

LAPORAN TUGAS BESAR 2

IF3270 PEMBELAJARAN MESIN

Implementasi Convolutional Neural Network dan Recurrent Neural Network



Disusun oleh:

Ahmad Farid Mudrika	13522008
Ibrahim Ihsan Rasyid	13522018
Akbar Al Fattah	13522036

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2025

DAFTAR ISI

DAFTAR ISI	2
BAB I	
DESKRIPSI PERSOALAN	3
BAB II	
PEMBAHASAN	5
PENJELASAN IMPLEMENTASI	5
A. Implementasi CNN	5
1. Deskripsi kelas	5
2. Forward propagation	5
B. Implementasi RNN	5
1. Deskripsi Kelas	5
C. Implementasi LSTM	6
1. Deskripsi Kelas	6
2. Forward Propagation	10
HASIL PENGUJIAN	14
A. Hasil Pengujian CNN	14
a. Pengaruh jumlah layer konvolusi	14
b. Pengaruh banyak filter per layer konvolusi	14
c. Pengaruh ukuran filter per layer konvolusi	14
d. Pengaruh jenis pooling layer	14
B. Hasil Pengujian RNN	14
a. Pengaruh jumlah layer RNN	14
b. Pengaruh banyak cell RNN per layer	15
c. Pengaruh jenis layer RNN berdasarkan arah	15
C. Hasil Pengujian LSTM	15
a. Pengaruh jumlah layer LSTM	15
b. Pengaruh banyak cell LSTM per layer	15
c. Pengaruh jenis layer LSTM berdasarkan arah	15
BAB III	
KESIMPULAN DAN SARAN	16
KESIMPULAN	16
SARAN	16
BAB IV	
PEMBAGIAN TUGAS	17
BAB V	
REFERENSI	18

BAB I

DESKRIPSI PERSOALAN

Pada era modern saat ini, *machine learning* telah menjadi salah satu teknologi inti yang digunakan dalam berbagai bidang, termasuk pengenalan citra (*image recognition*) dan analisis teks (*text analysis*). Di antara berbagai pendekatan dalam *machine learning*, *neural network* merupakan salah satu metode yang paling banyak digunakan karena kemampuannya dalam mempelajari representasi data yang kompleks secara otomatis. Dalam konteks ini, dua jenis arsitektur yang sangat berpengaruh adalah *Convolutional Neural Network* (CNN) dan *Recurrent Neural Network* (RNN), termasuk salah satu variannya yang lebih kompleks, yaitu *Long Short-Term Memory* (LSTM).

Tugas Besar ini bertujuan untuk memberikan pemahaman yang mendalam kepada mahasiswa mengenai mekanisme kerja dari CNN dan RNN, khususnya pada tahap *forward propagation*. Mahasiswa diminta untuk melakukan implementasi *forward propagation* dari masing-masing arsitektur tersebut *from scratch*, sehingga dapat memahami perhitungan dasar yang terjadi di setiap lapisan jaringan.

Selain itu, untuk memberikan konteks dan aplikasi nyata dari arsitektur yang diimplementasikan, mahasiswa juga diminta untuk melakukan pelatihan model menggunakan *library* Keras. Model CNN akan digunakan untuk menyelesaikan permasalahan klasifikasi citra pada dataset CIFAR-10, sedangkan model RNN dan LSTM akan digunakan untuk melakukan klasifikasi sentimen teks berbahasa Indonesia menggunakan dataset NusaX-Sentiment.

Adapun persoalan utama yang ingin diselesaikan dalam tugas ini meliputi:

1. **Klasifikasi Citra** menggunakan arsitektur **CNN**:

- Mengidentifikasi objek dalam gambar dari dataset CIFAR-10.
- Mengimplementasikan *forward propagation* CNN dari nol.
- Melatih model CNN menggunakan Keras dengan konfigurasi tertentu (Conv2D, Pooling, Flatten, Dense).

2. **Klasifikasi Teks** menggunakan arsitektur **RNN**:

- Mengklasifikasikan sentimen dari kalimat dalam Bahasa Indonesia (positif, netral, negatif).
- Melakukan *preprocessing* teks berupa tokenisasi dan embedding.
- Mengimplementasikan *forward propagation* untuk RNN sederhana dari nol.

3. **Klasifikasi Teks** menggunakan arsitektur **LSTM**:

- Menyelesaikan tugas klasifikasi sentimen dengan arsitektur LSTM.
- Mengimplementasikan *forward propagation* untuk LSTM dari nol.

- Memanfaatkan *Embedding layer*, *Bidirectional* atau *Unidirectional LSTM*, *Dropout*, dan *Dense layer*.

Dengan menyelesaikan tugas ini, diharapkan mahasiswa tidak hanya memahami penggunaan model CNN dan RNN secara praktis, tetapi juga mampu membongkar dan memahami komponen internal dari jaringan saraf tersebut, yang merupakan keterampilan penting dalam pengembangan model-model *deep learning* yang lebih kompleks di masa depan.

BAB II

PEMBAHASAN

PENJELASAN IMPLEMENTASI

A. Implementasi CNN

1. Deskripsi kelas

Ada 4 kelas pada implementasi CNN dari scratch, yaitu

a. Conv2DLayer yang merupakan layer konvolusi. Atributnya terdiri atas weights, bias, stride, padding, dan method forward yang merupakan implementasi forward propagation dari Convolutional Layer

b. PoolingLayer (Average atau Max) yang merupakan layer pooling.

c. FlattenLayer yang merupakan layer yang diflatten menjadi array.

d. DenseLayer yang merupakan dense layer

e. CNNScratch, yang merupakan kelas untuk melakukan forward propagation CNN

2. Forward propagation

Forward propagation pada CNN terdiri atas 4 tahap, yaitu tahap konvolusi, tahap pooling, tahap flattening, dan tahap Dense layer.

Pertama, gambar dikonvolusikan menggunakan kernel. setelah tahap konvolusi selesai, gambar masuk ke tahap pooling, yang akan membuat feature map yang lebih sederhana. Setelah itu, masuk ke tahap flattening yang akan membuat setiap hasil konvolusi menjadi layer yang bisa ditentukan labelnya, dan terakhir masuk ke tahap dense

B. Implementasi RNN

1. Deskripsi Kelas

Kelas SimpleRNN mengimplementasikan arsitektur dan metode *forward propagation Recurrent Neural Network* (RNN) sederhana menggunakan numpy. Kelas ini dapat mewakili rnn dengan n rnn layer, dengan arsitektur *unidirectional* atau *bidirectional*.

Atribut kelas SimpleRNN:

- embedding_weights (numpy.ndarray): matriks embedding berukuran (vocab_size, embedding_dim) yang menyimpan representasi vektor setiap token.
- rnn_weights(list[dict[str, dict[str, numpy.ndarray]]]): Daftar layer RNN. Setiap elemen berisi dictionary {'forward': ..., 'backward': ...} (jika bidirectional), dengan isi bobot input_weights, recurrent_weights, dan bias.
- dense_weights (numpy.ndarray): Matriks bobot untuk dense layer (fully connected).
- dense_bias (numpy.ndarray): Vektor bias untuk dense layer, berukuran (num_classes,).
- vocab_size (int): Jumlah token unik dalam vocabulary.

- `embedding_dim` (int): Dimensi vektor embedding untuk setiap token.
- `rnn_units` (int): Jumlah neuron di layer RNN.
- `num_classes` (int): Jumlah kelas keluaran dari model.
- `bidirectional` (bool): Menandakan apakah model menggunakan RNN dua arah.
- `num_layers` (int): Jumlah layer rnn yang digunakan dalam model.

Metode kelas SimpleRNN:

- `__init__` (`num_layers`: int = 1, `bidirectional`: bool = False) : Konstruktor untuk inisialisasi atribut model. Return None.
- `load_keras_weights` (`keras_model`) : Memuat bobot dari model Keras ke atribut internal seperti `embedding`, `RNN`, dan `dense` layer. Return None.
- `embedding_forward` (`input_ids`: np.ndarray) : Melakukan lookup vektor embedding berdasarkan indeks token. Return np.ndarray dengan shape (`batch_size`, `seq_len`, `embedding_dim`).
- `single_rnn_pass` (`x`: np.ndarray, `W_in`: np.ndarray, `W_rec`: np.ndarray, `bias`: np.ndarray, `reverse`: bool = False) : Menjalankan forward pass untuk satu arah (forward/backward) dari satu layer RNN. Return (np.ndarray, np.ndarray) berupa output sequence dan hidden state terakhir.
- `rnn_forward` (`embedded_input`: np.ndarray) : Menjalankan semua layer RNN (dengan dukungan `bidirectional` jika diaktifkan), dan mengembalikan hidden state terakhir. Return np.ndarray dengan shape (`batch_size`, `rnn_units`) atau (`batch_size`, `2*rnn_units`) jika `bidirectional`.
- `dense_forward` (`rnn_output`: np.ndarray) : Menghasilkan distribusi probabilitas kelas dengan menerapkan `dense` layer dan `softmax` ke output RNN. Return np.ndarray dengan shape (`batch_size`, `num_classes`).
- `forward` (`input_ids`: np.ndarray) : Menjalankan proses inferensi lengkap: `embedding` → `RNN` → `dense` → `softmax`. Return np.ndarray dengan probabilitas untuk setiap kelas.
- `predict` (`input_ids`: np.ndarray) : Mengembalikan prediksi label (kelas dengan probabilitas tertinggi) berdasarkan input. Return np.ndarray bertipe int dengan shape (`batch_size`,).

2. Forward Propagation

Forward propagation pada kelas SimpleRNN terdiri dari tiga tahap: `embedding`, `RNN`, dan `dense`. Pertama, input berupa token ID masuk ke tahap `embedding`, yang dikonversi menjadi vektor berdimensi `embedding_dim`. Kemudian, hasil `embedding` ini diteruskan ke tahap `RNN`. Tahap `RNN` memproses data secara berurutan. Untuk setiap `timestep`, hidden state diperbarui menggunakan input saat ini dan hidden state sebelumnya, menggunakan bobot input (`W_in`), bobot rekuren (`W_rec`), dan bias. Jika model bersifat `bidirectional`, maka proses ini dilakukan dua kali: maju dan mundur, lalu output dari keduanya digabungkan.

Setelah seluruh `timestep` diproses, output akhir RNN (yaitu hidden state terakhir) diteruskan ke tahap `dense`, yang mengubahnya menjadi skor (logits) untuk setiap kelas menggunakan bobot dan bias layer `dense`. Skor ini kemudian dinormalisasi menggunakan fungsi `softmax` agar menjadi distribusi probabilitas. Hasil akhir adalah probabilitas untuk masing-masing kelas yang dapat digunakan untuk klasifikasi. Jika dipanggil melalui method `predict`, kelas dengan probabilitas tertinggi akan dipilih sebagai output.

C. Implementasi LSTM

1. Deskripsi Kelas

- a. Kelas `EmbeddingLayer` – merupakan representasi dari sebuah embedding layer sederhana yang berfungsi untuk memetakan token-token dalam bentuk indeks integer ke dalam representasi vektor berdimensi tetap.

```
class EmbeddingLayer:
    def __init__(self):
        self.weights = None
```

Atribut Kelas `EmbeddingLayer`:

- `weights` (`numpy.ndarray`): matriks embedding berukuran (`vocab_size`, `embedding_dim`) yang menyimpan representasi vektor setiap token

Method Kelas `EmbeddingLayer`:

- `load_weights(self, weights)`: memuat matriks bobot embedding dari model Keras
- `forward(self, input_ids)`: method forward pass

- b. Kelas `LSTMCell` – merepresentasikan sel dasar dari jaringan LSTM untuk memproses data sekuensial secara bertahap

```
class LSTMCell:
    def __init__(self):
        self.Wf = None # Forget gate weights
        self.Wi = None # Input gate weights
        self.Wc = None # Cell state weights
        self.Wo = None # Output gate weights

        self.Uf = None # Forget gate recurrent weights
        self.Ui = None # Input gate recurrent weights
        self.Uc = None # Cell state recurrent weights
        self.Uo = None # Output gate recurrent weights

        self.bf = None # Forget gate bias
        self.bi = None # Input gate bias
        self.bc = None # Cell state bias
        self.bo = None # Output gate bias
```

Atribut kelas `LSTMCell`:

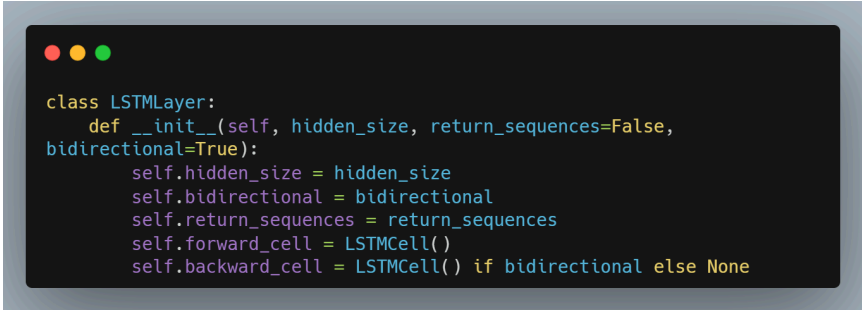
- `Wf, Wi, Wc, Wo` (`numpy.ndarray`): matriks bobot input masing-masing gerbang

- U_f, U_i, U_c, U_o (numpy.ndarray): matriks bobot rekuren masing-masing gerbang
- b_f, b_i, b_c, b_o (numpy.ndarray): vektor bias masing-masing gerbang

Method kelas LSTMCell:

- `load_weights(self, kernel, recurrent_kernel, bias)`: memuat bobot dan bias ke dalam sel. Parameter `kernel` adalah matriks bobot input gabungan, `recurrent_kernel` adalah bobot rekuren gabungan, dan `bias` adalah vektor bias gabungan
- `sigmoid(self, x)`: perhitungan fungsi aktivasi sigmoid
- `tanh(self, x)`: perhitungan fungsi aktivasi tanh
- `forward(self, x_t, h_prev, c_prev)`: melakukan forward pass

- c. Kelas `LSTMLayer` – merepresentasikan satu layer LSTM yang mampu menangani seluruh sekuens input. Layer dibangun dengan menggabungkan satu atau dua buah `LSTMCell` tergantung pada konfigurasi arah



```
class LSTMLayer:
    def __init__(self, hidden_size, return_sequences=False,
                 bidirectional=True):
        self.hidden_size = hidden_size
        self.bidirectional = bidirectional
        self.return_sequences = return_sequences
        self.forward_cell = LSTMCell()
        self.backward_cell = LSTMCell() if bidirectional else None
```

Atribut kelas `LSTMLayer`:

- `hidden_size` (int): ukuran hidden state yang dihasilkan setiap sel
- `bidirectional` (bool): menentukan apakah lapisan `bidirectional` (maju dan mundur) atau `unidirectional` (hanya maju)
- `return_sequences` (bool): menentukan apakah output akan dikembalikan dalam bentuk seluruh urutan keluaran per waktu (jika `True`) atau hanya keluaran pada waktu terakhir (jika `False`)
- `forward_cell` (`LSTMCell`): sel LSTM yang memproses input dari awal ke akhir
- `backward_cell` (`LSTMCell`) sel LSTM yang memproses input dari akhir ke awal jika `bidirectional` bernilai `True`

Method kelas `LSTMLayer`:

- `load_weights(self, forward_weights, backward_weights=None)`: memuat bobot ke masing-masing sel LSTM. Masing-masing argumen berisi tuple dari (`kernel`, `recurrent_kernel`, `bias`)

- `forward(self, x, mask=None)`: melakukan forward pass

d. Kelas `DropoutLayer` – merepresentasikan layer dropout yang digunakan sebagai teknik regularisasi dalam neural network

```
class DropoutLayer:
    def __init__(self, rate=0.5):
        self.rate = rate
```

Atribut kelas `DropoutLayer`:

- `rate (float)`: rasio neuron yang akan dinonaktifkan secara acak selama pelatihan

Method kelas `DropoutLayer`:

- `forward(self, x, training=False)`: melakukan forward pass

e. Kelas `DenseLayer` – merepresentasikan fully connected layer dalam neural network. Layer ini bertugas melakukan transformasi linear terhadap input dan menerapkan fungsi aktivasi

```
class DenseLayer:
    def __init__(self, activation=None):
        self.activation = activation
        self.weights = None
        self.bias = None
```

Atribut kelas `DenseLayer`:

- `activation (str)`: fungsi aktivasi yang digunakan. Jika `None`, tidak menerapkan fungsi aktivasi
- `weights (numpy.ndarray)`: matriks bobot berukuran `(input_size, output_size)`
- `bias (numpy.ndarray)`: vektor bias berukuran `(output_size,)`

Method kelas `DenseLayer`:

- `load_weights(self, weights, bias)`: memuat bobot `weights` dan bias ke dalam layer

- `relu(self, x)`: fungsi aktivasi ReLU
 - `softmax(self, x)`: fungsi aktivasi Softmax
 - `forward(self, x)`: melakukan forward pass
- f. Kelas `LSTMModel` – merupakan model neural network yang menggabungkan kelas-kelas layer embedding, LSTM berlapis, dropout, dan dense yang telah dideskripsikan sebelumnya

```
class LSTMModel:
    def __init__(self, num_units=64, num_layers=1, bidirectional=False):
        self.num_units = num_units
        self.num_layers = num_layers
        self.bidirectional = bidirectional
        self.embedding_layer = EmbeddingLayer()
        self.lstm_layers = [
            LSTMLayer(
                hidden_size=num_units,
                return_sequences=(i < num_layers - 1),
                bidirectional=bidirectional)
            for i in range(num_layers)
        ]
        self.dropout_layer = DropoutLayer(rate=0.5)
        self.dense_layer = DenseLayer(
            activation='softmax'
        )
```

Atribut kelas `LSTMModel`:

- `num_units (int)`: jumlah hidden unit pada setiap sel LSTM
- `num_layers (int)`: jumlah lapisan LSTM yang digunakan dalam model
- `bidirectional (bool)`: menentukan apakah tiap layer LSTM akan bersifat bidirectional (jika True) atau tidak (jika False)
- `embedding_layer (EmbeddingLayer)`: layer embedding pada model
- `lstm_layers (List[LSTMLayer])`: larik berisikan tiap layer LSTM secara berurutan
- `dropout_layer (DropoutLayer)`: layer dropout pada model
- `dense_layer (DenseLayer)`: layer dense pada model

Method kelas `LSTMModel`:

- `load_keras_weights(self, keras_model)`: memuat bobot model dari model Keras yang sudah dilatih sebelumnya
- `predict(self, input_ids, training=False)`: melakukan prediksi atau forward pass pada model. Jika training bernilai True, dropout diterapkan setelah setiap layer LSTM

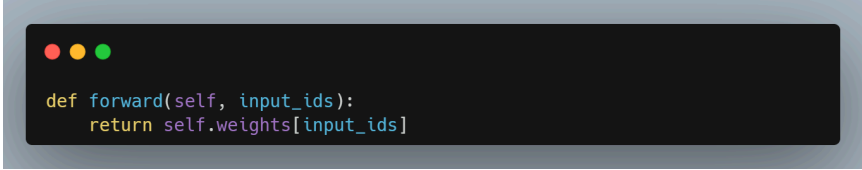
2. Forward Propagation

Pada model ini, forward propagation melibatkan beberapa tahap utama, yaitu: *embedding*, *LSTM* bertingkat, *dropout* (opsional), dan *dense layer* dengan *softmax*. Proses ini dilakukan menggunakan skema batch matrix processing

Forward propagation model LSTM ini mengikuti alur transformasi sebagai berikut:

a. Embedding Layer

Token input dalam bentuk integer (ID kata) diubah menjadi vektor embedding berdimensi tetap



```
def forward(self, input_ids):  
    return self.weights[input_ids]
```

b. LSTM Layer(s)

Proses internal sel LSTM terdiri dari empat komponen utama: tiga gerbang (*gate*) dan satu memori sel. Berikut adalah rumus dari keempat komponen tersebut:

Input activation:

$$a_t = \tanh(W_a \cdot x_t + U_a \cdot out_{t-1} + b_a)$$

Input gate:

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot out_{t-1} + b_i)$$

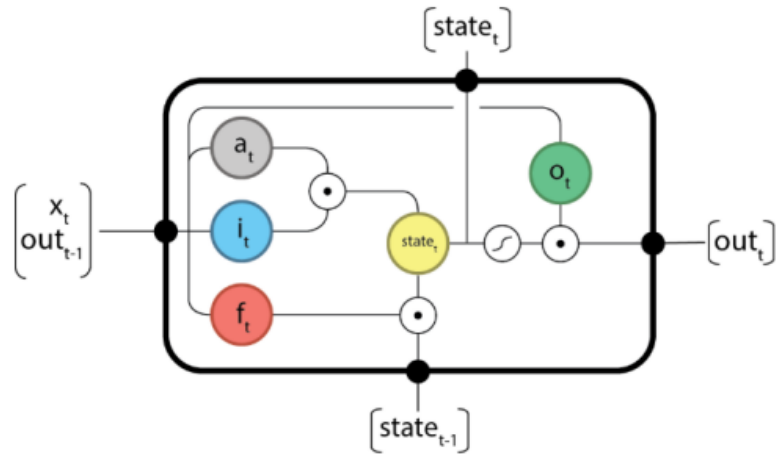
Forget gate:

$$f_t = \sigma(W_f \cdot x_t + U_f \cdot out_{t-1} + b_f)$$

Output gate:

$$o_t = \sigma(W_o \cdot x_t + U_o \cdot out_{t-1} + b_o)$$

Berikut adalah visualisasi aliran data pada satu sel LSTM



```
def forward(self, x_t, h_prev, c_prev):
    # Forget gate
    f = self.sigmoid(np.dot(x_t, self.Wf) + np.dot(h_prev, self.Uf) +
self.bf)

    # Input gate
    i = self.sigmoid(np.dot(x_t, self.Wi) + np.dot(h_prev, self.Ui) +
self.bi)

    # Cell state
    c_hat = self.tanh(np.dot(x_t, self.Wc) + np.dot(h_prev, self.Uc) +
self.bc)
    c = f * c_prev + i * c_hat

    # Output gate
    o = self.sigmoid(np.dot(x_t, self.Wo) + np.dot(h_prev, self.Uo) +
self.bo)

    # Hidden state
    h = o * self.tanh(c)

    return h, c
```

Selanjutnya, layer LSTM akan memproses urutan input seiring waktu, dengan cara menerapkan sel LSTM secara berulang pada setiap timestep. Setiap lapisan LSTM dapat bersifat *unidirectional* (searah) atau *bidirectional* (dua arah). Bila bersifat *bidirectional*, maka akan ada dua jalur propagasi informasi: forward yaitu dari timestep pertama hingga terakhir, dan backward yaitu dari timestep terakhir ke timestep pertama. Setelah itu, apabila layer tersebut *bidirectional*, output dari kedua arah tersebut akan digabungkan. Proses ini berlangsung untuk setiap lapisan dalam daftar layer pada model, dan output dari satu layer menjadi input ke layer berikutnya.

Pada `LSTMLayer`, jika `return_sequences` bernilai `True`, maka seluruh urutan output (hidden state) dikembalikan. Jika `return_sequences` bernilai `False`, maka hanya hidden state terakhir yang diteruskan.

```

def forward(self, x, mask=None):
    batch_size, seq_len, input_size = x.shape

    h_forward = np.zeros((batch_size, self.hidden_size))
    c_forward = np.zeros((batch_size, self.hidden_size))
    forward_output = []

    for t in range(seq_len):
        x_t = x[:, t, :]
        h_forward, c_forward = self.forward_cell.forward(x_t, h_forward, c_forward)
        forward_output.append(h_forward)

    if self.bidirectional:
        h_backward = np.zeros((batch_size, self.hidden_size))
        c_backward = np.zeros((batch_size, self.hidden_size))
        backward_output = []

        for t in reversed(range(seq_len)):
            x_t = x[:, t, :]
            h_backward, c_backward = self.backward_cell.forward(x_t, h_backward,
c_backward)
            backward_output.append(h_backward)

        backward_output.reverse()

    if self.return_sequences:
        if self.bidirectional:
            forward_stack = np.stack(forward_output, axis=1) # (batch_size, seq_len,
hidden_size)
            backward_stack = np.stack(backward_output, axis=1)
            return np.concatenate([forward_stack, backward_stack], axis=-1)
        else:
            return np.stack(forward_output, axis=1) # (batch_size, seq_len,
hidden_size)
    else:
        if self.bidirectional:
            return np.concatenate([h_forward, h_backward], axis=-1)
        else:
            return h_forward

```

c. Dropout Layer (saat training)

Dropout digunakan untuk mencegah overfitting dengan menghilangkan beberapa neuron secara acak pada saat training.

```

def forward(self, x, training=False):
    if training or self.rate == 0:
        return x
    mask = np.random.binomial(1, 1 - self.rate, size=x.shape)
    return x * mask / (1 - self.rate)

```

d. Dense Layer

Output dari LSTM terakhir (baik hidden state terakhir atau seluruh sequence) akan diteruskan ke dense layer untuk menghasilkan skor untuk setiap token dalam kosakata. Fungsi aktivasi softmax akan mengubah skor ini menjadi probabilitas.

```
def forward(self, x):  
    output = np.dot(x, self.weights) + self.bias  
    if self.activation == 'relu':  
        return self.relu(output)  
    elif self.activation == 'softmax':  
        return self.softmax(output)  
    else:  
        return output
```

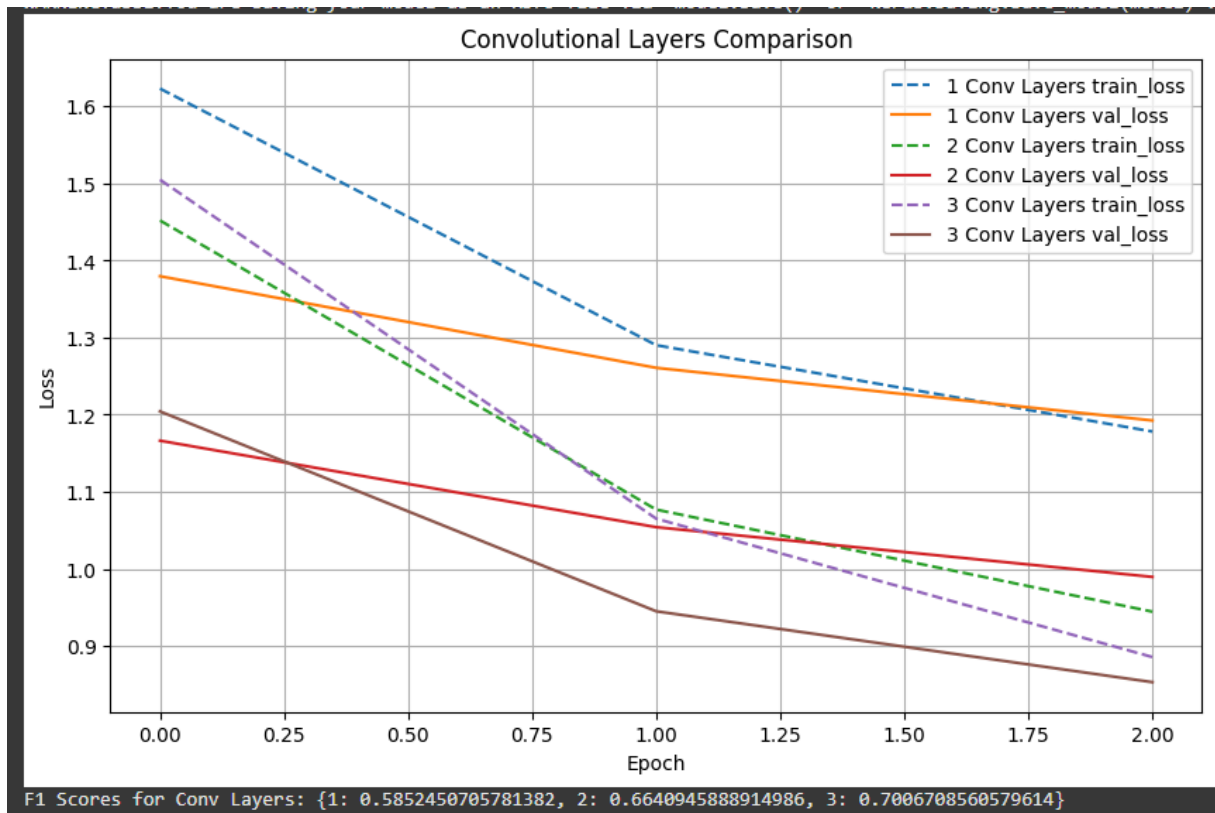
Seluruh method forward propagation dari kelas-kelas tersebut dikompilasi menjadi sebuah fungsi untuk melakukan prediksi pada kelas LSTMModel

```
def predict(self, input_ids, training=False):  
    x = self.embedding_layer.forward(input_ids)  
  
    for lstm_layer in self.lstm_layers:  
        x = lstm_layer.forward(x)  
        if training:  
            x = self.dropout_layer.forward(x, training)  
  
    output = self.dense_layer.forward(x)  
    return output
```

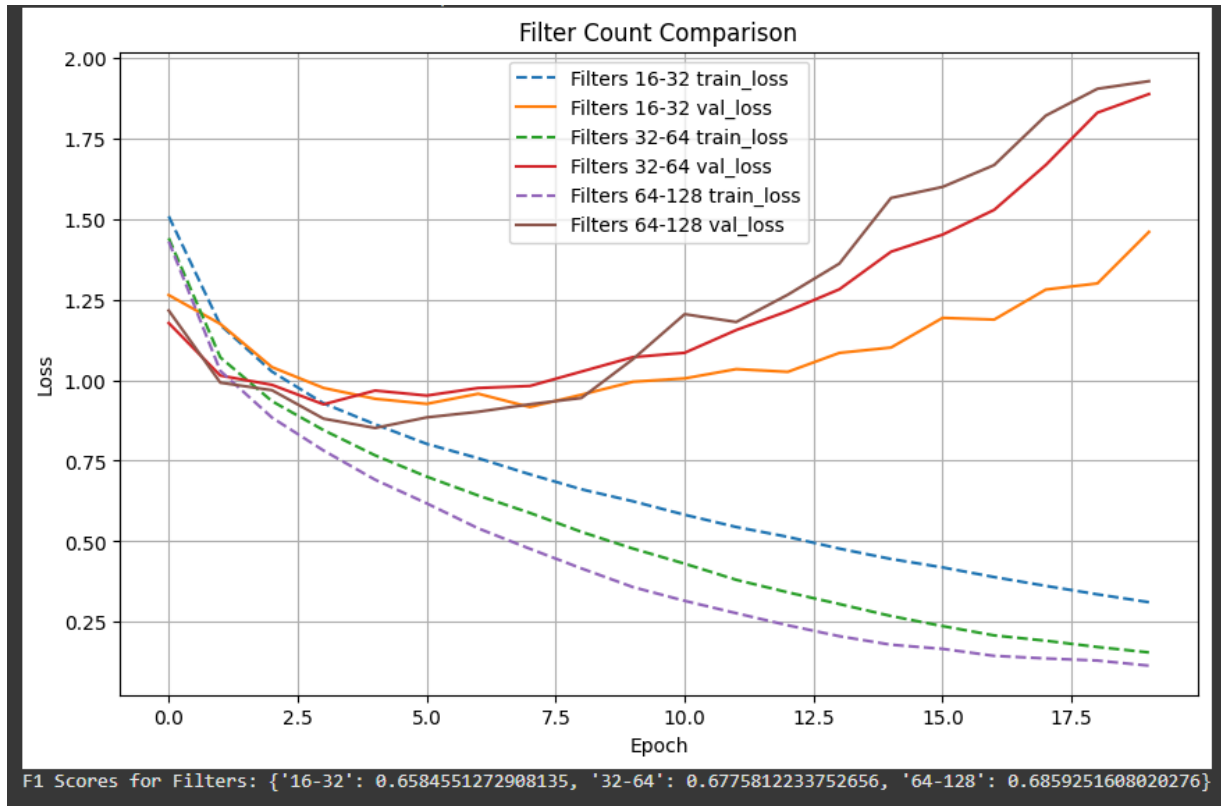
HASIL PENGUJIAN

A. Hasil Pengujian CNN

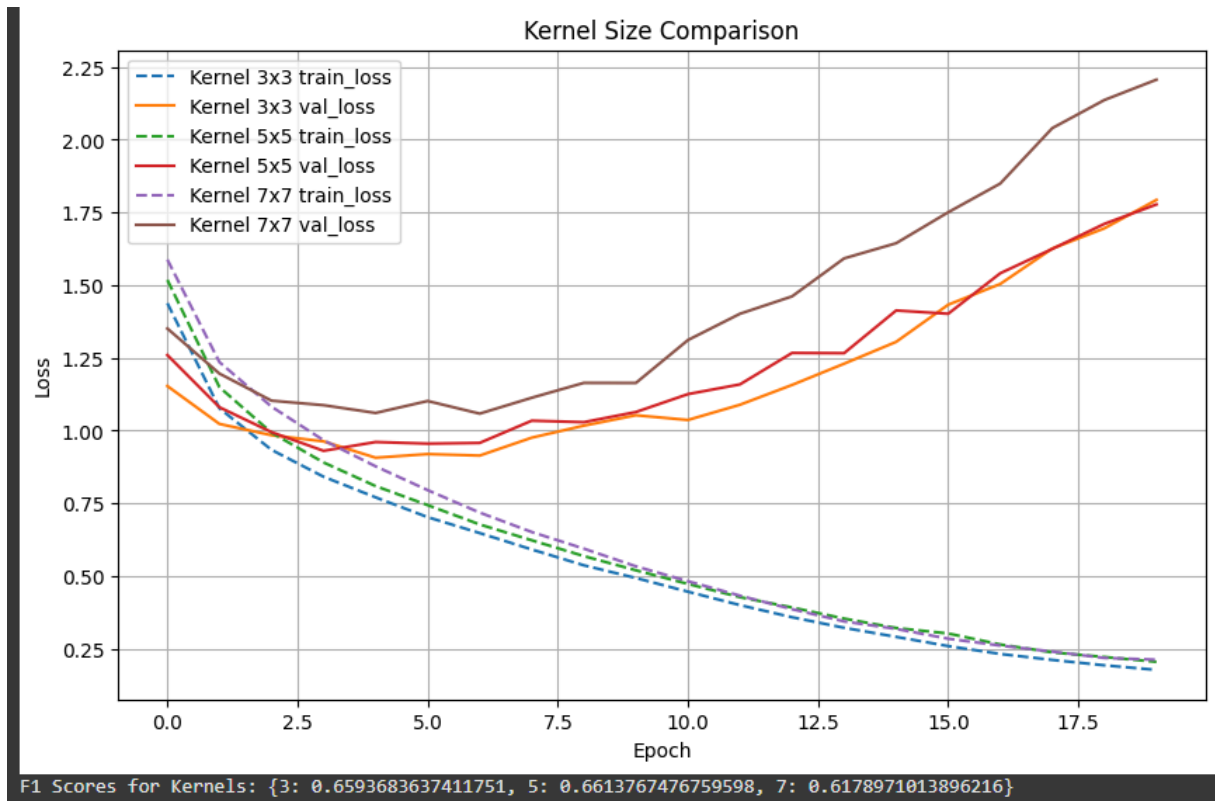
a. Pengaruh jumlah layer konvolusi



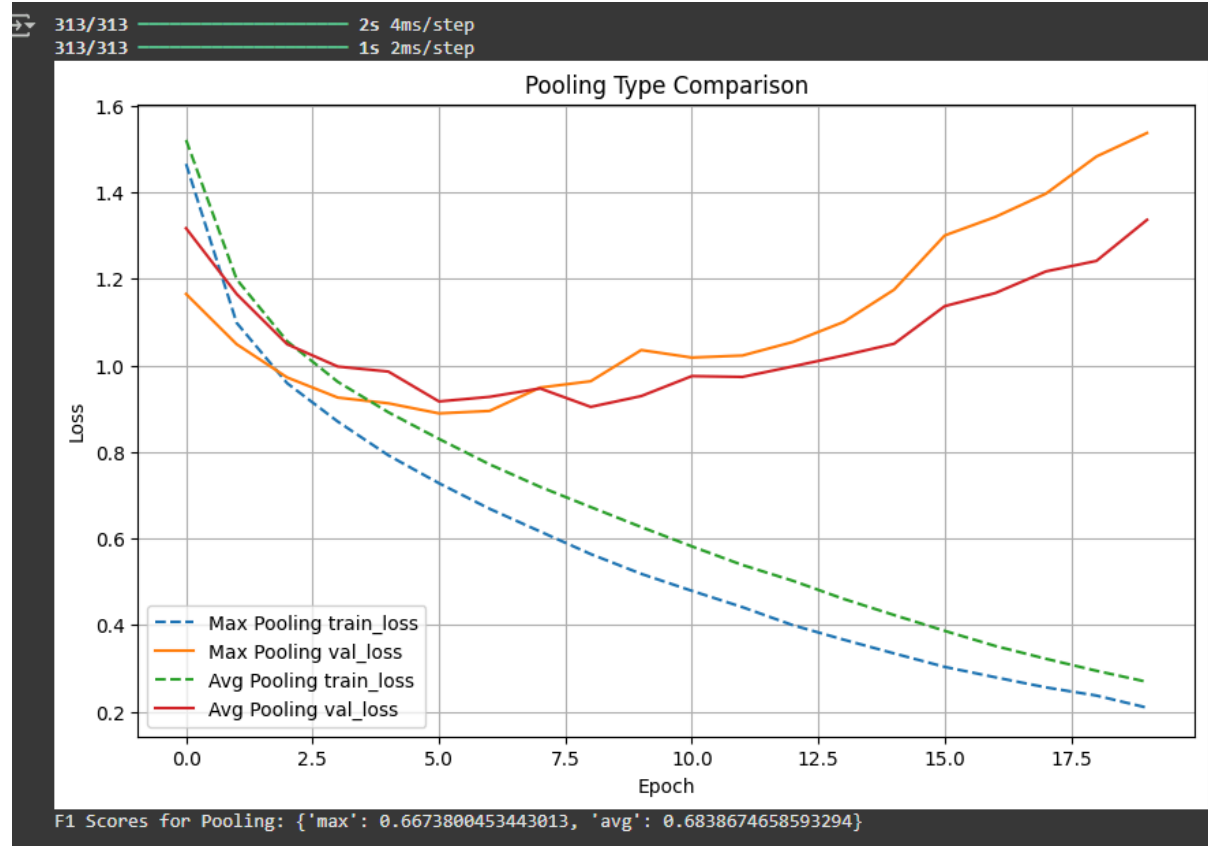
b. Pengaruh banyak filter per layer konvolusi



c. Pengaruh ukuran filter per layer konvolusi



d. Pengaruh jenis pooling layer



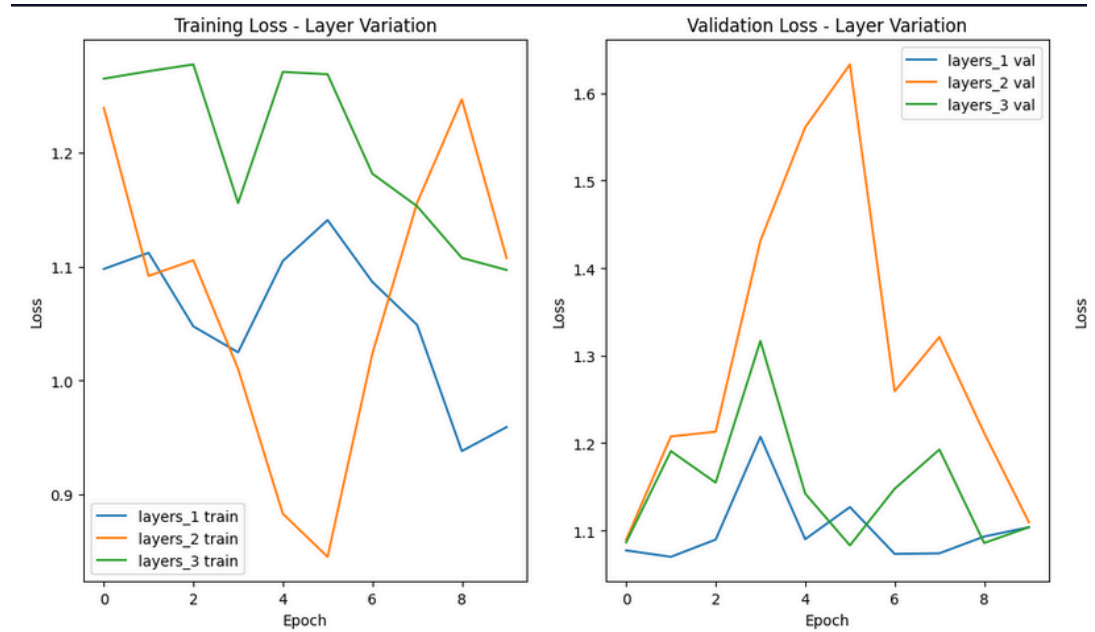
B. Hasil Pengujian RNN

Berikut adalah konfigurasi default apabila atribut bukan variabel dalam pengujian

Atribut	Nilai
Banyak layer RNN	1
Banyak sel RNN per layer	64
Jenis layer RNN berdasarkan arah	Unidirectional
Vocab size	20000
Sequence Length (Ukuran timestep)	128
Dimensi Embedding Layer	128

a. Pengaruh jumlah layer RNN

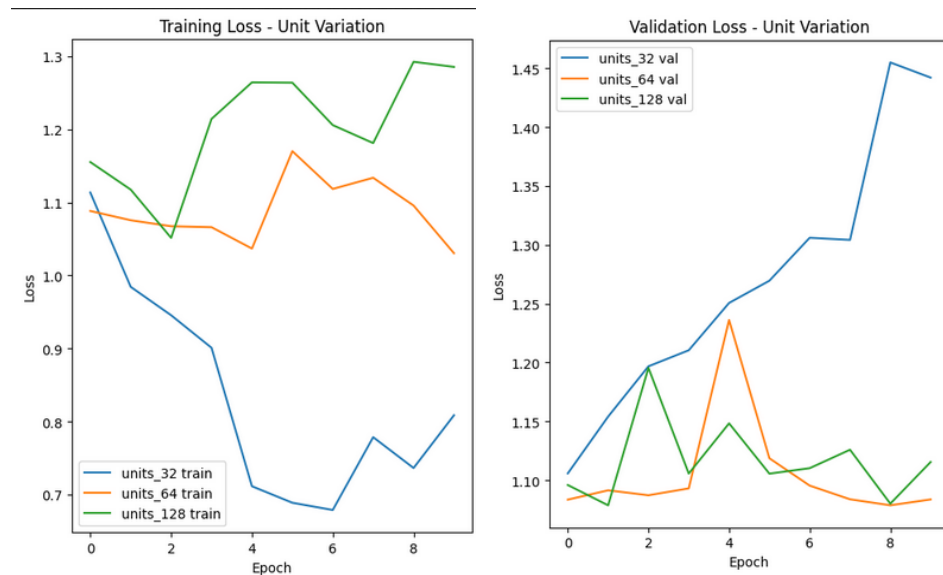
Berikut merupakan grafik training loss dan validation loss dengan variasi layer berjumlah 1, 2, dan 3



Gambar 1. Grafik training loss dan validation loss per epoch.

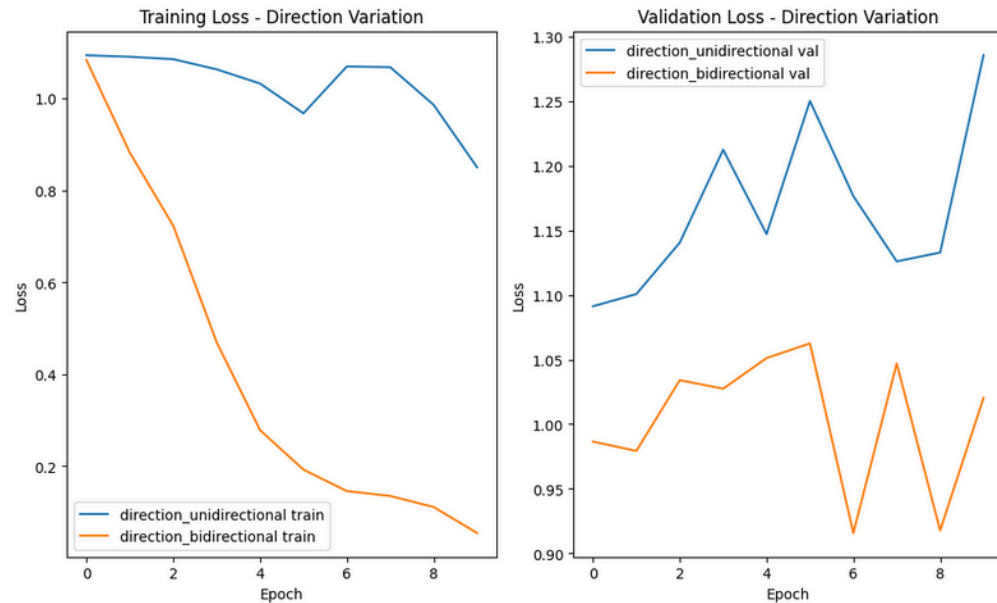
b. Pengaruh banyak cell RNN per layer

Berikut merupakan grafik *training loss* dan *validation loss* dengan variasi jumlah cell RNN per layer sebanyak 32, 64, dan 128.



c. Pengaruh jenis layer RNN berdasarkan arah

Berikut merupakan grafik *training loss* dan *validation loss* dengan variasi arah layer RNN (unidirectional atau bidirectional).



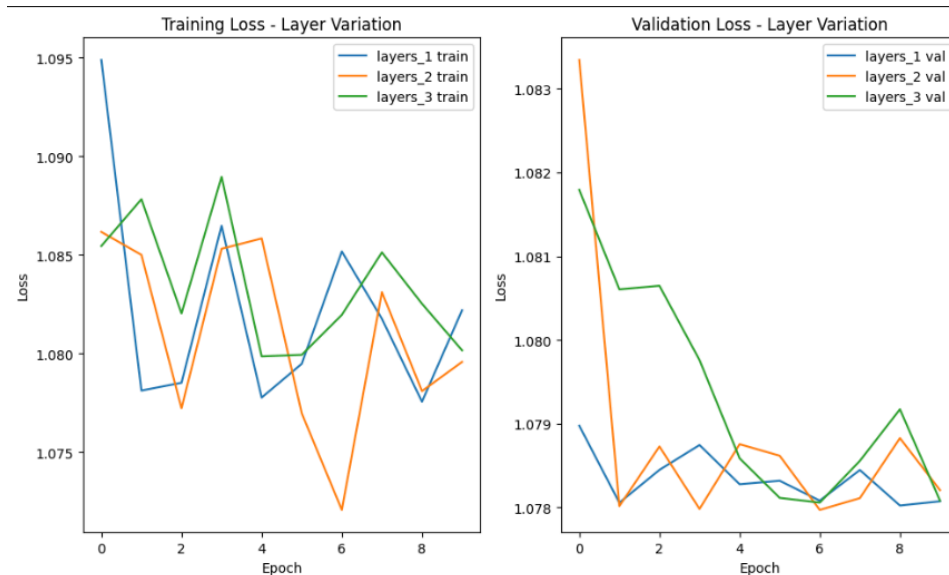
C. Hasil Pengujian LSTM

Berikut adalah konfigurasi default apabila atribut bukan variabel dalam pengujian

Atribut	Nilai
Banyak layer LSTM	1
Banyak sel LSTM per layer	64
Jenis layer LSTM berdasarkan arah	Unidirectional

a. Pengaruh jumlah layer LSTM

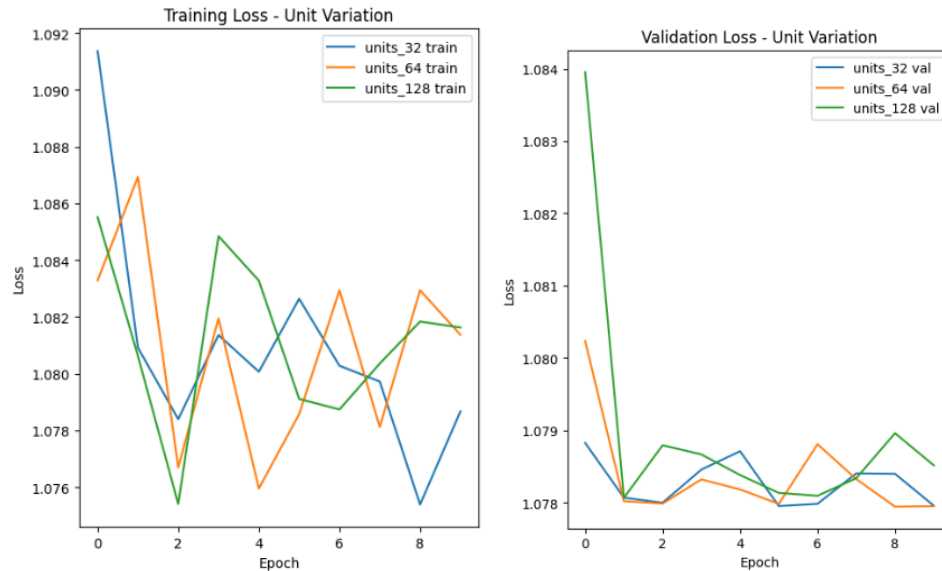
Berikut merupakan grafik training loss dan validation loss dalam pengujian jumlah layer. Variasi jumlah layer yang digunakan adalah 1, 2, dan 3



Berdasarkan grafik tersebut, dapat dilihat bahwa jumlah layer LSTM mempengaruhi hasil pembelajaran baik pada training loss maupun validation loss. Pada grafik training loss, model dengan 3 layer LSTM menunjukkan penurunan loss yang lebih signifikan dibandingkan model dengan 1 atau 2 layer, mengindikasikan kemampuan yang lebih baik dalam mempelajari pola data. Namun, pada grafik validation loss, model dengan 3 layer LSTM cenderung memiliki loss yang lebih tinggi dibandingkan model dengan 1 atau 2 layer, terutama setelah epoch tertentu. Hal ini menunjukkan bahwa model dengan lebih banyak layer LSTM mungkin mengalami overfitting, di mana model terlalu kompleks sehingga performa pada data validasi menurun meskipun performa pada data training terus membaik.

b. Pengaruh banyak cell LSTM per layer

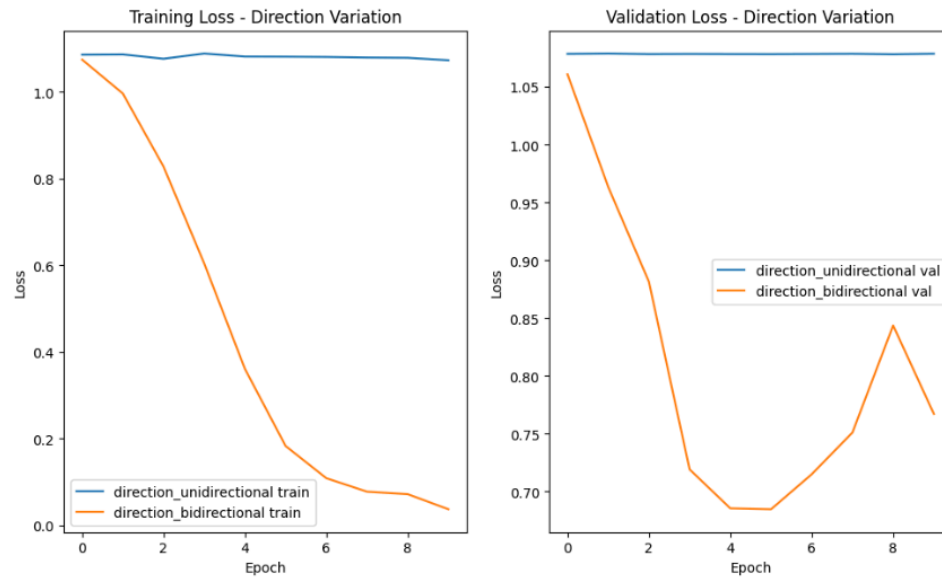
Berikut merupakan grafik training loss dan validation loss dalam pengujian banyak sel LSTM per layer. Variasi banyak sel yang digunakan adalah 32, 64, dan 128



Berdasarkan grafik tersebut, dapat diamati bahwa peningkatan jumlah unit pada lapisan LSTM mempengaruhi performa model. Pada grafik training loss, model dengan 128 unit menunjukkan penurunan loss yang lebih cepat dan lebih rendah dibandingkan model dengan unit lebih sedikit (32 atau 64), menandakan kapasitas pembelajaran yang lebih kuat. Namun, pada grafik validation loss, model dengan 128 unit cenderung memiliki loss yang lebih tinggi atau fluktuatif dibandingkan model dengan unit lebih sedikit, terutama setelah beberapa epoch. Hal ini mengindikasikan bahwa model dengan unit terlalu besar mungkin mulai overfitting, yaitu terlalu spesifik mempelajari data training sehingga kurang mampu beradaptasi dengan data validasi. Sebaliknya, model dengan unit lebih kecil (32 atau 64) menunjukkan performa yang lebih stabil pada data validasi, meskipun penurunan loss pada data training lebih lambat.

c. Pengaruh jenis layer LSTM berdasarkan arah

Berikut merupakan grafik training loss dan validation loss dalam pengujian jenis layer LSTM berdasarkan arah. Variasi jenis layer adalah unidirectional dan bidirectional



Berdasarkan grafik tersebut, model bidirectional menunjukkan performa lebih unggul dengan training loss yang turun lebih cepat dan mencapai nilai lebih rendah (mendekati 0.0), serta validation loss yang lebih stabil (0.70 vs 0.75 pada unidirectional). Hal ini membuktikan bahwa arsitektur bidirectional—yang memproses data dari dua arah—memiliki kapasitas pembelajaran lebih efektif dan generalisasi lebih baik. Meski membutuhkan komputasi lebih besar, bidirectional layak dipertimbangkan untuk tugas sekuensial selama sumber daya mencukupi, sementara unidirectional tetap menjadi pilihan efisien untuk kasus dengan keterbatasan komputasi.

BAB III

KESIMPULAN DAN SARAN

KESIMPULAN

Pada tugas besar ini, kami telah berhasil mengimplementasikan forward propagation untuk tiga arsitektur neural network yang berbeda, yaitu Convolutional Neural Network, Recurrent Neural Network, dan Long Short-Term Memory. Implementasi dilakukan from scratch yang berarti tanpa menggunakan library high-level seperti TensorFlow atau PyTorch, sehingga memungkinkan pemahaman yang lebih mendalam terhadap alur data dan perhitungan internal dari tiap jenis layer.

Selain itu, kami juga membangun dan melatih model CNN untuk klasifikasi citra menggunakan dataset CIFAR-10 serta model RNN dan LSTM untuk klasifikasi sentimen teks Bahasa Indonesia menggunakan dataset NusaX-Sentiment, dengan memanfaatkan Keras. Melalui serangkaian pengujian, kami mengevaluasi pengaruh dari berbagai aspek terhadap kinerja model: jumlah layer konvolusi, banyak filter per konvolusi, ukuran filter per konvolusi, dan jenis pooling layer pada arsitektur CNN, serta jumlah layer, banyak cell per layer, dan jenis layer berdasarkan arah (bidirectional atau unidirectional) pada RNN dan LSTM.

Berdasarkan pengujian CNN, dapat disimpulkan bahwa jumlah layer konvolusi, banyak filter tiap layer konvolusi, ukuran filter per layer konvolusi, dan jenis pooling layer berpengaruh terhadap hasil pelatihan. Selain itu, pada pengujian RNN dan LSTM, dapat disimpulkan pula bahwa jumlah layer, banyak cell per layer, dan jenis layer berdasarkan arah berpengaruh terhadap hasil pelatihan. Artinya, dalam membangun model, sangat penting untuk menentukan konfigurasi yang tepat.

SARAN

Terdapat beberapa saran yang ingin kami sampaikan dalam pengerjaan tugas besar ini, yaitu:

1. Membangun modul RNN dan LSTM dengan lebih modular, mengingat keduanya memiliki arsitektur yang serupa. Keduanya bisa memiliki beberapa atribut yang sama, dan dibedakan hanya pada sel LSTM yang memperhitungkan beberapa nilai sebagai gate dan cell state
2. Mengerjakan dan menyicil tugas jauh-jauh hari sebelum deadline supaya hasil kode maupun laporan lebih baik

BAB IV

PEMBAGIAN TUGAS

NIM	Nama	Tugas
13522008	Ahmad Farid Mudrika	Implementasi RNN
13522018	Ibrahim Ihsan Rasyid	Implementasi LSTM
13522036	Akbar Al Fattah	Implementasi CNN

BAB V

REFERENSI

- Link Github: <https://github.com/DeltDev/Tubes2-ML>
- Spesifikasi Tugas:  Spesifikasi Tugas Besar 2 IF3270 Pembelajaran Mesin
- Salindia Kuliah Pembelajaran Mesin CNN: [Salindia CNN](#)
- Salindia Kuliah Pembelajaran Mesin RNN: [Salindia RNN](#)
- Salindia Kuliah Pembelajaran Mesin LSTM: [Salindia LSTM](#)
- Dataset CIFAR-10: <https://www.tensorflow.org/datasets/catalog/cifar10>
- Dataset NusaX-Sentiment (Bahasa Indonesia):
<https://github.com/IndoNLP/nusax/tree/main/datasets/sentiment/indonesian>