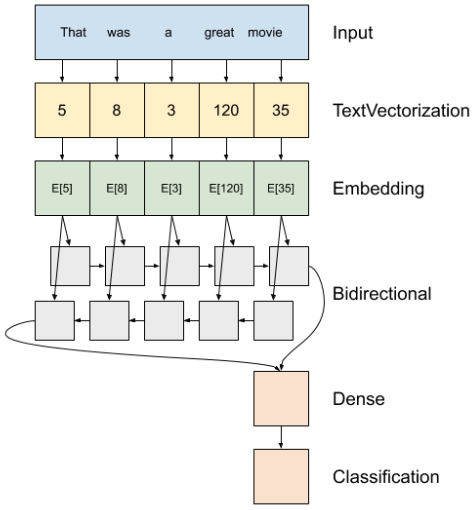
1.1. Convolutional Neural Network



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
- Lakukan variasi pelatihan sebagai berikut untuk analisis pengaruh beberapa hyperparameter dalam CNN
- Simpan hasil bobot dari pelatihan.
- Buatlah modul *forward propagation from scratch* dari model yang telah dibuat

1.2. Simple Recurrent Neural Network

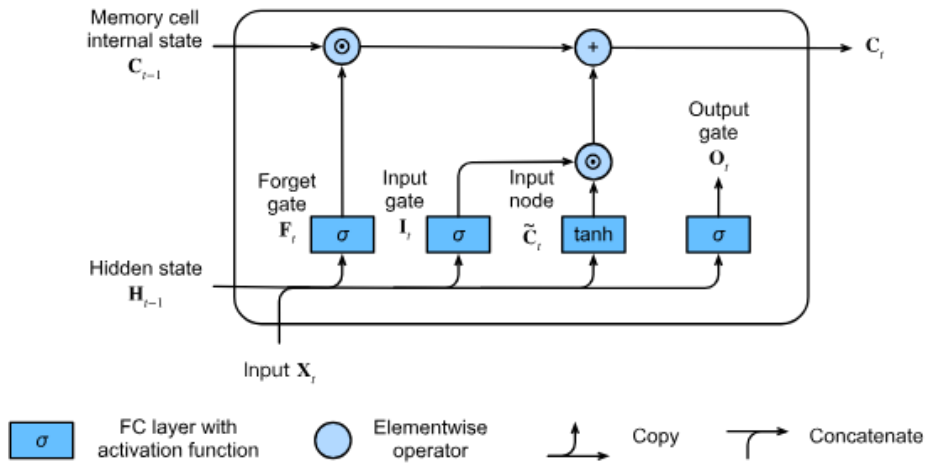


Untuk bagian ini, Anda diminta untuk melakukan beberapa hal berikut:

- Preprocessing data teks menjadi representasi numerik yang bisa diterima oleh model
- Lakukan pelatihan untuk suatu model RNN untuk *text classification* dengan dataset NusaX-Sentiment (Bahasa Indonesia) dan dengan menggunakan Keras

- Lakukan variasi pelatihan sebagai berikut untuk analisis pengaruh beberapa hyperparameter dalam RNN
- Simpan weight hasil pelatihan dengan Keras.
- Buatlah modul *forward propagation from scratch* dari model yang telah dibuat

1.3. 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
- Lakukan pelatihan untuk suatu model LSTM untuk *text classification* dengan dataset NusaX-Sentiment (Bahasa Indonesia) dan dengan menggunakan Keras
- Lakukan variasi pelatihan sebagai berikut untuk analisis pengaruh beberapa hyperparameter dalam LSTM
- Simpan weight hasil pelatihan dengan Keras.
- Buatlah modul *forward propagation from scratch* dari model yang telah dibuat

2. Pembahasan

2.1. Penjelasan Implementasi

2.1.1. Deskripsi kelas beserta deskripsi atribut dan methodnya

2.1.1.1. Kelas Layer Umum

- Embedding (LSTMlayers.py, RNNlayers.py):
  - Deskripsi: Kelas ini bertanggung jawab untuk mengubah sekuens token (integer) menjadi representasi vektor padat (embedding).
  - Atribut:
    - W (numpy array): Matriks bobot embedding dengan dimensi (vocab_size, embed_dim).
  - Method:
    - __init__(self, vocab_size, embed_dim): Konstruktor untuk inisialisasi matriks bobot.
    - forward(self, x): Melakukan lookup embedding berdasarkan input token x.
    - set_weights(self, W): Mengatur bobot matriks embedding.
- Dropout (LSTMlayers.py, RNNlayers.py):
  - Deskripsi: Mengimplementasikan mekanisme dropout untuk mencegah overfitting. Selama pelatihan, secara acak menonaktifkan sebagian unit dengan probabilitas tertentu.
  - Atribut:
    - rate (float): Probabilitas unit dinonaktifkan.
    - _training (bool): Status apakah model dalam mode pelatihan atau evaluasi.
  - Method:
    - __init__(self, rate): Konstruktor.
    - forward(self, x, training=None): Menerapkan dropout mask ke input x jika dalam mode pelatihan. Jika training tidak dispesifikasikan, menggunakan status _training internal.
    - set_training_mode(self, training: bool): Mengatur mode pelatihan/evaluasi.
- Dense / DenseLayer (LSTMlayers.py, RNNlayers.py, CNNlayers.py):
  - Deskripsi: Implementasi fully connected layer yang melakukan transformasi linear pada input, diikuti oleh fungsi aktivasi (opsional).
  - Atribut:
    - W / weights (numpy array): Matriks bobot dengan dimensi (input_dim, output_dim).
    - b / biases (numpy array): Vektor bias dengan dimensi (output_dim,).
    - activation (string, opsional): Jenis fungsi aktivasi yang digunakan (misalnya, 'relu', 'softmax', atau None).
  - Method:

        - __init__(self, input_dim, output_dim, activation=None): Konstruktor.
        - forward(self, x): Melakukan operasi output=activation(x · W+b).
        - set_weights(self, W, b): Mengatur bobot dan bias layer.

2.1.1.2. Kelas Layer Spesifik CNN

- Conv2DLayer:
    - Deskripsi: Mengimplementasikan operasi konvolusi 2D.
    - Atribut:
        - weights (numpy array): Kernel konvolusi dengan dimensi (kernel_h, kernel_w, input_channels, output_channels).
        - biases (numpy array): Bias untuk setiap output channel dengan dimensi (output_channels,).
        - activation (string): Fungsi aktivasi (misalnya, 'relu').
        - padding (string): Jenis padding ('same' atau 'valid').
    - Method:
        - __init__(self, weights, biases, activation='relu', padding='same'): Konstruktor.
        - forward(self, x): Melakukan operasi konvolusi pada input x, menerapkan padding, dan fungsi aktivasi.
- MaxPooling2DLayer:
    - Deskripsi: Mengimplementasikan operasi max pooling 2D.
    - Atribut:
        - pool_size (int atau tuple): Ukuran jendela pooling.
        - strides (int atau tuple): Langkah pergeseran jendela pooling.
    - Method:
        - __init__(self, pool_size=2, strides=2): Konstruktor.
        - forward(self, x): Melakukan operasi max pooling pada input x.
- AveragePooling2DLayer:
    - Deskripsi: Mengimplementasikan operasi average pooling 2D.
    - Atribut:
        - pool_size (int atau tuple): Ukuran jendela pooling.
        - strides (int atau tuple): Langkah pergeseran jendela pooling.
    - Method:
        - __init__(self, pool_size=2, strides=2): Konstruktor.
        - forward(self, x): Melakukan operasi average pooling pada input x.
- FlattenLayer:
    - Deskripsi: Meratakan input multidimensi menjadi vektor 1D.
    - Atribut: Tidak ada atribut spesifik selain yang diwariskan.
    - Method:
        - forward(self, x): Mengubah bentuk input x menjadi (batch_size, -1).

2.1.1.3. Kelas Layer Spesifik RNN/LSTM

- RNNCell (RNNlayers.py):
  - Deskripsi: Implementasi satu sel RNN sederhana.
  - Atribut:
    - Wx (numpy array): Matriks bobot untuk input xt.
    - Wh (numpy array): Matriks bobot untuk hidden state sebelumnya ht−1.
    - b (numpy array): Vektor bias.
  - Method:
    - __init__(self, input_dim, hidden_dim): Konstruktor.
    - forward(self, x_t, h_prev): Menghitung hidden state berikutnya ht=tanh(xt⋅Wx+ht−1⋅Wh+b).
    - set_weights(self, Wx, Wh, b): Mengatur bobot dan bias sel.
- RNN (RNNlayers.py):
  - Deskripsi: Implementasi layer RNN sederhana yang memproses sekuens data dengan menggunakan RNNCell.
  - Atribut:
    - hidden_dim (int): Dimensi hidden state.
    - cell (RNNCell): Instansi dari RNNCell.
    - return_sequences (bool): Jika True, mengembalikan seluruh sekuens hidden state. Jika False, hanya mengembalikan hidden state terakhir.
  - Method:
    - __init__(self, input_dim, hidden_dim, return_sequences=False): Konstruktor.
    - forward(self, x): Memproses sekuens input x melalui RNNCell untuk setiap timestep.
    - set_weights(self, Wx, Wh, b): Mengatur bobot untuk RNNCell internal.
- BiRNN (RNNlayers.py):
  - Deskripsi: Implementasi layer Bidirectional RNN yang terdiri dari dua layer RNN (satu forward dan satu backward).
  - Atribut:
    - fwd (RNN): Layer RNN untuk arah forward.
    - bwd (RNN): Layer RNN untuk arah backward.
    - return_sequences (bool): Menentukan apakah output berupa sekuens atau hidden state akhir.
  - Method:
    - __init__(self, input_dim, hidden_dim, return_sequences=False): Konstruktor.
    - forward(self, x): Memproses input x melalui RNN forward dan RNN backward (dengan input dibalik), kemudian menggabungkan hasilnya.

        - set_weights(self, Wx_f, Wh_f, b_f, Wx_b, Wh_b, b_b): Mengatur bobot untuk kedua RNN internal.
- LSTM (LSTMlayers.py):
    - Deskripsi: Implementasi satu unit LSTM.
    - Atribut:
        - D (int): Dimensi input.
        - H (int): Dimensi hidden state dan cell state.
        - W (numpy array): Matriks bobot untuk input xt, dimensi (D, 4H).
        - U (numpy array): Matriks bobot untuk hidden state ht−1, dimensi (H, 4H).
        - b (numpy array): Vektor bias, dimensi (4H,).
    - Method:
        - __init__(self, input_dim, hidden_dim): Konstruktor.
        - set_weights(self, W, U, b): Mengatur bobot dan bias LSTM.
        - _sigmoid(self, x): Fungsi aktivasi sigmoid.
        - forward(self, x_seq, h0=None, c0=None): Melakukan forward pass untuk satu sekuens input x_seq (dimensi (T, D)), menghitung forget gate, input gate, cell candidate, output gate, serta memperbarui cell state dan hidden state untuk setiap timestep. Mengembalikan sekuens hidden state dan final hidden & cell states.
- BidirectionalLSTM (LSTMlayers.py):
    - Deskripsi: Menggabungkan dua unit LSTM untuk memproses sekuens secara forward dan backward.
    - Atribut:
        - forward_lstm (LSTM): Unit LSTM untuk arah forward.
        - backward_lstm (LSTM): Unit LSTM untuk arah backward.
        - H (int): Dimensi hidden state (untuk satu arah).
    - Method:
        - __init__(self, input_dim, hidden_dim): Konstruktor.
        - set_weights(self, W_f, U_f, b_f, W_b, U_b, b_b): Mengatur bobot untuk kedua unit LSTM.
        - forward(self, x_seq): Memproses sekuens input x_seq melalui LSTM forward dan LSTM backward (dengan input dibalik). Hasil hidden state dari kedua arah digabungkan. Mengembalikan sekuens hidden state gabungan dan final hidden & cell states gabungan.

2.1.1.4. Kelas Model

- CNNFromScratch (CNNmodel.py):
    - Deskripsi: Kelas untuk membangun model CNN dari awal dengan memuat konfigurasi dan bobot dari model Keras yang sudah ada.

    - Atribut:
        - layers (list): Daftar layer yang membentuk model CNN (berisi instansi dari Conv2DLayer, MaxPooling2DLayer, FlattenLayer, DenseLayer, dll.).
    - Method:
        - __init__(self, keras_model): Konstruktor yang memanggil load_weights_from_keras.
        - load_weights_from_keras(self, keras_model): Mengiterasi layer-layer pada keras_model, menginisialisasi layer custom yang sesuai, dan memuat bobotnya.
        - predict(self, x): Melakukan forward propagation dengan melewatkan input x secara sekuensial melalui semua layer dalam atribut layers.
- SimpleRNNModel (RNNmodel.py):
    - Deskripsi: Kelas untuk membangun model RNN (bisa Simple RNN atau BiRNN, dan bisa bertumpuk) dari awal.
    - Atribut:
        - embed (Embedding): Layer embedding.
        - dropout (Dropout): Layer dropout.
        - rnns (list): Daftar layer RNN/BiRNN.
        - dense (Dense): Layer dense output.
        - _training (bool): Status mode pelatihan.
    - Method:
        - __init__(self, vocab_size, embed_dim, hidden_dim, num_classes, dropout_rate, num_layers, bidirectional, return_sequences_list): Konstruktor untuk membangun arsitektur model berdasarkan parameter yang diberikan.
        - forward(self, x, training=None): Melakukan forward propagation data input x melalui embedding, RNN layers, dropout, dan dense layer.
        - set_training_mode(self, training: bool): Mengatur mode pelatihan/evaluasi untuk model dan dropout layer-nya.
        - load_keras_weights(self, keras_weights): Memuat bobot dari model Keras yang sudah dilatih ke dalam layer-layer custom yang sesuai.
- ManualLSTMModel (LSTMmodel.py):
    - Deskripsi: Kelas untuk membangun model LSTM (khususnya Bidirectional LSTM diikuti Dense) dari awal dan memuat bobot dari file H5.
    - Atribut:
        - embedding (Embedding): Layer embedding.
        - bilstm (BidirectionalLSTM): Layer Bidirectional LSTM.
        - dropout (Dropout): Layer dropout.
        - dense (Dense): Layer dense output.
    - Method:

        - __init__(self, vocab_size, embedding_dim, lstm_units, dropout_rate, num_classes): Konstruktor untuk inisialisasi layers.
        - load_weights(self, filename: str): Memuat bobot dari file .h5 yang disimpan oleh Keras, memetakan nama layer dan variabel Keras ke struktur layer custom.
        - forward(self, x_batch: np.ndarray, training: bool = False): Melakukan forward propagation untuk satu batch data. Input x_batch dilewatkan melalui embedding, kemudian setiap sekuens dalam batch diproses oleh bilstm. Output hidden state akhir dari BiLSTM digabungkan, dilewatkan ke dropout dan dense layer, diakhiri dengan aktivasi softmax.

2.1.2. Penjelasan forward propagation

2.1.2.1. CNN

Proses forward propagation pada model CNN mengikuti alur sekuensial berdasarkan layer-layer yang telah dimuat dari model Keras:

- Input gambar X (misalnya, dengan dimensi (batch_size, height, width, channels)) diterima.
- Data dilewatkan secara berurutan melalui setiap layer yang ada dalam self.layers:
    - Jika layer adalah Conv2DLayer:
        - Input diproses dengan operasi konvolusi menggunakan kernel dan bias yang tersimpan.
        - Operasi padding ('same' atau 'valid') diterapkan sesuai konfigurasi.
        - Fungsi aktivasi (misalnya, ReLU: max(0,z)) diterapkan pada hasil konvolusi.
    - Jika layer adalah MaxPooling2DLayer atau AveragePooling2DLayer:
        - Operasi pooling (max atau average) diterapkan pada setiap feature map dari layer sebelumnya untuk mereduksi dimensi spasial.
    - Jika layer adalah FlattenLayer:
        - Output dari layer sebelumnya (biasanya layer konvolusi atau pooling) yang masih berbentuk multidimensi (misalnya, (batch_size, height', width', channels')) akan diratakan (di-flatten) menjadi vektor 1D per sampel (menjadi (batch_size, num_features)).
    - Jika layer adalah DenseLayer:
        - Input (yang biasanya sudah di-flatten) dikalikan dengan matriks bobot dense layer dan ditambahkan bias.
        - Fungsi aktivasi (misalnya, ReLU atau Softmax untuk layer output: $P(y_i) = \frac{e^{z_i}}{\sum j\, e^{z_j}}$) diterapkan.
- Output dari layer terakhir adalah hasil prediksi model. Untuk klasifikasi, ini biasanya berupa probabilitas kelas setelah melewati fungsi aktivasi Softmax.

2.1.2.2. Simple RNN

Proses forward propagation untuk model Simple RNN adalah sebagai berikut:

- Input: Sekuens token x dengan dimensi (batch_size, seq_len).
- Embedding: Token-token input diubah menjadi vektor embedding menggunakan self.embed.forward(x). Outputnya adalah h dengan dimensi (batch_size, seq_len, embed_dim).
- RNN Layers:
  - Output embedding h kemudian dilewatkan melalui satu atau lebih layer RNN (yang bisa berupa RNN atau BiRNN) yang tersimpan dalam self.rnns.
  - Untuk setiap layer RNN dalam self.rnns:
    - Jika input h ke layer RNN saat ini adalah 2D (yaitu, (batch_size, features)) dan ini bukan layer RNN pertama (artinya, layer RNN sebelumnya menghasilkan output state terakhir, bukan sekuens), maka h diubah menjadi 3D ((batch_size, 1, features)) agar dapat diproses sebagai sekuens dengan panjang 1.
    - h dilewatkan ke metode forward() dari layer RNN tersebut.
      - Untuk RNN: Hidden state ht dihitung untuk setiap timestep t sebagai $ht=\tanh(xt \cdot Wx+ht-1 \cdot Wh+b)$. Jika return_sequences=True, seluruh sekuens h0,...,hT−1 dikembalikan. Jika False, hanya hT−1 (state terakhir) yang dikembalikan.
      - Untuk BiRNN: Proses serupa terjadi untuk arah forward dan backward. h_f adalah output dari RNN forward, dan h_b adalah output dari RNN backward (yang memproses sekuens input terbalik). Jika return_sequences=True, h_f dan h_b (yang waktunya sudah disesuaikan) digabungkan pada dimensi fitur untuk setiap timestep. Jika False, final state hf dan final state (awal) hb digabungkan. Output dari layer BiRNN memiliki dimensi fitur dua kali lipat dari hidden_dim RNN penyusunnya.
    - Output dari layer RNN saat ini menjadi input untuk layer RNN berikutnya, atau untuk dropout layer jika ini adalah layer RNN terakhir.
- Dropout: Output dari layer RNN terakhir (h) dilewatkan ke self.dropout.forward(h, training) untuk regularisasi.
- Dense Layer: Output dari dropout (h, yang kini merupakan representasi akhir dari sekuens) dilewatkan ke self.dense.forward(h) untuk menghasilkan logit klasifikasi. Dimensi input dense layer disesuaikan dengan output dari layer RNN terakhir (misalnya,

hidden_dim untuk RNN unidirectional, 2 * hidden_dim untuk BiRNN, jika return_sequences=False pada RNN terakhir).

- Softmax: Logit diubah menjadi probabilitas kelas menggunakan fungsi softmax().

2.1.2.3. LSTM

Proses forward propagation untuk model LSTM, khususnya yang diimplementasikan dalam ManualLSTMModel, adalah sebagai berikut:

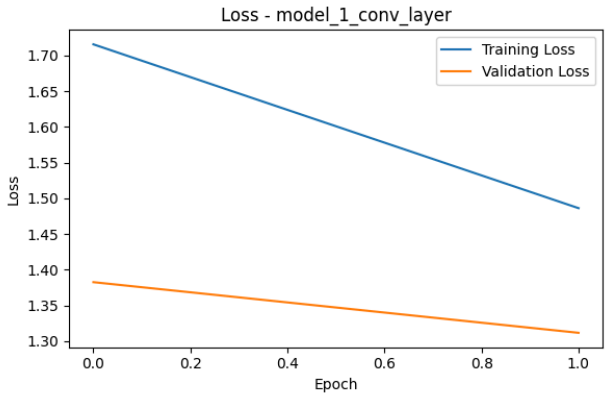
- Input: Batch sekuens token x_batch dengan dimensi (batch_size, seq_len).
- Embedding: Setiap sekuens token dalam batch diubah menjadi vektor embedding melalui self.embedding.forward(x_batch). Outputnya adalah embedded dengan dimensi (batch_size, seq_len, embedding_dim).
- Bidirectional LSTM:
  - Model ini memproses setiap sekuens dalam batch secara individual. Untuk satu sekuens seq (dimensi (seq_len, embedding_dim) dari embedded):
    - LSTM Forward Pass: Sekuens dilewatkan ke self.bilstm.forward_lstm. Untuk setiap timestep t:
      - Gerbang-gerbang LSTM dihitung:
        - Input gate: $it=\sigma(xtWxi+ht-1Uhi+bi)$
        - Forget gate: $ft=\sigma(xtWxf+ht-1Uhf+bf)$
        - Output gate: $ot=\sigma(xtWxo+ht-1Uho+bo)$
        - Cell candidate: $c\sim t=\tanh(xtWxc+ht-1Uhc+bc)$
      - Cell state diperbarui: $ct=ft\odot ct-1+it\odot c\sim t$
      - Hidden state diperbarui: $ht=ot\odot\tanh(ct)$
      - Proses ini menghasilkan sekuens hidden state forward hseq_f dan final hidden/cell state forward (hf,cf).
    - LSTM Backward Pass: Sekuens yang dibalik urutannya (xrev) dilewatkan ke self.bilstm.backward_lstm dengan cara yang sama, menghasilkan hseq_b_rev dan final hidden/cell state backward (hb_rev,cb_rev).
    - Karena model Keras Bidirectional(LSTM(...)) (tanpa return_sequences=True) hanya mengembalikan hidden state terakhir dari masing-masing arah, maka hf (hidden state terakhir dari forward pass) dan hb_rev (hidden state terakhir dari backward pass, yang sebenarnya adalah hidden state pertama dari sekuens asli) yang diambil.
    - Kedua hidden state akhir ini (hf dan hb_rev) digabungkan menjadi h_final untuk sekuens tersebut, dengan dimensi (2 * lstm_units,).
  - Setelah semua sekuens dalam batch diproses, h_final dari setiap sekuens ditumpuk (np.stack) menjadi h_stack dengan dimensi (batch_size, 2 * lstm_units).

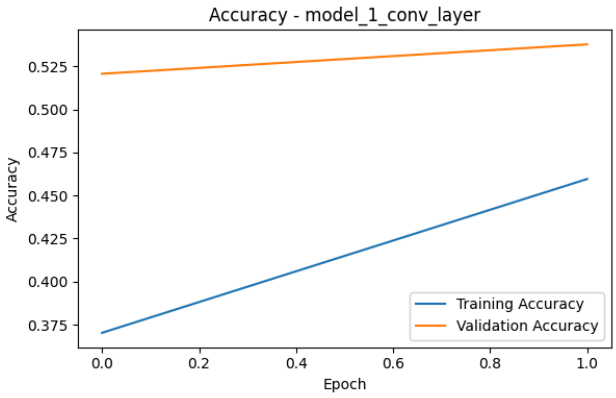
- Dropout: h_stack dilewatkan ke self.dropout.forward(h_stack, training) menghasilkan dropped.
- Dense Layer: dropped dilewatkan ke self.dense.forward(dropped) untuk menghasilkan logit klasifikasi.
- Softmax: Logit diubah menjadi probabilitas kelas (output akhir model) dengan menerapkan fungsi softmax.

2.2. Hasil Pengujian

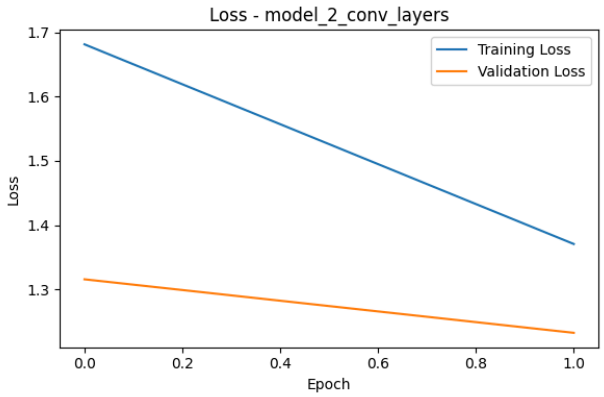
2.2.1. CNN

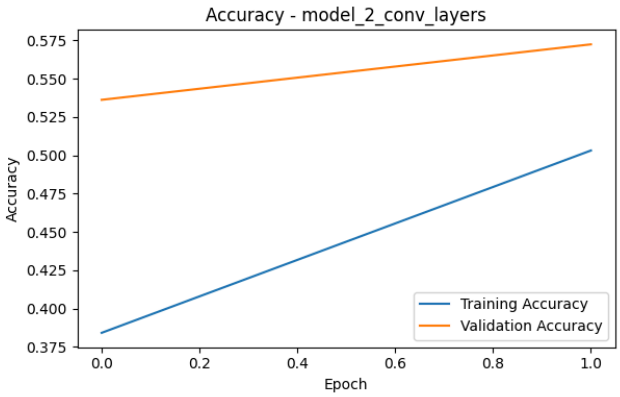
2.2.1.1. Pengaruh jumlah layer konvolusi





- Model dengan 1 Konvolusi Layer
  - Loss: Baik training loss maupun validation loss menurun, tetapi tidak terlalu signifikan
  - Accuracy: Training accuracy meningkat namun masih rendah (~0.42), dan validation accuracy stagnan di sekitar 0.53
  - Kesimpulan: Model terlalu sederhana (underfitting), kurang mampu mempelajari fitur kompleks dari data





- Model dengan 2 Konvolusi Layer

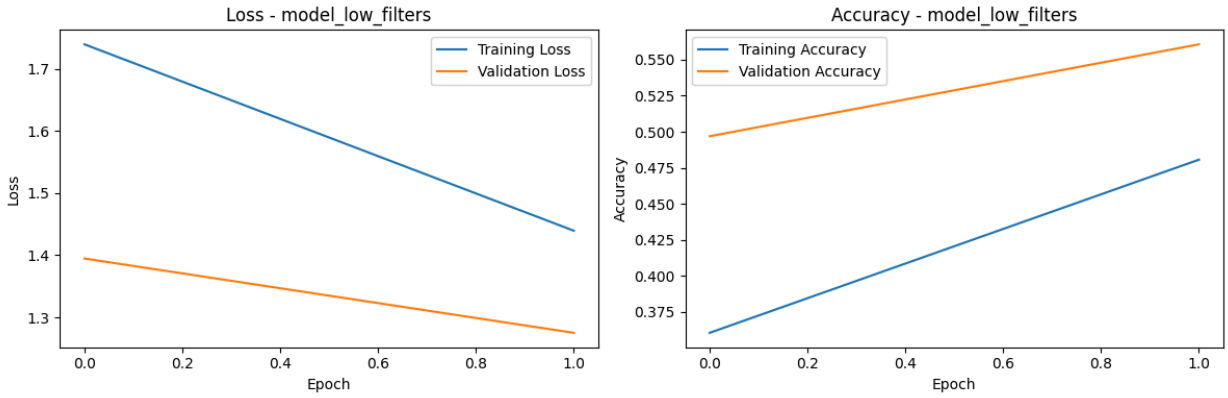
- Loss: Penurunan loss lebih stabil dan signifikan dibanding model 1 layer.
- Accuracy: Terdapat peningkatan akurasi baik pada training (~0.51) maupun validation (~0.56).
- Kesimpulan: Performa meningkat dengan kompleksitas tambahan. Model mampu belajar lebih baik terhadap data.

- Model dengan 3 Konvolusi Layer
  - Loss: Penurunan training dan validation loss sangat baik; lebih curam dibanding model sebelumnya.
  - Accuracy: Training accuracy meningkat hingga hampir 0.60, dan validation accuracy juga meningkat.
  - Kesimpulan: Model dengan 3 layer menunjukkan performa terbaik, menunjukkan bahwa penambahan konvolusi layer memperkaya kemampuan ekstraksi fitur.

Kesimpulan pengaruh jumlah layer konvolusi

- Meningkatkan jumlah layer konvolusi dari 1 ke 3 menunjukkan perbaikan signifikan dalam hal penurunan loss dan peningkatan akurasi. Ini mengindikasikan bahwa penambahan depth (jumlah layer) dalam CNN membantu model belajar representasi fitur yang lebih kompleks dan relevan, hingga titik tertentu. Namun, perlu diperhatikan potensi overfitting jika layer ditambah terus tanpa regularisasi yang memadai.

2.2.1.2. Pengaruh banyak filter per layer konvolusi



- Model dengan Low Filters ([16, 32]):
    - Loss: Penurunan training dan validation loss cukup stabil, tetapi masih lebih tinggi dibanding model lain.
    - Accuracy: Akurasi training dan validasi meningkat secara bertahap namun terbatas, hanya mencapai sekitar 0.53–0.55.
    - Kesimpulan: Jumlah filter yang terlalu kecil membatasi kapasitas model dalam mengekstraksi fitur yang kompleks, sehingga akurasi relatif rendah.

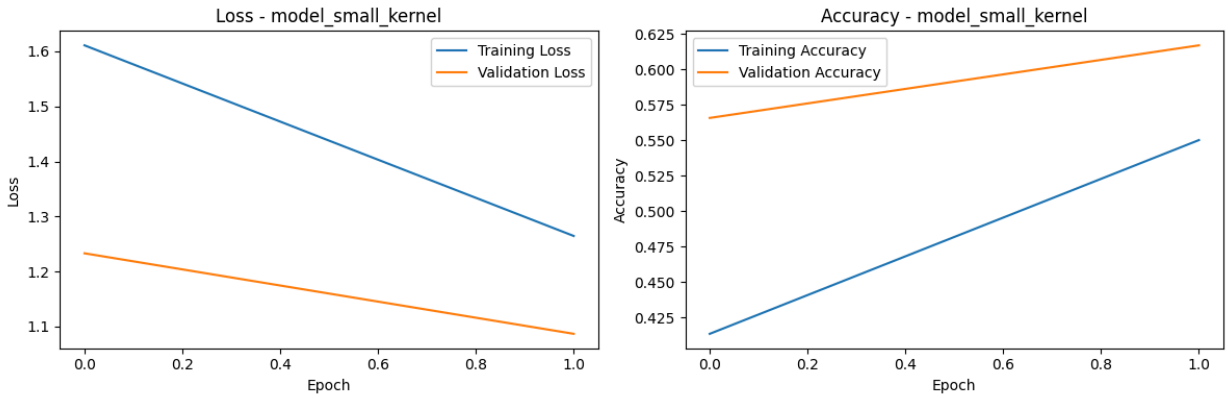
- Model dengan Medium Filters ([32, 64]):
    - Loss: Baik training loss maupun validation loss menurun lebih signifikan dibanding low filters.
    - Accuracy: Training dan validation accuracy meningkat dengan lebih tajam, mencapai kisaran 0.58–0.60.
    - Kesimpulan: Jumlah filter yang sedang memberikan keseimbangan baik antara kapasitas dan generalisasi. Ini tampaknya menjadi konfigurasi yang paling optimal dari tiga model.

- Model dengan High Filters ([64, 128]):
    - Loss: Penurunan loss cukup tajam dan lebih rendah dibanding dua model lainnya.
    - Accuracy: Validation accuracy sedikit lebih tinggi dibanding medium filters (~0.60), namun selisihnya kecil.
    - Kesimpulan: Meskipun performa sedikit lebih baik, peningkatan jumlah filter yang terlalu tinggi bisa berdampak pada risiko overfitting atau computational cost yang tidak sebanding dengan kenaikan performa.

Kesimpulan pengaruh banyak filter per layer konvolusi

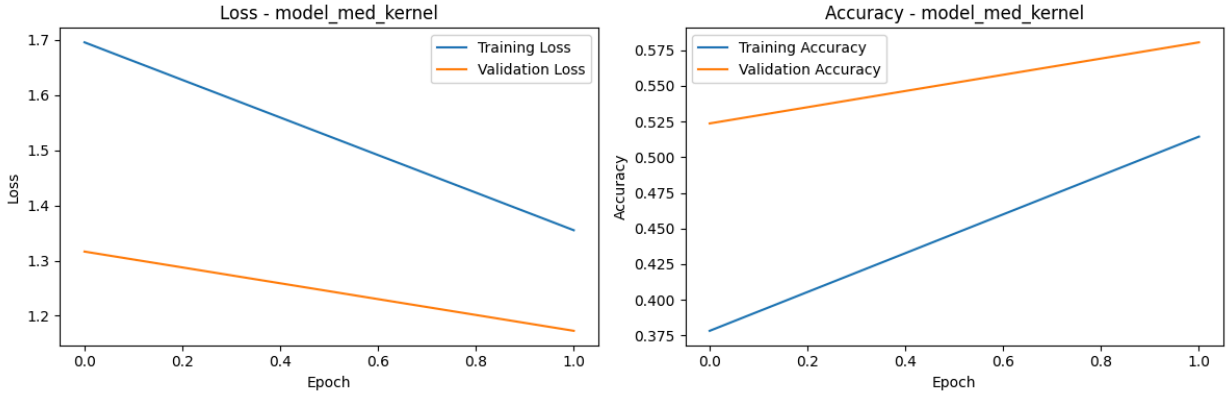
- Penambahan jumlah filter per layer konvolusi meningkatkan kemampuan model dalam mengekstraksi fitur yang kompleks, sehingga meningkatkan akurasi dan menurunkan loss.
- Namun, kenaikan performa dari medium ke high filters tidak terlalu signifikan, sehingga penggunaan jumlah filter sedang ([32, 64]) direkomendasikan sebagai trade-off terbaik antara kinerja dan efisiensi komputasi.

2.2.1.3. Pengaruh ukuran filter per layer konvolusi

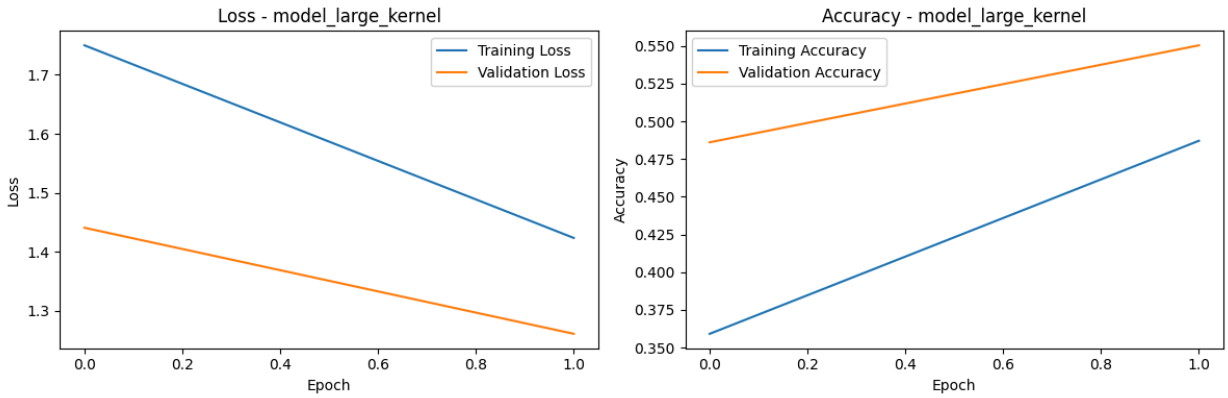


- Model dengan Kernel Kecil ([3, 3]):

- Loss: Penurunan training dan validation loss cukup signifikan.
- Accuracy: Validation accuracy tertinggi dari ketiganya, mendekati 0.62 pada epoch terakhir.
- Kesimpulan: Kernel kecil menangkap fitur lokal dengan baik dan mendorong generalisasi lebih tinggi. Ini menunjukkan kinerja terbaik untuk dataset ini.



- Model dengan Kernel Sedang ([5, 5]):
  - Loss: Menurun dengan cukup baik, tetapi tidak seefisien kernel kecil.
  - Accuracy: Validation accuracy juga meningkat, tetapi berada di bawah model dengan kernel kecil (~0.58).
  - Kesimpulan: Kernel sedang memberikan hasil yang lumayan baik, dengan trade-off antara local dan contextual features.

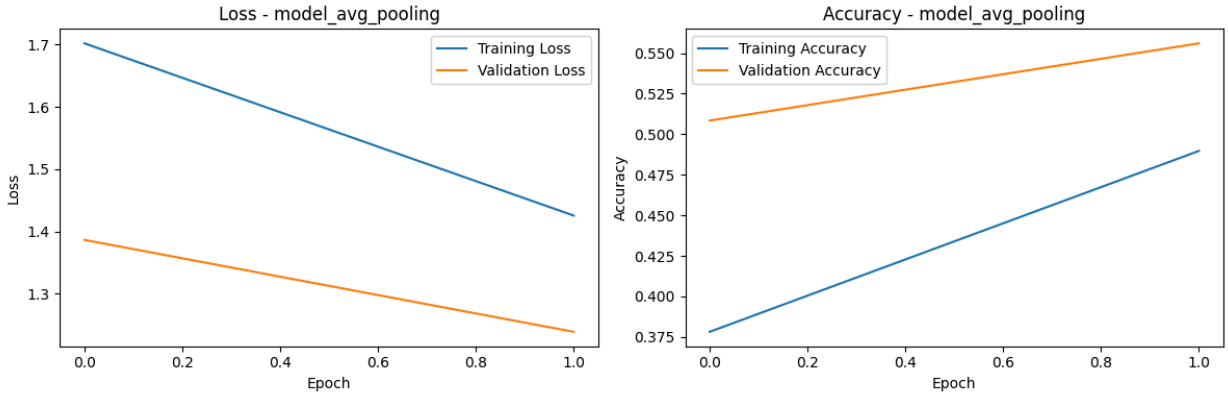


- Model dengan Kernel Besar ([7, 7]):
  - Loss: Penurunan loss tetap ada, tetapi tidak sebaik dua model sebelumnya.
  - Accuracy: Validation accuracy paling rendah (~0.54).
  - Kesimpulan: Kernel besar mungkin terlalu luas untuk menangkap fitur penting di awal lapisan CNN, menyebabkan model kurang efektif dalam generalisasi. Potensi over-smoothing fitur juga bisa terjadi.

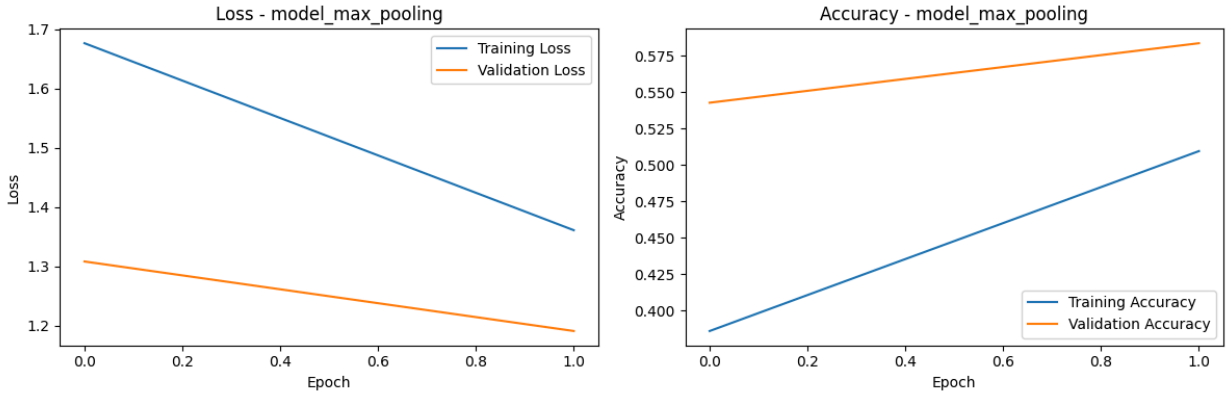
Kesimpulan pengaruh ukuran filter per layer konvolusi

- Ukuran kernel berpengaruh signifikan terhadap performa model CNN.
- Kernel kecil ([3,3]) memberikan hasil terbaik, karena mampu menangkap detail lokal dan mendorong akurasi validasi tertinggi.
- Semakin besar ukuran kernel, semakin lebar konteks yang ditangkap, namun risiko kehilangan detail kecil meningkat — yang dapat berdampak negatif pada akurasi, terutama di dataset dengan fitur halus.

2.2.1.4. Pengaruh jenis pooling layer



- Model dengan Average Pooling:
  - Loss: Penurunan training dan validation loss berlangsung stabil, tetapi tidak terlalu tajam.
  - Accuracy: Validation accuracy mencapai sekitar 0.55, dengan training accuracy lebih rendah (~0.49).
  - Kesimpulan: Average pooling memberikan hasil yang cukup baik, namun cenderung lebih halus dalam mengurangi dimensi dan mungkin kehilangan fitur penting karena rata-rata dari seluruh area.



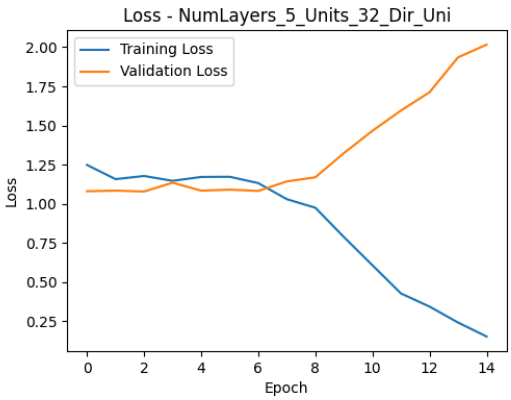
- Model dengan Max Pooling:
    - Loss: Penurunan loss lebih signifikan dibanding average pooling.
    - Accuracy: Validation accuracy lebih tinggi, mendekati 0.57, dan training accuracy juga lebih baik.
    - Kesimpulan: Max pooling menunjukkan performa yang lebih baik, karena lebih mampu mempertahankan fitur dominan dalam filter response, sehingga mendukung generalisasi model yang lebih kuat.

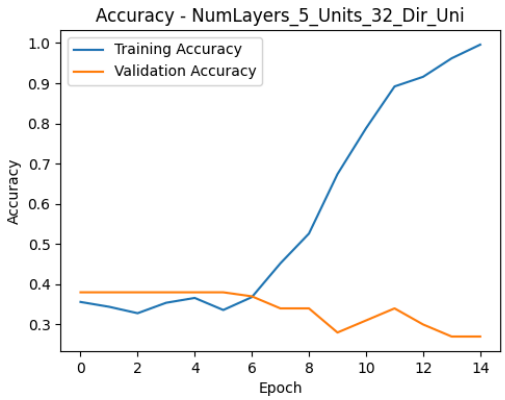
Kesimpulan pengaruh jenis pooling layer

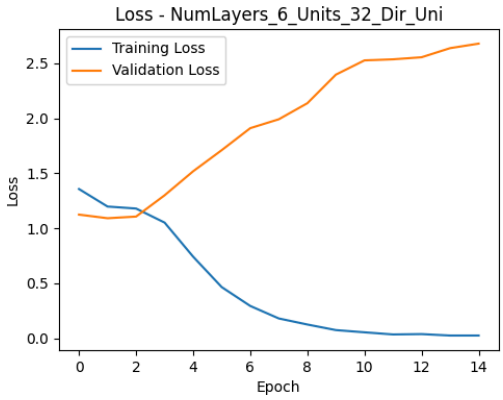
- Ukuran kernel berpengaruh signifikan terhadap performa model CNN.
- Kernel kecil ([3,3]) memberikan hasil terbaik, karena mampu menangkap detail lokal dan mendorong akurasi validasi tertinggi.
- Semakin besar ukuran kernel, semakin lebar konteks yang ditangkap, namun risiko kehilangan detail kecil meningkat — yang dapat berdampak negatif pada akurasi, terutama di dataset dengan fitur halus.

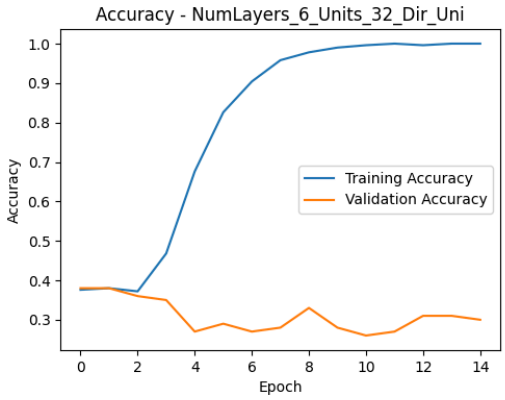
2.2.2. Simple RNN

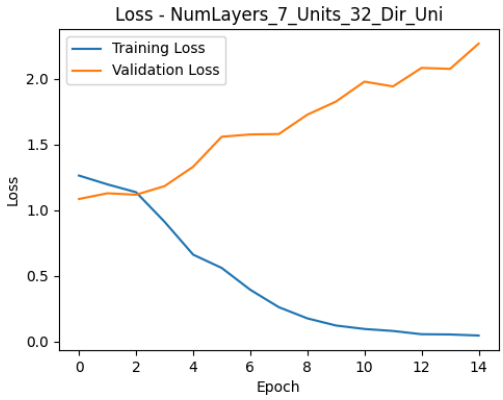
2.2.2.1. Pengaruh jumlah layer RNN

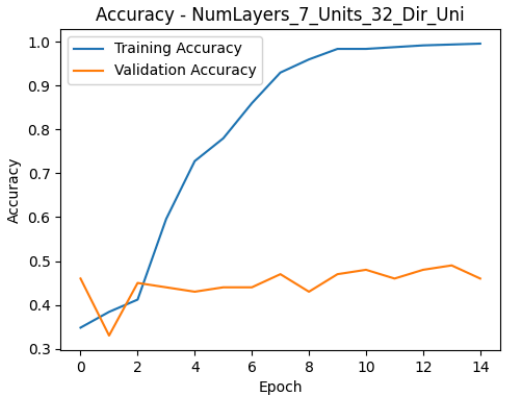












- Model dengan 5 Layer ():
  - *Validation loss* mulai menunjukkan tren kenaikan yang jelas sekitar epoch ke-7 atau ke-8.
  - *Validation accuracy* berfluktuasi dan tidak membaik secara signifikan, cenderung tetap rendah.
- Model dengan 6 Layer ():

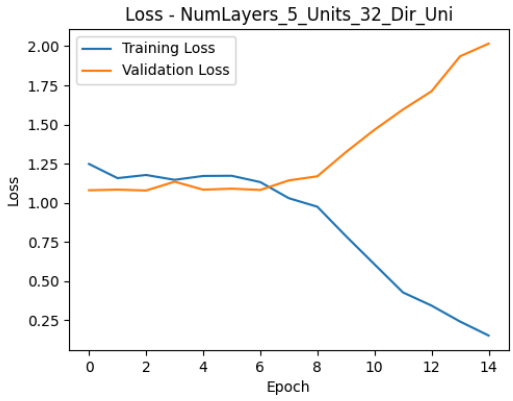
- *Validation loss* tampaknya mulai meningkat lebih awal atau setidaknya dengan lereng yang lebih curam dibandingkan model 5 *layer*, mungkin sekitar epoch ke-4 atau ke-5.
- *Validation accuracy* juga menunjukkan performa yang buruk dan tidak stabil.

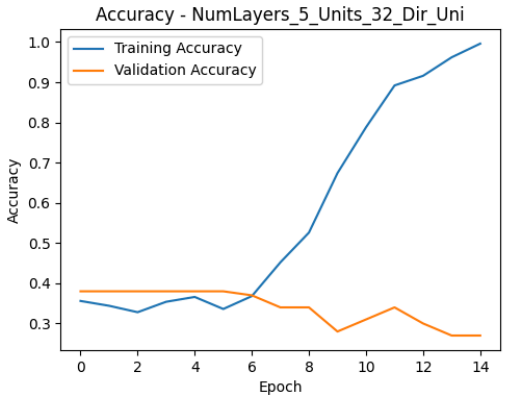
- Model dengan 7 Layer ():
  - Mirip dengan 6 *layer*, *validation loss* juga meningkat cukup dini, sekitar epoch ke-3 atau ke-4.
  - *Validation accuracy* juga menunjukkan volatilitas dan performa yang rendah.

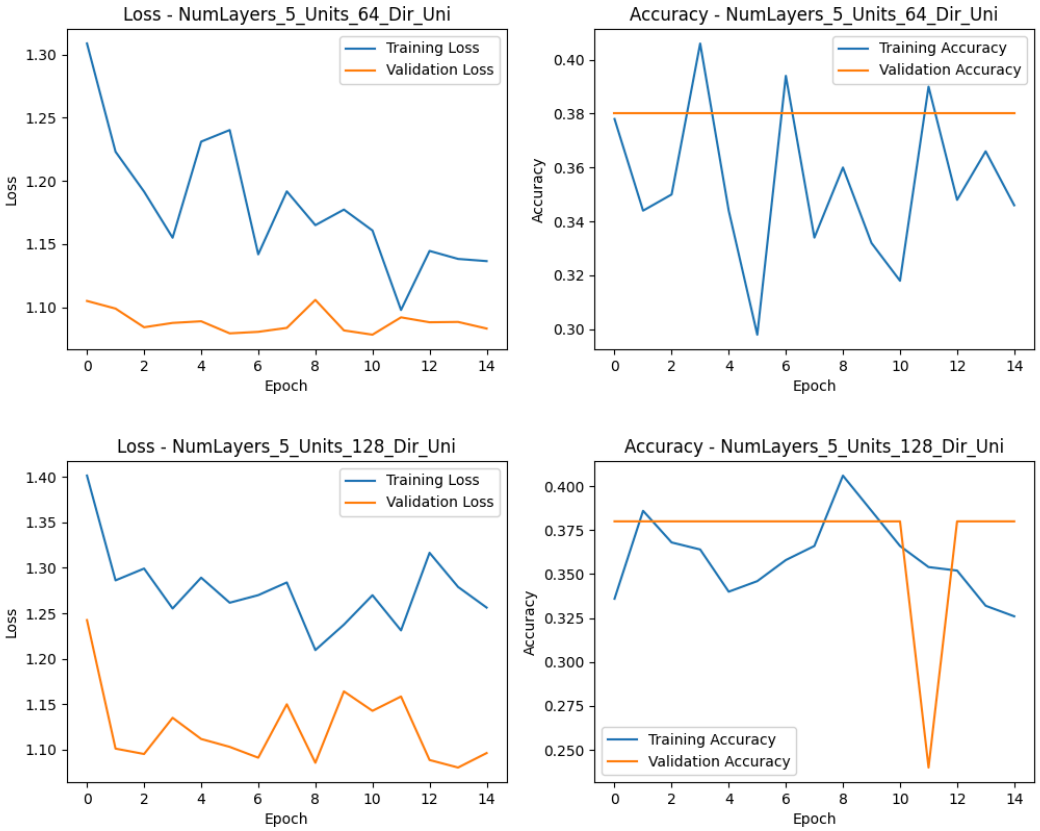
Kesimpulan Pengaruh Jumlah Layer:

1. Overfitting: Semua konfigurasi (5, 6, dan 7 *layer* RNN) menunjukkan tanda-tanda *overfitting*. Model menjadi terlalu kompleks untuk jumlah data yang tersedia atau arsitektur yang digunakan kurang optimal untuk generalisasi.
2. Penambahan Layer Cenderung Mempercepat/Memperparah Overfitting: Meskipun perbedaannya mungkin tidak drastis hanya dari visual, ada indikasi bahwa dengan bertambahnya jumlah *layer* (dari 5 ke 6, lalu ke 7), *validation loss* mulai meningkat lebih dini atau dengan lebih tajam, dan *validation accuracy* tetap rendah atau bahkan sedikit lebih buruk. Ini mengindikasikan bahwa menambah kedalaman model RNN dalam kasus ini tidak membantu meningkatkan performa pada data validasi, malah berpotensi membuat model lebih cepat *overfit*.

2.2.2.2. Pengaruh banyak cell RNN per layer





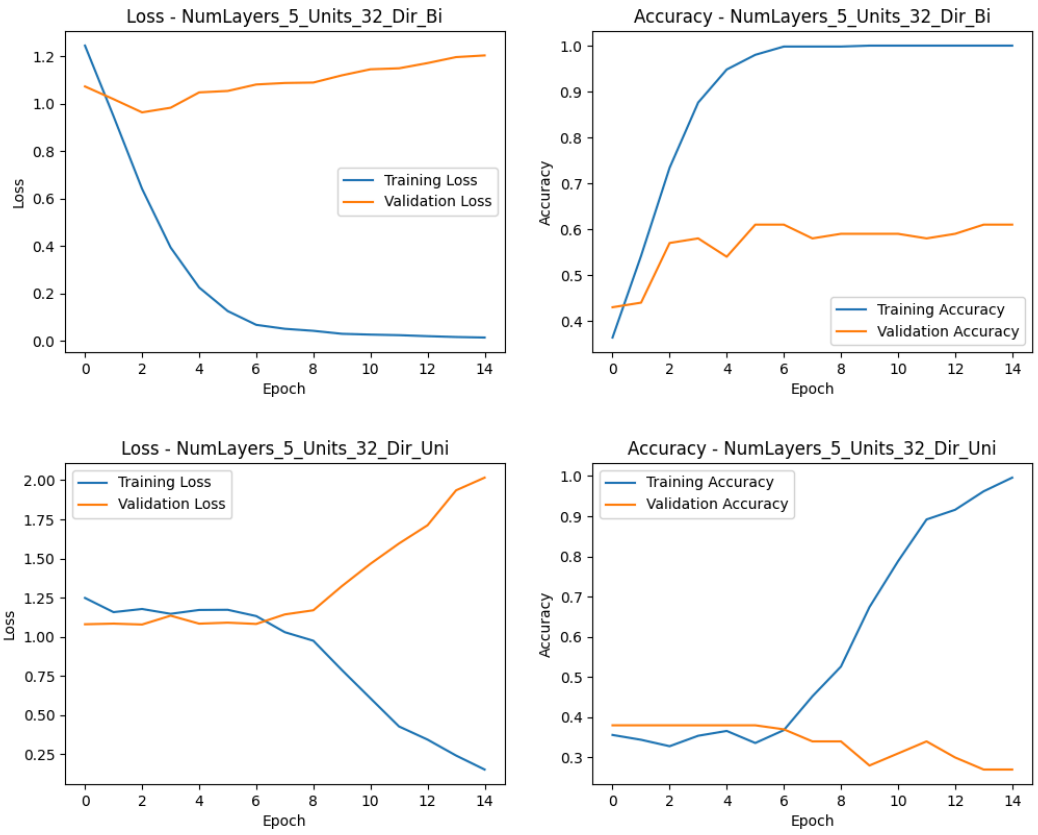


- 32 Unit/Cell: Model ini menunjukkan kemampuan belajar yang baik pada data training (loss turun, akurasi naik tinggi). Namun, ia mengalami overfitting yang jelas, di mana performa pada data validasi buruk dan memburuk seiring waktu. Ini mengindikasikan model terlalu kompleks atau kapasitasnya terlalu besar untuk data yang ada, sehingga menghafal data training.
- 64 Unit/Cell dan 128 Unit/Cell: Kedua konfigurasi ini menunjukkan masalah yang berbeda atau lebih parah.
    - Ketidakstabilan Pelatihan: Training loss dan training accuracy sangat berfluktuasi dan tidak menunjukkan konvergensi yang baik. Ini bisa disebabkan oleh beberapa faktor, seperti:
        - Vanishing/Exploding Gradients: Dengan lebih banyak unit dalam layer yang dalam (5 layer), masalah gradien bisa menjadi lebih signifikan, menyulitkan optimasi.
        - Optimasi yang Sulit: Ruang parameter yang lebih besar (karena lebih banyak unit) bisa jadi lebih sulit untuk dioptimalkan dengan optimizer dan learning rate default.
    - Performa Rendah Secara Umum: Baik training accuracy maupun validation accuracy tetap sangat rendah. Ini berarti model tidak hanya gagal menggeneralisasi (seperti pada kasus 32 unit yang overfit), tetapi juga gagal belajar secara efektif bahkan dari data training.

Kesimpulan Pengaruh banyak cell RNN per layer

1. Dalam analisis ini, dengan 5 layer RNN, menambah jumlah cell dari 32 menjadi 64 atau 128 tidak memberikan perbaikan.
2. Model dengan 32 unit/cell adalah yang paling "berhasil" dalam arti ia belajar dari data training, meskipun kemudian overfit.

2.2.2.3. Pengaruh jenis layer RNN berdasarkan arah



Kesimpulan Pengaruh Jenis Layer RNN Berdasarkan Arah

1. Overfitting Tetap Terjadi: Kedua jenis model (Unidirectional dan Bidirectional) dengan 5 layer dan 32 unit menunjukkan tanda-tanda overfitting. Training loss menurun dan training accuracy meningkat tinggi, sementara metrik validasi tidak mengikuti dan bahkan memburuk (terutama validation loss pada Unidirectional dan stagnasi pada Bidirectional).

2. Model Bidirectional Menunjukkan Performa Validasi Awal yang Lebih Baik:

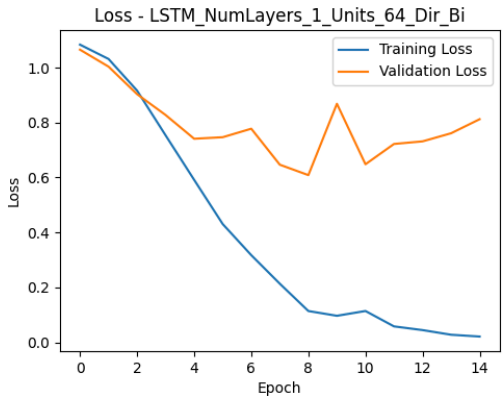
    a. Loss: Validation loss pada model Bidirectional sempat menurun dan mencapai titik terendah yang lebih baik sebelum mulai naik, dibandingkan model

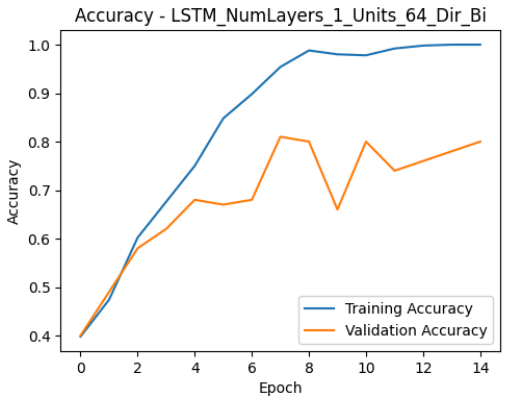
Unidirectional yang validation loss-nya lebih cepat divergen atau stagnan di nilai yang lebih tinggi.

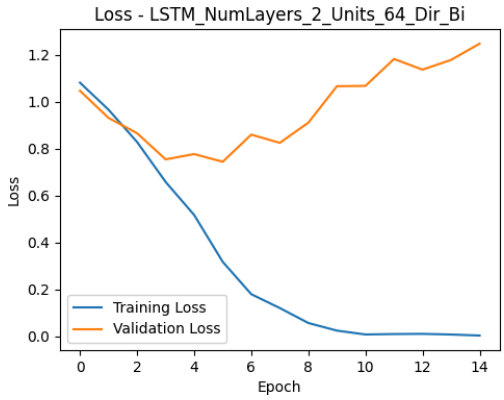
b. Accuracy: Validation accuracy pada model Bidirectional mencapai tingkat yang lebih tinggi (sekitar 0.55-0.60) dibandingkan model Unidirectional (sekitar 0.3-0.4). Ini menunjukkan bahwa kemampuan model Bidirectional untuk memproses informasi dari kedua arah dalam sekuens memberikan representasi yang lebih kaya dan berguna untuk klasifikasi pada data validasi, setidaknya sampai titik tertentu sebelum overfitting mengambil alih.

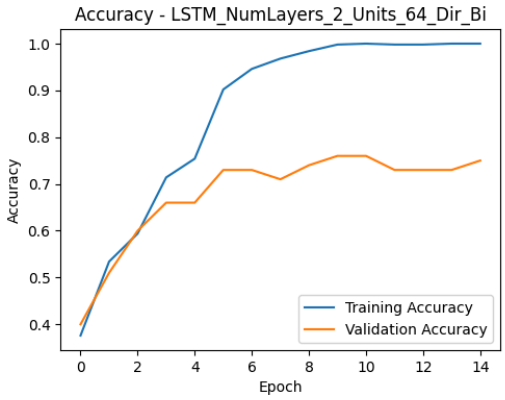
2.2.3. LSTM

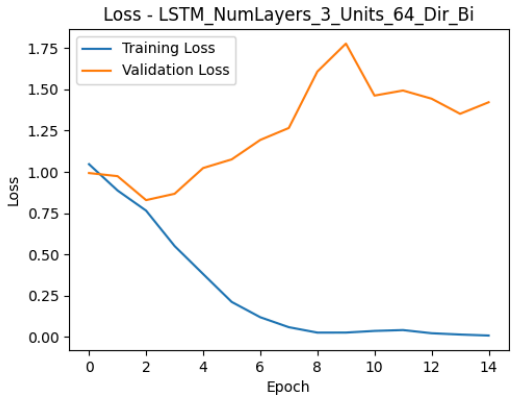
2.2.3.1. Pengaruh jumlah layer LSTM

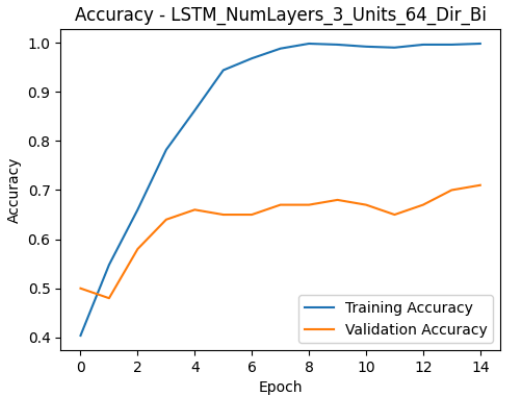










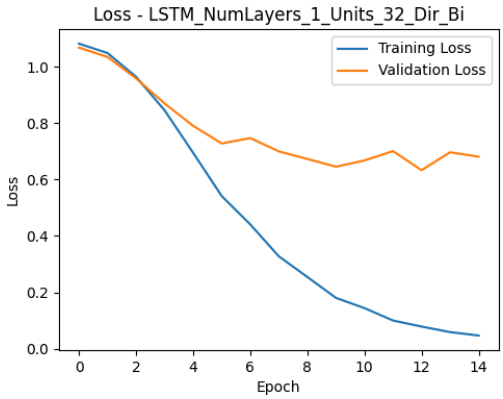


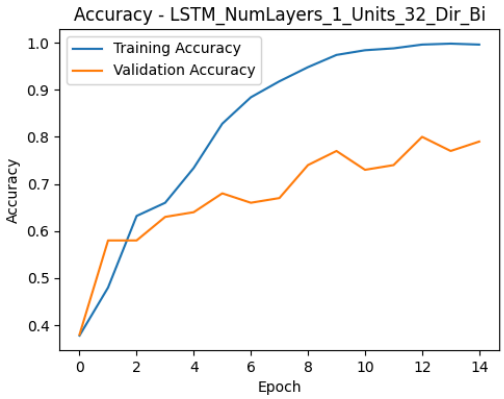
- Semua Model Mengalami Overfitting: Ketiga konfigurasi (1, 2, dan 3 layer BiLSTM) menunjukkan tanda-tanda overfitting yang jelas. Model dengan mudah mempelajari data training hingga mencapai akurasi dan loss yang sangat baik, namun gagal menggeneralisasi performa tersebut ke data validasi.

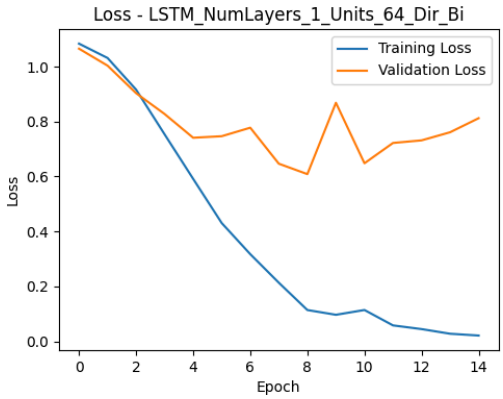
- Penambahan Layer Mempercepat dan Memperparah Overfitting:

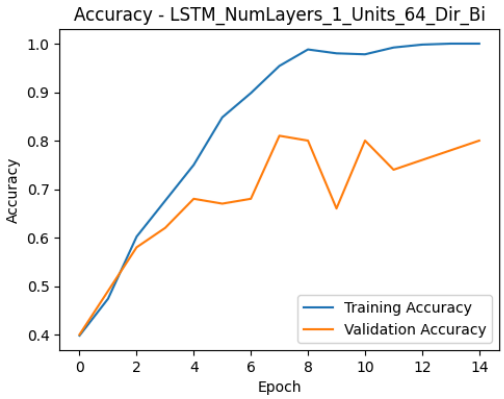
- Model dengan 1 layer BiLSTM menunjukkan overfitting yang paling lambat muncul dan mencapai performa validasi puncak (akurasi ~0.80) yang tertinggi di antara ketiganya sebelum overfitting mengambil alih.
- Model dengan 2 layer BiLSTM mulai overfit lebih awal (sekitar epoch ke-4) dan performa validasi puncaknya sedikit lebih rendah.
- Model dengan 3 layer BiLSTM menunjukkan overfitting yang paling parah dan paling cepat, dengan validation loss yang meningkat drastis sejak awal dan validation accuracy yang lebih rendah.

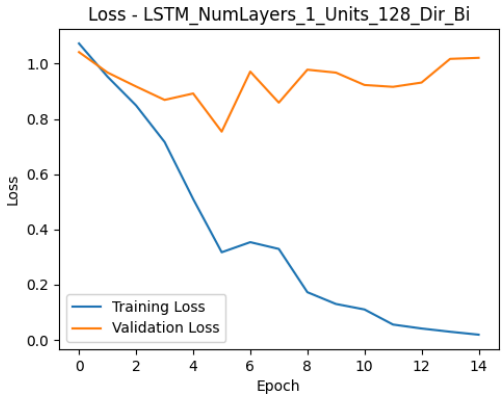
2.2.3.2. Pengaruh banyak cell LSTM per layer

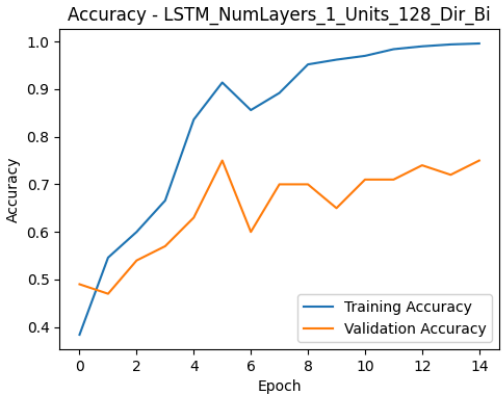










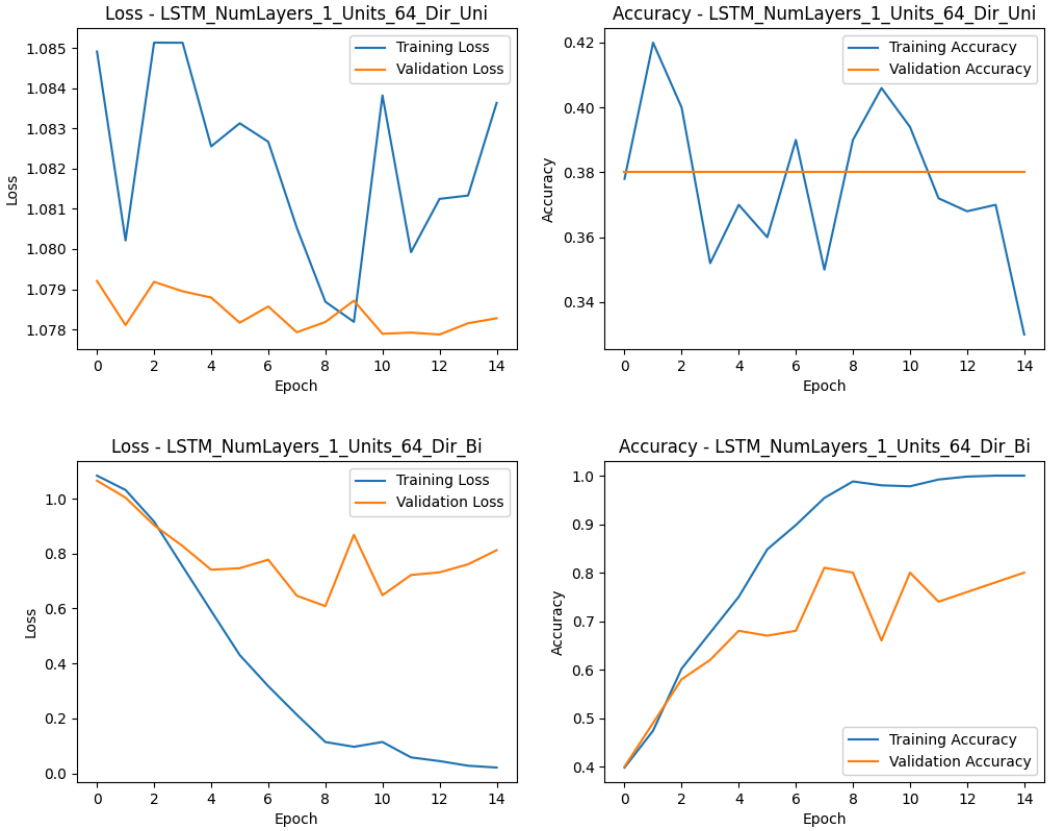


- Semua Model Mengalami Overfitting: Ketiga konfigurasi (32, 64, dan 128 unit) menunjukkan tanda-tanda overfitting. Model dengan mudah mempelajari data training, namun performa pada data validasi tidak sejalan dan cenderung memburuk atau stagnan setelah beberapa epoch awal.

- Peningkatan Jumlah Unit Cenderung Mempercepat Overfitting dengan Performa Validasi yang Tidak Selalu Lebih Baik:

    - Model dengan 32 unit menunjukkan overfitting yang relatif lebih lambat muncul, dengan validation accuracy yang mencapai level yang cukup baik (~0.75-0.80) dan cukup stabil sebelum fluktuasi meningkat.
    - Model dengan 64 unit juga mencapai validation accuracy puncak yang mirip (~0.80), namun dengan fluktuasi yang lebih terlihat pada validation loss dan accuracy-nya. Overfitting tampak dimulai sedikit lebih awal atau setidaknya kurva validasinya kurang mulus.
    - Model dengan 128 unit menunjukkan overfitting yang paling cepat. Meskipun training loss turun dengan cepat, validation loss mulai meningkat sangat dini, dan validation accuracy puncaknya sedikit lebih rendah dan kurang stabil dibandingkan konfigurasi dengan unit lebih sedikit.

2.2.3.3. Pengaruh jenis layer LSTM berdasarkan arah



- Model Bidirectional Jauh Lebih Unggul:
    - Kemampuan Belajar: Model Bidirectional LSTM (BiLSTM) menunjukkan kemampuan belajar yang jauh lebih baik pada data training dibandingkan model Unidirectional LSTM. Ini terlihat dari penurunan training loss yang signifikan dan peningkatan training accuracy yang tinggi pada model BiLSTM, sementara model Unidirectional gagal menunjukkan tren pembelajaran yang jelas.
    - Performa Validasi: Validation accuracy model BiLSTM mencapai tingkat yang jauh lebih tinggi (sekitar 0.80 pada puncaknya) dibandingkan model Unidirectional (yang stagnan di sekitar 0.38). Validation loss pada model BiLSTM juga sempat menurun sebelum akhirnya naik karena overfitting, sedangkan pada model Unidirectional, validation loss tidak menunjukkan perbaikan.
- Pentingnya Konteks Dua Arah: Perbedaan performa yang drastis ini sangat menyoroti pentingnya konteks dari kedua arah (masa lalu dan masa depan) untuk tugas pemrosesan sekuens pada dataset ini. Model BiLSTM mampu memanfaatkan informasi ini untuk membangun representasi yang lebih kaya dan bermakna, yang mengarah pada kemampuan klasifikasi yang lebih baik. Model Unidirectional, yang hanya melihat konteks masa lalu, tampaknya tidak mendapatkan informasi yang cukup untuk tugas ini.

2.2.3.4. Perbandingan model Tensorflow dengan implementasi

```
F1 Bawaan:    0.7203
              precision    recall  f1-score   support

           0     0.7122    0.6471    0.6781       153
           1     0.6915    0.6771    0.6842        96
           2     0.7605    0.8411    0.7987       151

    accuracy                         0.7275       400
   macro avg     0.7214    0.7217    0.7203       400
weighted avg     0.7255    0.7275    0.7251       400

==========
==========
==========

F1 Buatan:    0.7203
              precision    recall  f1-score   support

           0     0.7122    0.6471    0.6781       153
           1     0.6915    0.6771    0.6842        96
           2     0.7605    0.8411    0.7987       151

    accuracy                         0.7275       400
   macro avg     0.7214    0.7217    0.7203       400
weighted avg     0.7255    0.7275    0.7251       400
```

3. Kesimpulan dan Saran

Berdasarkan implementasi dan serangkaian pengujian yang telah dilakukan terhadap model Convolutional Neural Network (CNN) untuk klasifikasi gambar pada dataset CIFAR-10, serta model Simple Recurrent Neural Network (Simple RNN) dan Long-Short Term Memory (LSTM) untuk klasifikasi sentimen teks pada dataset NusaX (Bahasa Indonesia), dapat ditarik beberapa kesimpulan dan saran sebagai berikut:

3.1. Kesimpulan

3.1.1. Convolutional Neural Network (CNN)

- Pengaruh Jumlah Layer Konvolusi: Dari hasil pengujian, penambahan jumlah layer konvolusi (dari 1 hingga 3 layer) menunjukkan tren peningkatan performa, baik dari segi penurunan loss maupun peningkatan akurasi pada data training dan validasi. Model dengan 3 layer konvolusi cenderung memberikan hasil terbaik dibandingkan 1 atau 2 layer, mengindikasikan bahwa kedalaman tertentu membantu dalam ekstraksi fitur yang lebih kompleks dari gambar.
- Pengaruh Banyak Filter per Layer Konvolusi: Variasi jumlah filter menunjukkan bahwa penggunaan filter yang lebih banyak (misalnya, konfigurasi [64, 128] filter) dapat memberikan sedikit peningkatan performa dibandingkan jumlah filter yang lebih sedikit (misalnya, [16, 32] atau [32, 64]). Namun, perbedaan ini mungkin tidak selalu signifikan dan perlu dipertimbangkan dengan biaya komputasi yang meningkat.
- Pengaruh Ukuran Filter per Layer Konvolusi: Eksperimen dengan ukuran filter (kernel) yang berbeda (3x3, 5x5, 7x7) menunjukkan bahwa filter dengan ukuran yang lebih kecil atau sedang (misalnya, 3x3 atau 5x5) cenderung memberikan hasil yang lebih baik atau sebanding dengan filter yang lebih besar, dengan potensi efisiensi komputasi yang lebih tinggi. Filter yang terlalu besar mungkin tidak selalu optimal.
- Pengaruh Jenis Pooling Layer: Perbandingan antara Max Pooling dan Average Pooling menunjukkan bahwa Max Pooling secara umum memberikan hasil akurasi dan F1-score yang sedikit lebih baik daripada Average Pooling untuk arsitektur dan dataset yang diuji. Ini konsisten dengan praktik umum di mana Max Pooling lebih efektif dalam menangkap fitur yang paling dominan.
- Implementasi From Scratch: Model CNN from scratch yang dibangun berhasil memuat bobot dari model Keras dan menghasilkan prediksi yang identik (akurasi kesesuaian prediksi 1.0000 dan perbedaan F1-score 0.000000) pada subset data uji, menunjukkan validitas implementasi forward propagation secara manual.

3.1.2. Simple Recurrent Neural Network (Simple RNN)

- Pengaruh Jumlah Layer RNN: Penambahan jumlah layer RNN (dari 5, 6, hingga 7 layer) secara konsisten menunjukkan bahwa model mengalami overfitting yang signifikan. Semakin banyak layer, validation loss cenderung meningkat lebih cepat dan validation accuracy tetap rendah atau bahkan memburuk. Ini mengindikasikan bahwa menambah kedalaman model Simple RNN dalam kasus ini tidak membantu generalisasi dan justru mempercepat overfitting.
- Pengaruh Banyak Cell RNN per Layer: Untuk model dengan 5 layer RNN, penggunaan 32 unit/cell per layer menunjukkan kemampuan belajar pada data training yang baik namun kemudian overfit. Peningkatan jumlah unit menjadi 64 atau 128 justru menyebabkan ketidakstabilan pelatihan dan performa yang lebih buruk, baik pada data training maupun validasi. Model dengan 32 unit adalah yang paling "berhasil" dalam hal belajar dari data training di antara variasi ini.
- Pengaruh Jenis Layer RNN Berdasarkan Arah: Model Bidirectional RNN secara signifikan menunjukkan performa validasi awal yang lebih baik (akurasi lebih tinggi ~0.55-0.60) dibandingkan model Unidirectional RNN (akurasi ~0.3-0.4) dengan konfigurasi jumlah layer dan unit yang sama. Meskipun kedua model akhirnya overfit, kemampuan model Bidirectional untuk memproses informasi dari kedua arah sekuens memberikan representasi yang lebih kaya.

3.1.3. Long-Short Term Memory (LSTM)

- Pengaruh Jumlah Layer LSTM: Mirip dengan Simple RNN, penambahan jumlah layer BiLSTM (dari 1 ke 2, lalu ke 3 layer dengan 64 unit) juga menunjukkan bahwa semua model mengalami overfitting. Model dengan 1 layer BiLSTM menunjukkan overfitting yang paling lambat muncul dan mencapai performa validasi puncak tertinggi (~0.80 akurasi). Penambahan layer menjadi 2 atau 3 mempercepat dan memperparah overfitting, menghasilkan performa validasi yang lebih rendah.
- Pengaruh Banyak Cell LSTM per Layer: Untuk model dengan 1 layer BiLSTM, variasi jumlah unit (32, 64, 128) menunjukkan bahwa semua konfigurasi mengalami overfitting. Model dengan 32 dan 64 unit mencapai performa validasi yang sebanding (akurasi puncak ~0.80), meskipun model 64 unit menunjukkan fluktuasi yang lebih besar. Model dengan 128 unit mengalami overfitting yang lebih cepat dan performa validasi puncaknya sedikit lebih rendah.
- Pengaruh Jenis Layer LSTM Berdasarkan Arah: Model Bidirectional LSTM (BiLSTM) menunjukkan kemampuan belajar dan performa validasi yang jauh lebih unggul dibandingkan model Unidirectional LSTM dengan konfigurasi 1 layer dan 64 unit. Model

Unidirectional gagal belajar secara efektif, sementara model BiLSTM mampu menangkap pola yang lebih baik meskipun akhirnya juga overfit.

3.2. Saran

1. Penanganan Overfitting: Karena overfitting adalah masalah yang dominan pada sebagian besar eksperimen RNN dan LSTM, terutama pada model yang lebih dalam atau lebih lebar, teknik regularisasi yang lebih kuat atau strategi lain perlu dipertimbangkan. Ini bisa meliputi:
    - Early stopping berdasarkan performa validation loss.
    - Penyesuaian dropout rate yang lebih cermat.
    - Penggunaan regularisasi L1/L2 pada bobot layer.
    - Augmentasi data, terutama untuk dataset teks jika memungkinkan.
    - Menggunakan arsitektur model yang lebih sederhana (lebih sedikit layer atau unit) sebagai titik awal, seperti yang ditunjukkan oleh beberapa hasil eksperimen di mana model yang lebih sederhana berkinerja lebih baik pada data validasi.

2. Optimasi Hyperparameter Lebih Lanjut: Eksplorasi hyperparameter seperti learning rate optimizer, jenis optimizer, atau ukuran batch dapat dilakukan untuk menemukan konfigurasi yang lebih optimal, terutama untuk model yang menunjukkan ketidakstabilan pelatihan (seperti Simple RNN dengan banyak unit).
3. Pengembangan Model From Scratch: Untuk model RNN/LSTM from scratch, terutama SimpleRNNModel, perlu dipastikan bahwa penanganan return_sequences pada layer bertumpuk sudah sesuai dengan perilaku Keras untuk perbandingan yang adil saat memuat bobot dari model Keras dengan arsitektur bertumpuk. Kelas ManualLSTMModel saat ini spesifik untuk satu layer BiLSTM; pengembangannya untuk mendukung arsitektur LSTM bertumpuk atau unidirectional bisa menjadi langkah selanjutnya jika perbandingan from scratch untuk konfigurasi tersebut diperlukan.
4. Dataset dan Kompleksitas Model: Perlu adanya keseimbangan antara kompleksitas model dan ukuran serta kekayaan dataset. Jika dataset relatif kecil atau sederhana, model yang sangat kompleks (dalam atau lebar) akan cenderung overfit.
5. Analisis Lebih Mendalam untuk CNN: Meskipun tren umum terlihat, untuk CNN, pelatihan dengan jumlah epoch yang lebih banyak mungkin diperlukan untuk melihat

perbedaan yang lebih jelas antar konfigurasi hyperparameter, terutama terkait overfitting jika itu mulai muncul.

4. Pembagian Tugas

NIM	Nama	Tugas
13522130	Justin Aditya Putra P.	LSTM
13522155	Axel Santadi Warih	Simple RNN
13522163	Atqiya Haydar Luqman	CNN

5. Referensi

- Recurrent Neural Networks (RNNs), Clearly Explained!!!
- Keras Lecture 3: How to save training history and weights of your model
- Slide Perkuliahan IF3270 - Materi CNN