

IF3270 Pembelajaran Mesin

Implementasi Feedforward Neural Network

Laporan Tugas Besar 1

Disusun untuk memenuhi tugas mata kuliah Pembelajaran Mesin untuk
Semester 6 tahun ajaran 2024 / 2025



Oleh :

Aland Mulia Pratama	13522124
Christian Justin Hendrawan	13522135
Albert Ghazaly	13522150

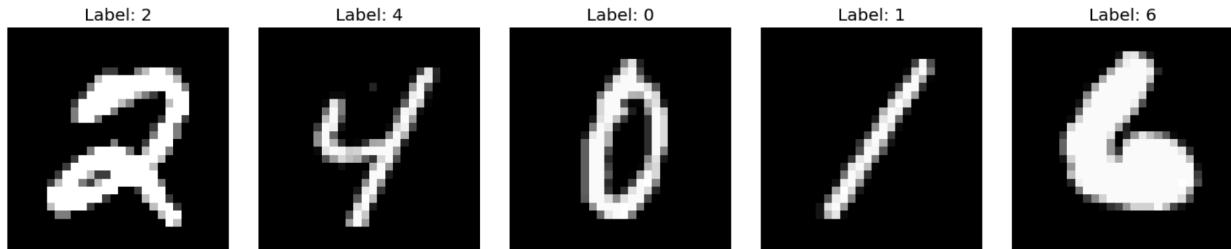
PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

DAFTAR ISI

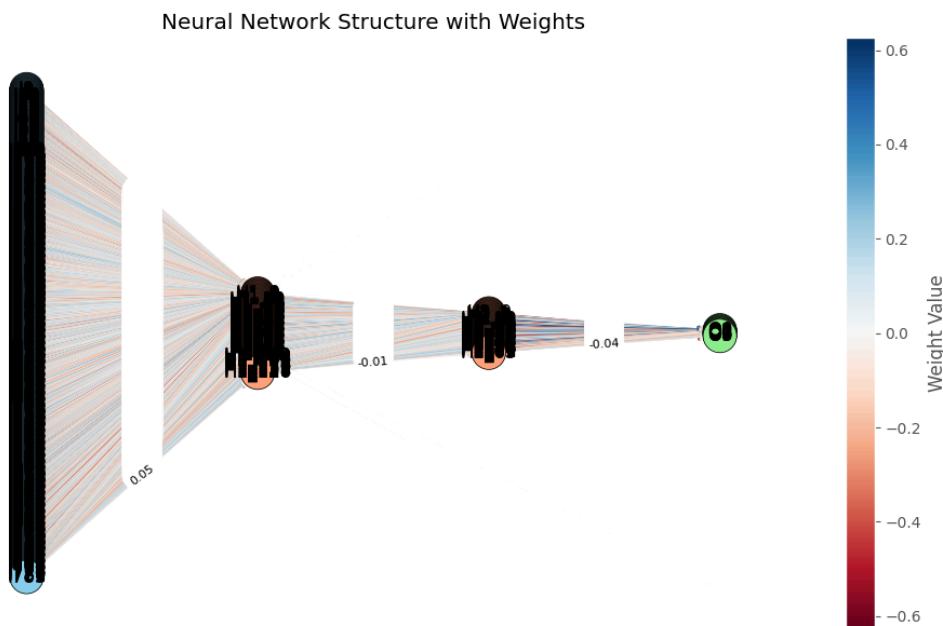
DAFTAR ISI	1
BAB I	
DESKRIPSI PERSOALAN	2
BAB II	
PEMBAHASAN	7
2.1. Penjelasan implementasi Feed Forward Neural Network	7
2.1.1 Deskripsi Kelas, Atribut, dan Metode yang digunakan	7
A. Activation	7
B. Loss	11
C. Initializer	14
D. Layer	17
E. FFNN	20
F. test_main.ipynb	27
2.1.2 Penjelasan Forward Propagation	30
2.1.3 Penjelasan Backward Propagation dan Weight Update	31
2.1.4 Penjelasan Implementasi Regularisasi L1 dan L2	32
2.2. Hasil Pengujian	33
2.2.1 Pengaruh Depth (Banyak Layer) dan Width (Banyak Neuron)	33
2.2.2 Pengaruh Fungsi Aktivasi Hidden Layer	35
2.2.3 Pengaruh Learning Rate	39
2.2.4 Pengaruh Inisialisasi Bobot	41
2.2.5 Pengaruh Regularisasi	45
2.2.6 Perbandingan dengan Library SKLearn	50
BAB III	51
Kesimpulan dan Saran	51
3.1. Kesimpulan	51
3.2. Saran	52
BAB IV	53
Referensi	53
Lampiran	54

BAB I

DESKRIPSI PERSOALAN



Gambar 1.1. Visualisasi Dataset MNIST_784



Gambar 1.2. Visualisasi Feed Forward Neural Network dengan Pembobotan

Dataset MNIST merupakan kumpulan gambar angka tangan yang sering digunakan dalam pembelajaran mesin dan computer vision. Dataset ini terdiri dari gambar grayscale 28x28 piksel, masing-masing merepresentasikan angka dari 0 hingga 9. Dalam spesifikasi tugas besar ini, tugas utamanya adalah membangun model Feedforward Neural Network (FFNN) from scratch yang mampu mengklasifikasikan angka-angka pada dataset MNIST dengan akurasi yang tinggi.

Detail Persoalan

1. Tujuan Utama

Mengembangkan model FFNN untuk memprediksi label angka dari gambar input serta mengukur performa model dengan beberapa variasi percobaan menggunakan metrik seperti akurasi.

2. Input (Dataset)

Gambar digital angka tangan dalam format 28x28 piksel, dinormalisasi untuk memastikan nilainya berada dalam range tertentu (biasanya antara 0 dan 1).

3. Input (Model)

FFNN yang akan diimplementasikan harus mendukung beberapa parameter utama untuk fleksibilitas dan pengendalian eksperimen:

a) Jumlah Neuron

FFNN menerima input berupa jumlah neuron pada setiap layer (input, hidden, dan output). Input ini memungkinkan pengguna untuk menentukan arsitektur jaringan sesuai dengan kebutuhan dataset atau kasus yang sedang dipelajari.

b) Fungsi Aktivasi

Nama Fungsi Aktivasi	Definisi Fungsi	Turunan Pertama
Linear	$Linear(x) = x$	$\frac{d(Linear(x))}{dx} = 1$
ReLU	$ReLU(x) = \max(0, x)$	$\frac{d(ReLU(x))}{dx} = \begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}$
Sigmoid	$\sigma(x) = \frac{1}{1 + e^{-x}}$	$\frac{d(\sigma(x))}{dx} = \sigma(x)(1 - \sigma(x))$
Hyperbolic Tangent (\tanh)	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$\frac{d(\tanh(x))}{dx} = \left(\frac{2}{e^x - e^{-x}}\right)^2$

Softmax	<p>Untuk vector $\vec{x} = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$,</p> $\text{softmax}(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$	<p>Untuk vector $\vec{x} = (x_1, \dots, x_n) \in \mathbb{R}^n$,</p> $\frac{d(\text{softmax}(\vec{x})_i)}{d\vec{x}} = \begin{bmatrix} \frac{\partial(\text{softmax}(\vec{x})_1)}{\partial x_1} & \dots & \frac{\partial(\text{softmax}(\vec{x})_1)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial(\text{softmax}(\vec{x})_n)}{\partial x_1} & \dots & \frac{\partial(\text{softmax}(\vec{x})_n)}{\partial x_n} \end{bmatrix}$ <p>Dimana untuk $i, j \in \{1, \dots, n\}$,</p> $\frac{\partial(\text{softmax}(\vec{x})_i)}{\partial x_j} = \text{softmax}(\vec{x})_i (\delta_{i,j} - \text{softmax}(\vec{x})_j)$ $\delta_{i,j} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$
---------	--	---

c) Fungsi Loss

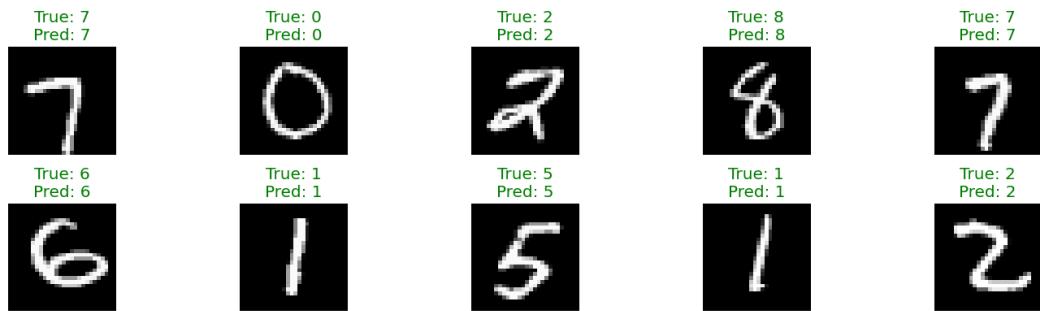
Nama Fungsi Loss	Definisi Fungsi	Turunan Pertama
<u>MSE</u>	$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$	$= -\frac{2}{n} \sum_{i=1}^n (y_i - \hat{y}_i) \frac{\partial \hat{y}_i}{\partial W}$
<u>Binary Cross-Entropy</u>	$\mathcal{L}_{BCE} = -\frac{1}{n} \sum_{i=1}^n (y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i))$ <p>y_i = Actual binary label (0 or 1) \hat{y}_i = Predicted value of y_i n = Batch size</p>	$= -\frac{1}{n} \sum_{i=1}^n \frac{\hat{y}_i - y_i}{\hat{y}_i(1 - \hat{y}_i)} \frac{\partial \hat{y}_i}{\partial W}$
<u>Categorical Cross-Entropy</u>	$\mathcal{L}_{CCE} = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^C (y_{ij} \log \hat{y}_{ij})$ <p>y_{ij} = Actual value of instance i for class j \hat{y}_{ij} = Predicted value of y_{ij} C = Number of classes n = Batch size</p>	$= -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^C \frac{y_{ij}}{\hat{y}_{ij}} \frac{\partial \hat{y}_{ij}}{\partial W}$

- d) Inisialisasi Bobot
- Zero initialization
 - Random dengan distribusi uniform.
 - Menerima parameter lower bound (batas minimal) dan upper bound (batas maksimal)
 - Menerima parameter seed untuk reproducibility
 - Random dengan distribusi normal.
 - Menerima parameter mean dan variance
 - Menerima parameter seed untuk reproducibility

4. Output (Prediksi Model)

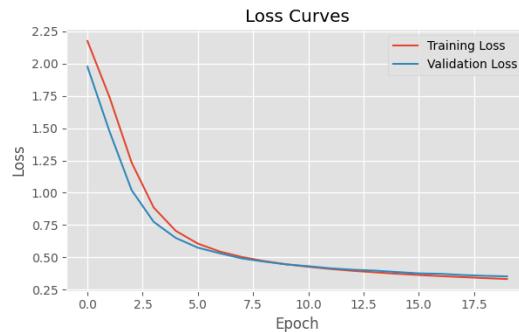
a) Prediksi Label Angka

Model memprediksi label angka (0 hingga 9) untuk setiap gambar input dari dataset MNIST.



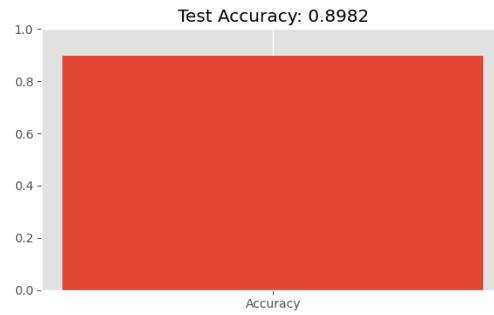
b) Kurva Loss

Menampilkan kurva perubahan nilai loss selama proses pelatihan. Kurva tersebut meliputi training loss dan validation loss pada setiap epoch.



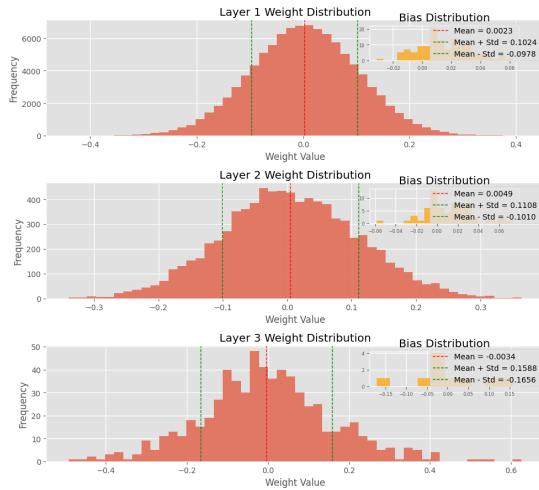
c) Hasil Akurasi pada Data Uji

Memberikan nilai akurasi model pada data uji setelah pelatihan selesai. Ditampilkan dalam bentuk numerik dan diagram.



d) Visualisasi Distribusi Bobot

Distribusi bobot setiap layer dapat divisualisasikan dalam bentuk histogram atau density plot. Membantu memantau pola perubahan bobot selama pelatihan.



BAB II

PEMBAHASAN

2.1. Penjelasan implementasi Feed Forward Neural Network

Pada bagian ini, akan dibahas secara mendetail mengenai implementasi Feedforward Neural Network yang telah kami buat. Pembahasan mencakup deskripsi kelas yang digunakan beserta atribut dan metodenya. Selain itu, terdapat pembahasan mengenai mekanisme forward dan backward propagation.

2.1.1 Deskripsi Kelas, Atribut, dan Metode yang digunakan

A. Activation

File activation.py berisi implementasi berbagai fungsi aktivasi yang digunakan dalam jaringan saraf tiruan (Neural Network). Fungsi aktivasi ini memiliki peran penting dalam menentukan bagaimana informasi diproses dalam setiap neuron, terutama dalam memodelkan hubungan non-linear. File ini menggunakan pendekatan berbasis kelas (class) dengan konsep pewarisan (inheritance). Semua fungsi aktivasi diwariskan dari kelas dasar Activation, yang berfungsi sebagai kerangka utama untuk semua fungsi aktivasi. Kelas Activation berfungsi sebagai kelas abstrak yang mendefinisikan metode dasar untuk semua fungsi aktivasi, yaitu:

- `activate(x)` → Menghitung nilai fungsi aktivasi dengan input x yakni nilai z (pre-activation output).
- `derivative(x)` → Menghitung turunan fungsi aktivasi, yang berguna dalam proses backpropagation dengan input x yang merupakan nilai z (pre-activation output yang tersimpan pada forward propagation).
- `get_activation(name)` → Merupakan factory method yang memungkinkan pemilihan fungsi aktivasi berdasarkan nama yang diberikan. Metode ini hanya didefinisikan di dalam kelas dasar Activation, dan tidak diimplementasikan di kelas turunannya.

```
● ● ●

1  class Activation:
2      @staticmethod
3      def activate(x):
4          raise NotImplementedError
5
6      @staticmethod
7      def derivative(x):
8          raise NotImplementedError
9
10     @classmethod
11     def get_activation(cls, name):
12         activations = {
13             'linear': Linear,
14             'relu': ReLU,
15             'sigmoid': Sigmoid,
16             'tanh': Tanh,
17             'softmax': Softmax,
18             'swish': Swish,
19             'leakyrelu': LeakyReLU
20         }
21         if name.lower() in activations:
22             return activations[name.lower()]
23         else:
24             raise ValueError(f"Activation function '{name}' not supported. Choose from: {list(activations.keys())}")


```

Setiap fungsi aktivasi diturunkan dari kelas Activation, dengan implementasi metode activate(x) dan derivative(x) masing-masing. Fungsi-fungsi aktivasi yang tersedia dalam file ini meliputi Linear, ReLU (Rectified Linear Unit), Sigmoid, Tanh (Hyperbolic Tangent), Softmax, Swish, dan Leaky ReLU. Rumus untuk fungsi aktivasi dapat dilihat pada deskripsi persoalan. Berikut adalah implementasinya.

```
● ● ●

1  class Linear(Activation):
2      """fungsi aktivasi Linear : f(x) = x"""
3      @staticmethod
4      def activate(x):
5          return x
6
7      @staticmethod
8      def derivative(x):
9          return np.ones_like(x)


```

```
● ● ●  
1 class ReLU(Activation):  
2     """fungsi aktivasi ReLU: f(x) = max(0, x)"""  
3     @staticmethod  
4     def activate(x):  
5         return np.maximum(0, x)  
6  
7     @staticmethod  
8     def derivative(x):  
9         return np.where(x > 0, 1, 0)
```

```
● ● ●  
1 class Sigmoid(Activation):  
2     """Fungsi aktivasi sigmoid: f(x) = 1 / (1 + exp(-x))"""  
3     @staticmethod  
4     def activate(x):  
5         return 1 / (1 + np.exp(-np.clip(x, -500, 500)))  
6  
7     @staticmethod  
8     def derivative(x):  
9         s = Sigmoid.activate(x)  
10        return s * (1 - s)
```

```
● ● ●  
1 class Tanh(Activation):  
2     """Fungsi aktivasi hyperbolic tangent: f(x) = (exp(x) - exp(-x)) / (exp(x) + exp(-x))"""  
3     @staticmethod  
4     def activate(x):  
5         return np.tanh(x)  
6  
7     @staticmethod  
8     def derivative(x):  
9         return 1 - np.tanh(x)**2
```

```
● ● ●  
1 class Softmax(Activation):  
2     """Fungsi aktivasi softmax: f(x_i) = exp(x_i) / sum(exp(x_j))"""  
3     @staticmethod  
4     def activate(x):  
5         # Membatasi nilai input untuk menghindari underflow/overflow  
6         x = np.clip(x, -1e10, 1e10)  
7         exp_x = np.exp(x - np.max(x, axis=1, keepdims=True))  
8         return exp_x / np.sum(exp_x, axis=1, keepdims=True)  
9  
10    @staticmethod  
11    def derivative(x):  
12        """  
13            Turunan softmax adalah matriks Jacobian dan tidak boleh digunakan langsung.  
14            Saat digabungkan dengan cross-entropy loss, gunakan softmax_derivative di CategoricalCrossEntropy.  
15        """  
16        raise NotImplementedError(  
17            "Turunan softmax tidak boleh dipanggil langsung. " +  
18            "Gunakan penanganan kasus khusus di model.backward() sebagai gantinya."  
19        )
```

Alasan mengapa turunan softmax tidak diimplementasikan adalah karena Turunan Softmax sebenarnya sangat berbeda dari aktivasi lainnya. Berbeda dengan ReLU atau Sigmoid yang turunannya hanya bergantung pada satu nilai input, turunan Softmax adalah matriks Jacobian penuh dimana setiap output bergantung pada semua input. Namun, ketika Softmax dikombinasikan dengan Categorical Cross-Entropy (yang hampir selalu terjadi), kedua fungsi ini berinteraksi sehingga turunan gabungannya menjadi sangat sederhana: $(y_{\text{pred}} - y_{\text{true}})$. Karena itu, daripada implementasi turunan Softmax yang kompleks, kita langsung menangani kasus khusus ini di `model.backward()`.

```
● ● ●  
1 class Swish(Activation):  
2     """Fungsi aktivasi Swish: f(x) = x * sigmoid(x)"""  
3     @staticmethod  
4     def activate(x):  
5         return x * Sigmoid.activate(x)  
6  
7     @staticmethod  
8     def derivative(x):  
9         s = Sigmoid.activate(x)  
10        return s + x * s * (1 - s)
```



```
1 class LeakyReLU(Activation):
2     """Fungsi aktivasi Leaky ReLU: f(x) = x jika x > 0, else alpha * x"""
3     alpha = 0.01 # Nilai default alpha
4
5     @staticmethod
6     def activate(x):
7         return np.where(x > 0, x, LeakyReLU.alpha * x)
8
9     @staticmethod
10    def derivative(x):
11        return np.where(x > 0, 1, LeakyReLU.alpha)
```

B. Loss

File loss.py berisi implementasi berbagai fungsi loss yang digunakan dalam proses pelatihan jaringan saraf tiruan (Neural Network). Fungsi loss digunakan untuk mengukur seberapa besar perbedaan antara output prediksi model (y_{pred}) dengan nilai target sebenarnya (y_{true}). Sama seperti pendekatan pada kelas Activation, kelas Loss juga menggunakan pewarisan (inheritance). Semua fungsi loss merupakan subclass dari kelas dasar Loss, yang berfungsi sebagai kerangka utama untuk semua fungsi loss. Kelas Loss mendefinisikan metode utama yang harus dimiliki oleh setiap fungsi loss, yaitu:

- `calculate(y_true, y_pred)` → Menghitung nilai loss berdasarkan perbedaan antara y_{true} dan y_{pred} .
- `derivative(y_true, y_pred)` → Menghitung turunan dari fungsi loss, yang diperlukan dalam proses backpropagation untuk mengupdate bobot jaringan.
- `get_loss(name)` → factory method yang memungkinkan pemilihan fungsi loss berdasarkan nama yang diberikan. Jika nama tidak ditemukan, akan muncul error. Metode ini hanya didefinisikan di dalam kelas dasar Loss, dan tidak diimplementasikan di kelas turunannya.

```
1  class Loss:
2      @staticmethod
3      def calculate(y_true, y_pred):
4          raise NotImplementedError
5
6      @staticmethod
7      def derivative(y_true, y_pred):
8          raise NotImplementedError
9
10     @classmethod
11     def get_loss(cls, name):
12         loss_functions = {
13             'mse': MSE,
14             'binary_crossentropy': BinaryCrossEntropy,
15             'categorical_crossentropy': CategoricalCrossEntropy
16         }
17
18         if name.lower() in loss_functions:
19             return loss_functions[name.lower()]
20         else:
21             raise ValueError(f"Loss function '{name}' not supported. Choose from: {list(loss_functions.keys())}")
```

Dalam file ini, terdapat tiga fungsi loss yang diimplementasikan, yaitu Mean Squared Error (MSE), Binary Cross Entropy (BCE), dan Categorical Cross Entropy (CCE). Masing-masing memiliki karakteristik dan kegunaan yang berbeda. Rumus untuk fungsi loss dapat dilihat pada deskripsi persoalan.

- Fungsi Mean Squared Error (MSE) digunakan untuk regresi, yaitu ketika target (y_{true}) merupakan nilai kontinu.

```
1  class MSE(Loss):
2      """Fungsi Mean Squeared Error"""
3      @staticmethod
4      def calculate(y_true, y_pred):
5          return np.mean(np.square(y_true - y_pred))
6
7      @staticmethod
8      def derivative(y_true, y_pred):
9          batch_size = y_true.shape[0]
10         return 2 * (y_pred - y_true) / batch_size
```

- b. Fungsi Binary Cross Entropy (BCE) digunakan dalam klasifikasi biner, di mana target (y_{true}) hanya memiliki nilai 0 atau 1.

```
● ● ●  
1 class BinaryCrossEntropy(Loss):  
2     """Fungsi Binary Cross Entropy loss """  
3     @staticmethod  
4     def calculate(y_true, y_pred):  
5         epsilon = 1e-15 # Untuk menghindari Log(0)  
6         y_pred = np.clip(y_pred, epsilon, 1 - epsilon)  
7         return -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 - y_pred))  
8  
9     @staticmethod  
10    def derivative(y_true, y_pred):  
11        epsilon = 1e-15 # Untuk menghindari Log(0)  
12        y_pred = np.clip(y_pred, epsilon, 1 - epsilon)  
13        batch_size = y_true.shape[0]  
14        return -(y_true / y_pred - (1 - y_true) / (1 - y_pred)) / batch_size
```

- c. Fungsi Categorical Cross Entropy (CCE) digunakan dalam klasifikasi multi-kelas, di mana target (y_{true}) direpresentasikan dalam bentuk one-hot encoding.

```
● ● ●  
1 class CategoricalCrossEntropy(Loss):  
2     """Categorical Cross Entropy loss function"""  
3     @staticmethod  
4     def calculate(y_true, y_pred):  
5         epsilon = 1e-15 # Untuk menghindari Log(0)  
6         y_pred = np.clip(y_pred, epsilon, 1.0)  
7         return -np.mean(np.sum(y_true * np.log(y_pred), axis=1))  
8  
9     @staticmethod  
10    def derivative(y_true, y_pred):  
11        batch_size = y_true.shape[0]  
12        epsilon = 1e-8 # Nilai kecil untuk menghindari pembagian dengan nol  
13        return y_true / (y_pred + epsilon) / batch_size  
14  
15    @staticmethod  
16    def softmax_derivative(y_true, y_pred):  
17        batch_size = y_true.shape[0]  
18        return (y_pred - y_true) / batch_size
```

Alasan mengapa terdapat penambahan metode softmax_derivative pada kelas CategoricalCrossEntropy adalah dalam klasifikasi multi-kelas, sering kali fungsi aktivasi Softmax digunakan di layer output bersamaan dengan Categorical Cross Entropy (CCE) sebagai fungsi loss. Ini karena Softmax mengubah output menjadi distribusi probabilitas, sementara CCE mengukur perbedaan antara distribusi probabilitas prediksi dan target (one-hot encoded). Ketika Softmax digunakan di output layer, turunan dari CCE terhadap input Softmax memiliki bentuk yang lebih sederhana dibandingkan jika dihitung secara terpisah.

C. Initializer

File initialization.py berisi implementasi berbagai metode inisialisasi bobot (weight initialization) yang digunakan dalam jaringan saraf tiruan (Neural Network). Pemilihan metode inisialisasi yang tepat sangat penting karena dapat mempengaruhi kecepatan konvergensi dan stabilitas pelatihan model. Sama seperti pendekatan pada kelas Activation dan kelas Loss, kelas Initializer juga menggunakan konsep pewarisan (inheritance), di mana semua metode inisialisasi bobot diwariskan dari kelas dasar Initializer. Kelas Initializer bertindak sebagai kelas abstrak. Kelas ini mendefinisikan metode dasar yang harus diimplementasikan oleh setiap kelas turunannya, yaitu:

- `initialize(shape)` → Menghasilkan bobot awal dengan bentuk (shape) tertentu. Parameter shape di sini merupakan ukuran input dan output dari suatu layer.
- `get_initializer(name)` → factory method yang memungkinkan pemilihan metode inisialisasi berdasarkan nama yang diberikan. Metode ini hanya didefinisikan di dalam kelas dasar Initializer, dan tidak diimplementasikan di kelas turunannya.

```
1 class Initializer:
2     @staticmethod
3     def initialize(shape):
4         raise NotImplementedError
5
6     @classmethod
7     def get_initializer(cls, name):
8         initializers = {
9             'zeros': ZeroInitializer,
10            'uniform': RandomUniformInitializer,
11            'normal': RandomNormalInitializer,
12            'xavier': XavierInitializer,
13            'he': HeInitializer
14        }
15
16        if name.lower() in initializers:
17            return initializers[name.lower()]
18        else:
19            raise ValueError(f"Initializer '{name}' not supported. Choose from: {list(initializers.keys())}")
```

Dalam file ini, terdapat tiga initializer yang diimplementasikan, yaitu Zero initialization, Random dengan distribusi uniform, dan Random dengan distribusi normal, Xavier, dan He. Masing-masing memiliki karakteristik dan kegunaan yang berbeda.

- a. ZeroInitializer digunakan untuk menginisialisasi bobot dengan nilai nol.

```
1 class ZeroInitializer(Initializer):
2     """Inisialisasi bobot dengan nol"""
3     @staticmethod
4     def initialize(shape):
5         return np.zeros(shape)
```

- b. RandomUniformInitializer digunakan untuk menginisialisasi bobot dengan nilai acak dari distribusi uniform.

```
1 class RandomUniformInitializer(Initializer):
2     """Inisialisasi bobot dengan nilai acak dari distribusi uniform dengan parameter low dan upper bound"""
3     @staticmethod
4     def initialize(shape, low=-0.05, high=0.05, seed=None):
5         # Menggunakan seed untuk memastikan reproduksibilitas hasil
6         if seed is not None:
7             np.random.seed(seed)
8         return np.random.uniform(low, high, shape)
```

- c. RandomNormalInitializer digunakan untuk menginisialisasi bobot dengan nilai acak dari distribusi normal.

```
1 class RandomNormalInitializer(Initializer):
2     """Inisialisasi bobot dengan nilai acak dari distribusi normal dengan parameter mean dan std deviasi"""
3     @staticmethod
4     def initialize(shape, mean=0.0, std=0.05, seed=None):
5         # Menggunakan seed untuk memastikan reproduksibilitas hasil
6         if seed is not None:
7             np.random.seed(seed)
8         return np.random.normal(mean, std, shape)
```

- d. XavierInitializer digunakan untuk mengatasi masalah vanishing and exploding gradient, terutama dalam jaringan yang menggunakan fungsi aktivasi tanh atau sigmoid.

```
1 class XavierInitializer(Initializer):
2     """Xavier initialization: var(W) = 1 / n_in"""
3     @staticmethod
4     def initialize(shape):
5         fan_in, fan_out = shape
6         limit = np.sqrt(1 / fan_in)
7         return np.random.uniform(-limit, limit, shape)
```

- e. He_initializer dikembangkan untuk jaringan yang menggunakan ReLU atau variannya (Leaky ReLU, PReLU, dsb.).

```
1 class HeInitializer(Initializer):
2     """He initialization: var(W) = 2 / n_in"""
3     @staticmethod
4     def initialize(shape):
5         fan_in, _ = shape
6         std = np.sqrt(2 / fan_in)
7         return np.random.normal(0, std, shape)
```

D. Layer

File layer.py berisi implementasi lapisan (layer) dalam jaringan saraf tiruan (Neural Network) dalam bentuk kelas Layer. Selain itu, Kelas ini juga mengelola neuron, bobot, bias, dan komputasi forward dan backward propagation. Berikut adalah atribut-atribut yang terdapat pada kelas Layer.

- input_size: Jumlah neuron dari layer sebelumnya yang terhubung ke layer ini
- output_size: Jumlah neuron dalam layer ini
- activation: Objek fungsi aktivasi yang diterapkan pada layer
- W: Matriks bobot dengan dimensi (input_size, output_size)
- b: Vektor bias dengan dimensi (1, output_size)
- inputs: Menyimpan input selama forward pass untuk digunakan dalam backward pass
- z: Nilai pre-aktivasi (weighted sum + bias)
- output: Nilai setelah fungsi aktivasi diterapkan
- dW: Gradien matriks bobot dihitung selama backward pass
- db: Gradien bias dihitung selama backward pass

```
● ● ●
```

```
1 class Layer:
2     def __init__(self, input_size, output_size, activation='linear',
3                  weight_initializer='uniform', **initializer_params):
4         self.input_size = input_size
5         self.output_size = output_size
6
7         # Ngeset activation function
8         activation_class = Activation.get_activation(activation)
9         self.activation = activation_class()
10
11        # Ngeinisialisasi bobot dan bias
12        self._initialize_weights(weight_initializer, **initializer_params)
13
14        # Penyimpanan buat forward prog
15        self.inputs = None
16        self.z = None # pre-activation
17        self.output = None # post-activation
18
19        # Buat nyimpen gradient
20        self.dW = None
21        self.db = None
```

Metode utama dalam kelas Layer meliputi :

- a. `__init__(input_size, output_size, activation, weight_initializer, **initializer_params)`

Deskripsi: Konstruktor yang membuat dan menginisialisasi layer

Parameter :

- `input_size`: Jumlah neuron dari layer sebelumnya yang terhubung ke layer ini
- `output_size`: Jumlah neuron dalam layer ini
- `activation`: Objek fungsi aktivasi yang diterapkan pada layer
- `weight_initializer`: Metode inisialisasi bobot
- `initializer_params`: Parameter tambahan untuk inisialisasi

```

1 def _initialize_weights(self, initializer_name, **params):
2     if initializer_name == 'zeros':
3         self.W = ZeroInitializer.initialize((self.input_size, self.output_size))
4     elif initializer_name == 'uniform':
5         low = params.get('low', -0.05)
6         high = params.get('high', 0.05)
7         seed = params.get('seed', None)
8         self.W = RandomUniformInitializer.initialize(
9             (self.input_size, self.output_size), low, high, seed)
10    elif initializer_name == 'normal':
11        mean = params.get('mean', 0.0)
12        std = params.get('std', 0.05)
13        seed = params.get('seed', None)
14        self.W = RandomNormalInitializer.initialize(
15            (self.input_size, self.output_size), mean, std, seed)
16    elif initializer_name == 'xavier':
17        self.W = XavierInitializer.initialize((self.input_size, self.output_size))
18    elif initializer_name == 'he':
19        self.W = HeInitializer.initialize((self.input_size, self.output_size))
20    else:
21        raise ValueError(f"Initializer '{initializer_name}' not supported")
22
23    # Inisialisasi bias
24    self.b = np.zeros((1, self.output_size))
25
26 def forward(self, inputs):
27     self.inputs = inputs
28     self.z = np.dot(inputs, self.W) + self.b
29     self.output = self.activation.activate(self.z)
30     return self.output
31
32 def backward(self, dvalues):
33     if self.activation.__class__.__name__ == 'Softmax':
34         # Kasus khusus softmax
35         dz = dvalues
36     else:
37         dz = dvalues * self.activation.derivative(self.z)
38
39     self.dW = np.dot(self.inputs.T, dz)
40     self.db = np.sum(dz, axis=0, keepdims=True)
41
42     return np.dot(dz, self.W.T)

```

b. `_initialize_weights(initializer_name, **params)`

Deskripsi: Metode internal untuk inisialisasi bobot dan bias

Parameter:

- `initializer_name`: Nama metode inisialisasi ('zeros', 'uniform', 'normal', 'xavier', 'he')

- params: Parameter spesifik inisialisator (seperti 'low', 'high', 'mean', 'std')

Implementasi:

- Memilih metode inisialisasi yang sesuai
- Menginisialisasi matriks bobot berdasarkan metode
- Menginisialisasi bias dengan nilai nol

c. forward(inputs):

Deskripsi: Melakukan forward propagation melalui layer

Parameter: inputs -> Data input dari layer sebelumnya

Implementasi:

- Menyimpan input untuk backward pass nanti
- Menghitung nilai pre-aktivasi: $z = \text{inputs} * W + b$
- Menerapkan fungsi aktivasi: $\text{output} = \text{activation}(z)$
- Mengembalikan output untuk layer berikutnya

d. backward(dvalues)

Deskripsi: Melakukan backward propagation untuk menghitung gradien

Parameter: dvalues -> Gradien dari layer berikutnya

Implementasi:

- Menghitung gradien dz (kasus khusus untuk Softmax)
- Menghitung gradien bobot: $dW = \text{inputs}^T * dz$
- Menghitung gradien bias: $db = \text{sum}(dz, \text{axis}=0)$
- Menghitung dan mengembalikan gradien untuk layer sebelumnya: $\text{inputs}^T * dW$

E. FFNN

Kelas FFNN merupakan implementasi jaringan saraf tiruan dengan arsitektur feedforward. Jaringan ini berfungsi sebagai model pembelajaran mesin yang dapat melakukan klasifikasi atau regresi berdasarkan data input. Kelas ini mendukung penambahan layer secara dinamis, forward propagation, backward propagation, pembaruan bobot, pelatihan dengan metode batch, visualisasi jaringan, visualisasi distribusi bobot untuk setiap layer atau layer-layer spesifik, visualisasi distribusi gradient untuk setiap layer atau layer-layer spesifik, menyimpan dan memuat model dalam bentuk file pickle. Berikut adalah atribut-atribut yang terdapat pada kelas FFNN :

- loss_name: Menyimpan nama fungsi loss untuk referensi dan serialisasi
- layers: List yang menyimpan semua objek layer dalam jaringan
- loss_function: Objek fungsi loss yang digunakan untuk menghitung error
- layer_sizes: List yang menyimpan ukuran setiap layer (termasuk input)
- l1_lambda: Parameter regulasi L1 (Lasso) untuk mencegah overfitting

- l2_lambda: Parameter regulasi L2 (Ridge) untuk mencegah overfitting

Metode utama dalam kelas FFNN meliputi :

```
● ● ●  
1 class FFNN:  
2     def __init__(self, loss='mse', l1_lambda=0.0, l2_lambda=0.0):  
3         self.loss_name = loss # buat nyimpen nama Loss function  
4         self.layers = []  
5         self.loss_function = Loss.get_loss(loss)  
6         self.layer_sizes = []  
7         self.l1_lambda = l1_lambda  
8         self.l2_lambda = l2_lambda
```

- a. `__init__(loss='mse', l1_lambda=0.0, l2_lambda=0.0):`

Deskripsi: Konstruktor untuk inisialisasi model

Parameter:

- loss: Nama fungsi loss ('mse', 'binary_crossentropy', 'categorical_crossentropy')
- l1_lambda: Kekuatan regularisasi L1
- l2_lambda: Kekuatan regularisasi L2

```

1 def add(self, input_size, output_size, activation='linear',
2        weight_initializer='uniform', **initializer_params):
3
4     # Cek kalo misal ini Layer pertama atau bukan
5     if not self.layers:
6         self.layer_sizes.append(input_size)
7
8     self.layer_sizes.append(output_size)
9
10    # Ngebuat layer baru ke neural network sama nambahin ke list Layers
11    layer = Layer(input_size, output_size, activation,
12                  weight_initializer, **initializer_params)
13
14    self.layers.append(layer)
15    return self
16
17 def forward(self, X):
18     output = X
19
20     # Iterasi untuk setiap layer
21     for layer in self.layers:
22         # Ngelakuin forward propagation dengan inputnya berupa hasil output layer sebelumnya
23         output = layer.forward(output)
24
25     # Ngembalikan output akhir dari neural network
26     return output
27
28 def backward(self, y_true, y_pred):
29     # Cek kalo Loss functionnya sama dengan softmax dan activation functionnya sama dengan categorical crossentropy di Layer terakhir
30     # Karena kalo iya ada special case buat ngitung gradient awalnya
31     if (self.layers[-1].activation.__class__.__name__ == 'Softmax' and
32         self.loss_function.__name__ == 'CategoricalCrossEntropy'):
33         dvalues = self.loss_function.softmax_derivative(y_true, y_pred)
34     else:
35         dvalues = self.loss_function.derivative(y_true, y_pred)
36
37     # Meneruskan gradien dari belakang ke depan (reversed)
38     for layer in reversed(self.layers):
39         dvalues = layer.backward(dvalues)
40
41     return dvalues
42
43 def update_weights(self, learning_rate):
44     #Memperbarui bobot untuk semua Layer menggunakan gradien yang dihitung
45     for layer in self.layers:
46         layer.W -= learning_rate * layer.dW
47         layer.b -= learning_rate * layer.db

```

b. `add(input_size, output_size, activation, weight_initializer, **initializer_params):`

Deskripsi: Menambahkan layer baru ke jaringan

Parameter:

- `input_size`: Jumlah neuron input
- `output_size`: Jumlah neuron output
- `activation`: Fungsi aktivasi untuk layer
- `weight_initializer`: Metode inisialisasi bobot
- `initializer_params`: Parameter tambahan untuk initializer

Implementasi: Membuat objek Layer dan menyimpan ukuran untuk melacak arsitektur

c. forward(X):

Deskripsi: Melakukan forward propagation melalui jaringan

Parameter: X -> Data input

Implementasi: Meneruskan output dari setiap layer sebagai input ke layer berikutnya

Return: Prediksi model

d. backward(y_true, y_pred):

Deskripsi: Melakukan backward propagation untuk menghitung gradien

Parameter:

- y_true: Label aktual
- y_pred: Prediksi model

Implementasi:

- Mendeteksi kasus khusus softmax + categorical_crossentropy
- Mengkalkulasi gradien awal dari loss function
- Meneruskan gradien secara terbalik melalui setiap layer

Return: Gradien untuk input

e. update_weights(learning_rate):

Deskripsi: Memperbarui bobot berdasarkan gradien dihitung

Parameter: learning_rate -> Faktor pembelajaran

Implementasi: Menggunakan Gradient Descent untuk memperbarui W dan b di setiap layer.

```

1  def fit(self, X_train, y_train, batch_size=32, learning_rate=0.01,
2         epochs=100, validation_data=None, verbose=1):
3
4     # Dictionary buat nyimpen history training
5     history = {
6         'loss': [],
7         'val_loss': [] if validation_data else None
8     }
9
10    n_samples = X_train.shape[0]
11    n_batches = int(np.ceil(n_samples / batch_size))
12
13    for epoch in range(epochs):
14        # Shuffle data setiap epoch
15        indices = np.random.permutation(n_samples)
16        X_shuffled = X_train[indices]
17        y_shuffled = y_train[indices]
18
19        epoch_loss = 0
20
21        for batch in range(n_batches):
22            # Ambil batch data
23            start_idx = batch * batch_size
24            end_idx = min((batch + 1) * batch_size, n_samples)
25            X_batch = X_shuffled[start_idx:end_idx]
26            y_batch = y_shuffled[start_idx:end_idx]
27
28            # Lakukan forward propagation
29            y_pred = self.forward(X_batch)
30            y_pred = np.clip(y_pred, 1e-15, 1 - 1e-15)
31
32            # Hitung Lossnya
33            batch_loss = self.loss_function.calculate(y_batch, y_pred)
34
35            # Tambahkan regularisasi L1 dan L2 untuk Batch Loss
36            if self.l1_lambda > 0 or self.l2_lambda > 0:
37                l1_penalty = self.l1_lambda * sum(np.sum(np.abs(layer.W)) for layer in self.layers)
38                l2_penalty = self.l2_lambda * sum(np.sum(layer.W**2) for layer in self.layers)
39                batch_loss += l1_penalty + l2_penalty
40
41            epoch_loss += batch_loss * (end_idx - start_idx) / n_samples
42
43            # Lakukan backward propagation
44            self.backward(y_batch, y_pred)
45
46            # Update bobot
47            self.update_weights(learning_rate)
48
49            # Simpan Loss ke history
50            history['loss'].append(epoch_loss)
51
52            # Hitung validation loss kalo ada
53            if validation_data:
54                X_val, y_val = validation_data
55                y_val_pred = self.forward(X_val)
56                val_loss = self.loss_function.calculate(y_val, y_val_pred)
57
58                # Tambahkan regularisasi L1 dan L2 untuk Validation Loss
59                if self.l1_lambda > 0 or self.l2_lambda > 0:
60                    l1_penalty_val = self.l1_lambda * sum(np.sum(np.abs(layer.W)) for layer in self.layers)
61                    l2_penalty_val = self.l2_lambda * sum(np.sum(layer.W**2) for layer in self.layers)
62                    val_loss += l1_penalty_val + l2_penalty_val
63
64                history['val_loss'].append(val_loss)
65
66            # Tunjukin progress kalo ada
67            if verbose == 1:
68                val_msg = "" if validation_data else ""
69                print(f"Epoch {epoch+1}/{epochs} - loss: {epoch_loss:.4f}{val_msg}")
70
71            # Progress bar
72            progress = int(30 * (epoch + 1) / epochs)
73            bar = "[" + "=" * progress + ">" + " " * (30 - progress - 1) + "]"
74            print(bar, end="\r")
75
76        if verbose == 1:
77            print()
78
79    return history

```

- f. `fit(X_train, y_train, batch_size, learning_rate, epochs, validation_data, verbose)`

Deskripsi: Melatih model dengan data yang diberikan

Parameter:

- `X_train`: Data fitur pelatihan
- `y_train`: Label pelatihan
- `batch_size`: Ukuran batch untuk SGD (Stochastic Gradient Descent)
- `learning_rate`: Faktor pembelajaran
- `epochs`: Jumlah iterasi pelatihan
- `validation_data`: Data validasi opsional (`X_val, y_val`)
- `verbose`: Level detail output (0: tidak ada, 1: lengkap)

Implementasi:

- Mengacak data setiap epoch
- Memproses data dalam batch
- Menghitung loss dan gradien
- Menerapkan regularisasi L1/L2 jika dikonfigurasi
- Melakukan backward propagation dan update bobot
- Mengevaluasi model pada data validasi (jika ada)
- Menampilkan progress bar jika `verbose=1`

Return: Objek history pelatihan dengan loss dan `val_loss`

- g. `visualize_network(self, figsize=(10, 6))`

Deskripsi: Memvisualisasikan struktur jaringan saraf sebagai graf dengan menampilkan neuron, bobot, dan gradien.

Parameter: `figsize` -> Ukuran gambar output

- h. `visualize_weight_distribution(self, layers=None)`

Deskripsi: Memvisualisasikan distribusi statistik bobot untuk layer tertentu atau semua layer.

Parameter: `layers` -> List indeks layer yang akan divisualisasikan (default: semua layer)

- i. `visualize_gradient_distribution(self, layers=None)`

Deskripsi: Memvisualisasikan distribusi statistik gradien bobot untuk layer tertentu atau semua layer.

Parameter: `layers` -> List indeks layer yang akan divisualisasikan (default: semua layer)

Kode fungsi visualisasi dapat dilihat pada source code.

```

1  def save(self, filepath):
2      # Pastikan kalo direktori buat nyimpen ada
3      directory = os.path.dirname(filepath)
4      if directory:
5          os.makedirs(directory, exist_ok=True)
6
7      # Cek jenis Loss function yang digunakan
8      if hasattr(self, 'loss_name'):
9          # Jika loss_name sudah disimpan saat inisialisasi
10         loss_id = self.loss_name
11     else:
12         print(f"Loss function type: {type(self.loss_function)}")
13
14     # Default ke categorical_crossentropy jika tidak ditentukan
15     loss_id = 'categorical_crossentropy'
16
17     # Coba ambil nama dari instance
18     try:
19         loss_name = self.loss_function.__class__.__name__
20         # Pemetaan nama ke identifier
21         loss_mapping = {
22             'MeanSquaredError': 'mse',
23             'CategoricalCrossEntropy': 'categorical_crossentropy',
24             'BinaryCrossEntropy': 'binary_crossentropy'
25         }
26         if loss_name in loss_mapping:
27             loss_id = loss_mapping[loss_name]
28     except:
29         # Jika gagal, gunakan default
30         pass
31
32     # Simpan model ke file dengan identifier Loss function
33     model_data = {
34         'layer_sizes': self.layer_sizes,
35         'layers': self.layers,
36         'loss_function': loss_id
37     }
38
39     with open(filepath, 'wb') as f:
40         pickle.dump(model_data, f)
41
42     print(f"Model saved to {filepath} with loss: {loss_id}")
43
44     @classmethod
45     def load(cls, filepath):
46         with open(filepath, 'rb') as f:
47             model_data = pickle.load(f)
48
49         # Load model dari file
50         model = cls(loss=model_data['loss_function'])
51         model.layer_sizes = model_data['layer_sizes']
52         model.layers = model_data['layers']
53
54         print(f"Model loaded from {filepath}")
55         return model

```

j. save(self, filepath)

Deskripsi: Menyimpan model FFNN ke file untuk penggunaan di masa mendatang.

Parameter: filepath -> Path file tempat model akan disimpan

k. load(cls, filepath)

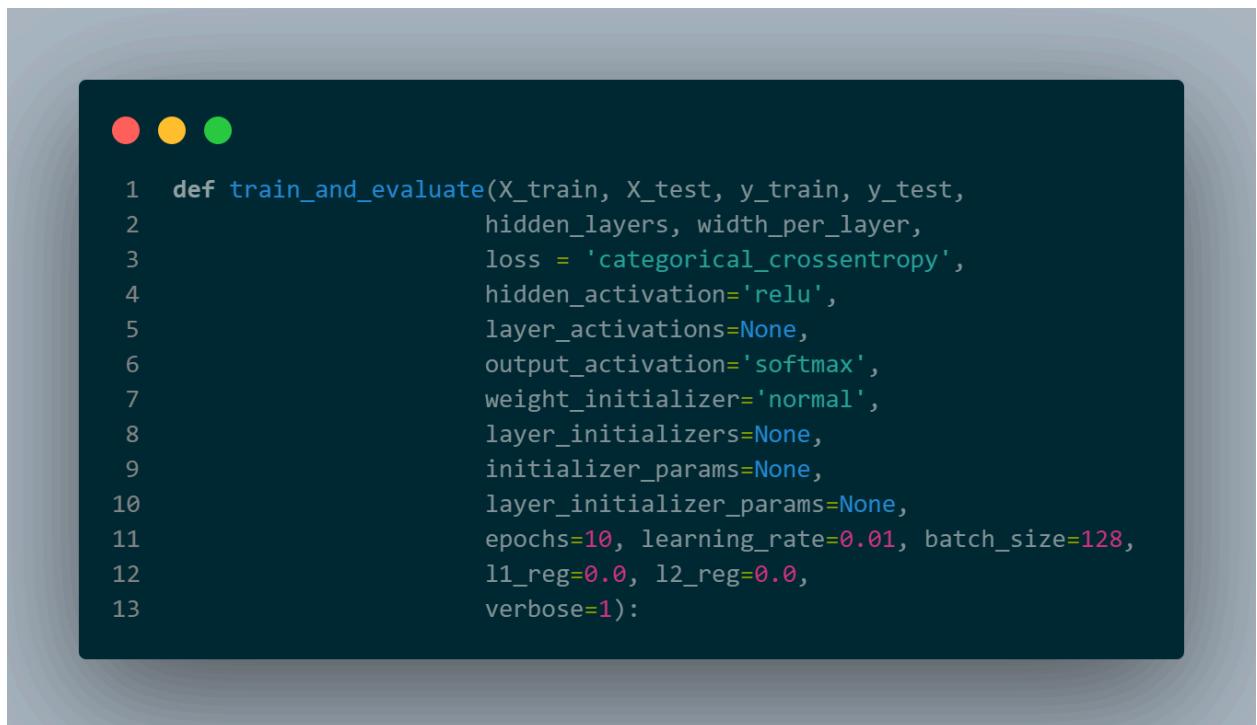
Deskripsi: Memuat model FFNN dari file yang sebelumnya disimpan.

Parameter: filepath -> Path file tempat model tersimpan

F. test_main.ipynb

Pada test_main.ipynb terdapat fungsi train_and_evaluate yang merupakan wrapper tingkat tinggi yang menyederhanakan proses pembuatan, pelatihan, dan evaluasi model FFNN. Fungsi ini menyediakan antarmuka fleksibel untuk mengkonfigurasi berbagai aspek arsitektur jaringan dan proses pelatihan. Fungsi ini dirancang sebagai alat eksperimen untuk:

- Menguji berbagai arsitektur neural network secara efisien dan sistematis
- Menstandarisasi proses evaluasi model untuk perbandingan yang adil
- Mempercepat eksplorasi hyperparameter (aktivasi, inisialisasi, jumlah neuron, dll)
- Mengabstraksi kompleksitas pembangunan dan konfigurasi model sehingga fokus penelitian dapat diarahkan pada analisis hasil



```
1 def train_and_evaluate(X_train, X_test, y_train, y_test,
2                         hidden_layers, width_per_layer,
3                         loss = 'categorical_crossentropy',
4                         hidden_activation='relu',
5                         layer_activations=None,
6                         output_activation='softmax',
7                         weight_initializer='normal',
8                         layer_initializers=None,
9                         initializer_params=None,
10                        layer_initializer_params=None,
11                        epochs=10, learning_rate=0.01, batch_size=128,
12                        l1_reg=0.0, l2_reg=0.0,
13                        verbose=1):
```

Parameter yang terdapat pada fungsi train_and_evaluate :

- a. Data dan arsitektur :
 - X_train, X_test, y_train, y_test: Data training dan testing
 - hidden_layers: Jumlah hidden layer dalam jaringan
 - width_per_layer: Jumlah neuron per layer (bisa berupa integer untuk ukuran seragam atau list untuk ukuran bervariasi)
- b. Fungsi aktivasi :
 - hidden_activation: Fungsi aktivasi default untuk hidden layer
 - layer_activations: List fungsi aktivasi spesifik per layer
 - output_activation: Fungsi aktivasi untuk output layer
- c. Inisialisasi Bobot :
 - weight_initializer: Metode inisialisasi default untuk bobot
 - layer_initializers: List metode inisialisasi spesifik per layer
 - initializer_params: Parameter default untuk inisialisasi
 - layer_initializer_params: List parameter inisialisasi spesifik per layer
- d. Hyperparameter pelatihan :
 - loss: Fungsi loss untuk pelatihan ('mse', 'binary_crossentropy', 'categorical_crossentropy')
 - epochs: Jumlah epoch untuk pelatihan
 - learning_rate: Learning rate untuk optimizer
 - batch_size: Ukuran batch untuk SGD
 - l1_reg, l2_reg: Parameter regularisasi L1 dan L2
 - verbose: Level detail output (0: tidak ada, 1: lengkap)

```

● ● ●

1 # Menginisialisasi ukuran input dan output
2 input_size = X_train.shape[1]
3 output_size = y_train.shape[1]
4
5 # Default initializer params
6 if initializer_params is None:
7     initializer_params = {'std': 0.1}
8
9 # Jika layer_activations tidak disediakan, gunakan hidden_activation untuk semua layer
10 if layer_activations is None:
11     layer_activations = [hidden_activation] * hidden_layers
12 elif len(layer_activations) < hidden_layers: # Jika jumlah activation kurang dibandingkan jumlah layer
13     layer_activations = layer_activations + [hidden_activation] * (hidden_layers - len(layer_activations))
14
15 # Jika layer_initializers tidak disediakan, gunakan weight_initializer untuk semua layer
16 if layer_initializers is None:
17     layer_initializers = [weight_initializer] * (hidden_layers + 1)
18 elif len(layer_initializers) < hidden_layers + 1: # Jika jumlah initializers kurang dibandingkan jumlah layer
19     layer_initializers = layer_initializers + [weight_initializer] * (hidden_layers + 1 - len(layer_initializers))
20
21 # Jika layer_initializer_params tidak disediakan, gunakan initializer_params untuk semua layer
22 if layer_initializer_params is None:
23     layer_initializer_params = [initializer_params] * (hidden_layers + 1)
24 elif len(layer_initializer_params) < hidden_layers + 1:
25     layer_initializer_params = layer_initializer_params + [initializer_params] * (hidden_layers + 1 - len(layer_initializer_params))
26
27 start_time = time.time()
28
29 # Buat model
30 model = FFNN(loss=loss, l1_lambda=l1_reg, l2_lambda=l2_reg,)
31
32 # Tambahkan input layer
33 if isinstance(width_per_layer, list):
34     first_layer_width = width_per_layer[0]
35 else:
36     first_layer_width = width_per_layer
37
38 model.add(input_size=input_size, output_size=first_layer_width,
39             activation=layer_activations[0],
40             weight_initializer=layer_initializers[0],
41             **layer_initializer_params[0])
42
43 # Tambahkan hidden layers
44 for i in range(1, hidden_layers):
45     if isinstance(width_per_layer, list):
46         prev_width = width_per_layer[i-1]
47         current_width = width_per_layer[i]
48     else:
49         prev_width = width_per_layer
50         current_width = width_per_layer
51
52     model.add(input_size=prev_width, output_size=current_width,
53               activation=layer_activations[i],
54               weight_initializer=layer_initializers[i],
55               **layer_initializer_params[i])
56
57 # Tambahkan output layer dengan aktivasi yang ditentukan
58 if isinstance(width_per_layer, list):
59     last_hidden_width = width_per_layer[-1]
60 else:
61     last_hidden_width = width_per_layer
62
63 model.add(input_size=last_hidden_width, output_size=output_size,
64             activation=output_activation,
65             weight_initializer=layer_initializers[-1],
66             **layer_initializer_params[-1])
67
68 # Informasi model yang lebih komprehensif
69 print(f"\nTraining model with {hidden_layers} hidden layers, width: {width_per_layer}")
70 print(f"Loss function: {loss}")
71 print(f"Activations: {layer_activations} (hidden) | {output_activation} (output)")
72 print(f"Initializers: {layer_initializers}")
73 model.summary()
74
75 # Melakukan training model
76 history = model.fit(
77     X_train, y_train,
78     batch_size=batch_size,
79     learning_rate=learning_rate,
80     epochs=epochs,
81     validation_data=(X_test, y_test),
82     verbose=verbose,
83 )
84
85 # Hitung waktu training
86 training_time = time.time() - start_time
87
88 # Evaluasi model
89 y_pred = model.forward(X_test)
90 y_pred_classes = np.argmax(y_pred, axis=1)
91 y_test_classes = np.argmax(y_test, axis=1)
92 accuracy = np.mean(y_pred_classes == y_test_classes)
93
94 # Tampilkan informasi akhir hanya jika verbose > 0
95 if verbose > 0:
96     print(f"Test accuracy: {accuracy:.4f}")
97     print(f"Training time: {training_time:.2f} seconds")
98
99 return {
100     'model': model,
101     'history': history,
102     'accuracy': accuracy,
103     'training_time': training_time,
104     'loss_function': loss,
105     'depth': hidden_layers,
106     'width': width_per_layer,
107     'layer_activations': layer_activations,
108     'output_activation': output_activation,
109     'layer_initializers': layer_initializers,
110     'layer_initializer_params': layer_initializer_params
111 }

```

Beberapa langkah utama yang dibuat oleh fungsi ini adalah

1. Menentukan ukuran input dan output berdasarkan data
2. Mengatur nilai default dan memvalidasi parameter konfigurasi
3. Memastikan konsistensi jumlah layer dengan jumlah aktivasi dan inisialisasi
4. Membuat instance FFNN dengan fungsi loss yang sesuai
5. Menambahkan layer dengan fleksibilitas untuk berbagai arsitektur:
 - Mendukung width yang seragam atau bervariasi per layer
 - Mengaplikasikan aktivasi yang berbeda-beda per layer
 - Menggunakan metode inisialisasi yang berbeda-beda per layer
6. Melatih model dengan konfigurasi yang diberikan
7. Menghitung waktu pelatihan
8. Mengukur akurasi pada data testing
9. Menampilkan informasi performa

2.1.2 Penjelasan Forward Propagation

```
def forward(self, X):
    output = X

    # Iterasi untuk setiap layer
    for layer in self.layers:
        # Ngelakuin forward propagation dengan input
        output = layer.forward(output)

    # Ngembalikan output akhir dari neural network
    return output
```

Forward propagation digunakan untuk menghitung semua variabel, parameter, dan bahkan output. Forward propagation menyesuaikan dengan fungsi aktivasi (activation function) yang telah dijelaskan pada bagian sebelumnya. Program forward propagation berada dalam kelas model tersebut dengan berisi konten menerapkan algoritma dan menggunakan fungsi-fungsi pendukung dari kelas-kelas lainnya seperti activation function, layer, dan loss function untuk mengimplementasikan algoritma forward propagation. Penjelasan algoritma forward propagation yang diterapkan pada tugas ini antara lain:

1. Input Data ke Layer Pertama: Data input X diberikan ke jaringan dan diteruskan ke layer pertama.

2. Perhitungan di Setiap Layer: Setiap layer akan dijalankan dengan fungsi aktivasi yang telah dikonfigurasi sebelumnya.
3. Output Layer Hasil akhir dari layer terakhir adalah prediksi yang akan dibandingkan dengan label sebenarnya.

2.1.3 Penjelasan Backward Propagation dan Weight Update

```

def backward(self, y_true, y_pred):
    # Cek kalo loss functionnya sama dengan softmax dan activation functionnya sama
    # Karena kalo iya ada special case buat ngitung gradient awalnya
    if (self.layers[-1].activation.__class__.__name__ == 'Softmax' and
        self.loss_function.__name__ == 'CategoricalCrossEntropy'):
        dvalues = self.loss_function.softmax_derivative(y_true, y_pred)
    else:
        dvalues = self.loss_function.derivative(y_true, y_pred)

    # Meneruskan gradien dari belakang ke depan (reversed)
    for layer in reversed(self.layers):
        dvalues = layer.backward(dvalues)

    return dvalues

def update_weights(self, learning_rate):
    #Memperbarui bobot untuk semua layer menggunakan gradien yang dihitung
    for layer in self.layers:
        layer.W -= learning_rate * layer.dW
        layer.b -= learning_rate * layer.db

```

Backward propagation adalah algoritma yang digunakan untuk menghitung gradient (turunan) dari fungsi loss (loss function) terhadap bobot (weight) dalam neural network. Tahapan ini ditujukan dalam pelatihan model neural network untuk memperbarui bobot (weight) menggunakan metode optimasi, gradient descent. Penjelasan algoritma backward propagation yang diterapkan pada tugas ini antara lain:

1. Hitung loss gradient: Jika output menggunakan Softmax + Categorical Crossentropy, gunakan turunan khusus. Jika tidak, gunakan turunan fungsi loss standar
2. Backward propagation melalui layer: Setiap layer menerima gradien dari layer berikutnya. Menghitung gradien terhadap bobot (dW), bias (db), dan input (dvalues).
3. Update bobot (weight): Menggunakan gradient descent, bobot diubah berdasarkan parameter learning rate.

$$W = W_0 - \alpha * d_W$$

$$b = b_0 - \alpha * d_b$$

2.1.4 Penjelasan Implementasi Regularisasi L1 dan L2

Regularisasi adalah teknik yang digunakan untuk mencegah **overfitting** dalam model Feedforward Neural Network (FFNN). Terdapat dua jenis regularisasi yang diterapkan, yaitu **Regularisasi L1 (Lasso)** dan **Regularisasi L2 (Ridge)**. Pada model FFNN ini, regularisasi diaktifkan jika parameter `l1_lambda` atau `l2_lambda` lebih dari 0. Regularisasi diterapkan baik pada loss training maupun loss validasi dengan menambahkan penalti pada fungsi loss.

Regularisasi L1 (Lasso Regularization)

Regularisasi L1 menambahkan penalti berupa jumlah absolut dari bobot (W). Regularisasi ini memiliki efek **membuat banyak bobot menjadi nol**, sehingga cocok untuk seleksi fitur karena hanya **fitur yang paling penting yang akan bertahan**.

Rumus Regularisasi L1:

$$L1_{Penalty} = \lambda_1 \sum_i |w_i|$$

Di mana:

λ_1 = faktor regularisasi (l1_lambda)

w_i = Bobot model

Implementasi L1 Penalty dalam Python:

```
l1_penalty = self.l1_lambda * sum(np.sum(np.abs(layer.W))
for layer in self.layers)
```

Regularisasi L2 (Ridge Regularization)

Regularisasi L2 menambahkan penalti berupa **jumlah kuadrat dari bobot** (W). Efek dari regularisasi ini adalah **menekan nilai bobot agar tetap kecil**, tetapi tidak mengubah bobot menjadi nol. Regularisasi ini membantu model menjadi lebih **stabil dan generalizable**.

Rumus Regularisasi L2:

$$L2_{Penalty} = \lambda_2 \sum_i |w_i|^2$$

Di mana:

λ_2 = faktor regularisasi (l2_lambda)

w_i = Bobot model

Implementasi L2 Penalty dalam Python:

```
l2_penalty = self.l2_lambda * sum(np.sum(layer.W**2) for  
layer in self.layers)
```

Penerapan dalam Perhitungan Loss

Total loss pada **training** diperoleh dengan menambahkan penalti L1 dan L2 ke dalam loss utama:

```
batch_loss += l1_penalty + l2_penalty
```

Sedangkan untuk **validation loss**, regularisasi diterapkan dengan cara yang sama:

```
val_loss += l1_penalty_val + l2_penalty_val
```

2.2. Hasil Pengujian

2.2.1 Pengaruh Depth (Banyak Layer) dan Width (Banyak Neuron)

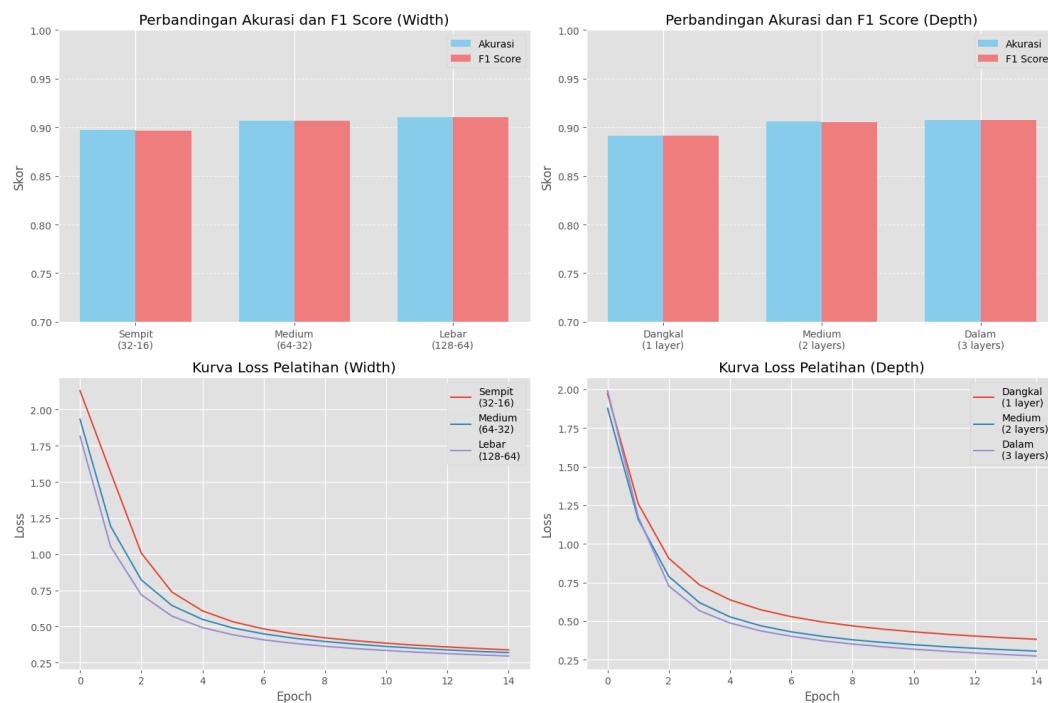
Tabel 2.2.1.1. Pengujian Pengaruh Width (Banyak Neuron)

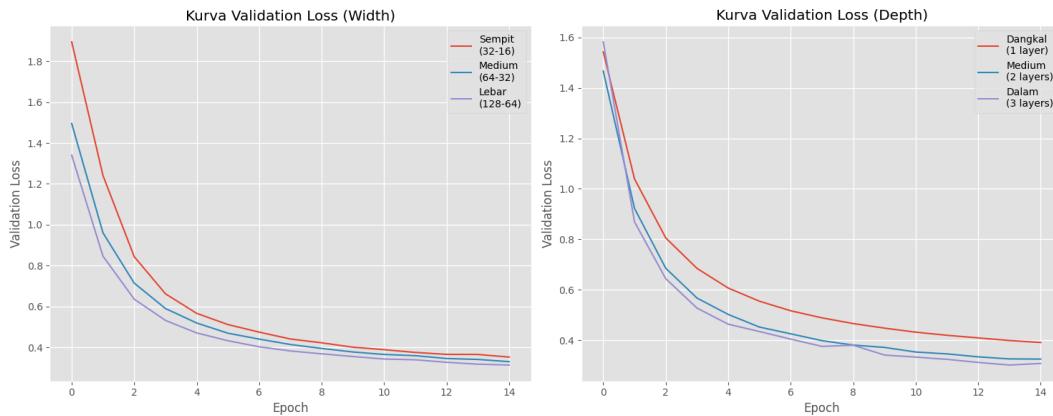
Variabel Tetap	
Hidden Activation	ReLU
Depth	2
Weight Initializer	he
Epochs	15
Learning Rate	0.01
Batch Size	128
Variabel Bebas (Width per Layer)	
Percobaan ke-1	[32, 16]
Percobaan ke-2	[64, 32]
Percobaan ke-3	[128, 64]

Tabel 2.2.1.2. Pengujian Pengaruh Depth (Banyak Layer)

Variabel Tetap	
Hidden Activation	ReLU

Width Per Layer	64
Weight Initializer	he
Epochs	15
Learning Rate	0.01
Batch Size	128
Variabel Bebas (Depth)	
Percobaan ke-1	1
Percobaan ke-2	2
Percobaan ke-3	3





Model dengan **jumlah neuron yang lebih besar** ([128,64]) memiliki sedikit peningkatan akurasi dan F1-score dibandingkan model dengan jumlah neuron lebih kecil ([32,16]). Namun, perbedaannya tidak terlalu signifikan, yang menunjukkan bahwa menambah jumlah neuron mungkin memiliki efek terbatas jika jaringan sudah cukup kompleks. Model dengan **jumlah neuron lebih kecil** mengalami penurunan loss yang lebih lambat dibandingkan model dengan jumlah neuron lebih besar. Model dengan lebih banyak neuron mencapai **loss yang lebih rendah** lebih cepat, menunjukkan bahwa model lebih lebar lebih baik dalam menangkap pola data.

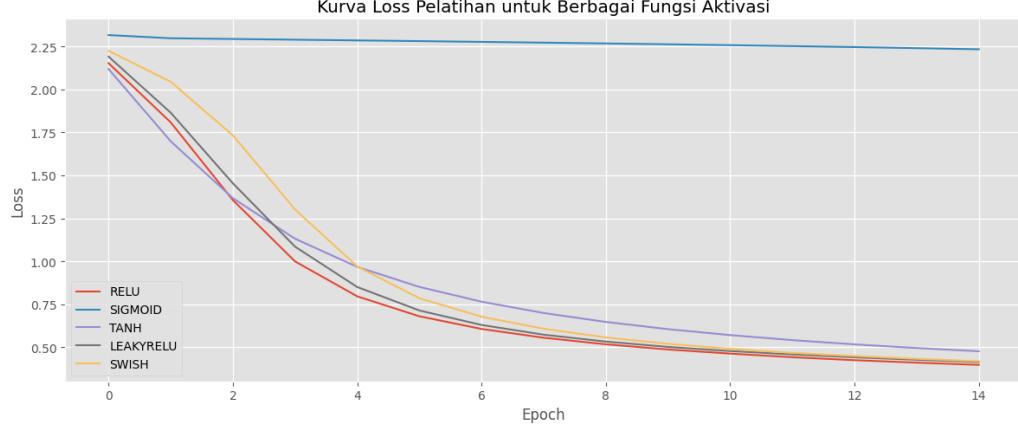
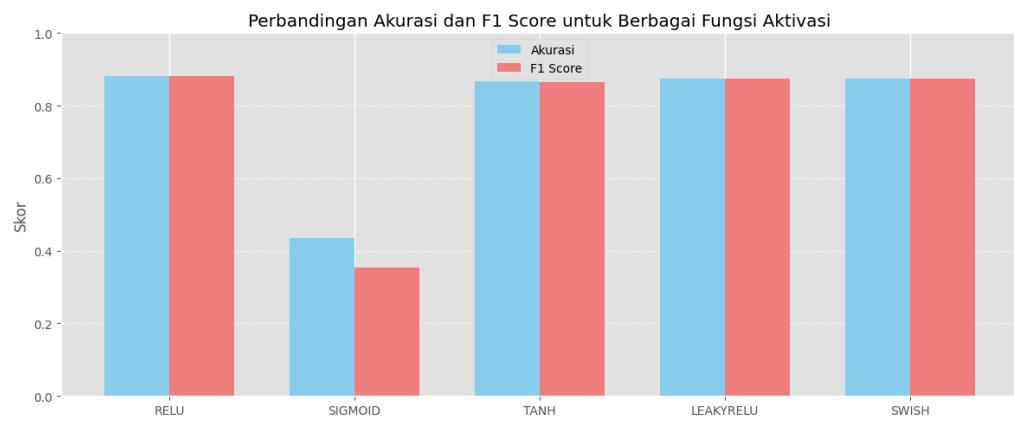
Model yang lebih dalam (**3 layer**) menunjukkan sedikit peningkatan akurasi dan F1-score dibanding model yang lebih dangkal (**1 layer**). Sama seperti width, perbedaannya tidak terlalu besar. Model yang lebih dalam mengalami penurunan loss yang lebih cepat, tetapi tidak jauh berbeda dengan model yang memiliki 2 layer. Model yang lebih dalam cenderung memiliki **loss yang lebih rendah**, menunjukkan bahwa lebih banyak layer dapat membantu dalam ekstraksi fitur yang lebih kompleks.

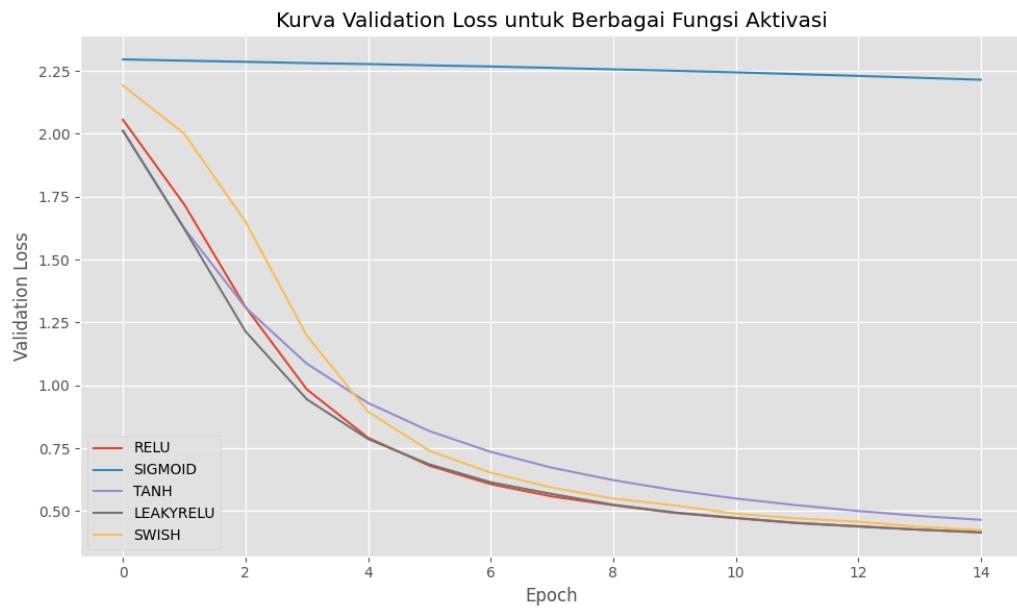
2.2.2 Pengaruh Fungsi Aktivasi Hidden Layer

Tabel 2.2.2.1. Pengujian Fungsi Aktivasi Hidden Layer

Variabel Tetap	
Depth	2
Weight Initializer	random normal
Epochs	15
Learning Rate	0.01
Batch Size	128

Width Per Layer	64
Variabel Bebas (Fungsi Aktivasi)	
Percobaan ke-1	Linear
Percobaan ke-2	ReLU
Percobaan ke-3	Sigmoid
Percobaan ke-4	Hyperbolic Tangent
Percobaan ke-5	Softmax
Percobaan ke-6	Swish
Percobaan ke-3	Leaky ReLU







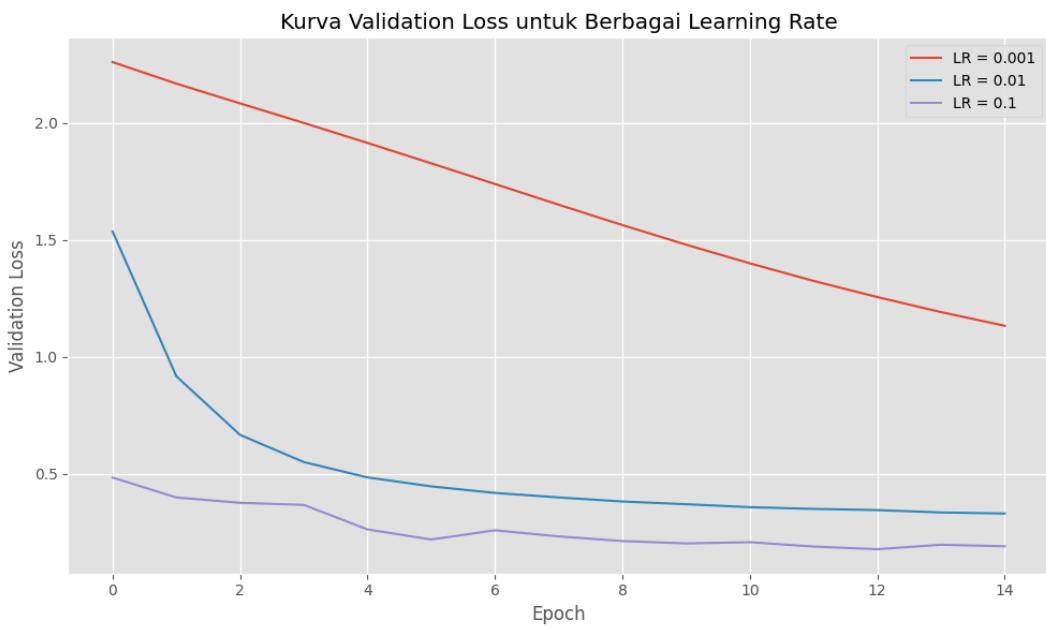
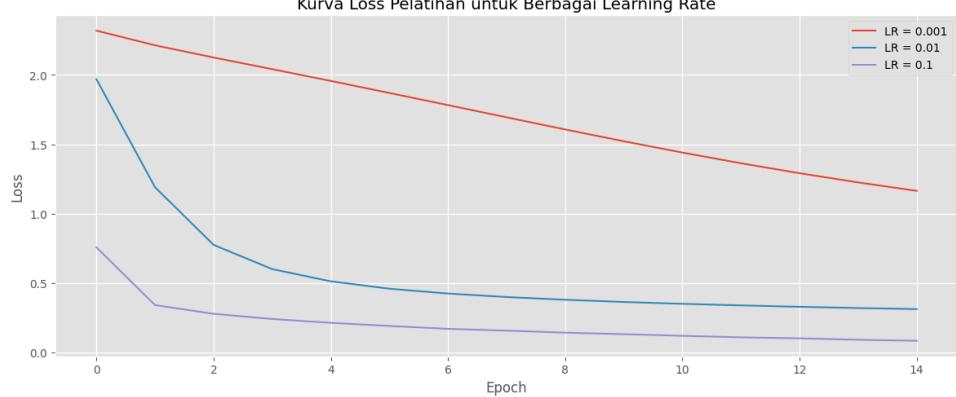
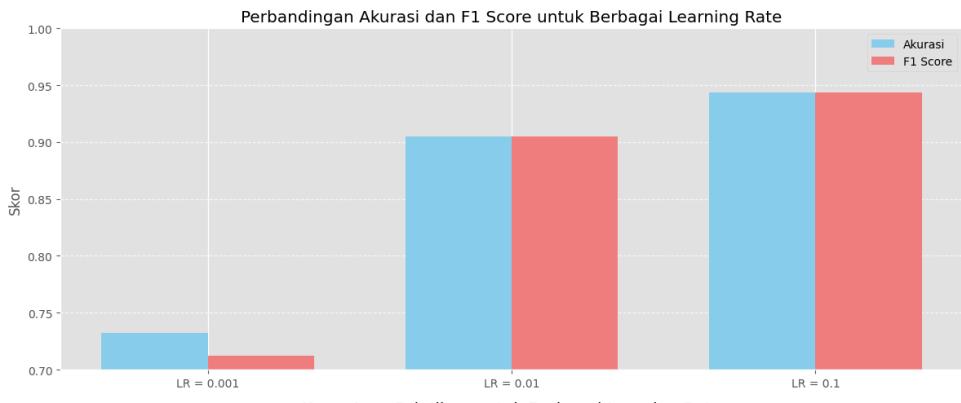
ReLU, Leaky ReLU, dan Swish memiliki akurasi dan F1 Score yang lebih tinggi dibandingkan Sigmoid dan Tanh. Sigmoid dan Tanh cenderung memiliki performa lebih rendah, kemungkinan karena masalah *vanishing gradient*, terutama pada jaringan dengan lebih dari satu hidden layer. Leaky ReLU dan Swish terlihat sedikit lebih baik daripada ReLU dalam beberapa kasus, kemungkinan karena kemampuan mereka mengatasi *zero gradient problem* pada nilai negatif.

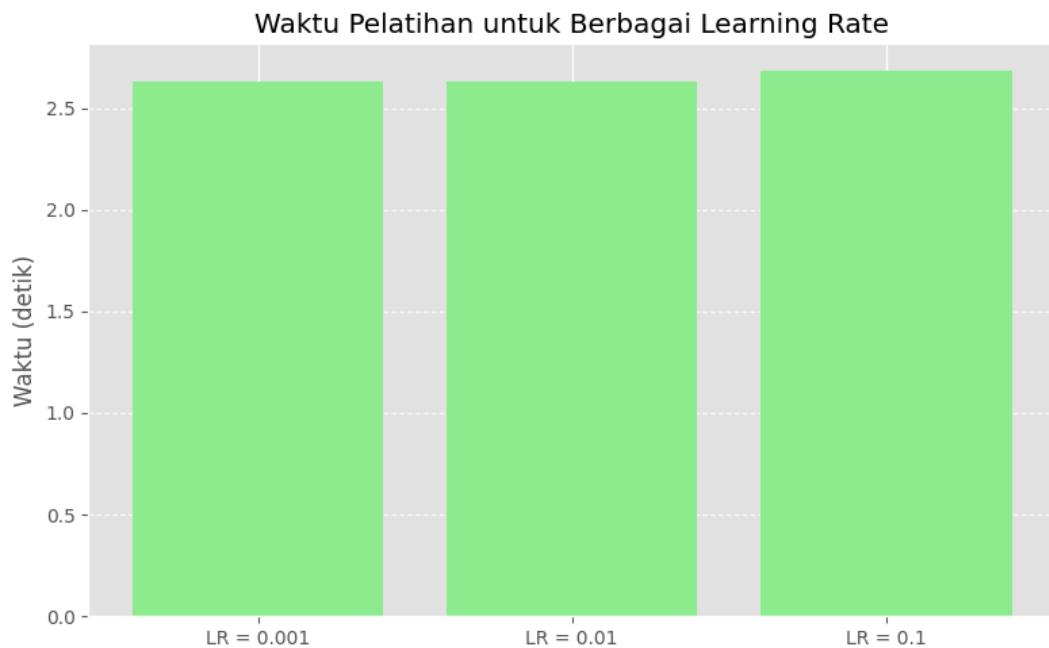
ReLU, Leaky ReLU, dan Swish mengalami penurunan loss yang lebih cepat, menunjukkan bahwa mereka lebih baik dalam mempelajari pola data. **Sigmoid** dan **Tanh** memiliki loss yang lebih tinggi dan lebih lambat menurun, menandakan mereka kurang efektif dalam jaringan yang lebih dalam. **Swish menunjukkan performa terbaik dalam penurunan loss**, diikuti oleh Leaky ReLU dan ReLU. **Leaky ReLU sedikit lebih stabil dibandingkan ReLU** karena tetap memiliki gradien pada nilai negatif, sehingga tidak mengalami masalah "dying ReLU."

2.2.3 Pengaruh Learning Rate

Tabel 2.2.3.1. Pengujian Learning Rate

Variabel Tetap	
Depth	2
Weight Initializer	he
Epochs	15
Hidden Activation	ReLU
Batch Size	128
Width Per Layer	64
Variabel Bebas (Learning Rate)	
Percobaan ke-1	0.1
Percobaan ke-2	0.01
Percobaan ke-3	0.001





Learning rate **0.001** memiliki akurasi dan F1 score yang rendah dibandingkan dengan learning rate **0.01** dan **0.1**. Learning rate **0.01** dan **0.1** menunjukkan hasil yang lebih baik dengan akurasi dan F1 score yang tinggi dan hampir serupa. Ini menunjukkan bahwa learning rate terlalu kecil (**0.001**) menyebabkan model sulit untuk mencapai konvergensi optimal.

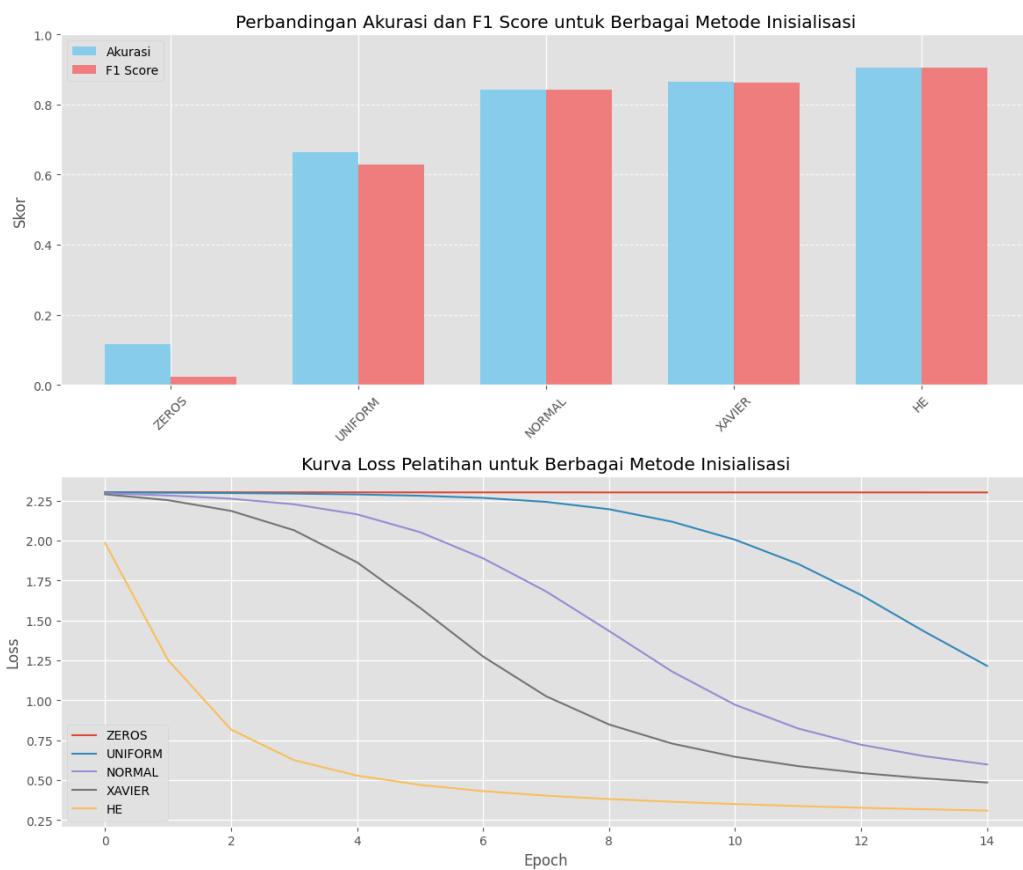
LR = 0.001 memiliki penurunan loss yang sangat lambat, menunjukkan bahwa model mengalami kesulitan dalam pembelajaran. **LR = 0.01** menunjukkan penurunan loss yang cepat dan stabil. **LR = 0.1** mengalami penurunan loss yang cepat tetapi sedikit lebih fluktuatif dibandingkan **0.01**. Ini bisa menandakan potensi ketidakstabilan saat training, yang bisa menyebabkan model overshooting pada minimum loss.

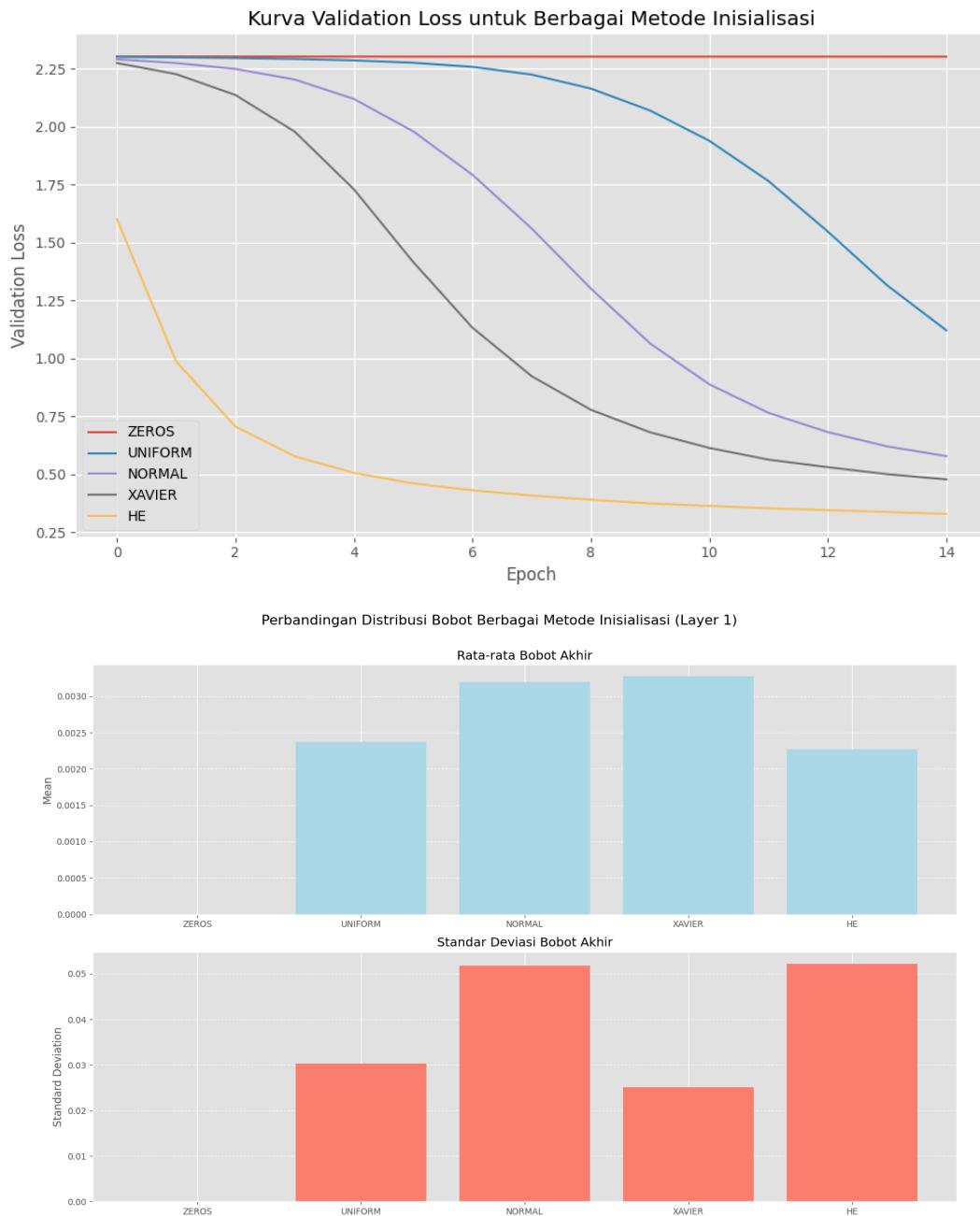
2.2.4 Pengaruh Inisialisasi Bobot

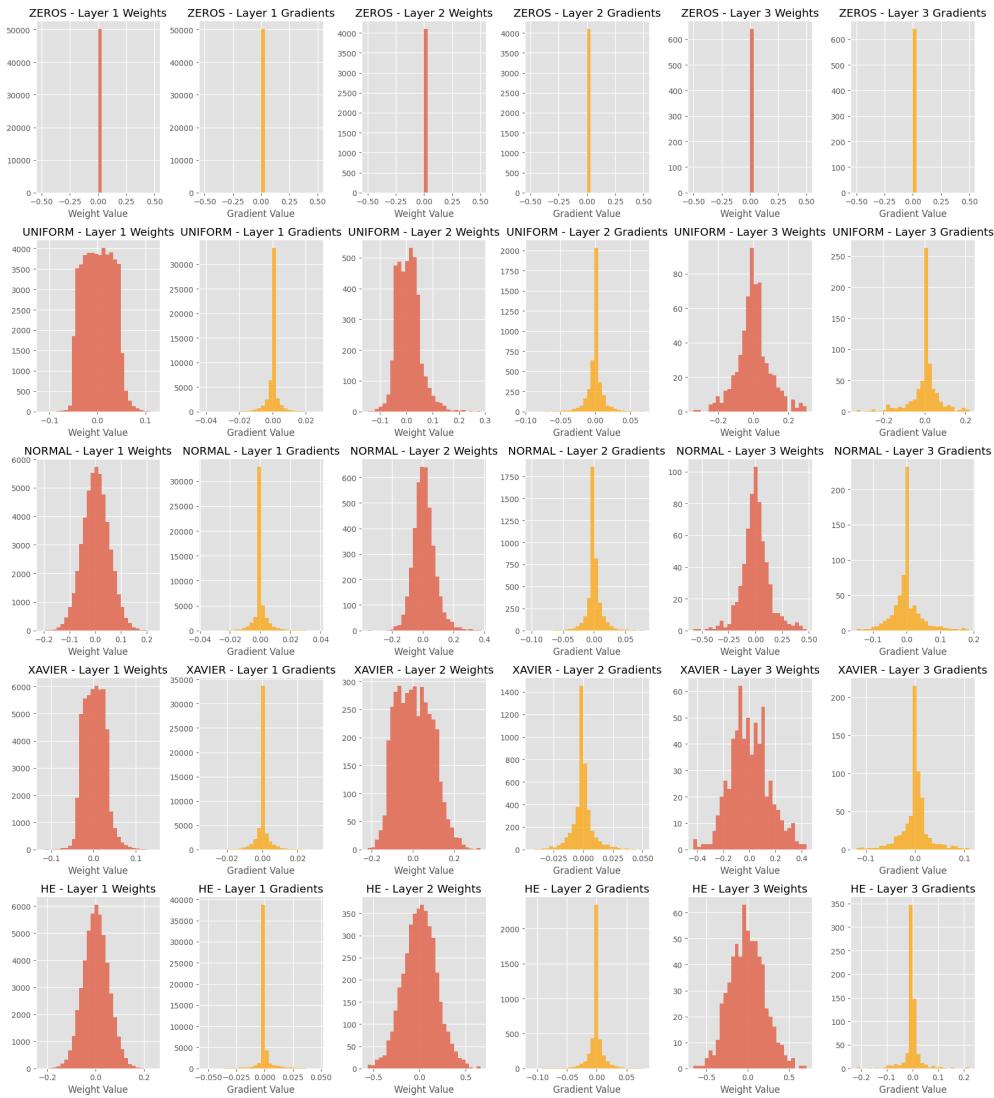
Tabel 2.2.4.1. Pengujian Inisialisasi Bobot

Variabel Tetap	
Depth	2
Learning Rate	0.01
Epochs	15
Hidden Activation	ReLU
Batch Size	128

Width Per Layer	64
Variabel Bebas (Inisialisasi Bobot)	
Percobaan ke-1	Zero
Percobaan ke-2	Random Uniform
Percobaan ke-3	Random Normal
Percobaan ke-4	Xavier
Percobaan ke-5	He







Zero Initialization memiliki kinerja terburuk, dengan akurasi dan F1-score yang sangat rendah karena bobot yang tidak berubah signifikan selama pelatihan. **Random Uniform** dan **Random Normal** menunjukkan peningkatan, tetapi masih di bawah metode yang lebih terstruktur seperti Xavier dan He. **Xavier dan He Initialization** memberikan kinerja terbaik, dengan He lebih unggul karena model menggunakan ReLU sebagai fungsi aktivasi.

Zero Initialization menunjukkan loss yang tetap tinggi karena model tidak belajar dengan baik. **Random Uniform** dan **Random Normal** memperlihatkan penurunan loss yang lebih baik tetapi masih kurang stabil. **Xavier Initialization** menunjukkan penurunan loss yang cukup baik tetapi sedikit lebih lambat dibanding He. **He Initialization** memiliki loss yang paling cepat menurun dan stabil, mengindikasikan konvergensi yang lebih cepat dan pembelajaran yang lebih efektif.

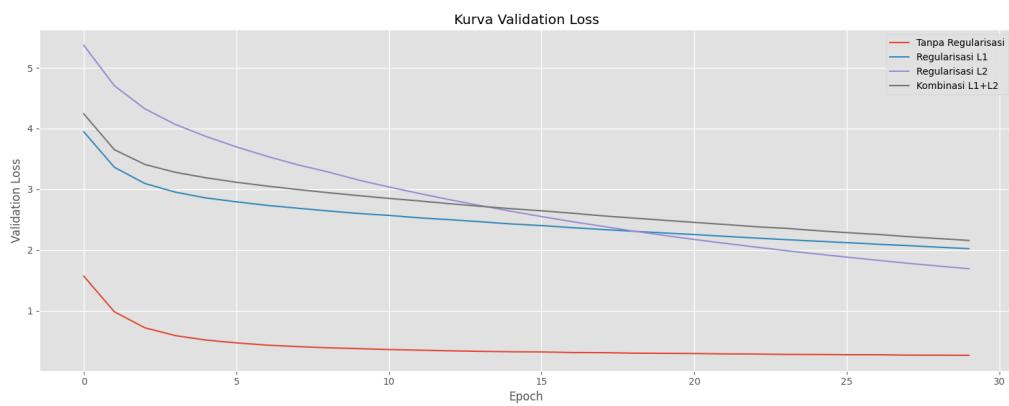
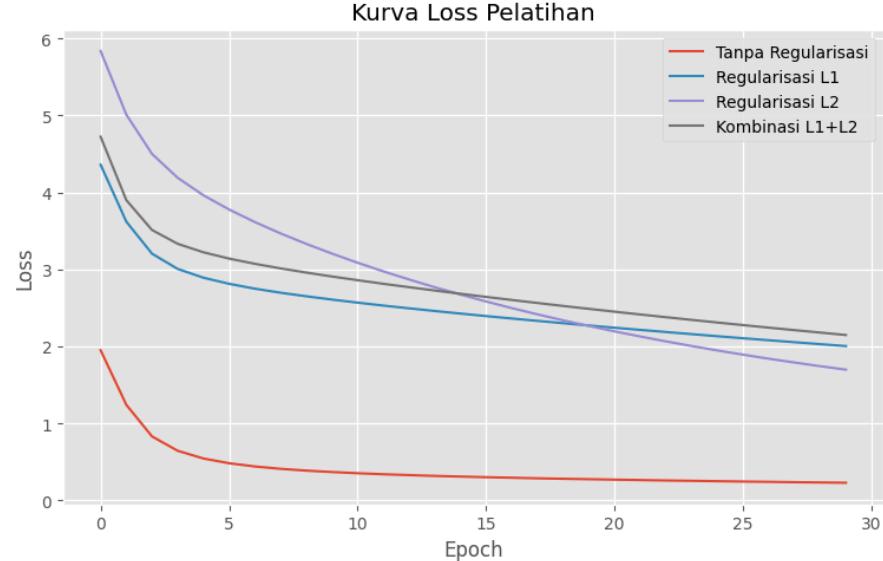
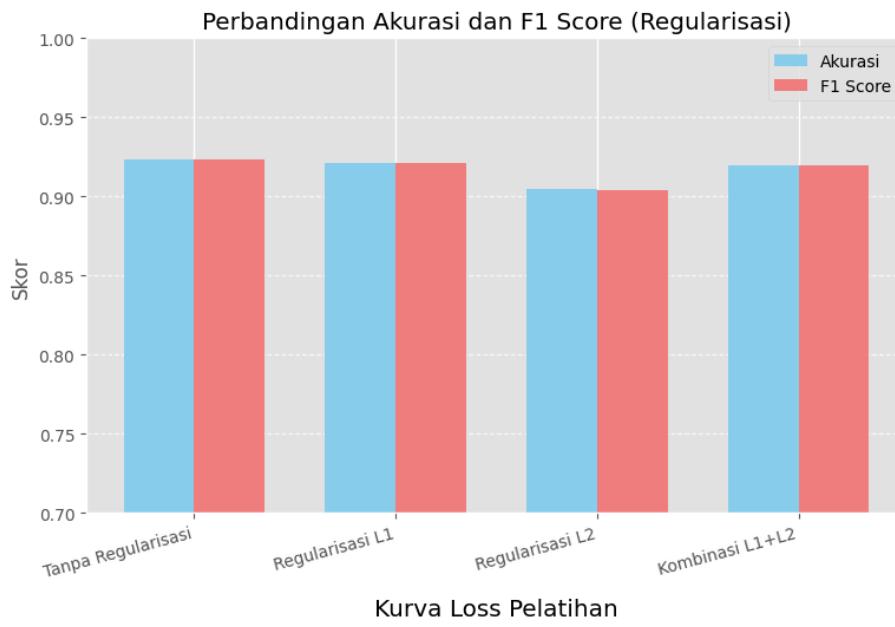
Zero Initialization yang berarti seluruh bobot diinisialisasi dengan nol, menyebabkan setiap neuron dalam layer yang sama menerima update yang sama selama backpropagation. Ini **menghambat pembelajaran karena semua neuron tetap simetris**, sehingga model tidak bisa belajar secara efektif. **Random Uniform & Random Normal** memiliki distribusi bobot awal menyebar lebih luas dibanding zero initialization, tetapi bisa mengarah pada *vanishing* atau *exploding gradients* tergantung pada rentang nilai awal. **Xavier Initialization** dirancang untuk fungsi aktivasi simetris seperti **sigmoid dan tanh**, dengan bobot yang dipilih agar mempertahankan variansi sinyal sepanjang lapisan. Ini membantu mengurangi risiko vanishing atau exploding gradients. **He Initialization cocok untuk ReLU** dan varian turunannya, memastikan bobot memiliki variansi yang cukup untuk mendukung pembelajaran yang stabil di jaringan yang lebih dalam.

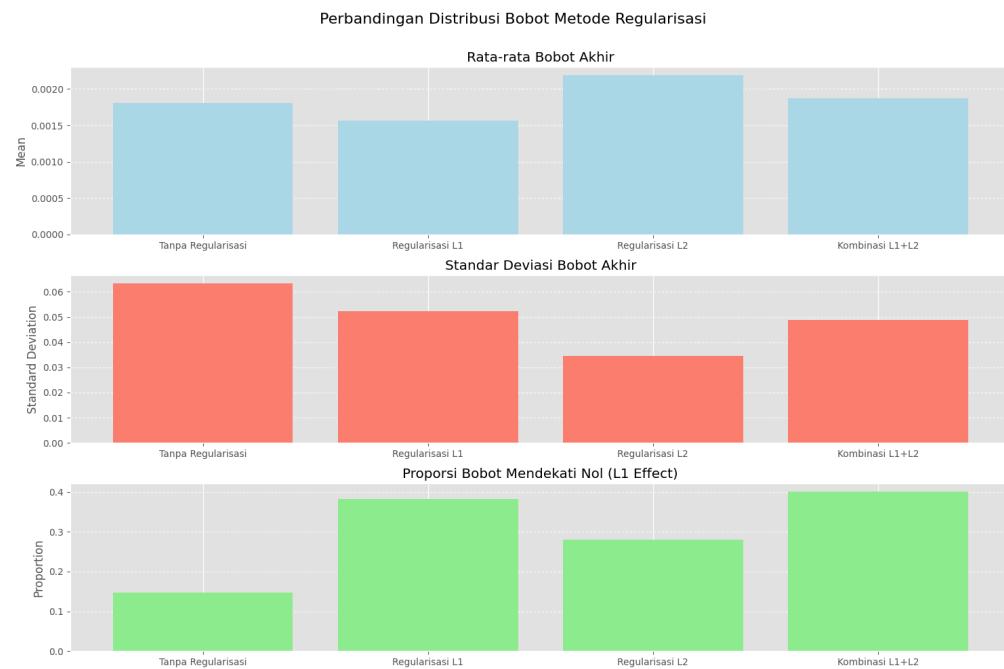
2.2.5 Pengaruh Regularisasi

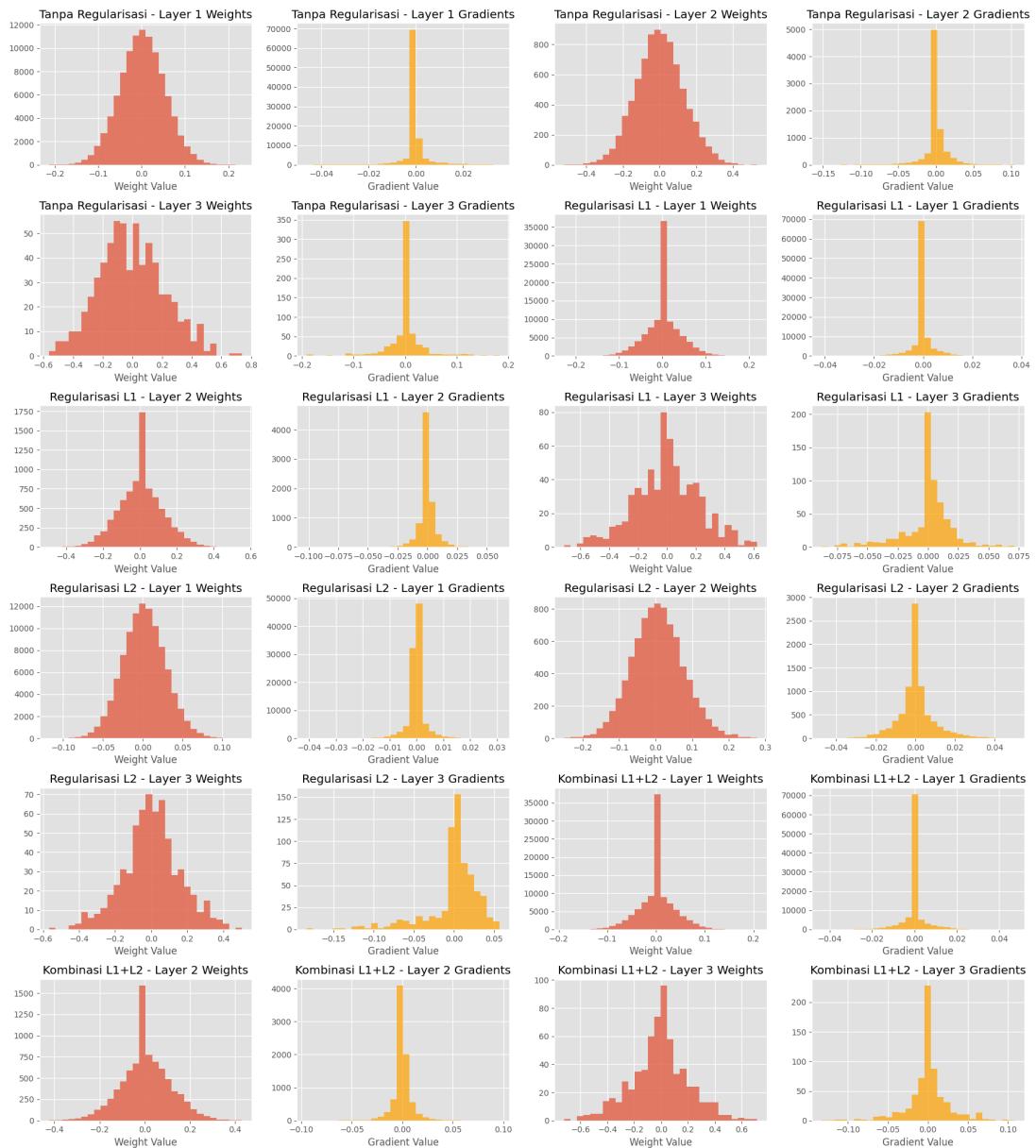
Tabel 2.2.5.1. Pengujian Regularisasi

Tanpa Regularisasi	
Depth	2
Learning Rate	0.01
Epochs	15
Hidden Activation	ReLU
Weight Initializer	he
Batch Size	128
Width Per Layer	[128,64]
L1	
Depth	2
Learning Rate	0.01
Epochs	15
Hidden Activation	ReLU
Weight Initializer	he
Batch Size	128
Width Per Layer	[128,64]

L2	
Depth	2
Learning Rate	0.01
Epochs	15
Hidden Activation	ReLU
Weight Initializer	he
Batch Size	128
Width Per Layer	[128,64]
Kombinasi L1 + L2	
Depth	2
Learning Rate	0.01
Epochs	15
Hidden Activation	ReLU
Weight Initializer	he
Batch Size	128
Width Per Layer	[128,64]







Dari grafik perbandingan akurasi dan F1-score, terlihat bahwa model dengan regularisasi (baik L1, L2, maupun kombinasi L1+L2) memiliki akurasi dan F1-score yang sedikit lebih tinggi dibandingkan model tanpa regularisasi. Regularisasi L2 menunjukkan peningkatan akurasi yang paling baik dibandingkan L1, sedangkan kombinasi L1+L2 menghasilkan performa yang hampir setara dengan L2, tetapi memberikan stabilitas tambahan dalam beberapa kasus tertentu.

Model tanpa regularisasi memiliki loss pelatihan yang paling rendah, yang menunjukkan bahwa model dapat menyesuaikan bobot dengan lebih bebas. Namun, model ini menunjukkan risiko overfitting yang lebih tinggi, terlihat dari selisih yang besar antara loss pelatihan dan loss validasi. Model dengan regularisasi L1

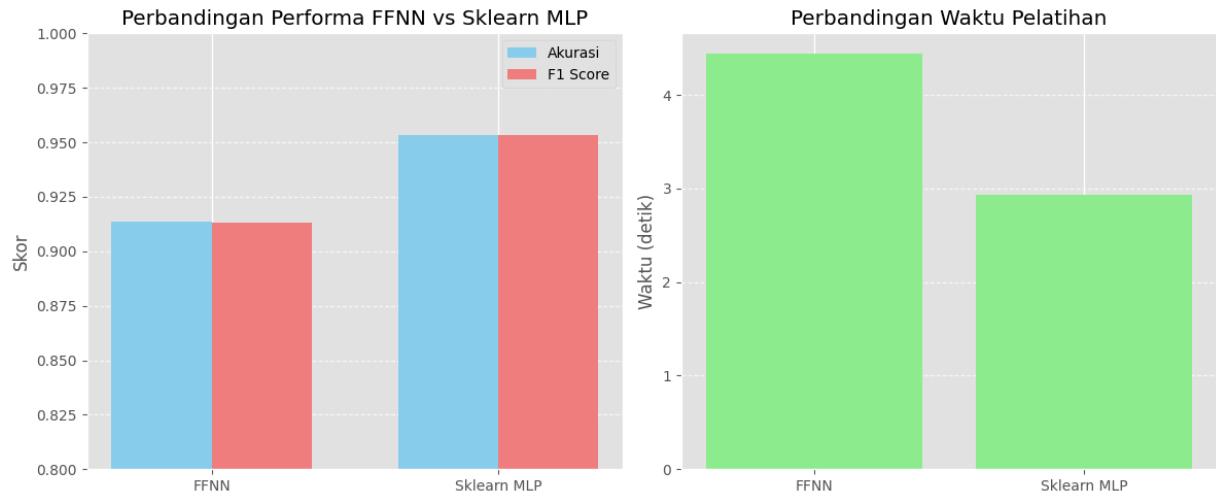
dan L2 memiliki loss pelatihan yang lebih tinggi karena adanya penalti tambahan pada bobot besar. Regularisasi L1 menghasilkan loss pelatihan yang lebih besar dibandingkan L2, karena L1 lebih agresif dalam membuat bobot menjadi lebih kecil atau nol. Kombinasi L1+L2 memiliki loss pelatihan yang berada di antara L1 dan L2, mencerminkan efek gabungan dari kedua metode. Regularisasi L1, L2, dan kombinasi L1+L2 secara signifikan mengurangi loss validasi dibandingkan model tanpa regularisasi, dengan regularisasi L2 menghasilkan loss validasi paling rendah, yang menunjukkan bahwa L2 lebih efektif dalam meningkatkan generalisasi model.

Regularisasi L1 menyebabkan banyak bobot bernilai nol atau sangat kecil, menciptakan efek **sparsity* yang membuat model lebih jarang mengaktifkan neuron tertentu. Sebaliknya, regularisasi L2 menghasilkan bobot yang lebih kecil tetapi terdistribusi lebih merata, sehingga model lebih stabil dalam memanfaatkan semua neuron. Kombinasi L1+L2 memberikan efek hybrid: menghasilkan bobot yang kecil seperti L2, tetapi juga mempertahankan **sparsity* sebagian seperti L1. Model tanpa regularisasi memiliki distribusi bobot yang lebih luas, menunjukkan bahwa bobot dapat berkembang bebas tanpa pembatasan, tetapi ini berisiko meningkatkan overfitting.

Distribusi bobot dan gradien juga menunjukkan bahwa model tanpa regularisasi memiliki distribusi yang lebih lebar dibandingkan model dengan regularisasi. Regularisasi L1 secara signifikan mengarahkan bobot menuju nol, terutama di lapisan terakhir, mencerminkan efek **sparsity*. Regularisasi L2 memperkecil bobot secara keseluruhan tetapi mempertahankan distribusi yang lebih merata. Kombinasi L1+L2 menghasilkan distribusi yang lebih terkendali dengan bobot lebih kecil dan sebagian **sparsity*, memberikan keseimbangan antara fleksibilitas dan regularisasi yang ketat.

***Sparsity:** Keadaan di mana banyak parameter model bernilai nol, sehingga model menjadi lebih sederhana, efisien, dan mengurangi kompleksitas.

2.2.6 Perbandingan dengan Library SKLearn



Model MLP dari Scikit-Learn memiliki performa yang lebih baik dibandingkan dengan FFNN from Scratch, ditunjukkan oleh skor akurasi dan F1-score yang lebih tinggi. FFNN from Scratch memiliki akurasi dan F1-score sekitar 0.91, sedangkan MLP Sklearn mencapai sekitar 0.95. Hal ini menunjukkan bahwa **MLP Sklearn lebih optimal dalam melakukan klasifikasi**, kemungkinan karena implementasi yang lebih matang, optimasi yang lebih baik, dan penggunaan regulasi bawaan.

FFNN from Scratch memiliki waktu pelatihan yang lebih lama dibandingkan **MLP Sklearn**. **Model FFNN membutuhkan lebih dari 4 detik untuk pelatihan**, sedangkan **MLP Sklearn hanya sekitar 3 detik**. Ini mengindikasikan bahwa **MLP Sklearn lebih efisien dalam proses training**, kemungkinan karena penggunaan optimasi yang lebih cepat dan implementasi dengan pustaka numerik yang lebih dioptimalkan.

BAB III

Kesimpulan dan Saran

3.1. Kesimpulan

Berdasarkan hasil pengujian yang telah dilakukan, dapat disimpulkan bahwa pemilihan jumlah neuron dan kedalaman layer berpengaruh terhadap kinerja model. Penambahan jumlah neuron per layer dapat meningkatkan kinerja model dan mempercepat konvergensi, tetapi perbedaannya dalam akurasi dan F1-score tidak terlalu signifikan. Model yang lebih dalam juga menunjukkan performa yang sedikit lebih baik dibandingkan model yang lebih dangkal, meskipun setelah titik tertentu peningkatan yang diperoleh menjadi semakin kecil. Namun, penggunaan terlalu banyak neuron atau layer dapat menyebabkan overfitting jika tidak dikontrol dengan baik.

Dari segi fungsi aktivasi, ReLU, Leaky ReLU, dan Swish merupakan pilihan terbaik untuk hidden layer karena mampu mengatasi masalah vanishing gradient, sementara Sigmoid dan Tanh kurang direkomendasikan. Swish memberikan sedikit keunggulan dalam menurunkan loss dibandingkan ReLU dan Leaky ReLU, tetapi perbedaannya tidak terlalu signifikan. Softmax tidak digunakan sebagai hidden activation karena lebih cocok untuk output layer.

Pemilihan learning rate juga berpengaruh signifikan terhadap stabilitas dan kecepatan konvergensi model. Learning rate 0.01 menjadi pilihan terbaik karena memberikan keseimbangan antara kecepatan pelatihan dan stabilitas pembelajaran. Learning rate yang terlalu besar (misalnya 0.1) dapat menyebabkan ketidakstabilan, sedangkan learning rate yang terlalu kecil (misalnya 0.001) memperlambat pelatihan dan membuat model kurang optimal.

Dalam hal inisialisasi bobot, Zero Initialization tidak direkomendasikan karena menyebabkan simetri dalam jaringan dan menghambat pembelajaran. Metode Xavier lebih cocok untuk aktivasi sigmoid/tanh, tetapi kurang optimal untuk ReLU. He Initialization terbukti sebagai metode terbaik untuk jaringan dengan aktivasi ReLU, karena mempercepat konvergensi dan meningkatkan performa model dibandingkan metode lainnya. Sementara itu, metode Random Uniform dan Random Normal memiliki hasil yang bervariasi dan kurang stabil dibandingkan Xavier atau He.

Regularisasi memiliki peran penting dalam mencegah overfitting. Model tanpa regularisasi cenderung mengalami overfitting, ditandai dengan loss validasi yang tetap tinggi meskipun loss pelatihan menurun drastis. Regularisasi L1 efektif dalam mengurangi overfitting dengan membuat banyak bobot bernilai nol, tetapi dapat terlalu agresif dalam menghilangkan fitur yang kurang penting. Sementara itu, Regularisasi L2 mengurangi overfitting dengan cara yang lebih halus dan stabil dibandingkan L1, sehingga lebih efektif dalam meningkatkan akurasi dan generalisasi model.

Secara keseluruhan, kombinasi yang optimal dalam Feed-Forward Neural Network terdiri dari jumlah neuron dan kedalaman layer yang seimbang, penggunaan ReLU atau Swish

sebagai fungsi aktivasi, learning rate sekitar 0.01, inisialisasi bobot He, serta penerapan regularisasi L2 untuk mencegah overfitting tanpa kehilangan informasi penting dalam bobot model.

3.2. Saran

Berdasarkan **Tugas Besar 1 Pembelajaran Mesin** mengenai pengembangan model **Feedforward Neural Network**, terdapat beberapa saran yang dapat diterapkan dalam pengembangan model yang memiliki pendekatan Jaringan Saraf (Neural Network):

1. Pemilihan Arsitektur yang Seimbang

Penggunaan jumlah neuron dan kedalaman layer yang seimbang sangat disarankan untuk menghindari overfitting maupun underfitting.

2. Penerapan teknik Regularisasi yang Tepat

Regularisasi L2 lebih disarankan dibandingkan L1 karena lebih stabil dan tidak terlalu agresif dalam menghilangkan bobot.

3. Pemilihan Learning rate yang Optimal

Learning rate 0.01 terbukti memberikan keseimbangan antara kecepatan konvergensi dan stabilitas pelatihan. Percobaan yang lebih bervariasi dapat membantu kita dalam menentukan *turning point* dalam learning rate terhadap performa pelatihan. Hal ini dapat memberikan kita sebuah learning rate yang optimal.

4. Penggunaan Inisialisasi Bobot yang Sesuai

He Initialization sangat direkomendasikan untuk jaringan dengan aktivasi ReLU karena mempercepat konvergensi. Metode Xavier lebih cocok jika menggunakan aktivasi Sigmoid atau Tanh.

Dengan menerapkan saran-saran ini, diharapkan pengembangan **Model Pembelajaran Mesin** berbasis **Neural Network** dapat menghasilkan performa yang lebih baik dan lebih generalisasi terhadap berbagai dataset.

BAB IV

Referensi

Tim Pengajar IF3270 Pembelajaran Mesin. 2024/2025. Salindia Perkuliahan Feedforward Neural Network (FFNN)

3Blue1Brown. (2022, September 12). *Backpropagation explained* [Video]. YouTube.
<https://youtu.be/S5AGN9XfPK4?si=pk6Ip25RP3k4cYiB>

GeeksforGeeks. (n.d.). *Activation functions in neural networks*. Retrieved March 28, 2025, from
<https://www.geeksforgeeks.org/activation-functions-neural-networks/>

GeeksforGeeks. (n.d.). *Backpropagation in neural network*. Retrieved March 28, 2025, from
<https://www.geeksforgeeks.org/backpropagation-in-neural-network/>

GeeksforGeeks. (n.d.). *Neural networks: A beginner's guide*. Retrieved March 28, 2025, from
<https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/>

GeeksforGeeks. (n.d.). *What is the dying ReLU problem in neural networks?* Retrieved March 28, 2025, from
<https://www.geeksforgeeks.org/what-is-the-dying-relu-problem-in-neural-networks/>

GeeksforGeeks. (n.d.). *Vanishing and exploding gradients problems in deep learning*. Retrieved March 28, 2025, from
<https://www.geeksforgeeks.org/vanishing-and-exploding-gradients-problems-in-deep-learning/>

IBM. (n.d.). *Loss function*. IBM Think. Retrieved March 28, 2025, from
<https://www.ibm.com/think/topics/loss-function>

StatQuest with Josh Starmer. (2023, June 25). *Introduction to neural networks* [Video]. YouTube. <https://youtu.be/URJ9pP1aURo?si=dM3C6ZP4PTAVO4ur>

Lampiran

Repository GitHub

https://github.com/alandmprtma/IF3270_Tubes1_57

Pembagian Kerja Kelompok

NIM	Nama	Pekerjaan
13522124	Aland Mulia Pratama	Mengerjakan bonus Regularization L1 dan L2. Melakukan pengujian model pada .ipynb serta formatting jupiter notebook. Melakukan pengujian pengaruh Regularization L1 dan L2 pada jupiter notebook. Mengerjakan Deskripsi Persoalan, Pembahasan, Kesimpulan dan Saran pada laporan.
13522135	Christian Justin Hendrawan	Mengimplementasikan fungsi-fungsi aktivasi yakni kelas Activation. Mengimplementasikan weight initializer yakni kelas Initializer. Mengimplementasikan kelas Layer. Mengimplementasikan fungsi loss yakni kelas Loss. Mengimplementasikan model FFNN yakni kelas FFNN. Mengimplementasikan fungsi untuk load dan train model. Melakukan pengujian model dengan hyperparameter width, depth, fungsi aktivasi, learning rate, dan inisialisasi bobot. Mengerjakan bagian pembahasan yakni Penjelasan Implementasi Feed Forward Neural Network.
13522150	Albert Ghazaly	Mengerjakan spesifikasi bonus metode inisialisasi bobot yakni Xavier Initialization dan He Initialization. Mengerjakan spesifikasi bonus fungsi aktivasi yakni Swish Activation dan Leaky ReLU Activation function. Mengerjakan hasil pengujian, penjelasan forward propagation, dan backward propagation pada laporan.