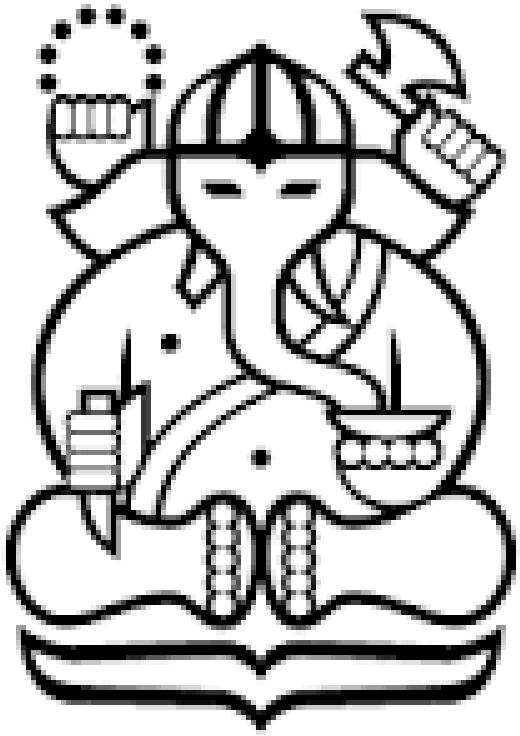
LAPORAN CNN, RNN, dan LSTM

IF3270 - Pembelajaran Mesin



Nama Kelompok:

1. Suthasoma Mahardhika Munthe (13522098)
2. Marvin Scifo Y. Hutahaean (13522110)
3. Berto Richardo Togatorop (13522118)

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

2025

Daftar Isi

BAB I
DESKRIPSI PERSOALAN

A. Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) adalah arsitektur jaringan saraf tiruan yang dirancang secara khusus untuk memproses data yang berbentuk grid, seperti gambar. CNN sangat populer dalam bidang Computer Vision karena kemampuannya mengekstraksi fitur spasial dari data.

Ciri khas CNN:

- Convolutional layer: Menerapkan filter (kernel) untuk mendeteksi fitur seperti tepi, tekstur, atau pola dalam input.
- Pooling layer: Mengurangi dimensi data (downsampling) dan meningkatkan efisiensi komputasi.
- Fully connected layer: Biasanya berada di akhir jaringan dan digunakan untuk klasifikasi berdasarkan fitur yang diekstrak.

Penggunaan CNN:

- Deteksi dan klasifikasi objek dalam gambar (misalnya pengenalan wajah, diagnosis medis berbasis citra).
- Pemrosesan citra dan video.
- Sistem pengenalan tulisan tangan.

Loss Function yang digunakan:

CNN yang digunakan untuk tugas klasifikasi dengan label berupa integer (bukan one-hot encoded) dapat memanfaatkan Sparse Categorical Cross Entropy. SCCE menghitung jarak antara output probabilistik (biasanya dari softmax) dan label target dalam bentuk integer.

B. Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) adalah jenis jaringan saraf tiruan yang dirancang untuk memproses data berurutan, seperti teks, sinyal suara, atau data deret waktu.

Ciri khas RNN:

- Memiliki loop dalam arsitekturnya, sehingga informasi dari langkah sebelumnya dapat digunakan pada langkah berikutnya (memori jangka pendek).
- Cocok untuk input berdimensi waktu (sequence).

Penggunaan RNN:

- Pemrosesan bahasa alami (Natural Language Processing / NLP), seperti text generation atau sentiment analysis.
- Pengenalan suara.
- Prediksi deret waktu.

Loss Function yang digunakan:
Jika tugas klasifikasi dilakukan pada setiap langkah urutan dengan output berupa label integer, Sparse Categorical Cross Entropy digunakan untuk menghitung loss antar waktu.

C. Long Short Term Memory (LSTM)

Long Short-Term Memory (LSTM) adalah varian dari RNN yang dirancang untuk mengatasi masalah long-term dependency pada RNN biasa, yaitu saat jaringan sulit mengingat informasi dari langkah-langkah yang jauh sebelumnya.
Ciri khas LSTM:

- Memiliki gating mechanism: input gate, forget gate, dan output gate, yang mengontrol aliran informasi dalam sel memori.
- Dapat mengingat informasi penting dalam waktu yang lebih panjang dibanding RNN biasa.

Penggunaan LSTM:

- Terjemahan mesin (machine translation).
- Analisis sentimen dan text summarization.
- Prediksi berbasis urutan seperti cuaca atau pasar saham.

Loss Function yang digunakan:
Sama seperti RNN, LSTM untuk klasifikasi sekuensial menggunakan Sparse Categorical Cross Entropy, cocok untuk label dalam bentuk indeks kelas.

BAB II
PEMBAHASAN

1. Penjelasan Implementasi

A. Deskripsi Kelas, Atribut, dan Metode

Pada tugas ini, kami membuat model *Convolution Neural Network (CNN), Recurrent Neural Network (RNN),* dan *Long Short Term Memory (LSTM)* dengan menggunakan library seperti numpy, sklearn untuk pengukuran kinerja, tensorflow untuk mendapatkan bobot dan bias untuk dipindahkan ke model-model yang dibuat *from scratch*, dll. Untuk mengimplementasikan ketiga model ini, kami membuat kelas UtilisationFunctions yang berfungsi sebagai membantu jalannya kode-kode *forward propagation* yang akan dijelaskan nantinya.

1. Kelas UtilisationFunctions

   Kelas UtilisationFunctions adalah sebuah kelas yang berisi fungsi aktivasi, loss, inisialisasi bobot, perkalian konvolusi, padding, dll yang digunakan dalam kode-kode *forward propagation*. Secara umum, metode-metode pada kelas ini bersifat *static* sehingga tidak ada instantiasi objek untuk kelas ini. Berikut adalah metode-metode yang terdapat dalam kelas ini:

   - Softmax Activation Function

     Softmax menghasilkan distribusi probabilitas dan memerlukan matriks Jacobian untuk perhitungan gradien.

```
@staticmethod
    def softmax(x, output=None, derivative=False):
        if derivative:
            batch_size, n_classes = output.shape
            jacobians = np.empty((batch_size,
n_classes, n_classes))
            for i in range(batch_size):
                s = output[i].reshape(-1, 1)
                jacobians[i] = np.diagflat(s) -
np.dot(s, s.T)
            return jacobians

        exp_x = np.exp(x - np.max(x, axis=-1,
keepdims=True))
        softmax_x = exp_x / np.sum(exp_x, axis=-1,
keepdims=True)
        return softmax_x
```

   - ReLU Activation Function

     ReLU digunakan untuk menghindari vanishing gradient pada jaringan dalam.

```
@staticmethod
    def reLU(x, output=None, derivative=False):
        return np.maximum(0,x) if not derivative else
UtilisationFunctions.reLUDeriv(x)

    @staticmethod
    def reLUDeriv(x):
        return np.where(x > 0, 1.0, 0.0)
```

- Sigmoid Activation Function
  Sigmoid cocok untuk output antara 0 dan 1 tetapi rawan mengalami vanishing gradient.

```
@staticmethod
    def sigmoid(x):
        return 1 / (1 + np.exp(-x))

    @staticmethod
    def sigmoid_derivative(x):
        return x / (1 - x)
```

- Hyperbolic Tangent (tanh) Activation Function
  Mengubah input menjadi nilai antara -1 dan 1. (Fungsi biasa diimplementasikan menggunakan pustaka numpy dan turunannya diimplementasikan seperti itu)

```
@staticmethod
    def tanh_derivative(x):
        return (1 - x ** 2)
```

- SCSE
  SCSE digunakan untuk klasifikasi multi kelas pada model CNN, RNN, dan LSTM

```
@staticmethod
    def sparse_categorical_entropy(y_true, y_pred):
        y_pred = np.clip(y_pred, 1e-15, 1.0)
        batch_size = y_pred.shape[0]
        return -np.log(y_pred[np.arange(batch_size),
y_true.flatten()])
```

- Padding
  Padding digunakan untuk menambahkan 0 sekeliling matriks input sebanyak *pad_num* kali

```
@staticmethod
def pad(x: np.ndarray, pad_num: int):
        if len(x.shape) == 3:
            C, H, W = x.shape
            new_arr = np.zeros((C, H + 2 * pad_num, W
+ 2 * pad_num))
            for c in range(C):
                for i in range(H):
                    for j in range(W):
                        new_arr[c, i + pad_num, j +
pad_num] = x[c, i, j]
            return new_arr
        elif len(x.shape) == 2:
            H, W = x.shape
            new_arr = np.zeros((H + 2 * pad_num, W +
2 * pad_num))
            for i in range(H):
                for j in range(W):
                    new_arr[i + pad_num, j + pad_num]
= x[i, j]
            return new_arr
        else:
            raise ValueError("Unsupported input shape
for padding")
```

- Perkalian Konvolusi
  Menerima matriks besar (input) dan matriks kecil (kernel) untuk melakukan konvolusi sesuai dengan padding dan stride yang diberikan

```
@staticmethod
    def convMul(big: np.ndarray, smol: np.ndarray,
stride: int):
        big_copy = np.array(big)
        smol_copy = np.array(smol)

        out_size = ((len(big) - len(smol)) // stride)
+ 1
        convRes = np.zeros((out_size, out_size))
        for i in range(len(convRes)):
            for j in range(len(convRes)):
                smoled_big =
big_copy[i*stride:i*stride+len(smol),
j*stride:j*stride+len(smol)]
                convRes[i][j] = np.sum(smoled_big *
smol_copy)
        return convRes
```

- Max dan Average Selector
  Memilih elemen terbesar atau rata-rata dari semua elemen pada matriks masukkan (input)

```
@staticmethod
    def maxArray(arr: np.ndarray):
        return np.max(arr)

    @staticmethod
    def avgArray(arr: np.ndarray):
        return np.mean(arr)
```

- Pooling
  Memilih elemen terbesar atau rata-rata berdasarkan kernel dan stride yang diberikan pada input

```
@staticmethod
    def pooling(big: np.ndarray, smol_size: int,
stride: int, method: Literal["max",
"average"]="max"):
        big_copy = np.array(big)

        out_size = ((big_copy.shape[0] - smol_size)
// stride) + 1
        # print(f"Size Detail: {big_copy.shape[0]},
{smol_size}, {stride}, {big.shape[0]},
{big.shape[1]}")
        poolRes = np.zeros((out_size, out_size))

        for i in range(out_size):
            for j in range(out_size):
                smoled_big =
big_copy[i*stride:i*stride+smol_size,
j*stride:j*stride+smol_size]
                if (method == "max"):
                    poolRes[i][j] =
UtilisationFunctions.maxArray(smoled_big)
                else:
                    poolRes[i][j] =
UtilisationFunctions.avgArray(smoled_big)
        return poolRes
```

2. RMSNorm (Bagian dari Implementasi Fully Connected Layer pada Tubes 1)

Kelas ini adalah implementasi dari *root mean square normalization* pada suatu layer. RMS Normalization sendiri adalah salah satu metode *layer normalization* yang melakukan normalisasi terhadap transformasi linear $W.x$ sebelum dijumlahkan dengan bias dan diberikan fungsi aktivasi. Secara matematis, bentuk umum RMSNorm adalah sebagai berikut (Zhang & Sennrich, 2019):

$$y = f\left(\frac{xW}{RMS(a)} \odot g + b\right)$$

Adapun RMS(a) dapat dituliskan sebagai berikut:

$$RMS(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$$

- Konstruktor kelas RMSNorm
  Pada konstruktor ini, kita menginisialisasi banyak dimensi (fitur) pada layer. Nantinya, kelas RMSNorm akan diinstansiasi oleh layer yang menggunakan normalisasi ini.

```
    def __init__(self, dim, eps=1e-8):
        self.dim = dim
        self.eps = eps
        self.g = np.ones(dim)
        self.grad_g = np.zeros_like(self.g)
        self.cache = {}
```

- Forward Propagation pada RMSNorm
  Fungsi ini melakukan normalisasi terhadap Wx sebelum ditambahkan dengan bias dan diberikan fungsi aktivasi. Cara kerja fungsi ini adalah dengan mengkalkulasi persamaan berikut:

$$\widehat{a} = \frac{xW}{RMS(a)} \odot g$$

```
    def forward(self, x):
        self.cache['x'] = x
        rms = np.sqrt(np.mean(x ** 2, axis=-1,
    keepdims=True) + self.eps)
        self.cache['rms'] = rms
        x_norm = x / rms
        self.cache['x_norm'] = x_norm
        out = x_norm * self.g
        return out
```

- Backward Propagation pada RMSNorm
  Fungsi ini melakukan backward propagation untuk melakukan pembelajaran terhadap g yang merupakan *learnable parameter* dalam RMSNorm. Sama

halnya dengan *weight* dan *bias*, kita melakukan *backward propagation* menggunakan *gradient descent*. Diberikan loss function L, kita dapat menemukan gradiennya terhadap parameter g (Zhang & Sennrich, 2019):

$$\frac{\partial L}{\partial g} = \frac{\partial L}{\partial v}\frac{Wx}{RMS(a)}$$

dengan v diberikan oleh persamaan berikut:

$$v = \frac{xW}{RMS(a)} \odot g + b$$

```
    def backward(self, grad_output):
        x = self.cache['x']
        x_norm = self.cache['x_norm']
        rms = self.cache['rms']
        dim = self.dim
        self.grad_g = np.sum(grad_output * x_norm,
    axis=0)
        grad_x_norm = grad_output * self.g
        sum_x_grad = np.sum(x * grad_x_norm,
    axis=-1, keepdims=True)
        dx = grad_x_norm / rms - (x * sum_x_grad) /
    (dim * rms**3)
        return dx
```

3. Conv2D
   - Konstruktor Kelas Conv2D

```
def __init__(self, in_channels: int, out_channels:
int, kernel_size: int, stride: int, padding: int,
bias=0.0):
        self.in_channels = in_channels
        self.out_channels = out_channels
        self.kernel_size = kernel_size
        self.stride = stride
        self.padding = padding
        self.bias = np.full((out_channels), bias)
        self.kernel =
np.random.randn(self.out_channels, self.in_channels,
self.kernel_size, self.kernel_size)
        self.last_image = None
```

   - Forward Propagation pada Conv2D

```
def forward(self, x: np.ndarray):
        self.last_image = x

        if self.padding > 0:
```

```
            x = np.pad(x, ((0, 0), (self.padding,
self.padding), (self.padding, self.padding)))


        size = (len(x[0]) - self.kernel_size) //
self.stride + 1
        out = np.zeros((self.out_channels, size,
size))
        for i in range(len(self.kernel)):
            temp_out = np.zeros((size, size))
            for j in range(len(self.kernel[i])):
                temp_conv =
UtilisationFunctions.convMul(x[j], self.kernel[i][j],
self.stride)
                temp_out = temp_out + temp_conv
            temp_out += self.bias[i]
            out[i] = temp_out
        return out
```

4. Pooling
   - Konstruktor Kelas Pooling

```
def __init__(self, kernel_size: int, stride: int,
method: Literal["max", "average"]="max"):
        self.kernel_size = kernel_size
        self.stride = stride
        self.method = method
        self.last_image = None
```

   - Forward Propagation pada Pooling

```
def forward(self, x: np.ndarray):
        self.last_image = x
        height = x.shape[1]
        size = (height - self.kernel_size) //
self.stride + 1
        # print(f"Size: {size}")
        out = np.zeros((x.shape[0], size, size))
        for i in range(x.shape[0]):
            out[i] =
UtilisationFunctions.pooling(x[i], self.kernel_size,
self.stride, self.method)
        return out
```

   - Backward Propagation pada Pooling

```
def backward(self, dLast: np.ndarray):
        in_channels = self.last_image.shape[0]
        hout, wout = dLast.shape[1], dLast.shape[2]
        dinput = np.zeros_like(self.last_image)

        for ic in range(in_channels):
            for i in range(hout):
                for j in range(wout):
                    col_start = i * self.stride
                    col_end = col_start +
self.kernel_size
                    row_start = j * self.stride
                    row_end = row_start +
self.kernel_size

                    last_image_slice =
self.last_image[ic, col_start:col_end,
row_start:row_end]

                    if (self.method == "max"):
                        maxe_index =
np.unravel_index(np.argmax(last_image_slice),
last_image_slice.shape)
                        dinput[ic, col_start:col_end,
row_start:row_end][maxe_index] += dLast[ic, i, j]
                    else:
                        dinput[ic, col_start:col_end,
row_start:row_end] += dLast[ic, i, j] /
(self.kernel_size * self.kernel_size)

        return dinput
```

5. ReLU
   - Konstruktor Kelas ReLU

```
def __init__(self):
        self.reLU =
np.vectorize(UtilisationFunctions.reLU)
        self.last_image = None
```

   - Forward Propagation pada ReLU

```
def forward(self, x: np.ndarray):
        self.last_image = x
        return self.reLU(x)
```

- Backward Propagation pada ReLU

```
def backward(self, dLast: np.ndarray):
        dinput = dLast * (self.last_image > 0)
        return dinput
```

6. Flatten
   - Konstruktor Kelas Flatten

```
def __init__(self):
        self.last_shape = None
```

   - Forward Propagation pada Flatten

```
def forward(self, x: np.ndarray):
        self.last_shape = x.shape
        reshaped = x.reshape(1, -1)
        return reshaped
```

   - Backward Propagation pada Flatten

```
def backward(self, dLast: np.ndarray):
        return dLast.reshape(self.last_shape)
```

7. Fully Connected
   - Konstruktor Kelas Fully Connected

```
def __init__(
        self,
        input_size: int,
        output_size: int,
        activation: Literal["reLU", "softmax"],
        weight_init: Literal["zero", "uniform",
"normal", "he", "xavier"],
        lower: float,
        upper: float,
        mean: float,
        variance: float,
        seed,
        use_rmsnorm: bool=True
    ):
        self.activation_name = activation
        self.activation =
getattr(UtilisationFunctions, activation)

        if seed is not None:
```

```
            np.random.seed(seed)

        if weight_init == "zero":
            self.weights = np.zeros((input_size,
output_size))
            self.biases = np.zeros((1, output_size))
        elif weight_init == "uniform":
            self.weights = np.random.uniform(lower,
upper, (input_size, output_size))
            self.biases = np.random.uniform(lower,
upper, (1, output_size))
        elif weight_init == "normal":
            self.weights = np.random.normal(
                mean, np.sqrt(variance), (input_size,
output_size)
            )
            self.biases = np.random.normal(mean,
np.sqrt(variance), (1, output_size))
        elif weight_init == "he":
            self.weights = np.random.normal(
                0, np.sqrt(2 / input_size),
(input_size, output_size)
            )
            self.biases = np.zeros((1, output_size))
        elif weight_init == "xavier":
            limit = np.sqrt(6 / (input_size +
output_size))
            self.weights = np.random.uniform(-limit,
limit, (input_size, output_size))
            self.biases = np.zeros((1, output_size))
        else:
            self.weights = np.zeros((input_size,
output_size))
            self.biases = np.zeros((1, output_size))

        self.input = None
        self.output = None
        self.z = None
        self.grad_weights = np.zeros((input_size,
output_size))
        self.grad_biases = np.zeros((1, output_size))

        self.use_rmsnorm = use_rmsnorm
        if self.use_rmsnorm:
            self.rmsnorm = RMSNorm(output_size)
        else:
```

```
            self.rmsnorm = None
```

- Forward Propagation pada Flatten

```
def forward(self, x):
        self.input = x
        z = x @ self.weights
        if self.use_rmsnorm:
            z = self.rmsnorm.forward(z)
        z = z + self.biases
        self.z = z
        self.output = self.activation(z)
        return self.output
```

- Backward Propagation pada Flatten

```
def backward(self, grad_output):
        if self.activation_name == "softmax":
            jacobians = self.activation(self.z,
self.output, derivative=True)
            grad_list = []
            for i in range(self.output.shape[0]):
                grad_i = np.dot(grad_output[i : i +
1], jacobians[i])
                grad_list.append(grad_i)
            grad = np.concatenate(grad_list, axis=0)
        else:
            activation_grad = self.activation(self.z,
self.output, derivative=True)
            grad = grad_output * activation_grad

        self.grad_biases = np.sum(grad, axis=0,
keepdims=True)

        if self.use_rmsnorm:
            grad = self.rmsnorm.backward(grad)

        self.grad_weights = self.input.T @ grad
        return grad @ self.weights.T
```

8. Embedded
   - Konstruktor Kelas Embedded

```
def __init__(self, input_dim, output_dim,
```

```
embeddings_initializer=Literal["uniform", "xavier"]):
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.embeddings_initializer =
embeddings_initializer
        self.last_input = None
        self.learning_rate = 0.005
        if (self.embeddings_initializer ==
"uniform"):
            self.weight = np.random.uniform(-0.05,
0.05, size=(input_dim, output_dim))
        elif (self.embeddings_initializer ==
"xavier"):
            limit = np.sqrt(6 / (input_dim +
output_dim))
            self.weight = np.random.uniform(-limit,
limit, size=(input_dim, output_dim))
```

- Forward Propagation pada Embedded

```
def forward(self, x: np.ndarray):
        self.last_input = x
        embedded_vector = self.weight[x]
        return embedded_vector
```

- Backward Propagation pada Embedded

```
def backward(self, dLast: np.ndarray):
        for i in range(self.last_input.shape[0]):
            for j in range(self.last_input.shape[1]):
                token_idx = self.last_input[i][j]
                self.weight[token_idx] -=
self.learning_rate * dLast[i][j]
```

9. Dropout
   - Konstruktor Kelas Dropout

```
def __init__(self, dropout_rate=0.3):
        self.dropout_rate = dropout_rate
        self.mask = None
        self.training_status = True
```

   - Forward Propagation pada Dropout

```
def forward(self, x):
```

```
        if self.training_status:
            self.mask = (np.random.rand(*x.shape) >
self.dropout_rate).astype(np.float32)
            return (x*self.mask)/(1.0 -
self.dropout_rate)
        else:
            return x
```

- Backward Propagation pada Dropout

```
def backward(self, dLast):
        return (dLast * self.mask) / (1.0 -
self.dropout_rate)
```

10. RNN
    - Konstruktor Kelas RNN

```
def __init__(self, unit, input_size, timestep,
return_sequence, bidirectional=False):
        self.unit = unit
        self.input_size = input_size
        self.timestep = timestep
        self.bidirectional = bidirectional
        self.last_input = None
        self.return_sequence = return_sequence

        self.input_weights =
UtilisationFunctions.xavier(input_size, unit,
'weight')
        self.hidden_weights =
UtilisationFunctions.xavier(unit, unit, 'weight')

        self.bias = UtilisationFunctions.xavier(1,
unit, 'bias')

        self.h_t = np.zeros((timestep, unit))

        if self.bidirectional:
            self.input_weights_reverse =
UtilisationFunctions.xavier(input_size, unit,
'weight')
            self.hidden_weights_reverse =
UtilisationFunctions.xavier(unit, unit, 'weight')

            self.bias_reverse =
UtilisationFunctions.xavier(1, unit, 'bias')
```

```
            self.h_t_reverse = np.zeros((timestep,
unit))
```

- Forward Propagation pada RNN

```
def forward(self, x: np.ndarray):
        self.last_input = x
        h = np.zeros((self.timestep, self.unit))

        if self.bidirectional:
            h_reverse = np.zeros((self.timestep,
self.unit))

        for t in range(self.timestep):
            h_used = np.zeros((self.unit)) if t == 0
else self.h_t[t-1]
            x_used = x[t]
            h[t] = np.tanh(x_used @
self.input_weights + h_used @ self.hidden_weights +
self.bias)
            self.h_t[t] = h[t]

        if self.bidirectional:
            for t in reversed(range(self.timestep)):
                h_used_reverse =
np.zeros((self.unit)) if t == self.timestep - 1 else
self.h_t_reverse[t+1]
                x_used_reverse = x[t]
                h_reverse[t] = np.tanh(x_used_reverse
@ self.input_weights_reverse + h_used_reverse @
self.hidden_weights_reverse + self.bias_reverse)
                self.h_t_reverse[t] = h_reverse[t]

        if self.return_sequence:
            if self.bidirectional:
                return np.concatenate([h, h_reverse],
axis=1)
            else:
                return h
        else:
            if self.bidirectional:
                return np.concatenate([h[-1],
h_reverse[0]])
            else:
                return h[-1]
```

11. LSTM

- Konstruktor Kelas LSTM

```
class LSTM:
    def __init__(self, unit, input_size, timestep,
return_sequence, bidirectional=False):
        self.unit = unit
        self.input_size = input_size
        self.timestep = timestep
        self.bidirectional = bidirectional
        self.last_input = None
        self.return_sequence = return_sequence

        self.input_weights =
[UtilisationFunctions.xavier(input_size, unit,
'weight') for _ in range(4)]
        self.hidden_weights =
[UtilisationFunctions.xavier(unit, unit, 'weight')
for _ in range(4)]

        self.bias = [UtilisationFunctions.xavier(1,
unit, 'bias') for _ in range(4)]

        self.h_t = np.zeros((timestep, unit))
        self.c_t = np.zeros((timestep, unit))

        if self.bidirectional:
            self.input_weights_reverse =
[UtilisationFunctions.xavier(input_size, unit,
'weight') for _ in range(4)]
            self.hidden_weights_reverse =
[UtilisationFunctions.xavier(unit, unit, 'weight')
for _ in range(4)]

            self.bias_reverse =
[UtilisationFunctions.xavier(1, unit, 'bias') for _
in range(4)]

            self.h_t_reverse = np.zeros((timestep,
unit))
            self.c_t_reverse = np.zeros((timestep,
unit))
```

- Forward Propagation pada LSTM

```
def forward(self, x: np.ndarray):
```

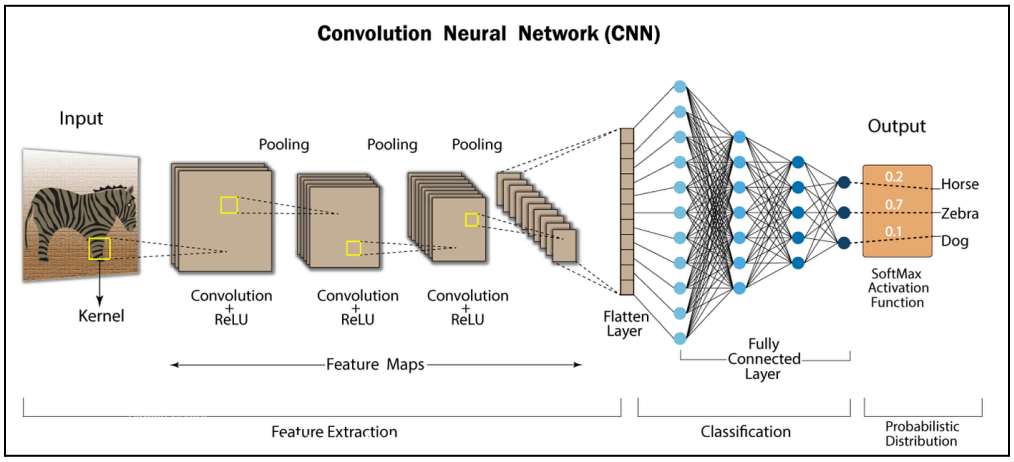
```
        self.last_input = x
        h = np.zeros((self.timestep, self.unit))
        c = np.zeros((self.timestep, self.unit))

        if self.bidirectional:
            h_reverse = np.zeros((self.timestep,
self.unit))
            c_reverse = np.zeros((self.timestep,
self.unit))

        for t in range(self.timestep):
            h_used = np.zeros((self.unit)) if t == 0
else self.h_t[t-1]
            c_used = np.zeros((self.unit)) if t == 0
else self.c_t[t-1]
            x_used = x[t]

            f = UtilisationFunctions.sigmoid(x_used @
self.input_weights[0] + h_used @
self.hidden_weights[0] + self.bias[0])
            i = UtilisationFunctions.sigmoid(x_used @
self.input_weights[1] + h_used @
self.hidden_weights[1] + self.bias[1])
            a = np.tanh(x_used @
self.input_weights[2] + h_used @
self.hidden_weights[2] + self.bias[2])
            o = UtilisationFunctions.sigmoid(x_used @
self.input_weights[3] + h_used @
self.hidden_weights[3] + self.bias[3])

            c[t] = (c_used * f) + (i * a)
            h[t] = np.tanh(c[t]) * o

            self.c_t[t] = c[t]
            self.h_t[t] = h[t]

        if self.bidirectional:
            for t in reversed(range(self.timestep)):
                h_used_reverse =
np.zeros((self.unit)) if t == self.timestep - 1 else
self.h_t_reverse[t+1]
                c_used_reverse =
np.zeros((self.unit)) if t == self.timestep - 1 else
self.c_t_reverse[t+1]
                x_used_reverse = x[t]
```

```
                f_reverse =
UtilisationFunctions.sigmoid(x_used_reverse @
self.input_weights_reverse[0] +

h_used_reverse @ self.hidden_weights_reverse[0] +

self.bias_reverse[0])
                i_reverse =
UtilisationFunctions.sigmoid(x_used_reverse @
self.input_weights_reverse[1] +

h_used_reverse @ self.hidden_weights_reverse[1] +

self.bias_reverse[1])
                a_reverse = np.tanh(x_used_reverse @
self.input_weights_reverse[2] +
                                    h_used_reverse @
self.hidden_weights_reverse[2] +

self.bias_reverse[2])
                o_reverse =
UtilisationFunctions.sigmoid(x_used_reverse @
self.input_weights_reverse[3] +

h_used_reverse @ self.hidden_weights_reverse[3] +

self.bias_reverse[3])

                c_reverse[t] = (c_used_reverse *
f_reverse) + (i_reverse * a_reverse)
                h_reverse[t] = np.tanh(c_reverse[t])
* o_reverse

                self.c_t_reverse[t] = c_reverse[t]
                self.h_t_reverse[t] = h_reverse[t]

        if self.return_sequence:
            if self.bidirectional:
                return np.concatenate([h, h_reverse],
axis=1)
            else:
                return h
        else:
            if self.bidirectional:
                return np.concatenate([h[-1],
h_reverse[0]])
```

```
            else:
                return h[-1]
```

B. Penjelasan Forward Propagation

1. Convolutional Neural Network (CNN)

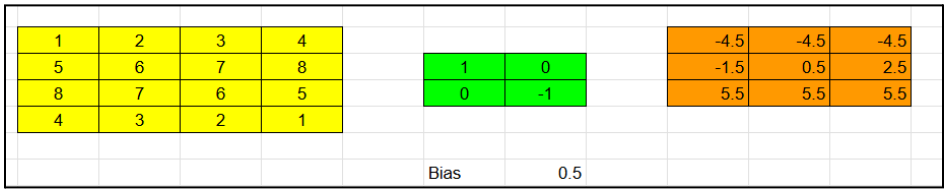


Gambar 1 - Contoh Arsitektur CNN

CNN terdiri dari beberapa jenis layer yang digunakan dan digabung menjadi satu jaringan saraf yang menerima input gambar dan mengembalikan output kategori dari gambar tersebut berdasarkan data-data yang sudah dilatih. Berikut adalah layer-layer yang digunakan pada CNN.

- **2 Dimensional Convolution**

  Convolution dilakukan dengan mengalikan (dot product) sebagian matriks dengan kernel. Terdapat kemungkinan matriks yang dioperasikan bisa dipadding.

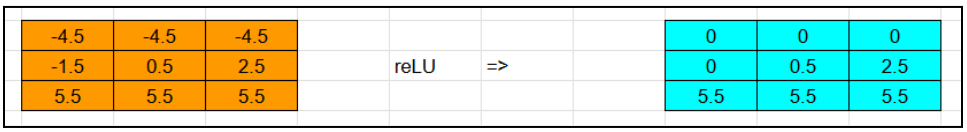


Gambar 2 - 2 Dimensional Convolution (Stride = 1, Padding = 0)

Untuk setiap langkah (stride), matriks yang dipotong sesuai ukuran kernel dikalikan lalu ditambahkan dengan bias. Perlu diketahui, matriks yang terlibat dalam output ke-n harus dijumlahkan semua sebelum ditambahkan dengan bias.

- **ReLU Activation**

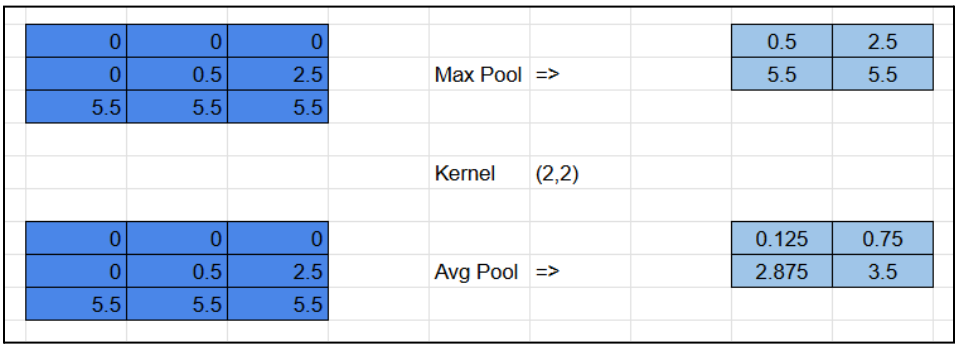
ReLU Activation dilakukan dengan mengubahkan angka-angka negatif yang ada pada matriks menjadi 0 dan membiarkan angka-angka lainnya yang positif.



Gambar 3 - ReLU Activation

- **Max or Average Pooling**

  Max or Average Pooling dilakukan dengan mengambil sebagian matriks sesuai dengan ukuran kernel, mencari nilai maksimum atau rata-rata dari matriks bagian tersebut lalu lanjut ke matriks lainnya sesuai dengan nilai stride yang ditentukan.

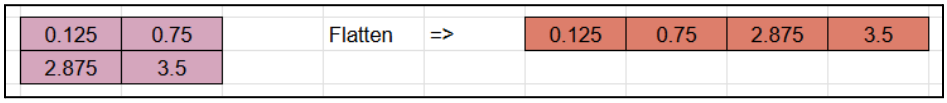


Gambar 4 - Max or Average Pooling (Stride = 1, Padding = 0)

Setelah pooling, biasanya layer selanjutnya adalah kombinasi Conv2D -> ReLU -> Pool, Conv2D -> ReLU, atau langsung ke tahap Flatten.

- **Flatten**

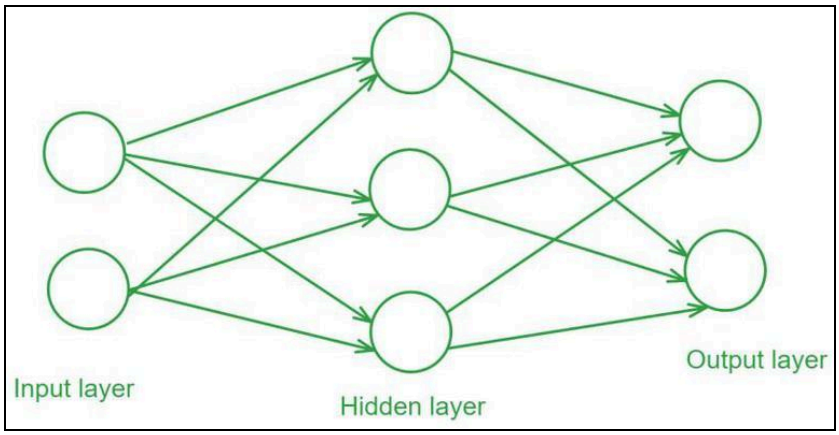
  Sebelum dibawa ke layer Fully Connected (FFNN), hasil output yang masih merupakan array 2 dimensi perlu diratakan agar bisa digunakan sebagai input nantinya pada layer Fully Connected. Berikut adalah cara flatten sebuah array 2 dimensi.



Gambar 5 - Flatten Layer

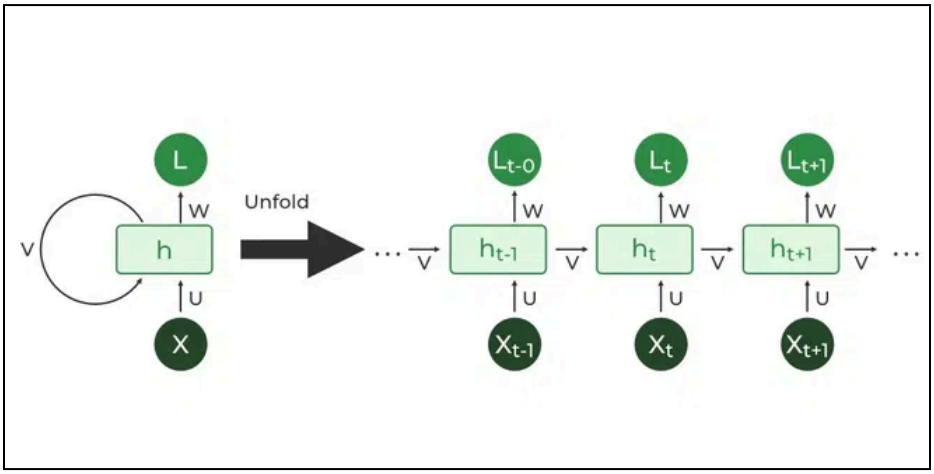
- **Fully Connected**

  Dengan input yang sudah di-*flatten*. Input sudah siap dikalikan dengan bobot dan dijumlahkan dengan bias yang ada lalu dimasukkan fungsi aktivasi layaknya sebuah layer FFNN.



Gambar 6 - Fully Connected Layer

2. Recurrent Neural Network (RNN)



Gambar 7 - Arsitektur RNN

RNN terdiri dari neuron yang memiliki loop (umpan balik) yang memungkinkan informasi dari langkah sebelumnya (t-1) diteruskan ke langkah sekarang (t). Setiap langkah waktu dalam urutan memiliki tiga komponen utama:

a. Input saat ini (x□)
b. Hidden state sebelumnya (h□-1)
c. Hidden state saat ini (h□)

Perhitungan hidden state pada waktu t dilakukan dengan persamaan:
h□ = f(W_xx□ + W□h□-1 + b)

Keterangan:

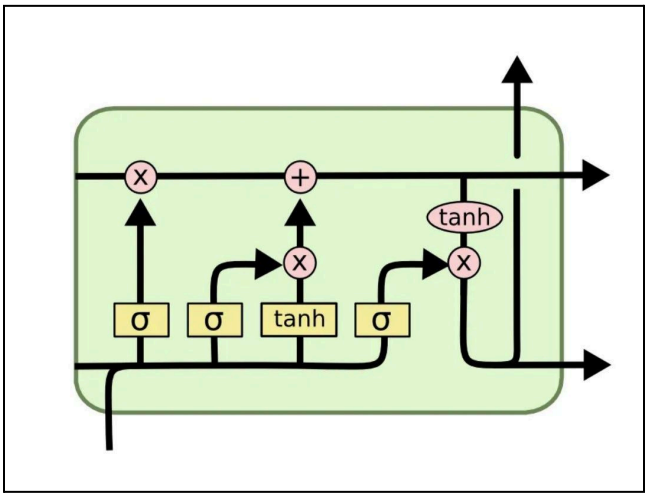
- W_x adalah bobot input ke hidden state.

- W□ adalah bobot dari hidden state sebelumnya.
- b adalah bias.
- f adalah fungsi aktivasi, biasanya tanh atau ReLU.

Output dari jaringan bisa diambil dari hidden state (h□) atau melalui lapisan tambahan (output layer):

$\hat{y}_t = \text{softmax}(W_o h_t + b_o)$

3. Long Short Term Memory (LSTM)



Gambar 8 - Arsitektur LSTM

LSTM memiliki dua state utama yang dibawa dari waktu ke waktu:

a. Hidden state: h□ – digunakan untuk output.
b. Cell state: c□ – digunakan sebagai memori jangka panjang.

Pada setiap langkah waktu t, input x□, hidden state h□₋₁, dan cell state c□₋₁ digunakan untuk menghitung state berikutnya.

Terdapat 4 "gate" utama dalam LSTM:

1. **Forget Gate**: Menentukan informasi apa dari cell state sebelumnya yang harus dilupakan.
   f□ = σ(W_f · [h□₋₁, x□] + b_f)

2. **Input Gate**: Menentukan informasi baru apa yang akan disimpan dalam cell state.
   g□ = tanh(W_g · [h□₋₁, x□] + b_g) [tanh activation]

$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$ [input activation]

3. **Cell State Update**: Cell state diperbarui dengan menggabungkan informasi lama dan baru.
   $c_t = f_t * c_{t-1} + i_t * g_t$

4. **Output Gate**: Menentukan bagian dari cell state yang akan digunakan untuk menghasilkan output hidden state h_t.
   $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$
   $h_t = o_t * \tanh(c_t)$

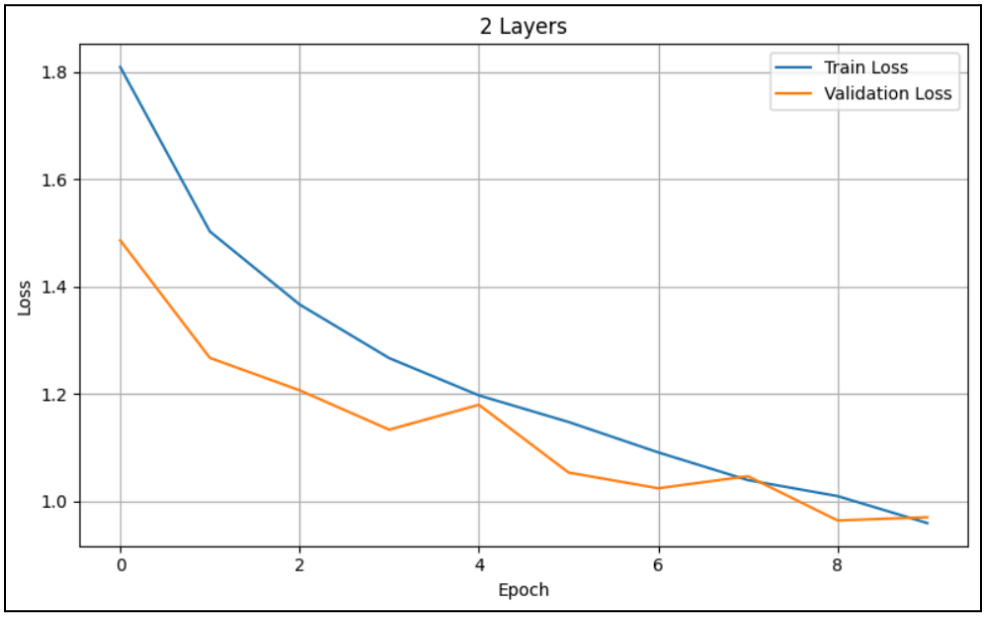
Di sini, σ adalah fungsi sigmoid, * adalah perkalian elemen (element-wise multiplication), dan $[h_{t-1}, x_t]$ menunjukkan konkatenasi dari hidden state sebelumnya dan input saat ini.

2. Hasil Pengujian

1. Convolutional Neural Network (CNN)

a. Jumlah Layer Konvolusi

i. 2 Layers



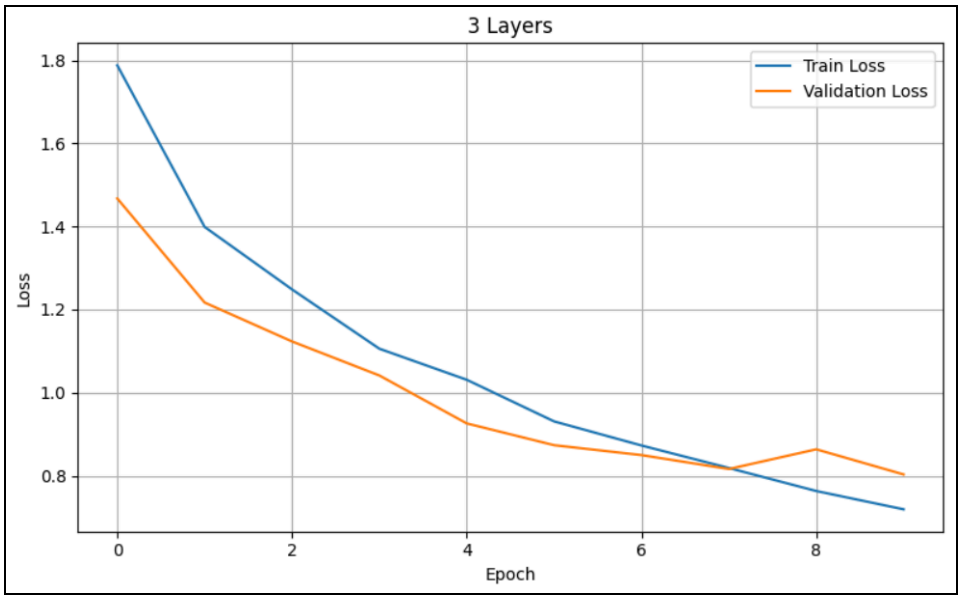

```
Layers:

layers.Conv2D(32, (3, 3), padding="same",
name='conv1', input_shape=(32, 32, 3)),
layers.ReLU(),
```

```
layers.MaxPooling2D(pool_size=(2, 2)),

layers.Conv2D(64, (3, 3), padding='same',
name='conv2'),
layers.ReLU(),
layers.MaxPooling2D(pool_size=(2, 2),
strides=1),

layers.Flatten(),
layers.Dropout(0.5),
layers.Dense(128, activation='relu',
name='fc1'),
layers.Dropout(0.5),
layers.Dense(num_classes, activation='softmax',
name='fc2')
```

Accuracy	0.6577
Macro F1	0.6489

Berdasarkan analisis yang dilakukan, model ini memberikan performa yang cukup baik terlihat dari nilai akurasi dan macro F1 nya yaitu 0.6577 dan 0.6489. Berdasarkan grafik yang ditunjukkan diatas, model ini tidak mengalami overfitting karena val loss jarang berada di atas train loss dan ini disebabkan karena adanya layer dropout yang membantu berkurangnya resiko overfitting. Namun, adanya dropout bisa juga mengurangi nilai akurasi tetapi ini lebih direkomendasikan dibandingkan nilai akurasi tinggi tetapi overfitting.

ii. 3 Layers



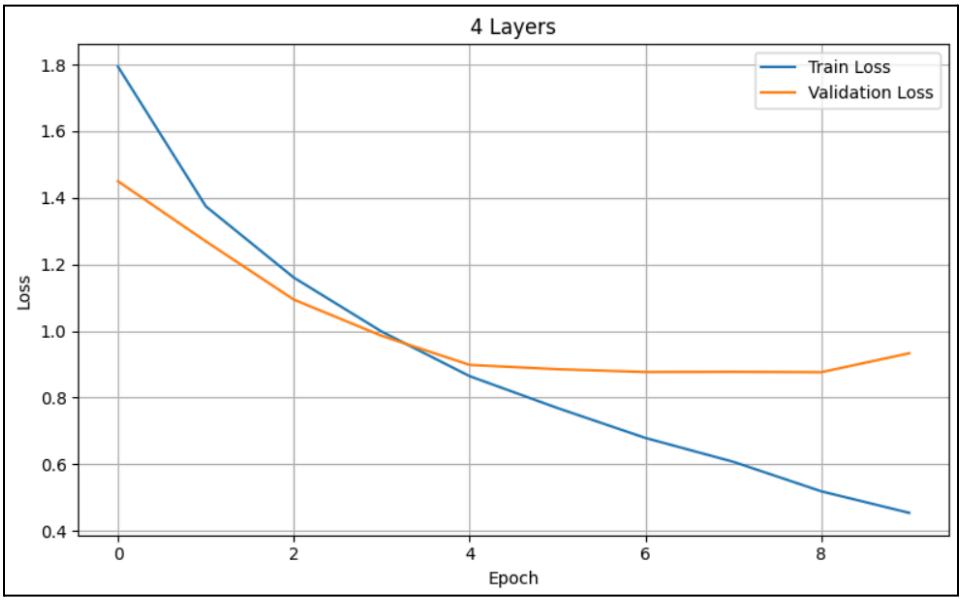

```
Layers:

layers.Conv2D(32, (3, 3), padding="same",
name='conv1', input_shape=(32, 32, 3)),
layers.ReLU(),
layers.MaxPooling2D(pool_size=(2, 2)),

layers.Conv2D(64, (3, 3), padding='same',
name='conv2'),
layers.ReLU(),
layers.MaxPooling2D(pool_size=(2, 2)),

layers.Conv2D(128, (3, 3), padding='same',
name='conv3'),
layers.ReLU(),
layers.MaxPooling2D(pool_size=(2, 2),
strides=1),

layers.Flatten(),
layers.Dropout(0.5),
layers.Dense(256, activation='relu',
name='fc1'),
layers.Dropout(0.5),
layers.Dense(num_classes, activation='softmax',
name='fc2')
```

Accuracy	0.7139

Macro F1	0.7116

Berdasarkan analisis yang dilakukan, model ini memberikan performa yang cukup baik terlihat dari nilai akurasi dan macro F1 nya yaitu 0.7139 dan 0.7116. Performa model ini terbukti lebih baik dibandingkan dengan model dengan 2 layer saja. Berdasarkan grafik yang ditunjukkan di atas, model ini tidak mengalami overfitting karena val loss jarang berada di atas train loss dan ini disebabkan karena adanya layer dropout yang membantu berkurangnya resiko overfitting. Namun, bisa dilihat pada epoch-epoch terakhir validation loss melebihi train loss. Ini artinya, model ini lebih overfitting dibandingkan dengan yang hanya memiliki 2 layer saja.

iii. 4 Layers




```
Layers:

layers.Conv2D(32, (3, 3), padding="same",
name='conv1', input_shape=(32, 32, 3)),
layers.ReLU(),
layers.MaxPooling2D(pool_size=(2, 2)),

layers.Conv2D(64, (3, 3), padding='same',
name='conv2'),
layers.ReLU(),
layers.MaxPooling2D(pool_size=(2, 2)),
```

```
layers.Conv2D(128, (3, 3), padding='same',
name='conv3'),
layers.ReLU(),
layers.MaxPooling2D(pool_size=(2, 2)),

layers.Conv2D(256, (3, 3), padding='same',
name='conv4'),
layers.ReLU(),
layers.MaxPooling2D(pool_size=(2, 2),
strides=1),

layers.Flatten(),
layers.Dropout(0.5),
layers.Dense(512, activation='relu',
name='fc1'),
layers.Dropout(0.5),
layers.Dense(num_classes, activation='softmax',
name='fc2')
```

Accuracy	0.6975
Macro F1	0.6951

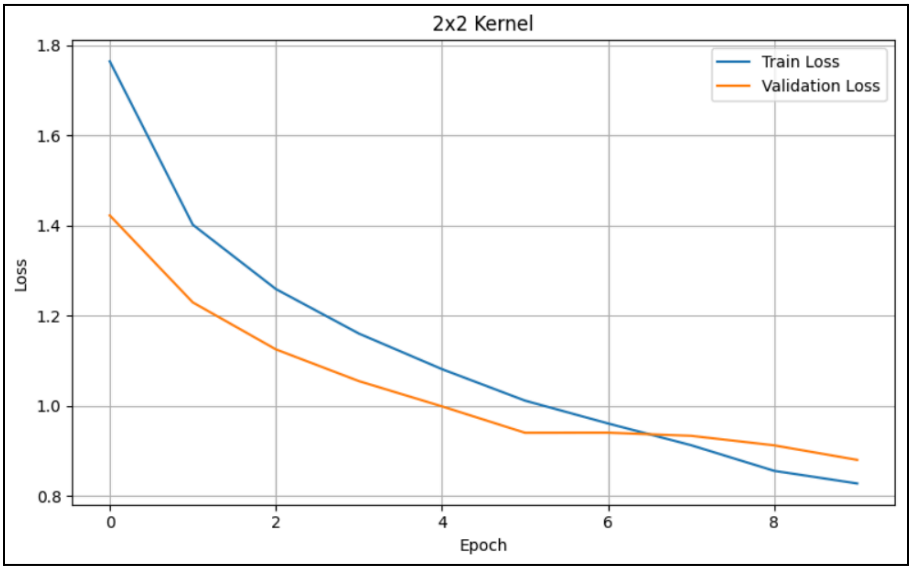
Berdasarkan analisis yang dilakukan, model ini memberikan performa yang cukup baik terlihat dari nilai akurasi dan macro F1 nya yaitu 0.6975 dan 0.6951. Performa model ini terbukti bahwa menambah lebih banyak layer tidak selalu menambah akurasi dari model tersebut. Berdasarkan grafik yang ditunjukkan di atas, model ini mengalami overfitting ketika memasuki epoch yang ke-4. Ini menunjukkan bahwa semakin sering model tersebut dilatih, semakin dia menghafal jawaban dari data latih saja dan bukan menyimpulkan sebuah generalisasi dari data-data yang dilatih tersebut.

iv. Pengaruh Jumlah Layer pada Konvolusi

Berdasarkan analisis yang dilakukan pada beberapa model. Meskipun lebih banyak layer bisa meningkatkan akurasi dan F1 dari model tersebut, akurasi tersebut bisa berkurang ketika model mulai mengalami overfitting. Model mengalami overfitting bila model terlalu besar dan data terlalu kecil untuk memuat model tersebut sehingga model hanya menghafal jawabannya saja dan itu dibuktikan dengan nilai validation loss yang melebihi nilai train loss sehingga akurasi berkurang karena model gagal untuk menyimpulkan generalisasi jawaban tersebut

b. Ukuran Filter per Konvolusi

i. 2x2 Kernel



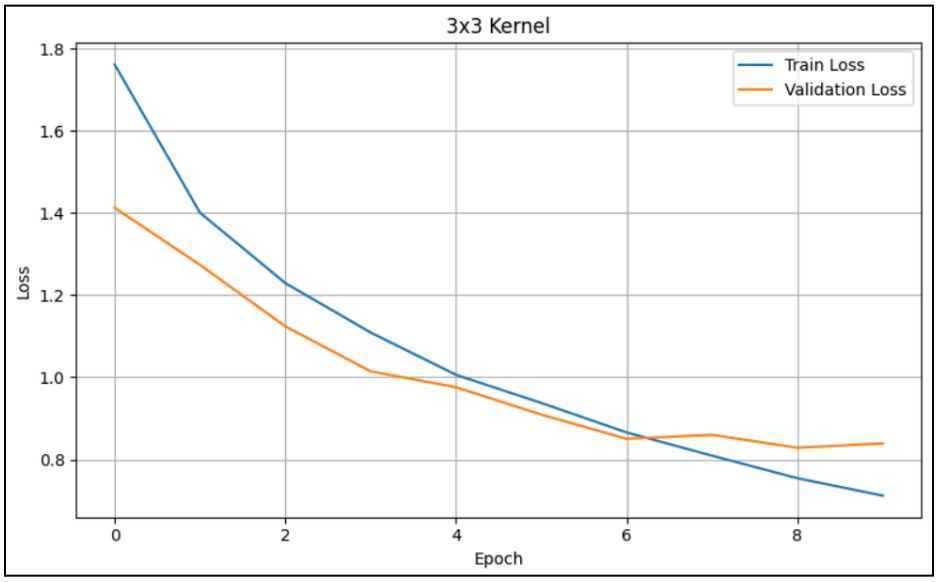

```
Layers:

layers.Conv2D(32, (2, 2), padding="same",
name='conv1')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2))

layers.Conv2D(64, (2, 2), padding='same',
name='conv2')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2))

layers.Conv2D(128, (2, 2), padding='same',
name='conv3')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2), strides=1)

layers.Flatten()
layers.Dropout(0.5)
layers.Dense(256, activation='relu', name='fc1')
layers.Dropout(0.5)
layers.Dense(num_classes, activation='softmax',
name='fc2')
```

Accuracy	0.6920

Macro F1	0.6864

Berdasarkan analisis yang dilakukan pada model ini, model dengan 2x2 Kernel memberikan akurasi sebesar 0.6920 dan macro f1 sebesar 0.6864. Tidak hanya itu, grafik menunjukkan bahwa validation loss lebih besar dibandingkan dengan training loss pada epoch terakhir sehingga model ini tidak terlalu overfitting. Kurangnya overfitting juga dibantu dengan adanya layer Dropout.

ii. 3x3 Kernel



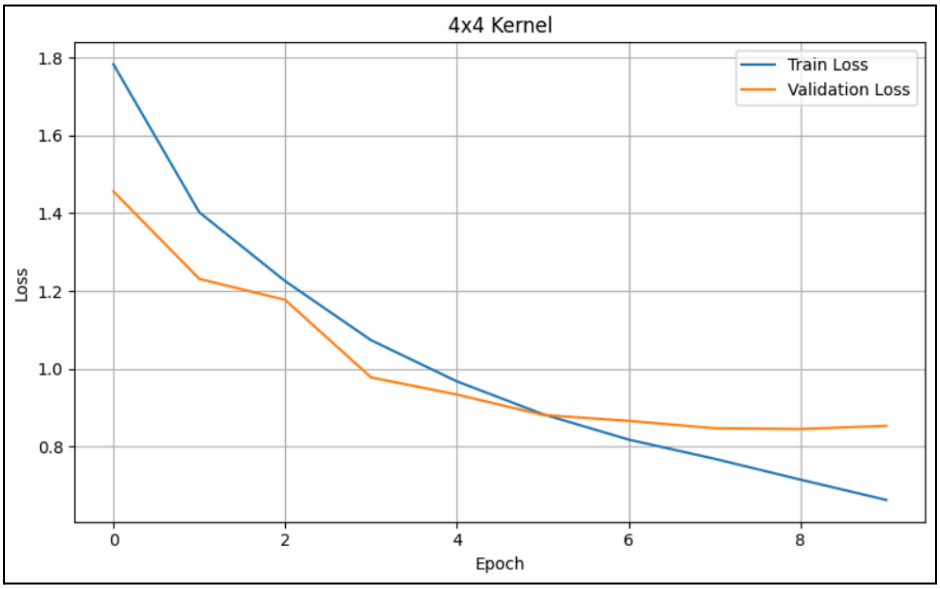

```
Layers:

layers.Conv2D(32, (3, 3), padding="same",
name='conv1')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2))

layers.Conv2D(64, (3, 3), padding='same',
name='conv2')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2))

layers.Conv2D(128, (3, 3), padding='same',
name='conv3')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2), strides=1)
```

```
layers.Flatten()
layers.Dropout(0.5)
layers.Dense(256, activation='relu', name='fc1')
layers.Dropout(0.5)
layers.Dense(num_classes, activation='softmax',
name='fc2')
```

Accuracy	0.7070
Macro F1	0.7025

Berdasarkan analisis yang dilakukan pada model ini, model dengan 3x3 Kernel memberikan akurasi sebesar 0.7070 dan macro f1 sebesar 0.7025. Tidak hanya itu, grafik menunjukkan bahwa validation loss lebih besar dibandingkan dengan training loss pada epoch terakhir sehingga model ini tidak terlalu overfitting. Kurangnya overfitting juga dibantu dengan adanya layer Dropout.

iii. 4x4 Kernel




```
Layers:

layers.Conv2D(32, (4, 4), padding="same",
name='conv1')
layers.ReLU()
```

```
layers.MaxPooling2D(pool_size=(2, 2))

layers.Conv2D(64, (4, 4), padding='same',
name='conv2')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2))

layers.Conv2D(128, (4, 4), padding='same',
name='conv3')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2), strides=1)

layers.Flatten()
layers.Dropout(0.5)
layers.Dense(256, activation='relu', name='fc1')
layers.Dropout(0.5)
layers.Dense(num_classes, activation='softmax',
name='fc2')
```

Accuracy	0.7167
Macro F1	0.7184

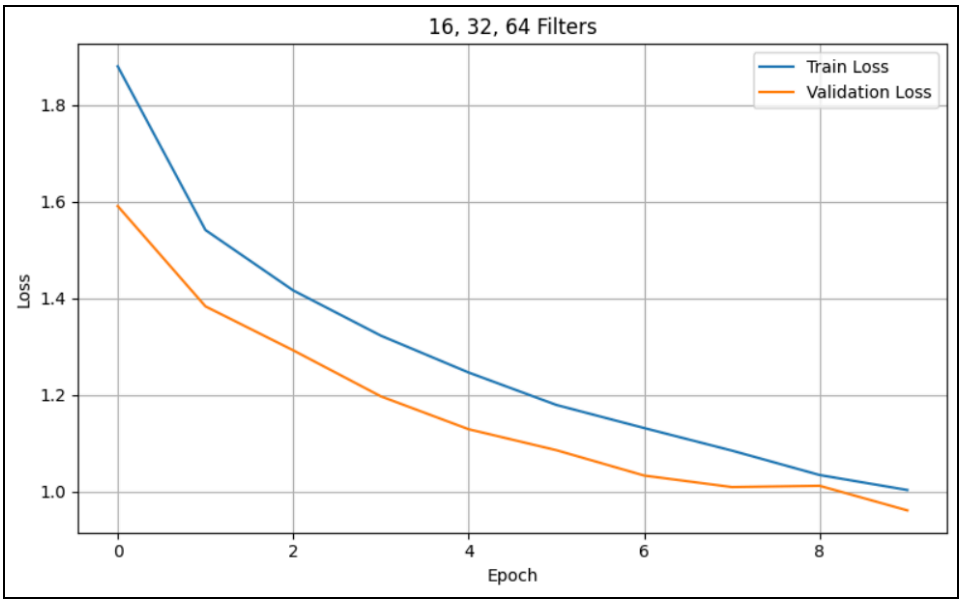
Berdasarkan analisis yang dilakukan pada model ini, model dengan 4x4 Kernel memberikan akurasi sebesar 0.7167 dan macro f1 sebesar 0.7184. Tidak hanya itu, grafik menunjukkan bahwa validation loss lebih besar dibandingkan dengan training loss pada epoch terakhir sehingga model ini tidak terlalu overfitting. Kurangnya overfitting juga dibantu dengan adanya layer Dropout.

iv. **Pengaruh Ukuran Filter per Konvolusi**

Berdasarkan analisis, bertambahnya ukuran filter bisa memberikan akurasi yang lebih besar. Namun, perlu diperhatikan juga karena semakin besar ukuran filter, semakin cepat juga validation loss nya menjadi lebih besar dari train loss sehingga akan lebih terkena resiko overfitting bila ukuran filter diperbesar.

c. Jumlah Filter per Konvolusi

i. 16, 32, 64 Filter

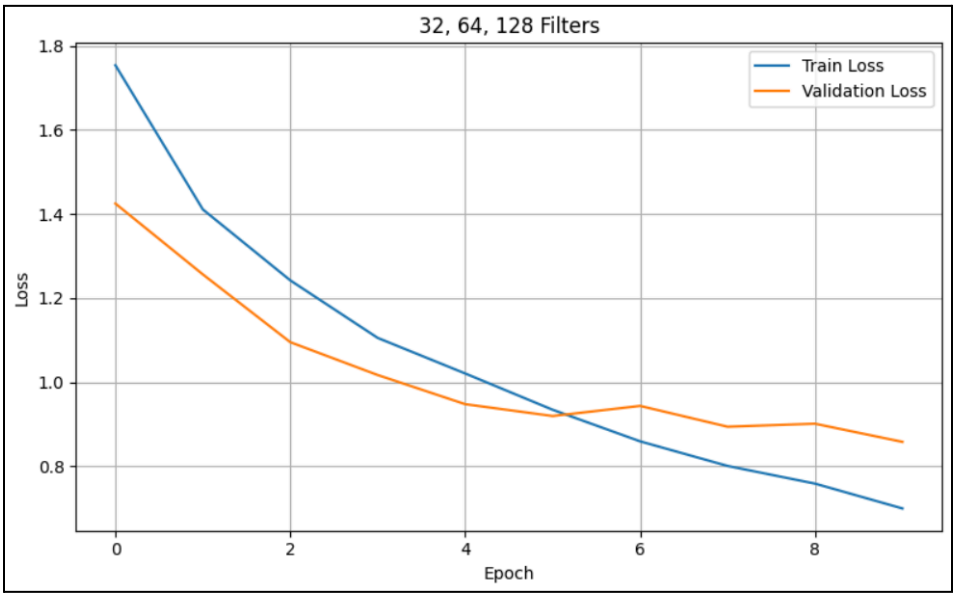



```
Layers:

layers.Conv2D(16, (3, 3), padding="same",
name='conv1')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2))

layers.Conv2D(32, (3, 3), padding='same',
name='conv2')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2))

layers.Conv2D(64, (3, 3), padding='same',
name='conv3')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2), strides=1)

layers.Flatten()
layers.Dropout(0.5)
layers.Dense(128, activation='relu', name='fc1')
layers.Dropout(0.5)
layers.Dense(num_classes, activation='softmax',
name='fc2')
```

Accuracy	0.6587
Macro F1	0.6570

Berdasarkan analisis yang dilakukan pada model ini, model dengan konfigurasi 16, 32, 64 filter ini memberikan akurasi sebesar 0.6587 dan macro f1 sebesar 0.6570. Tidak hanya itu, grafik menunjukkan bahwa validation loss lebih besar dibandingkan dengan training loss pada epoch terakhir sehingga model ini tidak terlalu overfitting. Kurangnya overfitting juga dibantu dengan adanya layer Dropout.

ii. 32, 64, 128 Filter



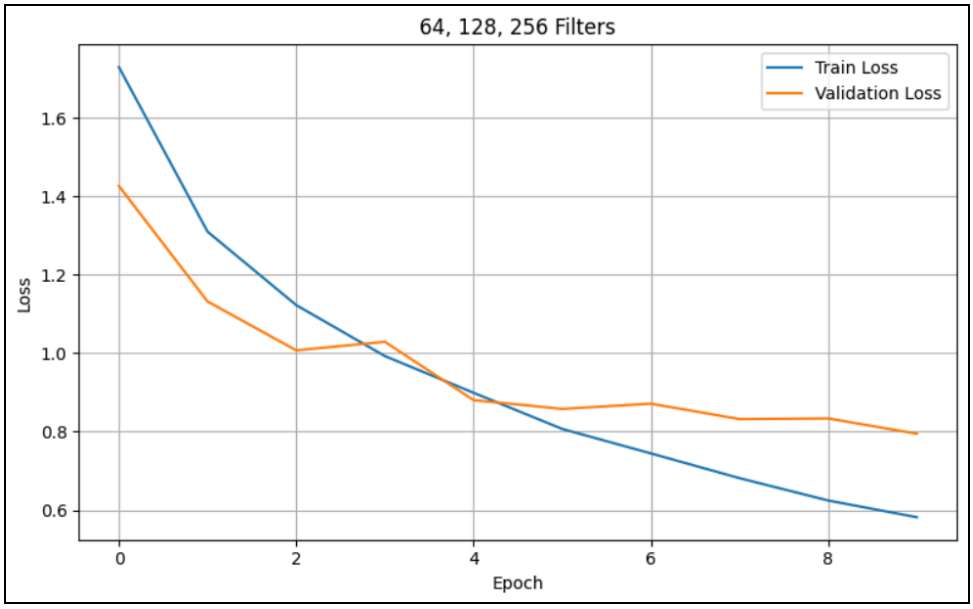

```
Layers:

layers.Conv2D(32, (3, 3), padding="same",
name='conv1')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2))

layers.Conv2D(64, (3, 3), padding='same',
name='conv2')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2))

layers.Conv2D(128, (3, 3), padding='same',
name='conv3')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2), strides=1)

layers.Flatten()
```

```
layers.Dropout(0.5)
layers.Dense(256, activation='relu', name='fc1')
layers.Dropout(0.5)
layers.Dense(num_classes, activation='softmax',
name='fc2')
```

Accuracy	0.6984
Macro F1	0.6947

Berdasarkan analisis yang dilakukan pada model ini, model dengan konfigurasi 32, 64, dan 128 filter memberikan akurasi sebesar 0.6984 dan macro f1 sebesar 0.6947. Tidak hanya itu, grafik menunjukkan bahwa validation loss lebih besar dibandingkan dengan training loss pada epoch terakhir sehingga model ini tidak terlalu overfitting. Kurangnya overfitting juga dibantu dengan adanya layer Dropout.

iii. 64, 128, 256 Filter




```
Layers:

layers.Conv2D(64, (3, 3), padding="same",
name='conv1')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2))
```

```
layers.Conv2D(128, (3, 3), padding='same',
name='conv2')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2))

layers.Conv2D(256, (3, 3), padding='same',
name='conv3')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2), strides=1)

layers.Flatten()
layers.Dropout(0.5)
layers.Dense(512, activation='relu', name='fc1')
layers.Dropout(0.5)
layers.Dense(num_classes, activation='softmax',
name='fc2')
```

Accuracy	0.7284
Macro F1	0.7283

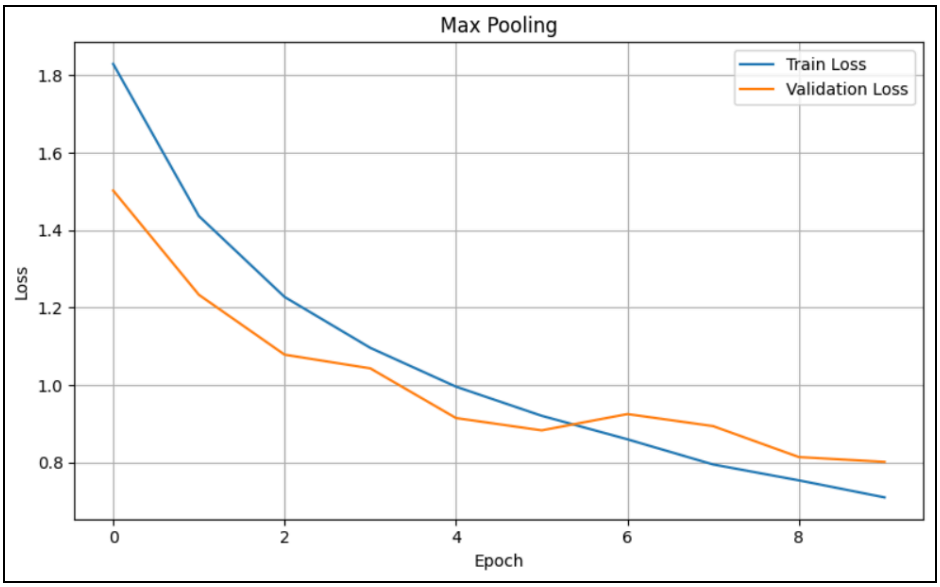
Berdasarkan analisis yang dilakukan pada model ini, model dengan konfigurasi 64, 128, dan 256 filter memberikan akurasi sebesar 0.7284 dan macro f1 sebesar 0.7283. Tidak hanya itu, grafik menunjukkan bahwa validation loss lebih besar dibandingkan dengan training loss pada epoch pertengahan sehingga model ini tidak terlalu overfitting. Kurangnya overfitting juga dibantu dengan adanya layer Dropout.

iv. Pengaruh Jumlah Filter per Layer Konvolusi

Berdasarkan analisis, bertambahnya ukuran filter bisa memberikan akurasi yang lebih besar. Namun, perlu diperhatikan juga karena semakin besar ukuran filter, semakin cepat juga validation loss nya menjadi lebih besar dari train loss sehingga akan lebih terkena resiko overfitting bila ukuran filter diperbesar.

d. Metode Pooling

i. Max Pooling

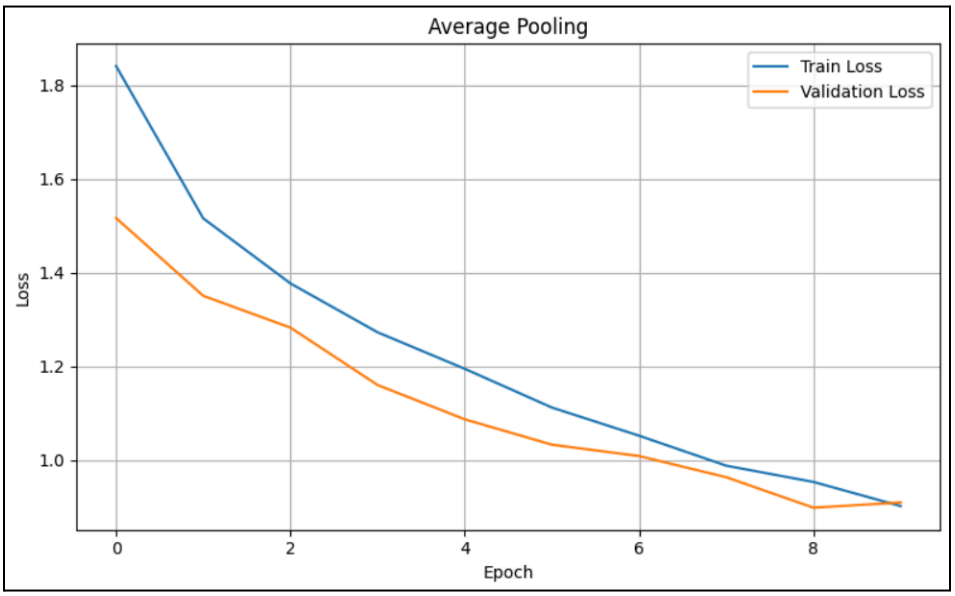



```
Layers:

layers.Conv2D(32, (3, 3), padding="same",
name='conv1')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2))

layers.Conv2D(64, (3, 3), padding='same',
name='conv2')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2))

layers.Conv2D(128, (3, 3), padding='same',
name='conv3')
layers.ReLU()
layers.MaxPooling2D(pool_size=(2, 2), strides=1)

layers.Flatten()
layers.Dropout(0.5)
layers.Dense(256, activation='relu', name='fc1')
layers.Dropout(0.5)
layers.Dense(num_classes, activation='softmax',
name='fc2')
```

Accuracy	0.7111
Macro F1	0.7088

Berdasarkan analisis yang dilakukan pada model ini, model dengan max pooling memberikan akurasi sebesar 0.7111 dan macro f1 sebesar 0.7088. Tidak hanya itu, grafik menunjukkan bahwa validation loss lebih besar dibandingkan dengan training loss pada epoch pertengahan sehingga model ini tidak terlalu overfitting. Kurangnya overfitting juga dibantu dengan adanya layer Dropout.

ii. Average Pooling




```
Layers:

layers.Conv2D(32, (3, 3), padding="same",
name='conv1')
layers.ReLU()
layers.AveragePooling2D(pool_size=(2, 2))

layers.Conv2D(64, (3, 3), padding='same',
name='conv2')
layers.ReLU()
layers.AveragePooling2D(pool_size=(2, 2))

layers.Conv2D(128, (3, 3), padding='same',
name='conv3')
layers.ReLU()
layers.AveragePooling2D(pool_size=(2, 2),
strides=1)

layers.Flatten()
```

```
layers.Dropout(0.5)
layers.Dense(256, activation='relu', name='fc1')
layers.Dropout(0.5)
layers.Dense(num_classes, activation='softmax',
name='fc2')
```

Accuracy	0.6773
Macro F1	0.6719

Berdasarkan analisis yang dilakukan pada model ini, model dengan average pooling memberikan akurasi sebesar 0.6773 dan macro f1 sebesar 0.6719. Tidak hanya itu, grafik menunjukkan bahwa validation loss lebih besar dibandingkan dengan training loss pada epoch pertengahan sehingga model ini tidak terlalu overfitting. Kurangnya overfitting juga dibantu dengan adanya layer Dropout.

iii. Pengaruh Metode Pooling

Berdasarkan analisis, max pooling memberikan jumlah akurasi dan macro f1 yang lebih besar dibandingkan average pooling. Namun perlu diperhatikan bahwa metode average pooling memiliki grafik yang lebih baik dibandingkan max pooling. Grafik max pooling menunjukkan bahwa modelnya lebih overfitting dibandingkan model average pooling.

e. Perbandingan CNN From Scratch dengan CNN Keras

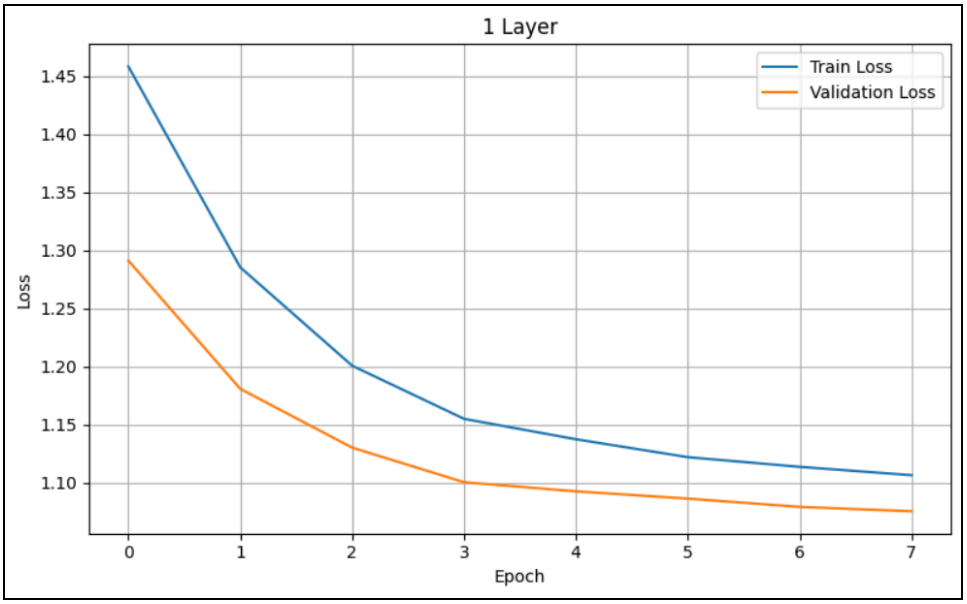
Difference	0.0
Macro F1	1.0

Berdasarkan nilai difference dan F1, terbukti bahwa model CNN sudah sama dengan model yang dibuat oleh Keras sehingga tidak ada kesalahan implementasi pada model CNN from scratch-nya.

2. Recurrent Neural Network (RNN)

a. Jumlah Layer RNN

i. 1 Layer

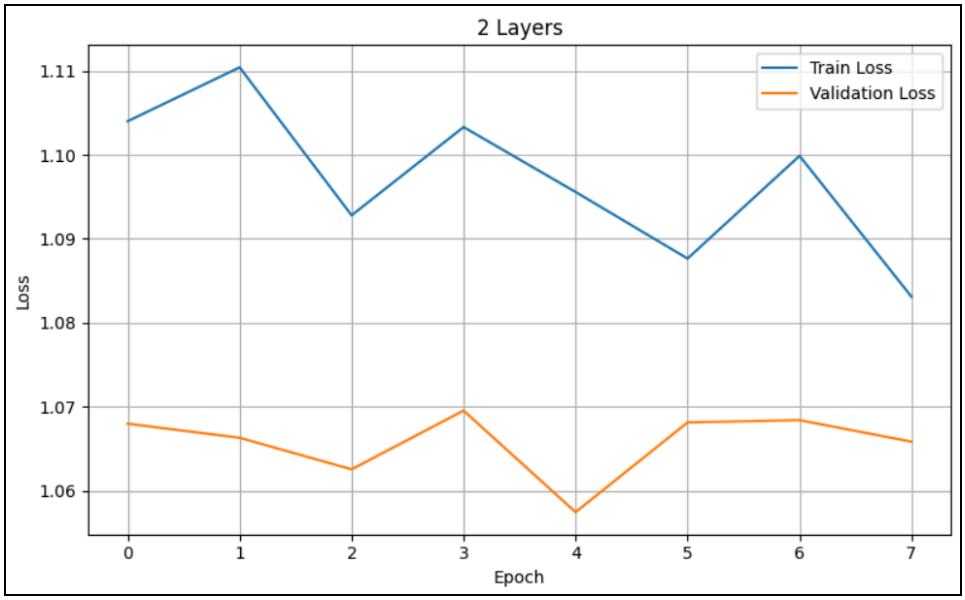



```
Layers:

layers.Embedding(vocabs, 100,
embeddings_initializer='uniform')
layers.SimpleRNN(units=64)
layers.Dropout(0.2)
layers.Dense(units=vocabs, activation='softmax')
```

Accuracy	0.3800
Macro F1	0.1836

Berdasarkan analisis yang diberikan (graf, akurasi, dan f1), akurasinya adalah 0.38 dan macro f1nya adalah 0.1836. Ini menunjukkan bahwa model hanya menjawab 1 label saja dan memerlukan data lebih banyak dan bentuk lain seperti LSTM dan Bidirectional. Berdasarkan graf, tidak ada masalah underfitting atau overfitting pada data tersebut. Itu juga dibantu oleh adanya Dropout.

ii. 2 Layers

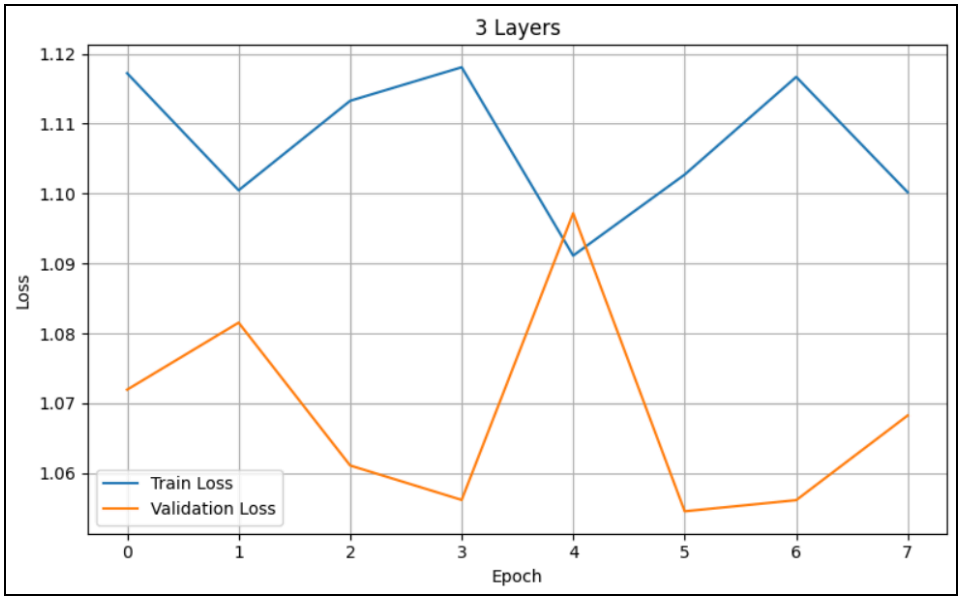



```
Layers:

layers.Embedding(vocabs, 100,
embeddings_initializer='uniform')
layers.SimpleRNN(units=64,
return_sequences=True)
layers.SimpleRNN(units=128)
layers.Dropout(0.2)
layers.Dense(units=vocabs, activation='softmax')
```

Accuracy	0.3800
Macro F1	0.1836

Berdasarkan analisis yang diberikan (graf, akurasi, dan f1), akurasinya adalah 0.38 dan macro f1nya adalah 0.1836. Ini menunjukkan bahwa model hanya menjawab 1 label saja dan memerlukan data lebih banyak dan bentuk lain seperti LSTM dan Bidirectional. Berdasarkan graf, tidak ada masalah underfitting atau overfitting pada data tersebut. Itu juga dibantu oleh adanya Dropout.

iii. 3 Layers




```
Layers:

layers.Embedding(vocabs, 100,
embeddings_initializer='uniform')
layers.SimpleRNN(units=64,
return_sequences=True)
layers.SimpleRNN(units=128,
return_sequences=True)
layers.SimpleRNN(units=256)
layers.Dropout(0.2)
layers.Dense(units=vocabs, activation='softmax')
```

Accuracy	0.3800
Macro F1	0.1836

Berdasarkan analisis yang diberikan (graf, akurasi, dan f1), akurasinya adalah 0.38 dan macro f1nya adalah 0.1836. Ini menunjukkan bahwa model hanya menjawab 1 label saja dan memerlukan data lebih banyak dan bentuk lain seperti LSTM dan Bidirectional. Berdasarkan graf, tidak ada masalah underfitting atau overfitting pada data tersebut. Itu juga dibantu oleh adanya Dropout.

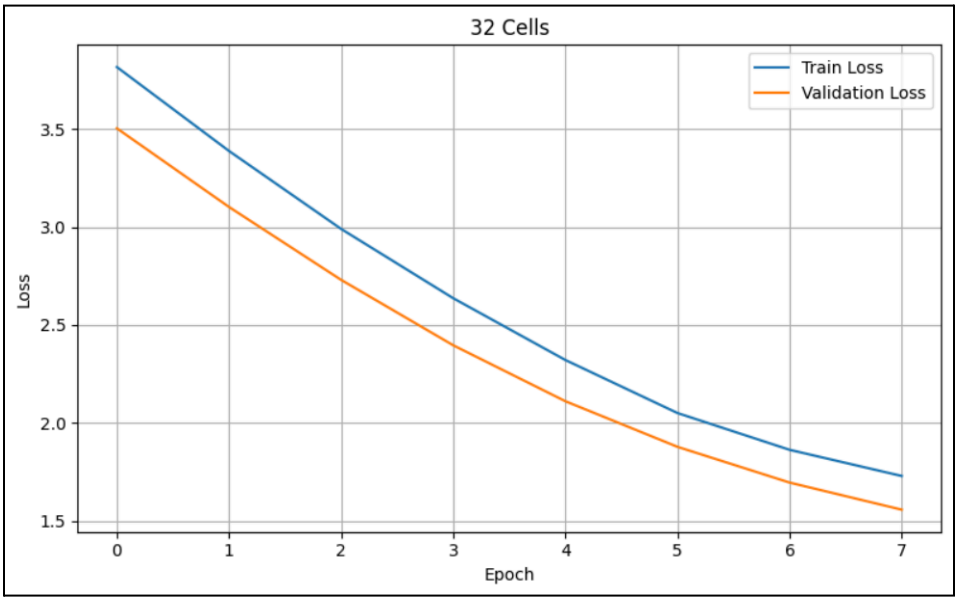
iv. Pengaruh Jumlah RNN Layer

Jumlah RNN tidak terlalu berpengaruh jika datanya terlalu sedikit atau model hanya bisa menjawab 1 label saja. Oleh karena itu RNN di kasus ini

tidak cukup baik sehingga perlu disarankan menggunakan LSTM atau menambah fitur Bidirectional.

b. Banyak Cell RNN per layer

i. 32 Cells

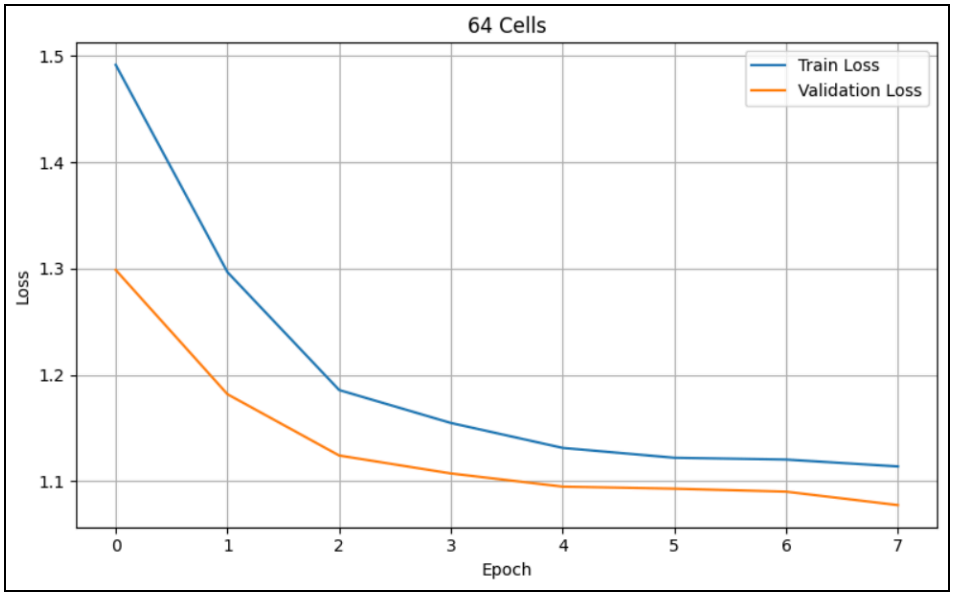



```
Layers:

layers.Embedding(vocabs, 100,
embeddings_initializer='uniform')
layers.SimpleRNN(units=32)
layers.Dropout(0.2)
layers.Dense(units=vocabs, activation='softmax')
```

Accuracy	0.3800
Macro F1	0.1836

Berdasarkan analisis yang diberikan (graf, akurasi, dan f1), akurasinya adalah 0.38 dan macro f1nya adalah 0.1836. Ini menunjukkan bahwa model hanya menjawab 1 label saja dan memerlukan data lebih banyak dan bentuk lain seperti LSTM dan Bidirectional. Berdasarkan graf, tidak ada masalah underfitting atau overfitting pada data tersebut. Itu juga dibantu oleh adanya Dropout.

ii. 64 Cells

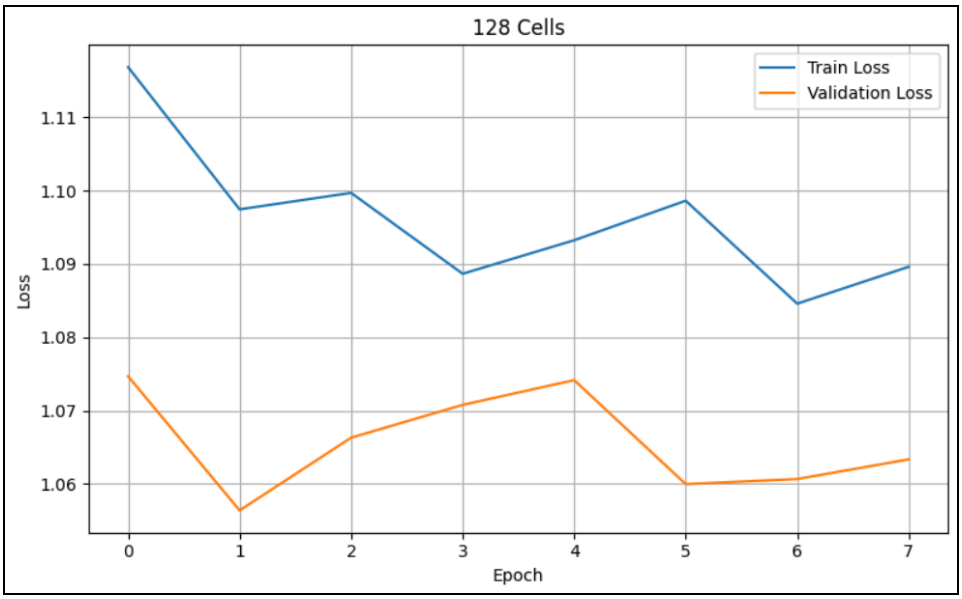



```
Layers:

layers.Embedding(vocabs, 100,
embeddings_initializer='uniform')
layers.SimpleRNN(units=64)
layers.Dropout(0.2)
layers.Dense(units=vocabs, activation='softmax')
```

Accuracy	0.3800
Macro F1	0.1836

Berdasarkan analisis yang diberikan (graf, akurasi, dan f1), akurasinya adalah 0.38 dan macro f1nya adalah 0.1836. Ini menunjukkan bahwa model hanya menjawab 1 label saja dan memerlukan data lebih banyak dan bentuk lain seperti LSTM dan Bidirectional. Berdasarkan graf, tidak ada masalah underfitting atau overfitting pada data tersebut. Itu juga dibantu oleh adanya Dropout.

iii. 128 Cells




```
Layers:

layers.Embedding(vocabs, 100,
embeddings_initializer='uniform')
layers.SimpleRNN(units=128)
layers.Dropout(0.2)
layers.Dense(units=vocabs, activation='softmax')
```

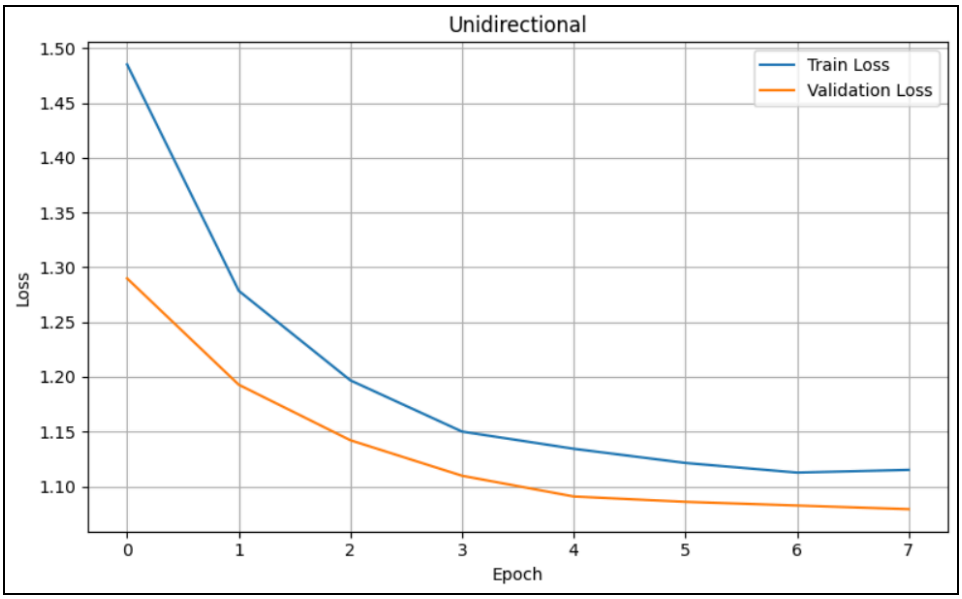
Accuracy	0.3800
Macro F1	0.1836

iv. Pengaruh Banyak Cell RNN per Layer

Jumlah Cell RNN tidak terlalu berpengaruh jika datanya terlalu sedikit atau model hanya bisa menjawab 1 label saja. Oleh karena itu RNN di kasus ini tidak cukup baik sehingga perlu disarankan menggunakan LSTM atau menambah fitur Bidirectional.

c. Unidirectional and Bidirectional

i. Unidirectional

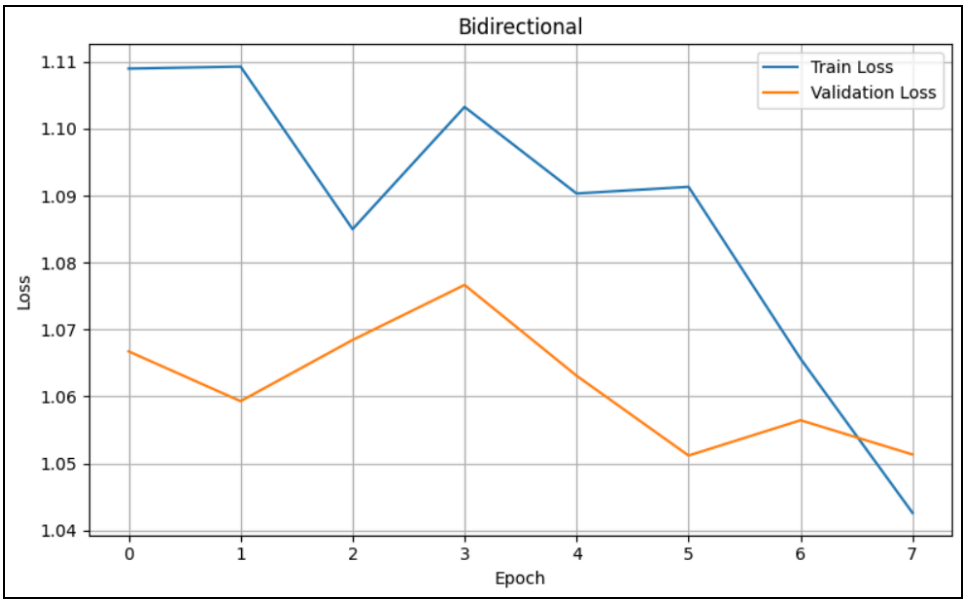



```
Layers:

layers.Embedding(vocabs, 100,
embeddings_initializer='uniform')
layers.SimpleRNN(units=64)
layers.Dropout(0.2)
layers.Dense(units=vocabs, activation='softmax')
```

Accuracy	0.3800
Macro F1	0.1836

Berdasarkan analisis yang diberikan (graf, akurasi, dan f1), akurasinya adalah 0.38 dan macro f1nya adalah 0.1836. Ini menunjukkan bahwa model hanya menjawab 1 label saja dan memerlukan data lebih banyak dan bentuk lain seperti LSTM dan Bidirectional. Berdasarkan graf, tidak ada masalah underfitting atau overfitting pada data tersebut. Itu juga dibantu oleh adanya Dropout.

ii. Bidirectional




```
Layers:

layers.Embedding(vocabs, 100,
embeddings_initializer='uniform')
layers.Bidirectional(layers.SimpleRNN(units=64))
layers.Dropout(0.2)
layers.Dense(units=vocabs, activation='softmax')
```

Accuracy	0.5700
Macro F1	0.4386

Berdasarkan analisis yang diberikan (graf, akurasi, dan f1), akurasinya adalah 0.57 dan macro f1nya adalah 0.4386. Meskipun nilai akurasi dan F1 naik, tetap saja perlu ditingkatkan lagi dengan model seperti LSTM. Ada masalah overfitting pada data tersebut jika ditrain terlalu sering (Tidak ada masalah jika dilatih 10 kali tetapi ada ketika lebih dari itu dan pada gambar sebenarnya model di run 2 kali sehingga epochnya adalah 2 * 8 = 16). Itu juga dibantu oleh adanya Dropout.

iii. Pengaruh Arah pada model

Terlihat bahwa adanya Bidirectional membantu performa dari model RNN ini. Ini bisa terjadi karena kalimat yang awalnya positif lalu anak kalimatnya negatif bisa dinilai positif meskipun intensinya adalah negatif.

Oleh karena itu perlu penilaian dari arah sebaliknya untuk melawan argumen dari arah yang satunya lagi.

d. Perbandingan RNN Keras dengan RNN From Scratch

Unidirectional

Difference	1.4155702962736557e-07
Macro F1	1.0

Bidirectional

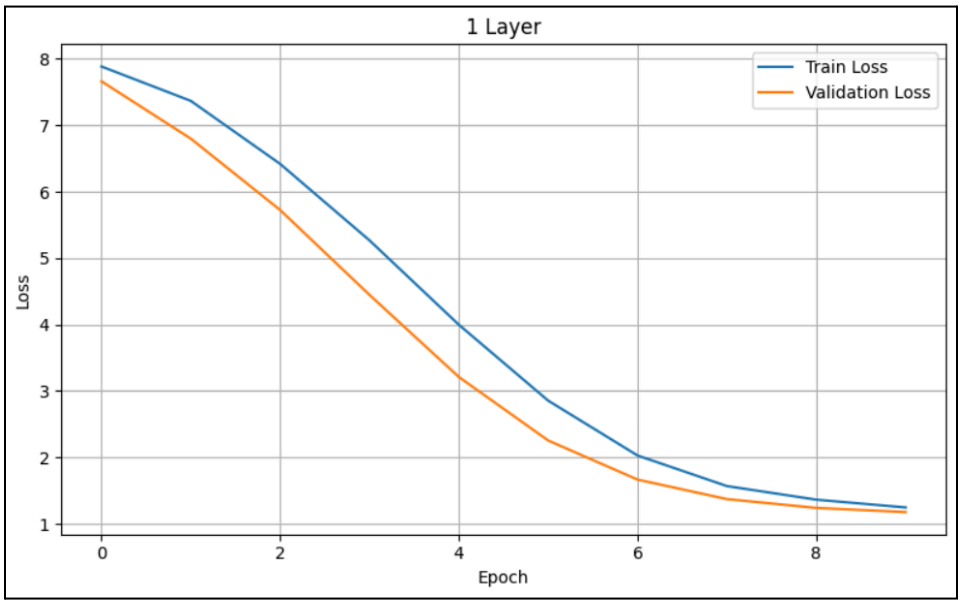
Difference	2.4797589996265234e-07
Macro F1	1.0

Berdasarkan nilai difference dan F1, terbukti bahwa model RNN baik Unidirectional atau Bidirectional sudah sama dengan model yang dibuat oleh Keras sehingga tidak ada kesalahan implementasi pada model RNN from scratch-nya.

3. Long Short Term Memory (LSTM)

a. Jumlah Layer LSTM

i. 1 Layer



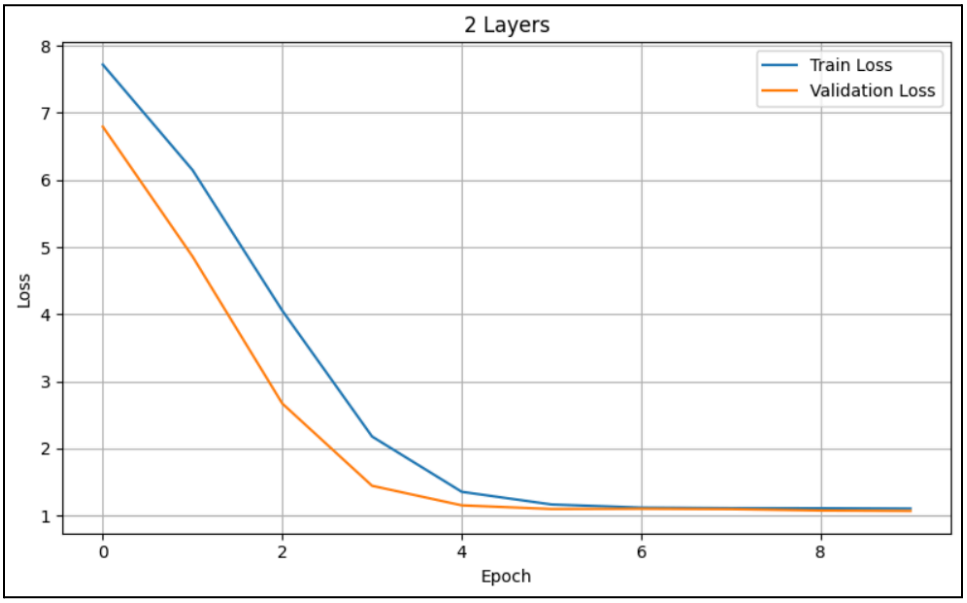

```
Layers:
```

```
layers.Embedding(vocabs, 100,
embeddings_initializer='uniform')
layers.LSTM(units=64)
layers.Dropout(0.2)
layers.Dense(units=vocabs, activation='softmax')
```

Accuracy	0.3800
Macro F1	0.1836

Berdasarkan analisis yang diberikan (graf, akurasi, dan f1), akurasinya adalah 0.38 dan macro f1nya adalah 0.1836. Ini menunjukkan bahwa model hanya menjawab 1 label saja dan memerlukan data lebih banyak dan bentuk lain seperti Bidirectional. Berdasarkan graf, tidak ada masalah underfitting atau overfitting pada data tersebut. Itu juga dibantu oleh adanya Dropout.

ii. 2 Layers



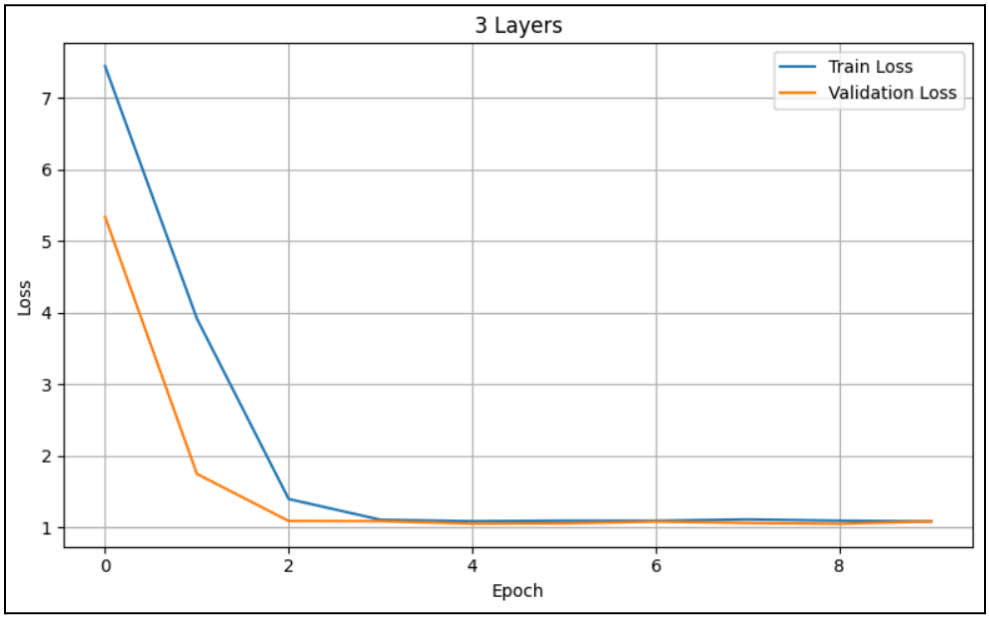

```
Layers:

layers.Embedding(vocabs, 100,
embeddings_initializer='uniform')
layers.LSTM(units=64, return_sequences=True)
layers.LSTM(units=128)
```

```
layers.Dropout(0.2)
layers.Dense(units=vocabs, activation='softmax')
```

Accuracy	0.3800
Macro F1	0.1836

Berdasarkan analisis yang diberikan (graf, akurasi, dan f1), akurasinya adalah 0.38 dan macro f1nya adalah 0.1836. Ini menunjukkan bahwa model hanya menjawab 1 label saja dan memerlukan data lebih banyak dan bentuk lain seperti Bidirectional. Berdasarkan graf, tidak ada masalah underfitting atau overfitting pada data tersebut. Itu juga dibantu oleh adanya Dropout.

iii. 3 Layers




```
Layers:

layers.Embedding(vocabs, 100,
embeddings_initializer='uniform')
layers.LSTM(units=64, return_sequences=True)
layers.LSTM(units=128, return_sequences=True)
layers.LSTM(units=256)
layers.Dropout(0.2)
layers.Dense(units=vocabs, activation='softmax')
```

Accuracy	0.3800
Macro F1	0.1836

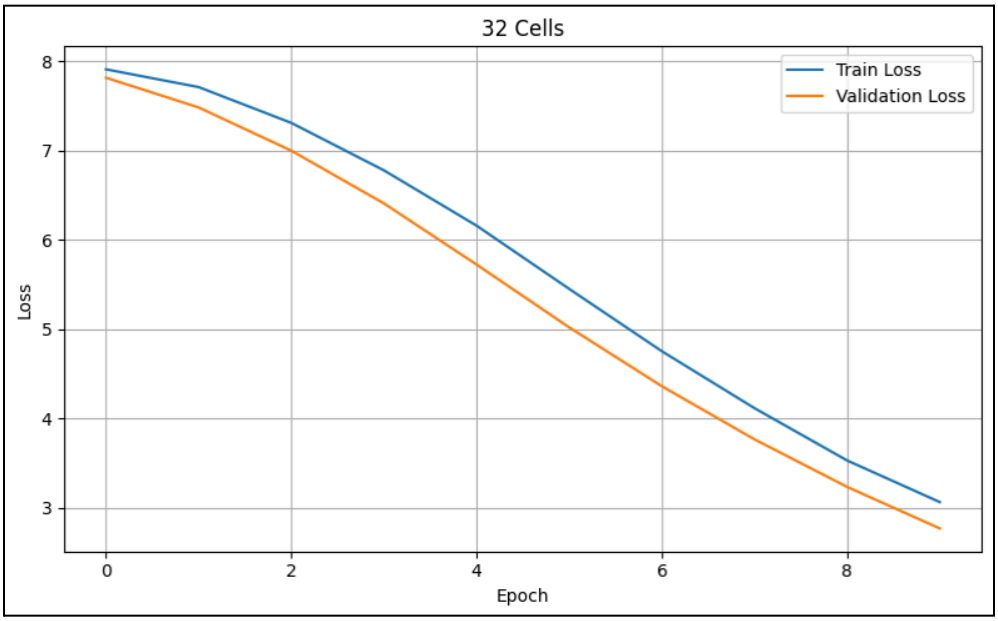
Berdasarkan analisis yang diberikan (graf, akurasi, dan f1), akurasinya adalah 0.38 dan macro f1nya adalah 0.1836. Ini menunjukkan bahwa model hanya menjawab 1 label saja dan memerlukan data lebih banyak dan bentuk lain seperti Bidirectional. Berdasarkan graf, tidak ada masalah underfitting atau overfitting pada data tersebut. Itu juga dibantu oleh adanya Dropout.

iv. Pengaruh Jumlah Layer LSTM

Jumlah Layer pada LSTM tidak terlalu berpengaruh jika datanya terlalu sedikit atau model hanya bisa menjawab 1 label saja. Oleh karena itu LSTM dengan satu arah di kasus ini tidak cukup baik sehingga perlu menambah fitur Bidirectional.

b. Banyak Cell RNN per layer

i. 32 Cells



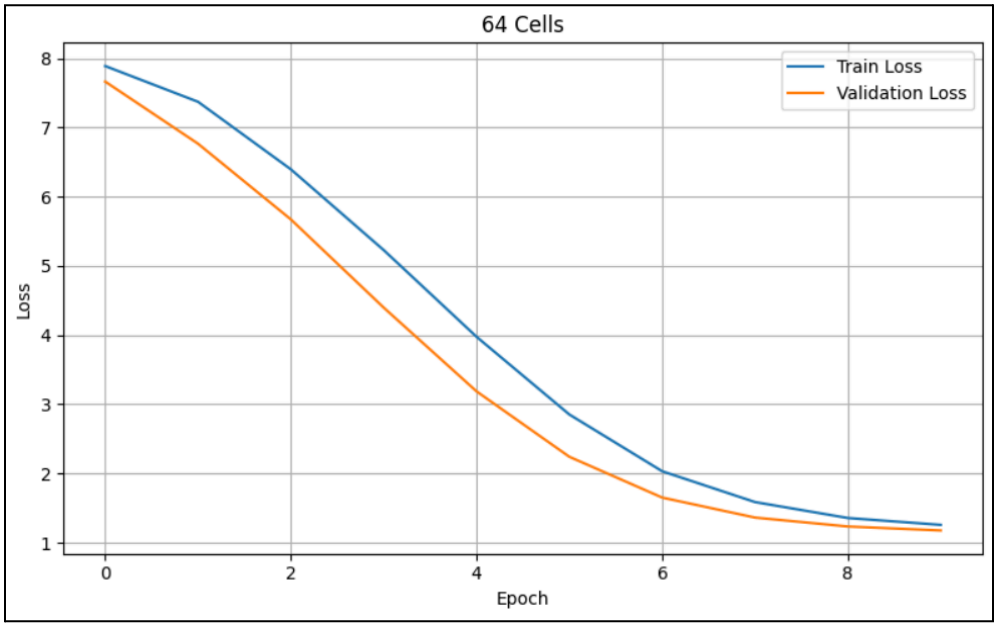

```
Layers:

layers.Embedding(vocabs, 100,
embeddings_initializer='uniform')
layers.LSTM(units=32)
```

```
layers.Dropout(0.2)
layers.Dense(units=vocabs, activation='softmax')
```

Accuracy	0.3800
Macro F1	0.1836

Berdasarkan analisis yang diberikan (graf, akurasi, dan f1), akurasinya adalah 0.38 dan macro f1nya adalah 0.1836. Ini menunjukkan bahwa model hanya menjawab 1 label saja dan memerlukan data lebih banyak dan bentuk lain seperti Bidirectional. Berdasarkan graf, tidak ada masalah underfitting atau overfitting pada data tersebut. Itu juga dibantu oleh adanya Dropout.

ii. 64 Cells

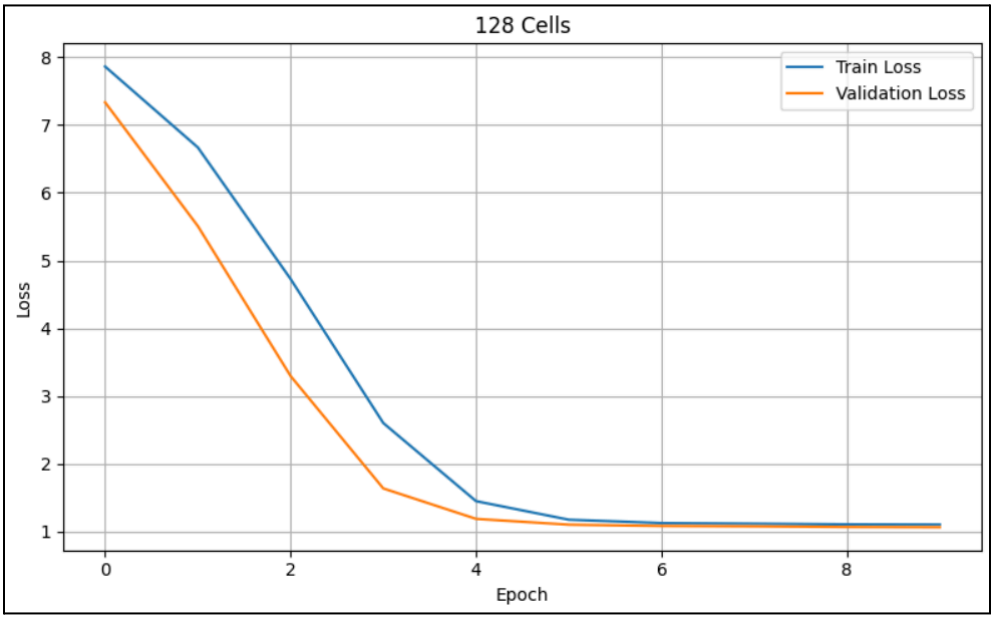



```
Layers:

layers.Embedding(vocabs, 100,
embeddings_initializer='uniform')
layers.LSTM(units=64)
layers.Dropout(0.2)
layers.Dense(units=vocabs, activation='softmax')
```

Accuracy	0.3800
Macro F1	0.1836

Berdasarkan analisis yang diberikan (graf, akurasi, dan f1), akurasinya adalah 0.38 dan macro f1nya adalah 0.1836. Ini menunjukkan bahwa model hanya menjawab 1 label saja dan memerlukan data lebih banyak dan bentuk lain seperti Bidirectional. Berdasarkan graf, tidak ada masalah underfitting atau overfitting pada data tersebut. Itu juga dibantu oleh adanya Dropout.

iii. 128 Cells




```
Layers:

layers.Embedding(vocabs, 100,
embeddings_initializer='uniform')
layers.LSTM(units=128)
layers.Dropout(0.2)
layers.Dense(units=vocabs, activation='softmax')
```

Accuracy	0.3800
Macro F1	0.1836

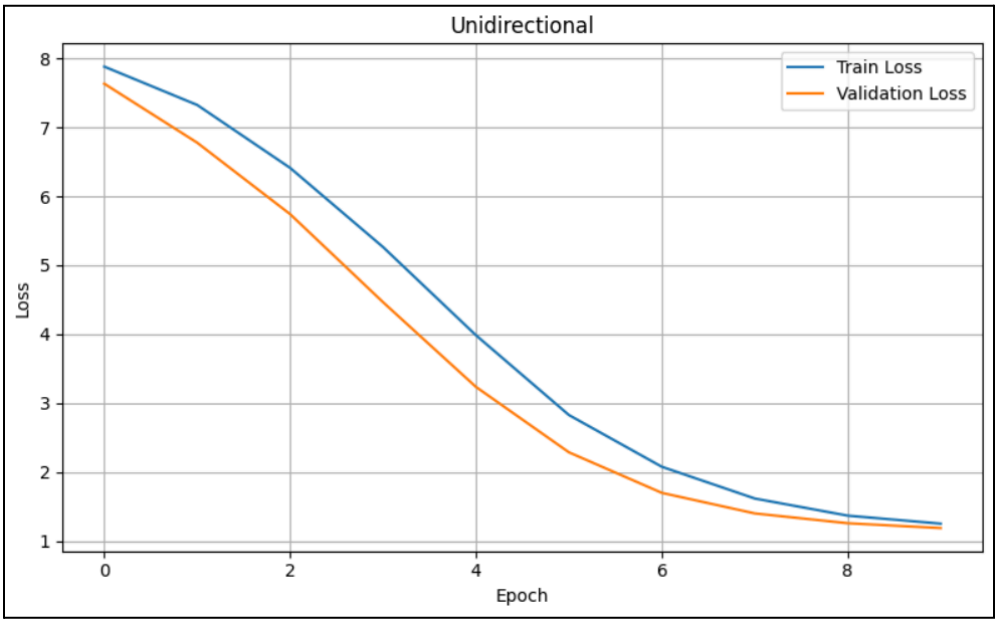
Berdasarkan analisis yang diberikan (graf, akurasi, dan f1), akurasinya adalah 0.38 dan macro f1nya adalah 0.1836. Ini menunjukkan bahwa model hanya menjawab 1 label saja dan memerlukan data lebih banyak dan bentuk lain seperti Bidirectional. Berdasarkan graf, tidak ada masalah underfitting atau overfitting pada data tersebut. Itu juga dibantu oleh adanya Dropout.

iv. Pengaruh Jumlah Cell pada LSTM

Jumlah Cell LSTM tidak terlalu berpengaruh jika datanya terlalu sedikit atau model hanya bisa menjawab 1 label saja. Oleh karena itu LSTM dengan satu arah di kasus ini tidak cukup baik sehingga perlu menambah fitur Bidirectional.

c. Unidirectional and Bidirectional

i. Unidirectional



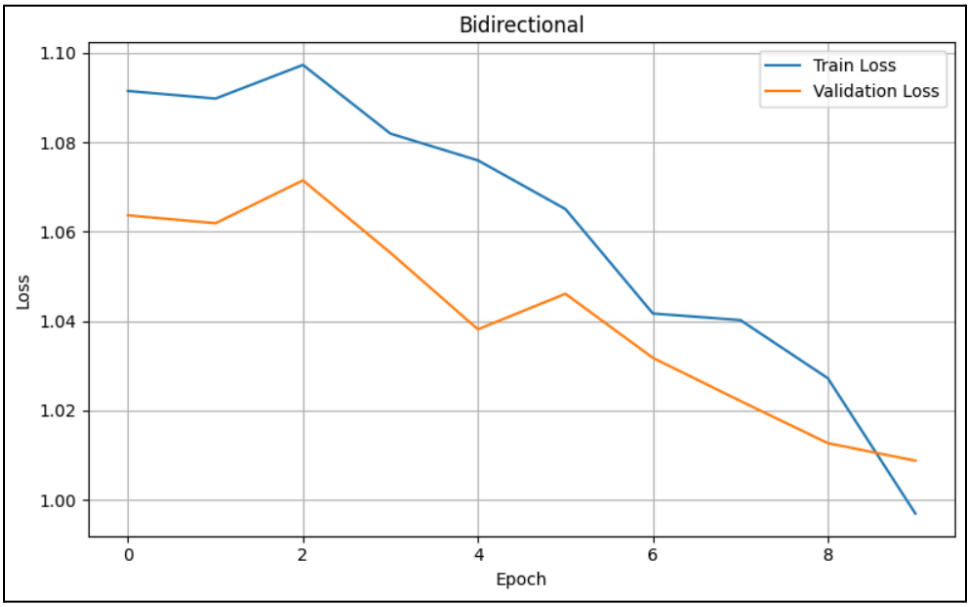

```
Layers:

layers.Embedding(vocabs, 100,
embeddings_initializer='uniform')
layers.LSTM(units=64)
layers.Dropout(0.2)
layers.Dense(units=vocabs, activation='softmax')
```

Accuracy	0.3800

Macro F1	0.1836

Berdasarkan analisis yang diberikan (graf, akurasi, dan f1), akurasinya adalah 0.38 dan macro f1nya adalah 0.1836. Ini menunjukkan bahwa model hanya menjawab 1 label saja dan memerlukan data lebih banyak dan bentuk lain seperti Bidirectional. Berdasarkan graf, tidak ada masalah underfitting atau overfitting pada data tersebut. Itu juga dibantu oleh adanya Dropout.

ii. Bidirectional




```
Layers:

layers.Embedding(vocabs, 100,
embeddings_initializer='uniform')
layers.Bidirectional(layers.LSTM(units=64))
layers.Dropout(0.2)
layers.Dense(units=vocabs, activation='softmax')
```

Accuracy	0.6300
Macro F1	0.4868

Berdasarkan analisis yang diberikan (graf, akurasi, dan f1), akurasinya adalah 0.63 dan macro f1nya adalah 0.4868. Meskipun nilai akurasi dan

F1 naik, tetap saja bisa ditingkatkan lagi dengan mungkin lebih menambah banyak data dan membantu mengurangi overfitting karena graf diatas kelihatannya akan overfitting karena validation loss lebih tinggi daripada train loss pada epoch terakhir.

iii. **Pengaruh Arah pada LSTM**

Terlihat bahwa adanya Bidirectional membantu performa dari model LSTM ini. Ini bisa terjadi karena kalimat yang awalnya positif lalu anak kalimatnya negatif bisa dinilai positif meskipun intensinya adalah negatif. Oleh karena itu perlu penilaian dari arah sebaliknya untuk melawan argumen dari arah yang satunya lagi.

d. Perbandingan LSTM Keras dengan LSTM From Scratch

Unidirectional

Difference	1.3434822786972365e-09
Macro F1	1.0

Bidirectional

Difference	9.468962328601842e-06
Macro F1	0.9639168644557059

Berdasarkan nilai difference dan F1, terbukti bahwa model LSTM Unidirectional sudah sama dengan model yang dibuat oleh Keras sehingga tidak ada kesalahan implementasi pada model LSTM from scratch-nya. Sedangkan pada Bidirectional, terdapat sedikit perbedaan nilai F1 sehingga bisa disimpulkan ada prediksi yang berbeda dari Keras dengan From Scratch. Ini bisa berbeda karena kemungkinan nilai desimal yang memengaruhi nilai final softmaxnya sehingga suatu kelas yang seharusnya punya nilai terbesar bisa kalah besar dengan kelas lain.

Kesimpulan & Saran

A. Kesimpulan

Pada CNN, performa bisa lebih meningkat jika layernya ditambah, jumlah filter diperbanyak, dan ukuran filter diperbesar. Namun perlu diperhatikan bahwa semakin besar atau banyak parameter yang ada di model, semakin besar juga resikonya terkena overfitting sehingga perlu berhati-hati dalam memperbesar model yang dibuat. Lebih baik memiliki akurasi kecil tetapi tidak overfitting dibandingkan akurasi besar tetapi overfitting karena overfitting artinya model hanya menghafal jawabannya saja.

Pada RNN dan LSTM, performa relatif sama saja. Ini bisa disebabkan karena kurangnya data yang dilatih (500 saja) dan model terlalu sering menjawab 1 kelas saja. Oleh karena itu, performa bisa lebih baik jika menggunakan fitur Bidirectional karena dalam sebuah kalimat perlu dinilai kalimatnya dari belakang dan dari depan.

B. Saran

Model-model yang dibuat perlu lebih sering dilatih dan di cek semua terhadap hyperparameter yang tersedia. Selain itu, karena performa yang cukup lama, ada pentingnya untuk menggunakan CuPy dibandingkan dengan NumPy karena CuPy menggunakan GPU dan CUDA untuk komputasinya sehingga waktu yang dibutuhkan untuk menjalankan program tidak terlalu lama.

Pembagian Tugas

NIM	Tugas
13522098	Laporan
13522110	CNN, RNN, LSTM, Laporan
13522118	Laporan

Referensi

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