TUGAS 1

ADVANCE DEEP LEARNING

NIM : 14230018

NAMA : ILHAM MAULANA

SourceCode : https://colab.research.google.com/drive/1K6xZbYdRAE7_QhEOlhq4zNqBAiZXI_XJ?usp=sharing

A. Dataset

Dataset ini adalah bagian dari proyek CUMIDA (CUtting-edge MIcroarray DAta analysis), yang merupakan sumber data yang disediakan oleh Laboratório de Genômica Computacional (LabGC) di Departamento de Genética da Universidade Federal do Rio Grande do Sul (UFRGS), Brazil. Dataset yang Anda tunjukkan adalah bagian dari dataset Prostate_GSE6919_U95B, yang merupakan hasil dari studi mikroarray pada kanker prostat.

https://sbcb.inf.ufrgs.br/data/cumida/Genes/Prostate/GSE6919_U95B/Prostate_GSE6919_U95B.csv

TYPE	GSE	GPL PLATFORM	SAMPLES	GENES	CLASSES	
Prostate	6919_U95B	92	124	12621	2	Download

ZEROR	SVM	MLP	DT	NB	RF	HC	KNN	K-MEANS
0.52	0.68	0.62	0.6	0.71	0.67	0.51	0.56	0.54

	samples	type	41880_at	41881_at	41882_at	41883_at	41884_at	41885_at	41886_r_at	41887_at	...	AFFX-ThrX-3_at	AFFX-ThrX-5_at	AFFX-ThrX-M_at	AFFX-TrpnX-3_at	AFFX-TrpnX-5_at
0	GSM152992.CEL	primary_prostate_tumor	2.414076	4.113824	2.035911	3.102248	2.115578	1.775455	6.107839	2.160168	...	2.955998	2.910953	2.095267	1.617076	2.060144
1	GSM152993.CEL	primary_prostate_tumor	2.385157	4.078664	2.123064	3.087631	2.254190	1.815183	5.708878	2.134447	...	3.196521	2.975412	2.249950	1.757867	2.352185
2	GSM152994.CEL	primary_prostate_tumor	2.295522	4.085505	2.144344	3.071539	2.229422	1.985899	5.679248	2.100443	...	2.929904	2.857025	2.047436	1.625339	2.065674
3	GSM152995.CEL	primary_prostate_tumor	2.260478	4.466391	2.206410	3.505265	2.605014	1.887307	5.935039	2.261295	...	3.578538	3.420946	2.736342	1.940826	2.713500
4	GSM152996.CEL	primary_prostate_tumor	2.229731	4.291435	2.506255	3.220628	2.404673	1.886664	5.965917	2.274317	...	3.558184	3.083316	2.562048	1.923414	2.775842
5	GSM152997.CEL	primary_prostate_tumor	2.323719	4.042794	2.219460	3.030761	2.539964	1.836586	5.970438	1.880250	...	3.442099	3.239850	2.373556	1.973410	2.563835
6	GSM152998.CEL	primary_prostate_tumor	2.557032	4.196903	2.122405	3.495034	2.353686	2.100858	6.338756	2.019285	...	3.542867	3.178389	2.413796	2.038962	2.723099
7	GSM152999.CEL	primary_prostate_tumor	2.441765	4.148545	1.968985	3.230116	2.240704	1.896051	6.166129	2.012576	...	3.363998	3.314865	2.345718	1.927122	2.600874
8	GSM153000.CEL	primary_prostate_tumor	2.604707	4.234341	2.167552	3.394769	2.294088	2.089270	6.190621	2.030608	...	3.703331	3.263648	2.769345	2.183191	2.62121[illegible]
9	GSM153001.CEL	primary_prostate_tumor	2.364670	4.183144	1.866217	3.169024	2.371755	1.983816	5.748455	2.142535	...	3.077640	3.048728	2.230446	1.928804	2.466[illegible]

10 rows × 12622 columns

1. Membangun 4 DNN dengan variasi

a. Hidden layer 1: 1000 neurons, ReLU activation

```
def build_model_one_hidden():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(1000, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # Hidden layer with 1000 neurons and
ReLU activation
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')           # Output layer with sigmoid activation
    ])
    return model
```

b. Hidden layer 2: 500 neurons, ReLU activation

```
def build_model_two_hidden():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(500, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # 1st hidden layer with 500 neurons
and ReLU activation
        tf.keras.layers.Dense(500, activation='relu'),
# 2nd hidden layer with 500 neurons and ReLU activation
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')          # Output layer with sigmoid activation
    ])
    return model
```

c. Hidden layer 3: 250 neurons, ReLU activation

```
def build_model_three_hidden():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(250, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # First hidden layer with 250 neurons
and ReLU activation
        tf.keras.layers.Dense(250, activation='relu'),
# Second hidden layer with 250 neurons and ReLU activation
        tf.keras.layers.Dense(250, activation='relu'),
# Third hidden layer with 250 neurons and ReLU activation
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model
```

d. Hidden layer 4: 100 neurons, ReLU activation

```
def build_model_four_hidden():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(100, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # First hidden layer with 100 neurons
and ReLU activation
        tf.keras.layers.Dense(100, activation='relu'),
# Second hidden layer with 100 neurons and ReLU activation
        tf.keras.layers.Dense(100, activation='relu'),
# Third hidden layer with 100 neurons and ReLU activation
        tf.keras.layers.Dense(100, activation='relu'),
# Fourth hidden layer with 100 neurons and ReLU activation
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model
```

e. Output Layer: (jlh kelas) neuron, Sigmoid activation, Epoch 100, batch size 10, Optimizer: SGD

```
model_one_hidden_layer = build_model_one_hidden()
model_two_hidden_layer = build_model_two_hidden()
model_three_hidden_layer = build_model_three_hidden()
model_four_hidden_layer = build_model_four_hidden()
```

```
model_one_hidden_layer.compile(optimizer='sgd',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_one_hidden_layer.fit(X_train_scaled, y_train, epochs=100,
batch_size=10, validation_data=(X_val_scaled, y_val))

model_two_hidden_layer.compile(optimizer='sgd',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_two_hidden_layer.fit(X_train_scaled, y_train, epochs=100,
batch_size=10, validation_data=(X_val_scaled, y_val))

model_three_hidden_layer.compile(optimizer='sgd',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_three_hidden_layer.fit(X_train_scaled, y_train, epochs=100,
batch_size=10, validation_data=(X_val_scaled, y_val))

model_four_hidden_layer.compile(optimizer='sgd',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_four_hidden_layer.fit(X_train_scaled, y_train, epochs=100,
batch_size=10, validation_data=(X_val_scaled, y_val))
```

f. Hasilnya

	Optimizer	Hidden Layers	Validation Accuracy	Test Accuracy
0	SGD	1	0.75	0.64
1	SGD	2	0.75	0.64
2	SGD	3	0.70	0.68
3	SGD	4	0.70	0.64

Hasil terbaik test accuracy menggunakan SGD Optimizer adalah pada **layer3** dengan Test Accuracy **0.68**

2. Menganti Optimizer

a. Adam

```
model_one_hidden_layer.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_one_hidden_layer.fit(X_train_scaled, y_train, epochs=100,
batch_size=10, validation_data=(X_val_scaled, y_val))
```

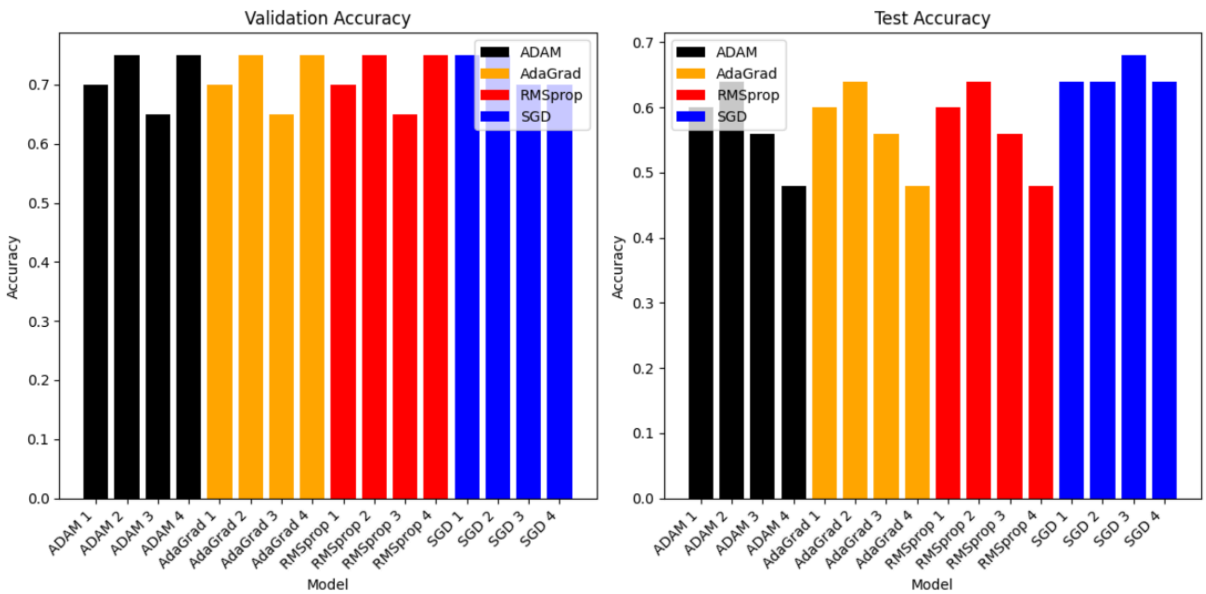
b. AdaGrad

```
optimizer_adagrad = tf.keras.optimizers.Adagrad()
model_one_hidden_layer.compile(optimizer=optimizer_adagrad,
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_one_hidden_layer.fit(X_train_scaled, y_train, epochs=100,
batch_size=10, validation_data=(X_val_scaled, y_val))
```

c. RMSProp

```
optimizer_rmsprop = tf.keras.optimizers.RMSprop()
model_one_hidden_layer.compile(optimizer=optimizer_rmsprop,
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_one_hidden_layer.fit(X_train_scaled, y_train, epochs=100,
batch_size=10, validation_data=(X_val_scaled, y_val))
```

d. Hasilnya



	Optimizer	Hidden Layers	Validation Accuracy	Test Accuracy
0	SGD	1	0.75	0.64
1	SGD	2	0.75	0.64
2	SGD	3	0.70	0.68
3	SGD	4	0.70	0.64
4	ADAM	1	0.70	0.60
5	ADAM	2	0.75	0.64
6	ADAM	3	0.65	0.56
7	ADAM	4	0.75	0.48
8	AdaGrad	1	0.70	0.60
9	AdaGrad	2	0.75	0.64
10	AdaGrad	3	0.65	0.56
11	AdaGrad	4	0.75	0.48
12	RMSprop	1	0.70	0.60
13	RMSprop	2	0.75	0.64
14	RMSprop	3	0.65	0.56
15	RMSprop	4	0.75	0.48

Hasil terbaik test accuracy yaitu menggunakan SGD Optimizer adalah pada **layer3**

Dengan Validation Accuracy **0.70 dan** Test Accuracy **0.68**

3. Menambahkan Dropout 50%

a. Di salah satu layer

```
def build_model_three_hidden_with_dropout():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(250, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # First hidden layer with 250 neurons
and ReLU activation
        tf.keras.layers.Dense(250, activation='relu'),
# Second hidden layer with 250 neurons and ReLU activation
        tf.keras.layers.Dense(250, activation='relu'),
# Third hidden layer with 250 neurons and ReLU activation
        tf.keras.layers.Dropout(0.5),  # Dropout layer with dropout rate of 50%
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model
```

b. Di semua layer

```
# Function to build the DNN model with one hidden layer
def build_model_one_hidden_with_dropout():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(1000, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # Hidden layer with 1000 neurons and
ReLU activation
        tf.keras.layers.Dropout(0.5),  # Dropout layer with dropout rate of 50%
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')          # Output layer with sigmoid activation
    ])
    return model

# Function to build the DNN model with two hidden layer
def build_model_two_hidden_with_dropout():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(500, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # 1st hidden layer with 500 neurons
and ReLU activation
        tf.keras.layers.Dense(500, activation='relu'),
# 2nd hidden layer with 500 neurons and ReLU activation
```

```
        tf.keras.layers.Dropout(0.5),  # Dropout layer with dropout rate of 50%
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model

# Function to build the DNN model with three hidden layer
def build_model_three_hidden_with_dropout():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(250, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # First hidden layer with 250 neurons
and ReLU activation
        tf.keras.layers.Dense(250, activation='relu'),
# Second hidden layer with 250 neurons and ReLU activation
        tf.keras.layers.Dense(250, activation='relu'),
# Third hidden layer with 250 neurons and ReLU activation
        tf.keras.layers.Dropout(0.5),  # Dropout layer with dropout rate of 50%
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model

# Function to build the DNN model with four hidden layer
def build_model_four_hidden_with_dropout():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(100, activation='relu',
input_shape=(X_train_scaled.shape[1],)),  # First hidden layer with 100 neurons
and ReLU activation
        tf.keras.layers.Dense(100, activation='relu'),
# Second hidden layer with 100 neurons and ReLU activation
        tf.keras.layers.Dense(100, activation='relu'),
# Third hidden layer with 100 neurons and ReLU activation
        tf.keras.layers.Dense(100, activation='relu'),
# Fourth hidden layer with 100 neurons and ReLU activation
        tf.keras.layers.Dropout(0.5),  # Dropout layer with dropout rate of 50%
        tf.keras.layers.Dense(len(label_encoder.classes_),
activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model
```

c. Hasil

Hasil terbaik test accuracy yaitu tetap pada SGD Optimizer adalah pada **layer3** Dengan Validation Accuracy **0.70 dan** Test Accuracy **0.68**

	Optimizer	Hidden Layers	Validation Accuracy	Test Accuracy
0	SGD	1	0.75	0.64
1	SGD	2	0.75	0.64
2	SGD	3	0.70	0.68
3	SGD	4	0.70	0.64
4	ADAM	1	0.70	0.60
5	ADAM	2	0.75	0.64
6	ADAM	3	0.65	0.56
7	ADAM	4	0.75	0.48
8	AdaGrad	1	0.70	0.60
9	AdaGrad	2	0.75	0.64
10	AdaGrad	3	0.65	0.56
11	AdaGrad	4	0.75	0.48
12	RMSprop	1	0.70	0.60
13	RMSprop	2	0.75	0.64
14	RMSprop	3	0.65	0.56
15	RMSprop	4	0.75	0.48
16	SGD+D	1	0.70	0.60
17	SGD+D	2	0.75	0.60
18	SGD+D	3	0.75	0.56
19	SGD+D	4	0.80	0.52

Hasil validation accuracy naik menjadi **0.80** pada SGD Optimizer layer 4 dengan menambahkan dropout 50%.

Information

TUGAS 1 - ADVANCE DEEP LEARNING

NIM : 14230018

NAMA : ILHAM MAULANA

Link Collab : https://colab.research.google.com/drive/1K6xZbYdRAE7_QhEOlhq4zNqBAiZXI_XJ?usp=sharing

Laporan : https://docs.google.com/document/d/1xYNUiVXY-jPQIBeP3bgkzJPTSXdF_Kyns9kOgp88i4o/edit?usp=sharing

Dataset Cumida

Double-click (or enter) to edit

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.activations import relu
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder, StandardScaler

import warnings;
warnings.filterwarnings('ignore');
```

TYPE	GSE	GPL PLATFORM	SAMPLES	GENES	CLASSES	
Prostate	6919_U95B	92	124	12621	2	Download

ZEROR	SVM	MLP	DT	NB	RF	HC	KNN	K-MEANS
0.52	0.68	0.62	0.6	0.71	0.67	0.51	0.56	0.54

https://sbcb.inf.ufrgs.br/cumida

https://sbcb.inf.ufrgs.br/data/cumida/Genes/Prostate/GSE6919_U95B/Prostate_GSE6919_U95B.csv

```
# Step 1: Load Data from CSV
df = pd.read_csv('https://raw.githubusercontent.com/k4ilham/dataset/main/Prostate_GSE6919_U95B.csv')
```

```
df.head(10)
```

	samples	type	41880_at	41881_at	41882_at	41883_at	4188
0	GSM152992.CEL	primary_prostate_tumor	2.414076	4.113824	2.035911	3.102248	2.11
1	GSM152993.CEL	primary_prostate_tumor	2.385157	4.078664	2.123064	3.087631	2.25
2	GSM152994.CEL	primary_prostate_tumor	2.295522	4.085505	2.144344	3.071539	2.22
3	GSM152995.CEL	primary_prostate_tumor	2.260478	4.466391	2.206410	3.505265	2.60
4	GSM152996.CEL	primary_prostate_tumor	2.229731	4.291435	2.506255	3.220628	2.40
5	GSM152997.CEL	primary_prostate_tumor	2.323719	4.042794	2.219460	3.030761	2.53
6	GSM152998.CEL	primary_prostate_tumor	2.557032	4.196903	2.122405	3.495034	2.35
7	GSM152999.CEL	primary_prostate_tumor	2.441765	4.148545	1.968985	3.230116	2.24
8	GSM153000.CEL	primary_prostate_tumor	2.604707	4.234341	2.167552	3.394769	2.29
9	GSM153001.CEL	primary_prostate_tumor	2.364670	4.183144	1.866217	3.169024	2.37

10 rows × 12622 columns

```
X = df.drop(df.columns[[0, 1]], axis=1)
y = df['type']
```

```
df.isnull().sum().sum()
```

```
0
```

```
# Encoding label menjadi angka menggunakan LabelEncoder
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
```

```
# Bagi data menjadi data pelatihan, validasi, dan pengujian
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

print(X_train.shape)
```

```
(79, 12620)
```

```
# Normalisasi data
#scaler = StandardScaler()
#X_train_scaled = scaler.fit_transform(X_train)
#X_val_scaled = scaler.transform(X_val)

# Normalisasi data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

```
# Membuat DataFrame kosong untuk menyimpan hasil evaluasi
results_df = pd.DataFrame(columns=['Optimizer', 'Hidden Layers', 'Validation Accuracy', 'Test Accuracy'])

# Fungsi untuk menambahkan atau memperbarui nilai dalam DataFrame
def update_results(optimizer, hidden_layers, val_acc, test_acc):
    global results_df
    # Mencari baris yang sesuai dengan kombinasi optimizer dan hidden layers
    mask = (results_df['Optimizer'] == optimizer) & (results_df['Hidden Layers'] == hidden_layers)
    # Jika kombinasi sudah ada, update nilai
    if mask.any():
        results_df.loc[mask, ['Validation Accuracy', 'Test Accuracy']] = val_acc, test_acc
    # Jika kombinasi belum ada, tambahkan baris baru
    else:
        results_df = pd.concat([results_df, pd.DataFrame({'Optimizer': [optimizer],
                                                          'Hidden Layers': [hidden_layers],
                                                          'Validation Accuracy': [val_acc],
                                                          'Test Accuracy': [test_acc]})],
                                ignore_index=True)
```

1. Membangun 4 DNN

1.A. Hidden layer 1: 1000 neurons, ReLU activation

Double-click (or enter) to edit

```
# Function to build the DNN model with one hidden layer
def build_model_one_hidden():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(1000, activation='relu', input_shape=(X_train_scaled.shape[1],)),  # Hidden layer with 1000 neurons and R
        tf.keras.layers.Dense(len(label_encoder.classes_), activation='sigmoid')          # Output layer with sigmoid activation
    ])
    return model
```

1.B. Hidden layer 2: 500 neurons, ReLU activation

```
# Function to build the DNN model with two hidden layer
def build_model_two_hidden():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(500, activation='relu', input_shape=(X_train_scaled.shape[1],)),  # 1st hidden layer with 500 neurons and
        tf.keras.layers.Dense(500, activation='relu'),                                   # 2nd hidden layer with 500 neurons and ReLU a
        tf.keras.layers.Dense(len(label_encoder.classes_), activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model
```

1.C. Hidden layer 3: 250 neurons, ReLU activation

```
# Function to build the DNN model with three hidden layer
def build_model_three_hidden():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(250, activation='relu', input_shape=(X_train_scaled.shape[1],)),  # First hidden layer with 250 neurons a
        tf.keras.layers.Dense(250, activation='relu'),                                   # Second hidden layer with 250 neurons and ReL
        tf.keras.layers.Dense(250, activation='relu'),                                   # Third hidden layer with 250 neurons and ReLU
        tf.keras.layers.Dense(len(label_encoder.classes_), activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model
```

1.D. Hidden layer 4: 100 neurons, ReLU activation

```
# Function to build the DNN model with four hidden layer
def build_model_four_hidden():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(100, activation='relu', input_shape=(X_train_scaled.shape[1],)),  # First hidden layer with 100 neurons a
        tf.keras.layers.Dense(100, activation='relu'),                                   # Second hidden layer with 100 neurons and ReL
        tf.keras.layers.Dense(100, activation='relu'),                                   # Third hidden layer with 100 neurons and ReLU
        tf.keras.layers.Dense(100, activation='relu'),                                   # Fourth hidden layer with 100 neurons and ReL
        tf.keras.layers.Dense(len(label_encoder.classes_), activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model
```

```
# Build the DNN model
model_one_hidden_layer = build_model_one_hidden()
model_two_hidden_layer = build_model_two_hidden()
model_three_hidden_layer = build_model_three_hidden()
model_four_hidden_layer = build_model_four_hidden()
```

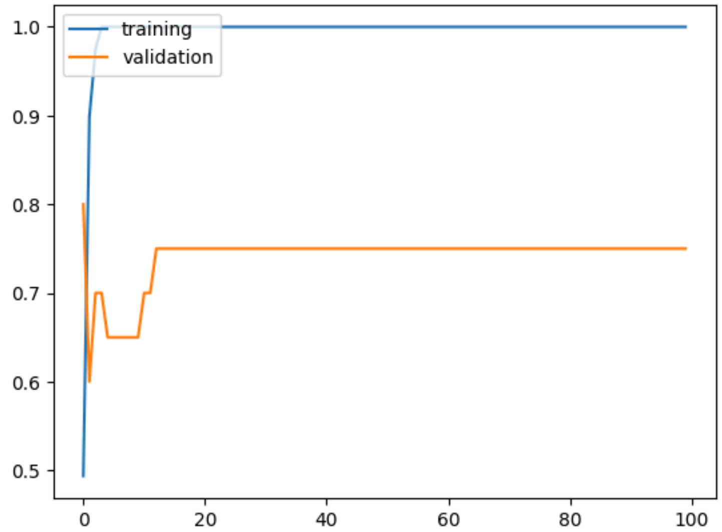
1.E. Optimizer: SGD

Train and evaluate the model with 1 hidden Layer

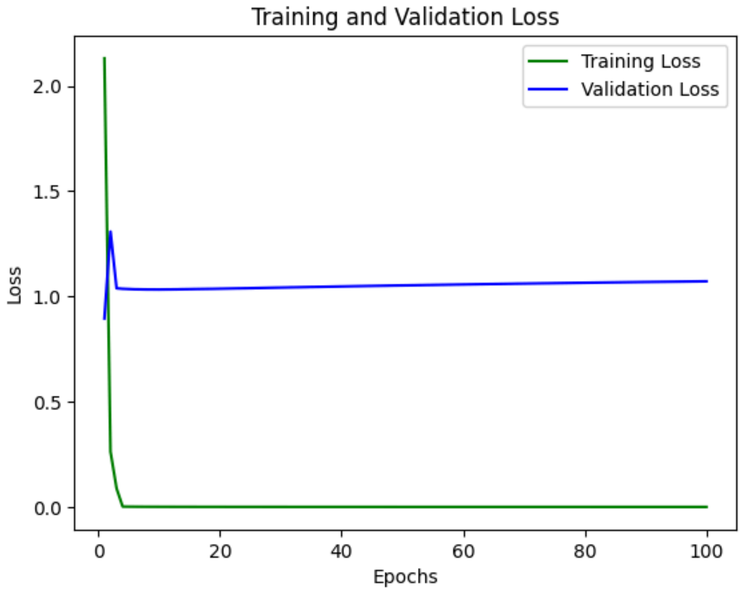
```
# Melatih model dengan data pelatihan
# optimizer_sgd = tf.keras.optimizers.SGD(learning_rate=0.01)  # Define the optimizer with learning rate
%%time
model_one_hidden_layer.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_one_hidden_layer.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_val))
```

```
Epoch 76/100
8/8 [==============================] - 1s 74ms/step - loss: 2.0608e-04 - accuracy: 1.0000 - val_loss: 1.0637 - val_accuracy: 0.75
Epoch 77/100
8/8 [==============================] - 1s 79ms/step - loss: 2.0409e-04 - accuracy: 1.0000 - val_loss: 1.0641 - val_accuracy: 0.75
Epoch 78/100
8/8 [==============================] - 1s 78ms/step - loss: 2.0199e-04 - accuracy: 1.0000 - val_loss: 1.0644 - val_accuracy: 0.75
Epoch 79/100
8/8 [==============================] - 1s 74ms/step - loss: 2.0005e-04 - accuracy: 1.0000 - val_loss: 1.0648 - val_accuracy: 0.75
Epoch 80/100
8/8 [==============================] - 1s 80ms/step - loss: 1.9822e-04 - accuracy: 1.0000 - val_loss: 1.0652 - val_accuracy: 0.75
Epoch 81/100
8/8 [==============================] - 1s 75ms/step - loss: 1.9623e-04 - accuracy: 1.0000 - val_loss: 1.0656 - val_accuracy: 0.75
Epoch 82/100
8/8 [==============================] - 1s 76ms/step - loss: 1.9440e-04 - accuracy: 1.0000 - val_loss: 1.0659 - val_accuracy: 0.75
Epoch 83/100
8/8 [==============================] - 1s 75ms/step - loss: 1.9257e-04 - accuracy: 1.0000 - val_loss: 1.0663 - val_accuracy: 0.75
Epoch 84/100
8/8 [==============================] - 1s 75ms/step - loss: 1.9071e-04 - accuracy: 1.0000 - val_loss: 1.0667 - val_accuracy: 0.75
Epoch 85/100
8/8 [==============================] - 1s 81ms/step - loss: 1.8904e-04 - accuracy: 1.0000 - val_loss: 1.0671 - val_accuracy: 0.75
Epoch 86/100
8/8 [==============================] - 1s 79ms/step - loss: 1.8735e-04 - accuracy: 1.0000 - val_loss: 1.0674 - val_accuracy: 0.75
Epoch 87/100
8/8 [==============================] - 1s 77ms/step - loss: 1.8574e-04 - accuracy: 1.0000 - val_loss: 1.0678 - val_accuracy: 0.75
Epoch 88/100
8/8 [==============================] - 1s 79ms/step - loss: 1.8411e-04 - accuracy: 1.0000 - val_loss: 1.0682 - val_accuracy: 0.75
Epoch 89/100
8/8 [==============================] - 1s 79ms/step - loss: 1.8241e-04 - accuracy: 1.0000 - val_loss: 1.0685 - val_accuracy: 0.75
Epoch 90/100
8/8 [==============================] - 1s 82ms/step - loss: 1.8088e-04 - accuracy: 1.0000 - val_loss: 1.0689 - val_accuracy: 0.75
Epoch 91/100
8/8 [==============================] - 1s 79ms/step - loss: 1.7934e-04 - accuracy: 1.0000 - val_loss: 1.0692 - val_accuracy: 0.75
Epoch 92/100
8/8 [==============================] - 1s 119ms/step - loss: 1.7781e-04 - accuracy: 1.0000 - val_loss: 1.0695 - val_accuracy: 0.7
Epoch 93/100
8/8 [==============================] - 1s 122ms/step - loss: 1.7639e-04 - accuracy: 1.0000 - val_loss: 1.0699 - val_accuracy: 0.7
Epoch 94/100
8/8 [==============================] - 1s 111ms/step - loss: 1.7486e-04 - accuracy: 1.0000 - val_loss: 1.0702 - val_accuracy: 0.7
Epoch 95/100
8/8 [==============================] - 1s 75ms/step - loss: 1.7349e-04 - accuracy: 1.0000 - val_loss: 1.0705 - val_accuracy: 0.75
Epoch 96/100
8/8 [==============================] - 1s 75ms/step - loss: 1.7207e-04 - accuracy: 1.0000 - val_loss: 1.0708 - val_accuracy: 0.75
Epoch 97/100
8/8 [==============================] - 1s 82ms/step - loss: 1.7063e-04 - accuracy: 1.0000 - val_loss: 1.0712 - val_accuracy: 0.75
Epoch 98/100
8/8 [==============================] - 1s 74ms/step - loss: 1.6931e-04 - accuracy: 1.0000 - val_loss: 1.0715 - val_accuracy: 0.75
Epoch 99/100
8/8 [==============================] - 1s 77ms/step - loss: 1.6796e-04 - accuracy: 1.0000 - val_loss: 1.0718 - val_accuracy: 0.75
Epoch 100/100
8/8 [==============================] - 1s 74ms/step - loss: 1.6666e-04 - accuracy: 1.0000 - val_loss: 1.0721 - val_accuracy: 0.75
CPU times: user 1min 9s, sys: 32.1 s, total: 1min 41s
Wall time: 1min 17s
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training', 'validation'], loc = 'upper left')
plt.show()
```



```
# Ambil loss dari history
train_loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(train_loss) + 1)



# Plot kurva loss
plt.plot(epochs, train_loss, 'g', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_one_hidden_layer.evaluate(X_val_scaled, y_val)
print("Model Validation Accuracy:", val_acc)

# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_one_hidden_layer  # Misalnya, model pertama dianggap sebagai model terbaik
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 155ms/step - loss: 1.0721 - accuracy: 0.7500
Model Validation Accuracy: 0.75
1/1 [==============================] - 0s 49ms/step - loss: 1.6707 - accuracy: 0.6400
Best Model Test Accuracy: 0.6399999856948853
```

```
results = model_one_hidden_layer.evaluate(X_test, y_test)
```

```
1/1 [==============================] - 0s 43ms/step - loss: 1.6707 - accuracy: 0.6400
```

```
update_results('SGD', 1, val_acc, test_acc)
print(results_df)
```

```
  Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0       SGD             1                 0.75           0.64
```

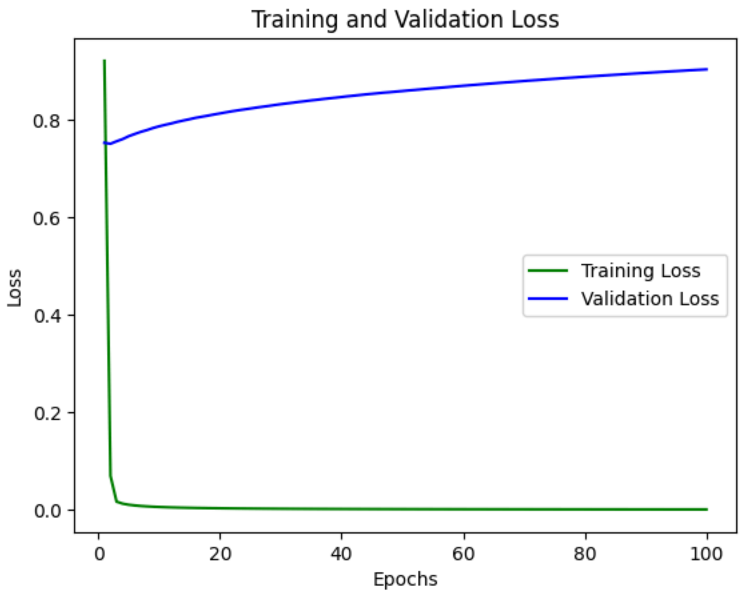
Train and evaluate the model with 2 hidden Layer

```
# Melatih model dengan data pelatihan
model_two_hidden_layer.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_two_hidden_layer.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_val))
```

```
Epoch 76/100
8/8 [==============================] - 0s 37ms/step - loss: 8.4737e-04 - accuracy: 1.0000 - val_loss: 0.8841 - val_accuracy: 0.75
Epoch 77/100
8/8 [==============================] - 0s 39ms/step - loss: 8.3655e-04 - accuracy: 1.0000 - val_loss: 0.8850 - val_accuracy: 0.75
Epoch 78/100
8/8 [==============================] - 0s 34ms/step - loss: 8.2549e-04 - accuracy: 1.0000 - val_loss: 0.8858 - val_accuracy: 0.75
Epoch 79/100
8/8 [==============================] - 0s 36ms/step - loss: 8.1503e-04 - accuracy: 1.0000 - val_loss: 0.8867 - val_accuracy: 0.75
Epoch 80/100
8/8 [==============================] - 0s 35ms/step - loss: 8.0493e-04 - accuracy: 1.0000 - val_loss: 0.8875 - val_accuracy: 0.75
Epoch 81/100
8/8 [==============================] - 0s 33ms/step - loss: 7.9500e-04 - accuracy: 1.0000 - val_loss: 0.8883 - val_accuracy: 0.75
Epoch 82/100
8/8 [==============================] - 0s 35ms/step - loss: 7.8523e-04 - accuracy: 1.0000 - val_loss: 0.8891 - val_accuracy: 0.75
Epoch 83/100
8/8 [==============================] - 0s 35ms/step - loss: 7.7525e-04 - accuracy: 1.0000 - val_loss: 0.8899 - val_accuracy: 0.75
Epoch 84/100
8/8 [==============================] - 0s 36ms/step - loss: 7.6604e-04 - accuracy: 1.0000 - val_loss: 0.8907 - val_accuracy: 0.75
Epoch 85/100
8/8 [==============================] - 0s 32ms/step - loss: 7.5700e-04 - accuracy: 1.0000 - val_loss: 0.8915 - val_accuracy: 0.75
Epoch 86/100
8/8 [==============================] - 0s 33ms/step - loss: 7.4797e-04 - accuracy: 1.0000 - val_loss: 0.8923 - val_accuracy: 0.75
Epoch 87/100
8/8 [==============================] - 0s 34ms/step - loss: 7.3928e-04 - accuracy: 1.0000 - val_loss: 0.8931 - val_accuracy: 0.75
Epoch 88/100
8/8 [==============================] - 0s 33ms/step - loss: 7.3083e-04 - accuracy: 1.0000 - val_loss: 0.8939 - val_accuracy: 0.75
Epoch 89/100
8/8 [==============================] - 0s 30ms/step - loss: 7.2243e-04 - accuracy: 1.0000 - val_loss: 0.8947 - val_accuracy: 0.75
Epoch 90/100
8/8 [==============================] - 0s 33ms/step - loss: 7.1416e-04 - accuracy: 1.0000 - val_loss: 0.8953 - val_accuracy: 0.75
Epoch 91/100
8/8 [==============================] - 0s 43ms/step - loss: 7.0606e-04 - accuracy: 1.0000 - val_loss: 0.8961 - val_accuracy: 0.75
Epoch 92/100
8/8 [==============================] - 0s 33ms/step - loss: 6.9807e-04 - accuracy: 1.0000 - val_loss: 0.8969 - val_accuracy: 0.75
Epoch 93/100
8/8 [==============================] - 0s 38ms/step - loss: 6.9046e-04 - accuracy: 1.0000 - val_loss: 0.8976 - val_accuracy: 0.75
Epoch 94/100
8/8 [==============================] - 0s 40ms/step - loss: 6.8313e-04 - accuracy: 1.0000 - val_loss: 0.8984 - val_accuracy: 0.75
Epoch 95/100
8/8 [==============================] - 0s 35ms/step - loss: 6.7569e-04 - accuracy: 1.0000 - val_loss: 0.8990 - val_accuracy: 0.75
Epoch 96/100
8/8 [==============================] - 0s 36ms/step - loss: 6.6830e-04 - accuracy: 1.0000 - val_loss: 0.8998 - val_accuracy: 0.75
Epoch 97/100
8/8 [==============================] - 0s 33ms/step - loss: 6.6115e-04 - accuracy: 1.0000 - val_loss: 0.9006 - val_accuracy: 0.75
Epoch 98/100
8/8 [==============================] - 0s 49ms/step - loss: 6.5425e-04 - accuracy: 1.0000 - val_loss: 0.9012 - val_accuracy: 0.75
Epoch 99/100
8/8 [==============================] - 0s 48ms/step - loss: 6.4751e-04 - accuracy: 1.0000 - val_loss: 0.9019 - val_accuracy: 0.75
Epoch 100/100
8/8 [==============================] - 0s 51ms/step - loss: 6.4077e-04 - accuracy: 1.0000 - val_loss: 0.9026 - val_accuracy: 0.75
```

```
# Ambil loss dari history
train_loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(train_loss) + 1)
```

```
# Plot kurva loss
plt.plot(epochs, train_loss, 'g', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_two_hidden_layer.evaluate(X_val_scaled, y_val)
print("Model 2 Hidden Layer SGD Optimizer Validation Accuracy:", val_acc)

# 5. Pilih model terbaik berdasarkan kinerja validasi
best_model = model_two_hidden_layer  # Misalnya, model pertama dianggap sebagai model terbaik

# 6. Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 142ms/step - loss: 0.9026 - accuracy: 0.7500
Model 2 Hidden Layer SGD Optimizer Validation Accuracy: 0.75
1/1 [==============================] - 0s 62ms/step - loss: 0.6905 - accuracy: 0.6400
Best Model Test Accuracy: 0.6399999856948853
```

```
update_results('SGD', 2, val_acc, test_acc)
print(results_df)
```

```
  Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0       SGD             1                 0.75           0.64
1       SGD             2                 0.75           0.64
```

Train and evaluate the model with 3 hidden Layer

```
# Melatih model dengan data pelatihan
model_three_hidden_layer.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_three_hidden_layer.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_val))
```

```
8/8 [==============================] - 0s 19ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.7643 - val_accuracy: 0.7000
Epoch 87/100
8/8 [==============================] - 0s 22ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.7655 - val_accuracy: 0.7000
Epoch 88/100
8/8 [==============================] - 0s 18ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.7667 - val_accuracy: 0.7000
Epoch 89/100
8/8 [==============================] - 0s 18ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.7679 - val_accuracy: 0.7000
Epoch 90/100
8/8 [==============================] - 0s 19ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.7690 - val_accuracy: 0.7000
Epoch 91/100
8/8 [==============================] - 0s 19ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.7703 - val_accuracy: 0.7000
Epoch 92/100
8/8 [==============================] - 0s 23ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 0.7714 - val_accuracy: 0.7000
Epoch 93/100
8/8 [==============================] - 0s 19ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 0.7726 - val_accuracy: 0.7000
Epoch 94/100
8/8 [==============================] - 0s 20ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 0.7737 - val_accuracy: 0.7000
Epoch 95/100
8/8 [==============================] - 0s 19ms/step - loss: 9.9855e-04 - accuracy: 1.0000 - val_loss: 0.7748 - val_accuracy: 0.70
Epoch 96/100
8/8 [==============================] - 0s 18ms/step - loss: 9.8534e-04 - accuracy: 1.0000 - val_loss: 0.7758 - val_accuracy: 0.70
Epoch 97/100
8/8 [==============================] - 0s 18ms/step - loss: 9.7267e-04 - accuracy: 1.0000 - val_loss: 0.7769 - val_accuracy: 0.70
Epoch 98/100
8/8 [==============================] - 0s 18ms/step - loss: 9.6005e-04 - accuracy: 1.0000 - val_loss: 0.7780 - val_accuracy: 0.70
Epoch 99/100
8/8 [==============================] - 0s 17ms/step - loss: 9.4755e-04 - accuracy: 1.0000 - val_loss: 0.7791 - val_accuracy: 0.70
Epoch 100/100
8/8 [==============================] - 0s 22ms/step - loss: 9.3555e-04 - accuracy: 1.0000 - val_loss: 0.7800 - val_accuracy: 0.70
```

```python
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_three_hidden_layer.evaluate(X_val_scaled, y_val)
print("Model 3 Hidden Layer SGD Optimizer Validation Accuracy:", val_acc)

# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_three_hidden_layer  # Misalnya, model pertama dianggap sebagai model terbaik

# Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 170ms/step - loss: 0.7800 - accuracy: 0.7000
Model 3 Hidden Layer SGD Optimizer Validation Accuracy: 0.699999988079071
1/1 [==============================] - 0s 31ms/step - loss: 0.7084 - accuracy: 0.6800
Best Model Test Accuracy: 0.6800000071525574
```

```python
update_results('SGD', 3, val_acc, test_acc)
print(results_df)
```

```
  Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0       SGD             1                 0.75           0.64
1       SGD             2                 0.75           0.64
2       SGD             3                 0.70           0.68
```

Train and evaluate the model with 4 hidden Layer

```python
# Melatih model dengan data pelatihan
model_four_hidden_layer.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_four_hidden_layer.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_val))
```

```
8/8 [==============================] - 0s 11ms/step - loss: 9.0420e-04 - accuracy: 1.0000 - val_loss: 1.1804 - val_accuracy: 0.70
Epoch 85/100
8/8 [==============================] - 0s 12ms/step - loss: 8.8982e-04 - accuracy: 1.0000 - val_loss: 1.1825 - val_accuracy: 0.70
Epoch 86/100
8/8 [==============================] - 0s 11ms/step - loss: 8.7556e-04 - accuracy: 1.0000 - val_loss: 1.1848 - val_accuracy: 0.70
Epoch 87/100
8/8 [==============================] - 0s 13ms/step - loss: 8.6183e-04 - accuracy: 1.0000 - val_loss: 1.1865 - val_accuracy: 0.70
Epoch 88/100
8/8 [==============================] - 0s 16ms/step - loss: 8.4806e-04 - accuracy: 1.0000 - val_loss: 1.1888 - val_accuracy: 0.70
Epoch 89/100
8/8 [==============================] - 0s 11ms/step - loss: 8.3518e-04 - accuracy: 1.0000 - val_loss: 1.1907 - val_accuracy: 0.70
Epoch 90/100
8/8 [==============================] - 0s 14ms/step - loss: 8.2222e-04 - accuracy: 1.0000 - val_loss: 1.1929 - val_accuracy: 0.70
Epoch 91/100
8/8 [==============================] - 0s 11ms/step - loss: 8.0997e-04 - accuracy: 1.0000 - val_loss: 1.1949 - val_accuracy: 0.70
Epoch 92/100
8/8 [==============================] - 0s 11ms/step - loss: 7.9800e-04 - accuracy: 1.0000 - val_loss: 1.1969 - val_accuracy: 0.70
Epoch 93/100
8/8 [==============================] - 0s 11ms/step - loss: 7.8619e-04 - accuracy: 1.0000 - val_loss: 1.1986 - val_accuracy: 0.70
Epoch 94/100
8/8 [==============================] - 0s 11ms/step - loss: 7.7479e-04 - accuracy: 1.0000 - val_loss: 1.2005 - val_accuracy: 0.70
Epoch 95/100
8/8 [==============================] - 0s 12ms/step - loss: 7.6337e-04 - accuracy: 1.0000 - val_loss: 1.2025 - val_accuracy: 0.70
Epoch 96/100
8/8 [==============================] - 0s 11ms/step - loss: 7.5243e-04 - accuracy: 1.0000 - val_loss: 1.2045 - val_accuracy: 0.70
Epoch 97/100
8/8 [==============================] - 0s 13ms/step - loss: 7.4200e-04 - accuracy: 1.0000 - val_loss: 1.2063 - val_accuracy: 0.70
Epoch 98/100
8/8 [==============================] - 0s 11ms/step - loss: 7.3169e-04 - accuracy: 1.0000 - val_loss: 1.2081 - val_accuracy: 0.70
Epoch 99/100
8/8 [==============================] - 0s 15ms/step - loss: 7.2161e-04 - accuracy: 1.0000 - val_loss: 1.2100 - val_accuracy: 0.70
Epoch 100/100
8/8 [==============================] - 0s 13ms/step - loss: 7.1172e-04 - accuracy: 1.0000 - val_loss: 1.2117 - val_accuracy: 0.70
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_four_hidden_layer.evaluate(X_val_scaled, y_val)
print("Model 3 Hidden Layer SGD Optimizer Validation Accuracy:", val_acc)

# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_four_hidden_layer  # Misalnya, model pertama dianggap sebagai model terbaik

# Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 150ms/step - loss: 1.2117 - accuracy: 0.7000
Model 3 Hidden Layer SGD Optimizer Validation Accuracy: 0.699999988079071
1/1 [==============================] - 0s 29ms/step - loss: 1.1738 - accuracy: 0.6400
Best Model Test Accuracy: 0.6399999856948853
```

```
update_results('SGD', 4, val_acc, test_acc)
print(results_df)
```

```
  Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0       SGD             1                 0.75           0.64
1       SGD             2                 0.75           0.64
2       SGD             3                 0.70           0.68
3       SGD             4                 0.70           0.64
```

2. Mengganti Optimizer

2.A Adam

Train and evaluate the model with 1 hidden Layer with Adam Optimizer

```
# Melatih model dengan data pelatihan
# optimizer_adam = tf.keras.optimizers.Adam(learning_rate=0.001)  # Define the optimizer with learning rate
model_one_hidden_layer.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_one_hidden_layer.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_val))
```

```
8/8 [==============================] - 2s 285ms/step - loss: 9.5065e-08 - accuracy: 1.0000 - val_loss: 19.6395 - val_accuracy: 0.
Epoch 78/100
8/8 [==============================] - 2s 272ms/step - loss: 9.3556e-08 - accuracy: 1.0000 - val_loss: 19.6393 - val_accuracy: 0.
Epoch 79/100
8/8 [==============================] - 2s 213ms/step - loss: 9.3556e-08 - accuracy: 1.0000 - val_loss: 19.6392 - val_accuracy: 0.
Epoch 80/100
8/8 [==============================] - 2s 213ms/step - loss: 9.3556e-08 - accuracy: 1.0000 - val_loss: 19.6390 - val_accuracy: 0.
Epoch 81/100
8/8 [==============================] - 2s 220ms/step - loss: 9.3556e-08 - accuracy: 1.0000 - val_loss: 19.6388 - val_accuracy: 0.
Epoch 82/100
8/8 [==============================] - 2s 215ms/step - loss: 9.2047e-08 - accuracy: 1.0000 - val_loss: 19.6386 - val_accuracy: 0.
Epoch 83/100
8/8 [==============================] - 2s 220ms/step - loss: 9.2047e-08 - accuracy: 1.0000 - val_loss: 19.6384 - val_accuracy: 0.
Epoch 84/100
8/8 [==============================] - 3s 332ms/step - loss: 9.2047e-08 - accuracy: 1.0000 - val_loss: 19.6383 - val_accuracy: 0.
Epoch 85/100
8/8 [==============================] - 2s 223ms/step - loss: 9.0539e-08 - accuracy: 1.0000 - val_loss: 19.6380 - val_accuracy: 0.
Epoch 86/100
8/8 [==============================] - 2s 222ms/step - loss: 9.0539e-08 - accuracy: 1.0000 - val_loss: 19.6379 - val_accuracy: 0.
Epoch 87/100
8/8 [==============================] - 2s 215ms/step - loss: 9.0539e-08 - accuracy: 1.0000 - val_loss: 19.6377 - val_accuracy: 0.
Epoch 88/100
8/8 [==============================] - 2s 214ms/step - loss: 8.9030e-08 - accuracy: 1.0000 - val_loss: 19.6375 - val_accuracy: 0.
Epoch 89/100
8/8 [==============================] - 2s 218ms/step - loss: 8.9030e-08 - accuracy: 1.0000 - val_loss: 19.6373 - val_accuracy: 0.
Epoch 90/100
8/8 [==============================] - 2s 223ms/step - loss: 8.9030e-08 - accuracy: 1.0000 - val_loss: 19.6372 - val_accuracy: 0.
Epoch 91/100
8/8 [==============================] - 3s 331ms/step - loss: 8.9030e-08 - accuracy: 1.0000 - val_loss: 19.6369 - val_accuracy: 0.
Epoch 92/100
8/8 [==============================] - 2s 213ms/step - loss: 8.7521e-08 - accuracy: 1.0000 - val_loss: 19.6367 - val_accuracy: 0.
Epoch 93/100
8/8 [==============================] - 2s 218ms/step - loss: 8.7521e-08 - accuracy: 1.0000 - val_loss: 19.6366 - val_accuracy: 0.
Epoch 94/100
8/8 [==============================] - 2s 213ms/step - loss: 8.7521e-08 - accuracy: 1.0000 - val_loss: 19.6364 - val_accuracy: 0.
Epoch 95/100
8/8 [==============================] - 2s 217ms/step - loss: 8.6012e-08 - accuracy: 1.0000 - val_loss: 19.6362 - val_accuracy: 0.
Epoch 96/100
8/8 [==============================] - 2s 214ms/step - loss: 8.6012e-08 - accuracy: 1.0000 - val_loss: 19.6360 - val_accuracy: 0.
Epoch 97/100
8/8 [==============================] - 2s 245ms/step - loss: 8.4503e-08 - accuracy: 1.0000 - val_loss: 19.6358 - val_accuracy: 0.
Epoch 98/100
8/8 [==============================] - 3s 309ms/step - loss: 8.4503e-08 - accuracy: 1.0000 - val_loss: 19.6356 - val_accuracy: 0.
Epoch 99/100
8/8 [==============================] - 2s 220ms/step - loss: 8.2994e-08 - accuracy: 1.0000 - val_loss: 19.6354 - val_accuracy: 0.
Epoch 100/100
8/8 [==============================] - 2s 219ms/step - loss: 8.2994e-08 - accuracy: 1.0000 - val_loss: 19.6353 - val_accuracy: 0.
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_one_hidden_layer.evaluate(X_val_scaled, y_val)
print("Model 1 Hidden Layer Adam Optimizer Validation Accuracy:", val_acc)

# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_one_hidden_layer  # Misalnya, model pertama dianggap sebagai model terbaik

# Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 144ms/step - loss: 19.6353 - accuracy: 0.7000
Model 1 Hidden Layer Adam Optimizer Validation Accuracy: 0.699999988079071
1/1 [==============================] - 0s 49ms/step - loss: 19.2272 - accuracy: 0.6000
Best Model Test Accuracy: 0.6000000238418579
```

```
update_results('ADAM', 1, val_acc, test_acc)
print(results_df)
```

```
  Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0       SGD             1                 0.75           0.64
1       SGD             2                 0.75           0.64
2       SGD             3                 0.70           0.68
3       SGD             4                 0.70           0.64
4      ADAM             1                 0.70           0.60
```

Train and evaluate the model with 2 hidden Layer with Adam Optimizer

```
# Melatih model dengan data pelatihan
model_two_hidden_layer.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_two_hidden_layer.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_val))
```

```
8/8 [==============================] - 1s 72ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 75/100
8/8 [==============================] - 1s 78ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 76/100
8/8 [==============================] - 1s 77ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 77/100
8/8 [==============================] - 1s 81ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 78/100
8/8 [==============================] - 1s 79ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 79/100
8/8 [==============================] - 1s 87ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 80/100
8/8 [==============================] - 1s 98ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 81/100
8/8 [==============================] - 1s 123ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.
Epoch 82/100
8/8 [==============================] - 1s 116ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.
Epoch 83/100
8/8 [==============================] - 1s 98ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 84/100
8/8 [==============================] - 1s 71ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 85/100
8/8 [==============================] - 1s 77ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 86/100
8/8 [==============================] - 1s 81ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 87/100
8/8 [==============================] - 1s 73ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 88/100
8/8 [==============================] - 1s 83ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 89/100
8/8 [==============================] - 1s 76ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 90/100
8/8 [==============================] - 1s 84ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 91/100
8/8 [==============================] - 1s 80ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 92/100
8/8 [==============================] - 1s 86ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 93/100
8/8 [==============================] - 1s 82ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 94/100
8/8 [==============================] - 1s 84ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 95/100
8/8 [==============================] - 1s 79ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3902 - val_accuracy: 0.7
Epoch 96/100
8/8 [==============================] - 1s 83ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3903 - val_accuracy: 0.7
Epoch 97/100
8/8 [==============================] - 1s 82ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3903 - val_accuracy: 0.7
Epoch 98/100
8/8 [==============================] - 1s 79ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3903 - val_accuracy: 0.7
Epoch 99/100
8/8 [==============================] - 1s 131ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3903 - val_accuracy: 0.
Epoch 100/100
8/8 [==============================] - 1s 113ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3903 - val_accuracy: 0.
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_two_hidden_layer.evaluate(X_val_scaled, y_val)
print("Model 2 Hidden Layer Adam Optimizer Validation Accuracy:", val_acc)

# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_two_hidden_layer  # Misalnya, model pertama dianggap sebagai model terbaik

# Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 145ms/step - loss: 11.3903 - accuracy: 0.7500
Model 2 Hidden Layer Adam Optimizer Validation Accuracy: 0.75
1/1 [==============================] - 0s 37ms/step - loss: 19.7588 - accuracy: 0.6400
Best Model Test Accuracy: 0.6399999856948853
```

```
update_results('ADAM', 2, val_acc, test_acc)
print(results_df)
```

```
  Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0       SGD             1                 0.75           0.64
1       SGD             2                 0.75           0.64
2       SGD             3                 0.70           0.68
3       SGD             4                 0.70           0.64
4      ADAM             1                 0.70           0.60
5      ADAM             2                 0.75           0.64
```

Train and evaluate the model with 3 hidden Layer with Adam Optimizer

```
# Melatih model dengan data pelatihan
model_three_hidden_layer.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_three_hidden_layer.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_val))
```

```
8/8 [==============================] - 0s 38ms/step - loss: 1.9586e-06 - accuracy: 1.0000 - val_loss: 5.0317 - val_accuracy: 0.65
Epoch 73/100
8/8 [==============================] - 0s 40ms/step - loss: 1.9194e-06 - accuracy: 1.0000 - val_loss: 5.0313 - val_accuracy: 0.65
Epoch 74/100
8/8 [==============================] - 0s 39ms/step - loss: 1.8787e-06 - accuracy: 1.0000 - val_loss: 5.0308 - val_accuracy: 0.65
Epoch 75/100
8/8 [==============================] - 0s 38ms/step - loss: 1.8349e-06 - accuracy: 1.0000 - val_loss: 5.0305 - val_accuracy: 0.65
Epoch 76/100
8/8 [==============================] - 0s 39ms/step - loss: 1.8032e-06 - accuracy: 1.0000 - val_loss: 5.0301 - val_accuracy: 0.65
Epoch 77/100
8/8 [==============================] - 0s 38ms/step - loss: 1.7685e-06 - accuracy: 1.0000 - val_loss: 5.0295 - val_accuracy: 0.65
Epoch 78/100
8/8 [==============================] - 0s 40ms/step - loss: 1.7308e-06 - accuracy: 1.0000 - val_loss: 5.0288 - val_accuracy: 0.65
Epoch 79/100
8/8 [==============================] - 0s 40ms/step - loss: 1.6900e-06 - accuracy: 1.0000 - val_loss: 5.0283 - val_accuracy: 0.65
Epoch 80/100
8/8 [==============================] - 0s 39ms/step - loss: 1.6583e-06 - accuracy: 1.0000 - val_loss: 5.0276 - val_accuracy: 0.65
Epoch 81/100
8/8 [==============================] - 0s 43ms/step - loss: 1.6297e-06 - accuracy: 1.0000 - val_loss: 5.0271 - val_accuracy: 0.65
Epoch 82/100
8/8 [==============================] - 0s 41ms/step - loss: 1.5965e-06 - accuracy: 1.0000 - val_loss: 5.0265 - val_accuracy: 0.65
Epoch 83/100
8/8 [==============================] - 0s 40ms/step - loss: 1.5708e-06 - accuracy: 1.0000 - val_loss: 5.0259 - val_accuracy: 0.65
Epoch 84/100
8/8 [==============================] - 0s 39ms/step - loss: 1.5437e-06 - accuracy: 1.0000 - val_loss: 5.0256 - val_accuracy: 0.65
Epoch 85/100
8/8 [==============================] - 0s 40ms/step - loss: 1.5105e-06 - accuracy: 1.0000 - val_loss: 5.0253 - val_accuracy: 0.65
Epoch 86/100
8/8 [==============================] - 0s 38ms/step - loss: 1.4878e-06 - accuracy: 1.0000 - val_loss: 5.0249 - val_accuracy: 0.65
Epoch 87/100
8/8 [==============================] - 0s 40ms/step - loss: 1.4622e-06 - accuracy: 1.0000 - val_loss: 5.0247 - val_accuracy: 0.65
Epoch 88/100
8/8 [==============================] - 0s 40ms/step - loss: 1.4350e-06 - accuracy: 1.0000 - val_loss: 5.0247 - val_accuracy: 0.65
Epoch 89/100
8/8 [==============================] - 0s 40ms/step - loss: 1.4139e-06 - accuracy: 1.0000 - val_loss: 5.0246 - val_accuracy: 0.65
Epoch 90/100
8/8 [==============================] - 0s 40ms/step - loss: 1.3837e-06 - accuracy: 1.0000 - val_loss: 5.0245 - val_accuracy: 0.65
Epoch 91/100
8/8 [==============================] - 0s 40ms/step - loss: 1.3656e-06 - accuracy: 1.0000 - val_loss: 5.0244 - val_accuracy: 0.65
Epoch 92/100
8/8 [==============================] - 0s 38ms/step - loss: 1.3505e-06 - accuracy: 1.0000 - val_loss: 5.0241 - val_accuracy: 0.65
Epoch 93/100
8/8 [==============================] - 0s 39ms/step - loss: 1.3234e-06 - accuracy: 1.0000 - val_loss: 5.0239 - val_accuracy: 0.65
Epoch 94/100
8/8 [==============================] - 0s 43ms/step - loss: 1.3037e-06 - accuracy: 1.0000 - val_loss: 5.0235 - val_accuracy: 0.65
Epoch 95/100
8/8 [==============================] - 0s 39ms/step - loss: 1.2811e-06 - accuracy: 1.0000 - val_loss: 5.0232 - val_accuracy: 0.65
Epoch 96/100
8/8 [==============================] - 0s 47ms/step - loss: 1.2675e-06 - accuracy: 1.0000 - val_loss: 5.0227 - val_accuracy: 0.65
Epoch 97/100
8/8 [==============================] - 0s 58ms/step - loss: 1.2404e-06 - accuracy: 1.0000 - val_loss: 5.0225 - val_accuracy: 0.65
Epoch 98/100
8/8 [==============================] - 0s 52ms/step - loss: 1.2238e-06 - accuracy: 1.0000 - val_loss: 5.0222 - val_accuracy: 0.65
Epoch 99/100
8/8 [==============================] - 0s 56ms/step - loss: 1.2072e-06 - accuracy: 1.0000 - val_loss: 5.0221 - val_accuracy: 0.65
Epoch 100/100
8/8 [==============================] - 0s 55ms/step - loss: 1.1921e-06 - accuracy: 1.0000 - val_loss: 5.0217 - val_accuracy: 0.65
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_three_hidden_layer.evaluate(X_val_scaled, y_val)
print("Model 3 Hidden Layer Adam Optimizer Validation Accuracy:", val_acc)

# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_three_hidden_layer  # Misalnya, model pertama dianggap sebagai model terbaik

# Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 148ms/step - loss: 5.0217 - accuracy: 0.6500
Model 3 Hidden Layer Adam Optimizer Validation Accuracy: 0.6499999761581421
1/1 [==============================] - 0s 35ms/step - loss: 6.8095 - accuracy: 0.5600
Best Model Test Accuracy: 0.5600000023841858
```

```
update_results('ADAM', 3, val_acc, test_acc)
print(results_df)
```

```
  Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0       SGD             1                 0.75           0.64
1       SGD             2                 0.75           0.64
2       SGD             3                 0.70           0.68
3       SGD             4                 0.70           0.64
```

```
4      ADAM            1                  0.70          0.60
5      ADAM            2                  0.75          0.64
6      ADAM            3                  0.65          0.56
```

Train and evaluate the model with 4 hidden Layer with Adam Optimizer

```
# Melatih model dengan data pelatihan
model_four_hidden_layer.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_four_hidden_layer.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_val))
```

```
8/8 [==============================] - 0s 20ms/step - loss: 5.6826e-06 - accuracy: 1.0000 - val_loss: 4.4464 - val_accuracy: 0.75
Epoch 73/100
8/8 [==============================] - 0s 21ms/step - loss: 5.4593e-06 - accuracy: 1.0000 - val_loss: 4.4503 - val_accuracy: 0.75
Epoch 74/100
8/8 [==============================] - 0s 19ms/step - loss: 5.2420e-06 - accuracy: 1.0000 - val_loss: 4.4546 - val_accuracy: 0.75
Epoch 75/100
8/8 [==============================] - 0s 21ms/step - loss: 4.9116e-06 - accuracy: 1.0000 - val_loss: 4.4576 - val_accuracy: 0.75
Epoch 76/100
8/8 [==============================] - 0s 19ms/step - loss: 4.7531e-06 - accuracy: 1.0000 - val_loss: 4.4615 - val_accuracy: 0.75
Epoch 77/100
8/8 [==============================] - 0s 20ms/step - loss: 4.5690e-06 - accuracy: 1.0000 - val_loss: 4.4674 - val_accuracy: 0.75
Epoch 78/100
8/8 [==============================] - 0s 21ms/step - loss: 4.4197e-06 - accuracy: 1.0000 - val_loss: 4.4738 - val_accuracy: 0.75
Epoch 79/100
8/8 [==============================] - 0s 22ms/step - loss: 4.2869e-06 - accuracy: 1.0000 - val_loss: 4.4807 - val_accuracy: 0.75
Epoch 80/100
8/8 [==============================] - 0s 19ms/step - loss: 4.1586e-06 - accuracy: 1.0000 - val_loss: 4.4879 - val_accuracy: 0.75
Epoch 81/100
8/8 [==============================] - 0s 19ms/step - loss: 4.0681e-06 - accuracy: 1.0000 - val_loss: 4.4914 - val_accuracy: 0.75
Epoch 82/100
8/8 [==============================] - 0s 20ms/step - loss: 3.9413e-06 - accuracy: 1.0000 - val_loss: 4.4927 - val_accuracy: 0.75
Epoch 83/100
8/8 [==============================] - 0s 24ms/step - loss: 3.8478e-06 - accuracy: 1.0000 - val_loss: 4.4957 - val_accuracy: 0.75
Epoch 84/100
8/8 [==============================] - 0s 29ms/step - loss: 3.7452e-06 - accuracy: 1.0000 - val_loss: 4.4988 - val_accuracy: 0.75
Epoch 85/100
8/8 [==============================] - 0s 29ms/step - loss: 3.6592e-06 - accuracy: 1.0000 - val_loss: 4.5044 - val_accuracy: 0.75
Epoch 86/100
8/8 [==============================] - 0s 28ms/step - loss: 3.5445e-06 - accuracy: 1.0000 - val_loss: 4.5134 - val_accuracy: 0.75
Epoch 87/100
8/8 [==============================] - 0s 26ms/step - loss: 3.3996e-06 - accuracy: 1.0000 - val_loss: 4.5194 - val_accuracy: 0.75
Epoch 88/100
8/8 [==============================] - 0s 26ms/step - loss: 3.2472e-06 - accuracy: 1.0000 - val_loss: 4.5274 - val_accuracy: 0.75
Epoch 89/100
8/8 [==============================] - 0s 25ms/step - loss: 3.1461e-06 - accuracy: 1.0000 - val_loss: 4.5369 - val_accuracy: 0.75
Epoch 90/100
8/8 [==============================] - 0s 27ms/step - loss: 3.0858e-06 - accuracy: 1.0000 - val_loss: 4.5436 - val_accuracy: 0.75
Epoch 91/100
8/8 [==============================] - 0s 30ms/step - loss: 2.9530e-06 - accuracy: 1.0000 - val_loss: 4.5456 - val_accuracy: 0.75
Epoch 92/100
8/8 [==============================] - 0s 29ms/step - loss: 2.8941e-06 - accuracy: 1.0000 - val_loss: 4.5493 - val_accuracy: 0.75
Epoch 93/100
8/8 [==============================] - 0s 28ms/step - loss: 2.8353e-06 - accuracy: 1.0000 - val_loss: 4.5546 - val_accuracy: 0.75
Epoch 94/100
8/8 [==============================] - 0s 30ms/step - loss: 2.7689e-06 - accuracy: 1.0000 - val_loss: 4.5606 - val_accuracy: 0.75
Epoch 95/100
8/8 [==============================] - 0s 29ms/step - loss: 2.6935e-06 - accuracy: 1.0000 - val_loss: 4.5664 - val_accuracy: 0.75
Epoch 96/100
8/8 [==============================] - 0s 19ms/step - loss: 2.6346e-06 - accuracy: 1.0000 - val_loss: 4.5718 - val_accuracy: 0.75
Epoch 97/100
8/8 [==============================] - 0s 19ms/step - loss: 2.5743e-06 - accuracy: 1.0000 - val_loss: 4.5772 - val_accuracy: 0.75
Epoch 98/100
8/8 [==============================] - 0s 19ms/step - loss: 2.5260e-06 - accuracy: 1.0000 - val_loss: 4.5819 - val_accuracy: 0.75
Epoch 99/100
8/8 [==============================] - 0s 19ms/step - loss: 2.4596e-06 - accuracy: 1.0000 - val_loss: 4.5874 - val_accuracy: 0.75
Epoch 100/100
8/8 [==============================] - 0s 21ms/step - loss: 2.4203e-06 - accuracy: 1.0000 - val_loss: 4.5917 - val_accuracy: 0.75
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_four_hidden_layer.evaluate(X_val_scaled, y_val)
print("Model 4 Hidden Layer Adam Optimizer Validation Accuracy:", val_acc)

# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_four_hidden_layer  # Misalnya, model pertama dianggap sebagai model terbaik

# Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 159ms/step - loss: 4.5917 - accuracy: 0.7500
Model 4 Hidden Layer Adam Optimizer Validation Accuracy: 0.75
1/1 [==============================] - 0s 31ms/step - loss: 5.2001 - accuracy: 0.4800
Best Model Test Accuracy: 0.47999998927116394
```

```
update_results('ADAM', 4, val_acc, test_acc)
print(results_df)
```

```
  Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0       SGD             1                 0.75           0.64
1       SGD             2                 0.75           0.64
2       SGD             3                 0.70           0.68
3       SGD             4                 0.70           0.64
4      ADAM             1                 0.70           0.60
5      ADAM             2                 0.75           0.64
6      ADAM             3                 0.65           0.56
7      ADAM             4                 0.75           0.48
```

2.B AdaGrad

Train and evaluate the model with 1 hidden Layer with AdaGrad Optimizer

```
# Melatih model dengan data pelatihan
optimizer_adagrad = tf.keras.optimizers.Adagrad()
model_one_hidden_layer.compile(optimizer=optimizer_adagrad, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_one_hidden_layer.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_val))
```

```
8/8 [==============================] - 2s 188ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6346 - val_accuracy: 0.
Epoch 73/100
8/8 [==============================] - 1s 177ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6346 - val_accuracy: 0.
Epoch 74/100
8/8 [==============================] - 1s 174ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6346 - val_accuracy: 0.
Epoch 75/100
8/8 [==============================] - 1s 171ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6346 - val_accuracy: 0.
Epoch 76/100
8/8 [==============================] - 1s 177ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6346 - val_accuracy: 0.
Epoch 77/100
8/8 [==============================] - 1s 174ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6346 - val_accuracy: 0.
Epoch 78/100
8/8 [==============================] - 1s 177ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6346 - val_accuracy: 0.
Epoch 79/100
8/8 [==============================] - 2s 239ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6346 - val_accuracy: 0.
Epoch 80/100
8/8 [==============================] - 2s 230ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6346 - val_accuracy: 0.
Epoch 81/100
8/8 [==============================] - 1s 173ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6345 - val_accuracy: 0.
Epoch 82/100
8/8 [==============================] - 1s 173ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6345 - val_accuracy: 0.
Epoch 83/100
8/8 [==============================] - 1s 179ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6345 - val_accuracy: 0.
Epoch 84/100
8/8 [==============================] - 1s 174ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6345 - val_accuracy: 0.
Epoch 85/100
8/8 [==============================] - 1s 173ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6345 - val_accuracy: 0.
Epoch 86/100
8/8 [==============================] - 1s 188ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6345 - val_accuracy: 0.
Epoch 87/100
8/8 [==============================] - 2s 226ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6345 - val_accuracy: 0.
Epoch 88/100
8/8 [==============================] - 2s 272ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6345 - val_accuracy: 0.
Epoch 89/100
8/8 [==============================] - 1s 181ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6345 - val_accuracy: 0.
Epoch 90/100
8/8 [==============================] - 1s 184ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6345 - val_accuracy: 0.
Epoch 91/100
8/8 [==============================] - 2s 196ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6345 - val_accuracy: 0.
Epoch 92/100
8/8 [==============================] - 1s 185ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6345 - val_accuracy: 0.
Epoch 93/100
8/8 [==============================] - 1s 179ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6344 - val_accuracy: 0.
Epoch 94/100
8/8 [==============================] - 2s 189ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6344 - val_accuracy: 0.
Epoch 95/100
8/8 [==============================] - 2s 236ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6344 - val_accuracy: 0.
Epoch 96/100
8/8 [==============================] - 2s 271ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6344 - val_accuracy: 0.
Epoch 97/100
8/8 [==============================] - 1s 186ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6344 - val_accuracy: 0.
Epoch 98/100
8/8 [==============================] - 1s 185ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6344 - val_accuracy: 0.
Epoch 99/100
8/8 [==============================] - 1s 186ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6344 - val_accuracy: 0.
Epoch 100/100
8/8 [==============================] - 1s 183ms/step - loss: 8.1485e-08 - accuracy: 1.0000 - val_loss: 19.6344 - val_accuracy: 0.
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_one_hidden_layer.evaluate(X_val_scaled, y_val)
print("Model 1 Hidden Layer AdaGrad Optimizer Validation Accuracy:", val_acc)

# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_one_hidden_layer  # Misalnya, model pertama dianggap sebagai model terbaik

# Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 174ms/step - loss: 19.6344 - accuracy: 0.7000
Model 1 Hidden Layer AdaGrad Optimizer Validation Accuracy: 0.699999988079071
1/1 [==============================] - 0s 48ms/step - loss: 19.2263 - accuracy: 0.6000
Best Model Test Accuracy: 0.6000000238418579
```

```
update_results('AdaGrad', 1, val_acc, test_acc)
print(results_df)
```

```
  Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0       SGD             1                 0.75           0.64
1       SGD             2                 0.75           0.64
2       SGD             3                 0.70           0.68
3       SGD             4                 0.70           0.64
4      ADAM             1                 0.70           0.60
5      ADAM             2                 0.75           0.64
6      ADAM             3                 0.65           0.56
7      ADAM             4                 0.75           0.48
8   AdaGrad             1                 0.70           0.60
```

Train and evaluate the model with 2 hidden Layer with AdaGrad Optimizer

```
# Melatih model dengan data pelatihan
optimizer_adagrad = tf.keras.optimizers.Adagrad()
model_two_hidden_layer.compile(optimizer=optimizer_adagrad, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_two_hidden_layer.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_val))
```

```
8/8 [==============================] - 0s 56ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3903 - val_accuracy: 0.7
Epoch 97/100
8/8 [==============================] - 0s 51ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3903 - val_accuracy: 0.7
Epoch 98/100
8/8 [==============================] - 1s 80ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3903 - val_accuracy: 0.7
Epoch 99/100
8/8 [==============================] - 1s 77ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3903 - val_accuracy: 0.7
Epoch 100/100
8/8 [==============================] - 1s 84ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 11.3903 - val_accuracy: 0.7
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_two_hidden_layer.evaluate(X_val_scaled, y_val)
print("Model 2 Hidden Layer AdaGrad Optimizer Validation Accuracy:", val_acc)

# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_two_hidden_layer  # Misalnya, model pertama dianggap sebagai model terbaik

# Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 157ms/step - loss: 11.3903 - accuracy: 0.7500
Model 2 Hidden Layer AdaGrad Optimizer Validation Accuracy: 0.75
1/1 [==============================] - 0s 37ms/step - loss: 19.7588 - accuracy: 0.6400
Best Model Test Accuracy: 0.6399999856948853
```

```
update_results('AdaGrad', 2, val_acc, test_acc)
print(results_df)
```

```
  Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0       SGD             1                 0.75           0.64
1       SGD             2                 0.75           0.64
2       SGD             3                 0.70           0.68
3       SGD             4                 0.70           0.64
4      ADAM             1                 0.70           0.60
5      ADAM             2                 0.75           0.64
6      ADAM             3                 0.65           0.56
7      ADAM             4                 0.75           0.48
8   AdaGrad             1                 0.70           0.60
9   AdaGrad             2                 0.75           0.64
```

Train and evaluate the model with 3 hidden Layer with AdaGrad Optimizer

```
# Melatih model dengan data pelatihan
optimizer_adagrad = tf.keras.optimizers.Adagrad()
model_three_hidden_layer.compile(optimizer=optimizer_adagrad, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_three_hidden_layer.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_val))
```

```
8/8 [==============================] - 0s 30ms/step - loss: 1.1679e-06 - accuracy: 1.0000 - val_loss: 5.0218 - val_accuracy: 0.65
Epoch 91/100
8/8 [==============================] - 0s 28ms/step - loss: 1.1664e-06 - accuracy: 1.0000 - val_loss: 5.0218 - val_accuracy: 0.65
Epoch 92/100
8/8 [==============================] - 0s 30ms/step - loss: 1.1664e-06 - accuracy: 1.0000 - val_loss: 5.0218 - val_accuracy: 0.65
Epoch 93/100
8/8 [==============================] - 0s 29ms/step - loss: 1.1664e-06 - accuracy: 1.0000 - val_loss: 5.0218 - val_accuracy: 0.65
Epoch 94/100
8/8 [==============================] - 0s 31ms/step - loss: 1.1664e-06 - accuracy: 1.0000 - val_loss: 5.0218 - val_accuracy: 0.65
Epoch 95/100
8/8 [==============================] - 0s 29ms/step - loss: 1.1664e-06 - accuracy: 1.0000 - val_loss: 5.0218 - val_accuracy: 0.65
Epoch 96/100
8/8 [==============================] - 0s 28ms/step - loss: 1.1649e-06 - accuracy: 1.0000 - val_loss: 5.0218 - val_accuracy: 0.65
Epoch 97/100
8/8 [==============================] - 0s 29ms/step - loss: 1.1649e-06 - accuracy: 1.0000 - val_loss: 5.0218 - val_accuracy: 0.65
Epoch 98/100
8/8 [==============================] - 0s 31ms/step - loss: 1.1649e-06 - accuracy: 1.0000 - val_loss: 5.0218 - val_accuracy: 0.65
Epoch 99/100
8/8 [==============================] - 0s 29ms/step - loss: 1.1649e-06 - accuracy: 1.0000 - val_loss: 5.0218 - val_accuracy: 0.65
Epoch 100/100
8/8 [==============================] - 0s 28ms/step - loss: 1.1649e-06 - accuracy: 1.0000 - val_loss: 5.0218 - val_accuracy: 0.65
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_three_hidden_layer.evaluate(X_val_scaled, y_val)
print("Model 3 Hidden Layer AdaGrad Optimizer Validation Accuracy:", val_acc)

# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_three_hidden_layer  # Misalnya, model pertama dianggap sebagai model terbaik

# Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 153ms/step - loss: 5.0218 - accuracy: 0.6500
Model 3 Hidden Layer AdaGrad Optimizer Validation Accuracy: 0.6499999761581421
1/1 [==============================] - 0s 37ms/step - loss: 6.8103 - accuracy: 0.5600
Best Model Test Accuracy: 0.5600000023841858
```

```
update_results('AdaGrad', 3, val_acc, test_acc)
print(results_df)
```

```
   Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0        SGD             1                 0.75           0.64
1        SGD             2                 0.75           0.64
2        SGD             3                 0.70           0.68
3        SGD             4                 0.70           0.64
4       ADAM             1                 0.70           0.60
5       ADAM             2                 0.75           0.64
6       ADAM             3                 0.65           0.56
7       ADAM             4                 0.75           0.48
8    AdaGrad             1                 0.70           0.60
9    AdaGrad             2                 0.75           0.64
10   AdaGrad             3                 0.65           0.56
```

Train and evaluate the model with 4 hidden Layer with AdaGrad Optimizer

```
# Melatih model dengan data pelatihan
optimizer_adagrad = tf.keras.optimizers.Adagrad()
model_four_hidden_layer.compile(optimizer=optimizer_adagrad, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_four_hidden_layer.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_val))
```

```
Epoch 84/100
8/8 [==============================] - 0s 20ms/step - loss: 2.3208e-06 - accuracy: 1.0000 - val_loss: 4.5927 - val_accuracy: 0.75
Epoch 85/100
8/8 [==============================] - 0s 16ms/step - loss: 2.3208e-06 - accuracy: 1.0000 - val_loss: 4.5928 - val_accuracy: 0.75
Epoch 86/100
8/8 [==============================] - 0s 18ms/step - loss: 2.3192e-06 - accuracy: 1.0000 - val_loss: 4.5928 - val_accuracy: 0.75
Epoch 87/100
8/8 [==============================] - 0s 17ms/step - loss: 2.3177e-06 - accuracy: 1.0000 - val_loss: 4.5928 - val_accuracy: 0.75
Epoch 88/100
8/8 [==============================] - 0s 18ms/step - loss: 2.3177e-06 - accuracy: 1.0000 - val_loss: 4.5928 - val_accuracy: 0.75
Epoch 89/100
8/8 [==============================] - 0s 16ms/step - loss: 2.3177e-06 - accuracy: 1.0000 - val_loss: 4.5928 - val_accuracy: 0.75
Epoch 90/100
8/8 [==============================] - 0s 15ms/step - loss: 2.3162e-06 - accuracy: 1.0000 - val_loss: 4.5928 - val_accuracy: 0.75
Epoch 91/100
8/8 [==============================] - 0s 18ms/step - loss: 2.3162e-06 - accuracy: 1.0000 - val_loss: 4.5928 - val_accuracy: 0.75
Epoch 92/100
8/8 [==============================] - 0s 16ms/step - loss: 2.3147e-06 - accuracy: 1.0000 - val_loss: 4.5928 - val_accuracy: 0.75
Epoch 93/100
8/8 [==============================] - 0s 15ms/step - loss: 2.3132e-06 - accuracy: 1.0000 - val_loss: 4.5929 - val_accuracy: 0.75
Epoch 94/100
8/8 [==============================] - 0s 16ms/step - loss: 2.3132e-06 - accuracy: 1.0000 - val_loss: 4.5929 - val_accuracy: 0.75
Epoch 95/100
8/8 [==============================] - 0s 16ms/step - loss: 2.3132e-06 - accuracy: 1.0000 - val_loss: 4.5929 - val_accuracy: 0.75
Epoch 96/100
8/8 [==============================] - 0s 22ms/step - loss: 2.3132e-06 - accuracy: 1.0000 - val_loss: 4.5929 - val_accuracy: 0.75
Epoch 97/100
8/8 [==============================] - 0s 24ms/step - loss: 2.3117e-06 - accuracy: 1.0000 - val_loss: 4.5929 - val_accuracy: 0.75
Epoch 98/100
8/8 [==============================] - 0s 24ms/step - loss: 2.3102e-06 - accuracy: 1.0000 - val_loss: 4.5929 - val_accuracy: 0.75
Epoch 99/100
8/8 [==============================] - 0s 25ms/step - loss: 2.3102e-06 - accuracy: 1.0000 - val_loss: 4.5929 - val_accuracy: 0.75
Epoch 100/100
8/8 [==============================] - 0s 22ms/step - loss: 2.3087e-06 - accuracy: 1.0000 - val_loss: 4.5930 - val_accuracy: 0.75
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_four_hidden_layer.evaluate(X_val_scaled, y_val)
print("Model 4 Hidden Layer AdaGrad Optimizer Validation Accuracy:", val_acc)
# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_four_hidden_layer  # Misalnya, model pertama dianggap sebagai model terbaik
# Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 226ms/step - loss: 4.5930 - accuracy: 0.7500
Model 4 Hidden Layer AdaGrad Optimizer Validation Accuracy: 0.75
1/1 [==============================] - 0s 39ms/step - loss: 5.2078 - accuracy: 0.4800
Best Model Test Accuracy: 0.47999998927116394
```

```
update_results('AdaGrad', 4, val_acc, test_acc)
print(results_df)
```

```
   Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0        SGD             1                 0.75           0.64
1        SGD             2                 0.75           0.64
2        SGD             3                 0.70           0.68
3        SGD             4                 0.70           0.64
4       ADAM             1                 0.70           0.60
5       ADAM             2                 0.75           0.64
6       ADAM             3                 0.65           0.56
7       ADAM             4                 0.75           0.48
8    AdaGrad             1                 0.70           0.60
9    AdaGrad             2                 0.75           0.64
10   AdaGrad             3                 0.65           0.56
11   AdaGrad             4                 0.75           0.48
```

2.C RMSProp

Train and evaluate the model with 1 hidden Layer with RMSProp Optimizer

```
# Melatih model dengan data pelatihan
optimizer_rmsprop = tf.keras.optimizers.RMSprop()
model_one_hidden_layer.compile(optimizer=optimizer_rmsprop, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_one_hidden_layer.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_val))
```

```
Epoch 76/100
8/8 [==============================] - 2s 191ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 19.5032 - val_accuracy: 0.
Epoch 77/100
8/8 [==============================] - 1s 186ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 19.5025 - val_accuracy: 0.
Epoch 78/100
8/8 [==============================] - 1s 187ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 19.5019 - val_accuracy: 0.
Epoch 79/100
8/8 [==============================] - 2s 221ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 19.5012 - val_accuracy: 0.
Epoch 80/100
8/8 [==============================] - 2s 262ms/step - loss: 9.0539e-09 - accuracy: 1.0000 - val_loss: 19.5006 - val_accuracy: 0.
Epoch 81/100
8/8 [==============================] - 2s 194ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.5001 - val_accuracy: 0.
Epoch 82/100
8/8 [==============================] - 2s 210ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4996 - val_accuracy: 0.
Epoch 83/100
8/8 [==============================] - 1s 184ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4991 - val_accuracy: 0.
Epoch 84/100
8/8 [==============================] - 1s 182ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4986 - val_accuracy: 0.
Epoch 85/100
8/8 [==============================] - 1s 186ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4981 - val_accuracy: 0.
Epoch 86/100
8/8 [==============================] - 1s 185ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4976 - val_accuracy: 0.
Epoch 87/100
8/8 [==============================] - 2s 259ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4971 - val_accuracy: 0.
Epoch 88/100
8/8 [==============================] - 2s 229ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4966 - val_accuracy: 0.
Epoch 89/100
8/8 [==============================] - 1s 181ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4961 - val_accuracy: 0.
Epoch 90/100
8/8 [==============================] - 1s 183ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4956 - val_accuracy: 0.
Epoch 91/100
8/8 [==============================] - 2s 283ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4952 - val_accuracy: 0.
Epoch 92/100
8/8 [==============================] - 2s 236ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4947 - val_accuracy: 0.
Epoch 93/100
8/8 [==============================] - 2s 190ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4942 - val_accuracy: 0.
Epoch 94/100
8/8 [==============================] - 2s 230ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4937 - val_accuracy: 0.
Epoch 95/100
8/8 [==============================] - 2s 252ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4932 - val_accuracy: 0.
Epoch 96/100
8/8 [==============================] - 1s 188ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4928 - val_accuracy: 0.
Epoch 97/100
8/8 [==============================] - 1s 185ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4923 - val_accuracy: 0.
Epoch 98/100
8/8 [==============================] - 2s 191ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4919 - val_accuracy: 0.
Epoch 99/100
8/8 [==============================] - 1s 181ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4914 - val_accuracy: 0.
Epoch 100/100
8/8 [==============================] - 1s 183ms/step - loss: 6.0359e-09 - accuracy: 1.0000 - val_loss: 19.4910 - val_accuracy: 0.
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_one_hidden_layer.evaluate(X_val_scaled, y_val)
print("Model 1 Hidden Layer RMSprop Optimizer Validation Accuracy:", val_acc)
# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_one_hidden_layer  # Misalnya, model pertama dianggap sebagai model terbaik
# Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 143ms/step - loss: 19.4910 - accuracy: 0.7000
Model 1 Hidden Layer RMSprop Optimizer Validation Accuracy: 0.699999988079071
1/1 [==============================] - 0s 44ms/step - loss: 19.0706 - accuracy: 0.6000
Best Model Test Accuracy: 0.6000000238418579
```

```
update_results('RMSprop', 1, val_acc, test_acc)
print(results_df)
```

```
   Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0        SGD             1                 0.75           0.64
1        SGD             2                 0.75           0.64
2        SGD             3                 0.70           0.68
3        SGD             4                 0.70           0.64
4       ADAM             1                 0.70           0.60
5       ADAM             2                 0.75           0.64
6       ADAM             3                 0.65           0.56
7       ADAM             4                 0.75           0.48
8    AdaGrad             1                 0.70           0.60
9    AdaGrad             2                 0.75           0.64
10   AdaGrad             3                 0.65           0.56
11   AdaGrad             4                 0.75           0.48
12   RMSprop             1                 0.70           0.60
```

Train and evaluate the model with 2 hidden Layer with RMSProp Optimizer

```
# Melatih model dengan data pelatihan
optimizer_rmsprop = tf.keras.optimizers.RMSprop()
model_two_hidden_layer.compile(optimizer=optimizer_rmsprop, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_two_hidden_layer.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_val))
```

```
8/8 [==============================] - 1s 97ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3910 - val_accuracy: 0.7
Epoch 73/100
8/8 [==============================] - 1s 87ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3910 - val_accuracy: 0.7
Epoch 74/100
8/8 [==============================] - 1s 96ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3910 - val_accuracy: 0.7
Epoch 75/100
8/8 [==============================] - 1s 67ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3910 - val_accuracy: 0.7
Epoch 76/100
8/8 [==============================] - 1s 65ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3910 - val_accuracy: 0.7
Epoch 77/100
8/8 [==============================] - 1s 66ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3910 - val_accuracy: 0.7
Epoch 78/100
8/8 [==============================] - 0s 62ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3910 - val_accuracy: 0.7
Epoch 79/100
8/8 [==============================] - 1s 68ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3911 - val_accuracy: 0.7
Epoch 80/100
8/8 [==============================] - 1s 64ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3911 - val_accuracy: 0.7
Epoch 81/100
8/8 [==============================] - 1s 69ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3911 - val_accuracy: 0.7
Epoch 82/100
8/8 [==============================] - 0s 63ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3911 - val_accuracy: 0.7
Epoch 83/100
8/8 [==============================] - 0s 64ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3911 - val_accuracy: 0.7
Epoch 84/100
8/8 [==============================] - 0s 63ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3911 - val_accuracy: 0.7
Epoch 85/100
8/8 [==============================] - 0s 63ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3911 - val_accuracy: 0.7
Epoch 86/100
8/8 [==============================] - 0s 63ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3912 - val_accuracy: 0.7
Epoch 87/100
8/8 [==============================] - 1s 67ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3912 - val_accuracy: 0.7
Epoch 88/100
8/8 [==============================] - 0s 62ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3912 - val_accuracy: 0.7
Epoch 89/100
8/8 [==============================] - 0s 63ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3912 - val_accuracy: 0.7
Epoch 90/100
8/8 [==============================] - 0s 62ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3912 - val_accuracy: 0.7
Epoch 91/100
8/8 [==============================] - 0s 62ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3912 - val_accuracy: 0.7
Epoch 92/100
8/8 [==============================] - 1s 64ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3913 - val_accuracy: 0.7
Epoch 93/100
8/8 [==============================] - 1s 65ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3913 - val_accuracy: 0.7
Epoch 94/100
8/8 [==============================] - 1s 77ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3913 - val_accuracy: 0.7
Epoch 95/100
8/8 [==============================] - 1s 96ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3913 - val_accuracy: 0.7
Epoch 96/100
8/8 [==============================] - 1s 90ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3913 - val_accuracy: 0.7
Epoch 97/100
8/8 [==============================] - 1s 87ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3913 - val_accuracy: 0.7
Epoch 98/100
8/8 [==============================] - 1s 66ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3913 - val_accuracy: 0.7
Epoch 99/100
8/8 [==============================] - 1s 66ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3913 - val_accuracy: 0.7
Epoch 100/100
8/8 [==============================] - 1s 65ms/step - loss: 7.5449e-09 - accuracy: 1.0000 - val_loss: 11.3914 - val_accuracy: 0.7
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_two_hidden_layer.evaluate(X_val_scaled, y_val)
print("Model 2 Hidden Layer RMSprop Optimizer Validation Accuracy:", val_acc)
# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_two_hidden_layer  # Misalnya, model pertama dianggap sebagai model terbaik
# Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 147ms/step - loss: 11.3914 - accuracy: 0.7500
Model 2 Hidden Layer RMSprop Optimizer Validation Accuracy: 0.75
1/1 [==============================] - 0s 38ms/step - loss: 19.7389 - accuracy: 0.6400
Best Model Test Accuracy: 0.6399999856948853
```

```
update_results('RMSprop', 2, val_acc, test_acc)
print(results_df)
```

```
  Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0       SGD             1                 0.75           0.64
1       SGD             2                 0.75           0.64
2       SGD             3                 0.70           0.68
3       SGD             4                 0.70           0.64
4      ADAM             1                 0.70           0.60
```

```
5       ADAM              2                    0.75           0.64
6       ADAM              3                    0.65           0.56
7       ADAM              4                    0.75           0.48
8    AdaGrad              1                    0.70           0.60
9    AdaGrad              2                    0.75           0.64
10   AdaGrad              3                    0.65           0.56
11   AdaGrad              4                    0.75           0.48
12   RMSprop              1                    0.70           0.60
13   RMSprop              2                    0.75           0.64
```

Train and evaluate the model with 3 hidden Layer with RMSProp Optimizer

```
# Melatih model dengan data pelatihan
optimizer_rmsprop = tf.keras.optimizers.RMSprop()
model_three_hidden_layer.compile(optimizer=optimizer_rmsprop, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_three_hidden_layer.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_val))
```

```
8/8 [==============================] - 0s 32ms/step - loss: 1.6599e-07 - accuracy: 1.0000 - val_loss: 5.0186 - val_accuracy: 0.65
Epoch 73/100
8/8 [==============================] - 0s 32ms/step - loss: 1.6448e-07 - accuracy: 1.0000 - val_loss: 5.0187 - val_accuracy: 0.65
Epoch 74/100
8/8 [==============================] - 0s 32ms/step - loss: 1.6297e-07 - accuracy: 1.0000 - val_loss: 5.0188 - val_accuracy: 0.65
Epoch 75/100
8/8 [==============================] - 0s 35ms/step - loss: 1.6146e-07 - accuracy: 1.0000 - val_loss: 5.0188 - val_accuracy: 0.65
Epoch 76/100
8/8 [==============================] - 0s 32ms/step - loss: 1.5844e-07 - accuracy: 1.0000 - val_loss: 5.0188 - val_accuracy: 0.65
Epoch 77/100
8/8 [==============================] - 0s 32ms/step - loss: 1.5844e-07 - accuracy: 1.0000 - val_loss: 5.0188 - val_accuracy: 0.65
Epoch 78/100
8/8 [==============================] - 0s 32ms/step - loss: 1.5693e-07 - accuracy: 1.0000 - val_loss: 5.0187 - val_accuracy: 0.65
Epoch 79/100
8/8 [==============================] - 0s 33ms/step - loss: 1.5542e-07 - accuracy: 1.0000 - val_loss: 5.0188 - val_accuracy: 0.65
Epoch 80/100
8/8 [==============================] - 0s 32ms/step - loss: 1.5542e-07 - accuracy: 1.0000 - val_loss: 5.0188 - val_accuracy: 0.65
Epoch 81/100
8/8 [==============================] - 0s 32ms/step - loss: 1.5392e-07 - accuracy: 1.0000 - val_loss: 5.0189 - val_accuracy: 0.65
Epoch 82/100
8/8 [==============================] - 0s 33ms/step - loss: 1.5090e-07 - accuracy: 1.0000 - val_loss: 5.0190 - val_accuracy: 0.65
Epoch 83/100
8/8 [==============================] - 0s 34ms/step - loss: 1.5090e-07 - accuracy: 1.0000 - val_loss: 5.0192 - val_accuracy: 0.65
Epoch 84/100
8/8 [==============================] - 0s 34ms/step - loss: 1.4788e-07 - accuracy: 1.0000 - val_loss: 5.0192 - val_accuracy: 0.65
Epoch 85/100
8/8 [==============================] - 0s 34ms/step - loss: 1.4486e-07 - accuracy: 1.0000 - val_loss: 5.0192 - val_accuracy: 0.65
Epoch 86/100
8/8 [==============================] - 0s 33ms/step - loss: 1.4184e-07 - accuracy: 1.0000 - val_loss: 5.0193 - val_accuracy: 0.65
Epoch 87/100
8/8 [==============================] - 0s 31ms/step - loss: 1.4184e-07 - accuracy: 1.0000 - val_loss: 5.0193 - val_accuracy: 0.65
Epoch 88/100
8/8 [==============================] - 0s 35ms/step - loss: 1.4033e-07 - accuracy: 1.0000 - val_loss: 5.0193 - val_accuracy: 0.65
Epoch 89/100
8/8 [==============================] - 0s 35ms/step - loss: 1.4033e-07 - accuracy: 1.0000 - val_loss: 5.0193 - val_accuracy: 0.65
Epoch 90/100
8/8 [==============================] - 0s 34ms/step - loss: 1.3883e-07 - accuracy: 1.0000 - val_loss: 5.0193 - val_accuracy: 0.65
Epoch 91/100
8/8 [==============================] - 0s 33ms/step - loss: 1.3732e-07 - accuracy: 1.0000 - val_loss: 5.0194 - val_accuracy: 0.65
Epoch 92/100
8/8 [==============================] - 0s 33ms/step - loss: 1.3732e-07 - accuracy: 1.0000 - val_loss: 5.0194 - val_accuracy: 0.65
Epoch 93/100
8/8 [==============================] - 0s 31ms/step - loss: 1.3732e-07 - accuracy: 1.0000 - val_loss: 5.0194 - val_accuracy: 0.65
Epoch 94/100
8/8 [==============================] - 0s 36ms/step - loss: 1.3430e-07 - accuracy: 1.0000 - val_loss: 5.0196 - val_accuracy: 0.65
Epoch 95/100
8/8 [==============================] - 0s 30ms/step - loss: 1.3430e-07 - accuracy: 1.0000 - val_loss: 5.0197 - val_accuracy: 0.65
Epoch 96/100
8/8 [==============================] - 0s 32ms/step - loss: 1.3279e-07 - accuracy: 1.0000 - val_loss: 5.0197 - val_accuracy: 0.65
Epoch 97/100
8/8 [==============================] - 0s 31ms/step - loss: 1.3279e-07 - accuracy: 1.0000 - val_loss: 5.0197 - val_accuracy: 0.65
Epoch 98/100
8/8 [==============================] - 0s 33ms/step - loss: 1.2977e-07 - accuracy: 1.0000 - val_loss: 5.0197 - val_accuracy: 0.65
Epoch 99/100
8/8 [==============================] - 0s 31ms/step - loss: 1.2977e-07 - accuracy: 1.0000 - val_loss: 5.0198 - val_accuracy: 0.65
Epoch 100/100
8/8 [==============================] - 0s 34ms/step - loss: 1.2977e-07 - accuracy: 1.0000 - val_loss: 5.0198 - val_accuracy: 0.65
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_three_hidden_layer.evaluate(X_val_scaled, y_val)
print("Model 3 Hidden Layer RMSprop Optimizer Validation Accuracy:", val_acc)
# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_three_hidden_layer  # Misalnya, model pertama dianggap sebagai model terbaik
# Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 210ms/step - loss: 5.0198 - accuracy: 0.6500
Model 3 Hidden Layer RMSprop Optimizer Validation Accuracy: 0.6499999761581421
1/1 [==============================] - 0s 57ms/step - loss: 7.0093 - accuracy: 0.5600
Best Model Test Accuracy: 0.5600000023841858
```

```
update_results('RMSprop', 3, val_acc, test_acc)
print(results_df)
```

```
   Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0        SGD             1                 0.75           0.64
1        SGD             2                 0.75           0.64
2        SGD             3                 0.70           0.68
3        SGD             4                 0.70           0.64
4       ADAM             1                 0.70           0.60
5       ADAM             2                 0.75           0.64
6       ADAM             3                 0.65           0.56
7       ADAM             4                 0.75           0.48
8    AdaGrad             1                 0.70           0.60
9    AdaGrad             2                 0.75           0.64
10   AdaGrad             3                 0.65           0.56
11   AdaGrad             4                 0.75           0.48
12   RMSprop             1                 0.70           0.60
13   RMSprop             2                 0.75           0.64
14   RMSprop             3                 0.65           0.56
```

Train and evaluate the model with 4 hidden Layer with RMSProp Optimizer

```
# Melatih model dengan data pelatihan
optimizer_rmsprop = tf.keras.optimizers.RMSprop()
model_four_hidden_layer.compile(optimizer=optimizer_rmsprop, loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_four_hidden_layer.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_val))
```

```
8/8 [==============================] - 0s 20ms/step - loss: 2.0522e-07 - accuracy: 1.0000 - val_loss: 4.8447 - val_accuracy: 0.75
Epoch 100/100
8/8 [==============================] - 0s 17ms/step - loss: 2.0371e-07 - accuracy: 1.0000 - val_loss: 4.8459 - val_accuracy: 0.75
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_four_hidden_layer.evaluate(X_val_scaled, y_val)
print("Model 4 Hidden Layer RMSprop Optimizer Validation Accuracy:", val_acc)
# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_four_hidden_layer  # Misalnya, model pertama dianggap sebagai model terbaik
# Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

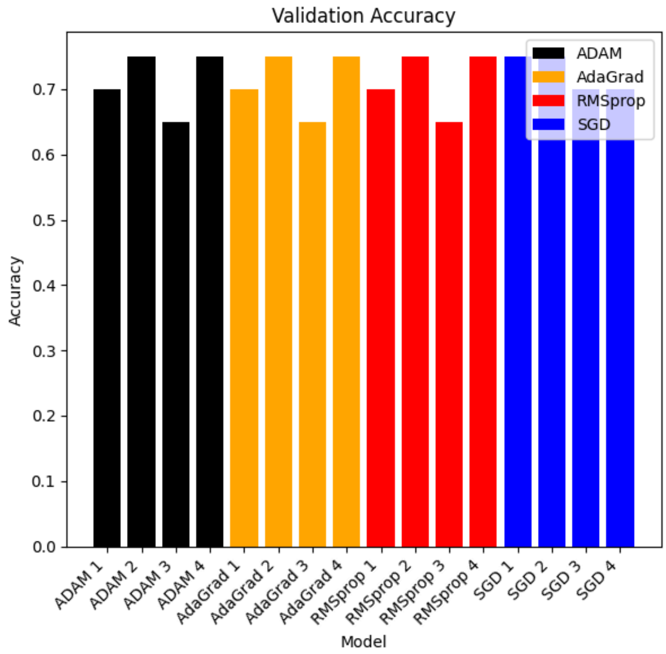
```
1/1 [==============================] - 0s 150ms/step - loss: 4.8459 - accuracy: 0.7500
Model 4 Hidden Layer RMSprop Optimizer Validation Accuracy: 0.75
1/1 [==============================] - 0s 36ms/step - loss: 5.7399 - accuracy: 0.4800
Best Model Test Accuracy: 0.47999998927116394
```

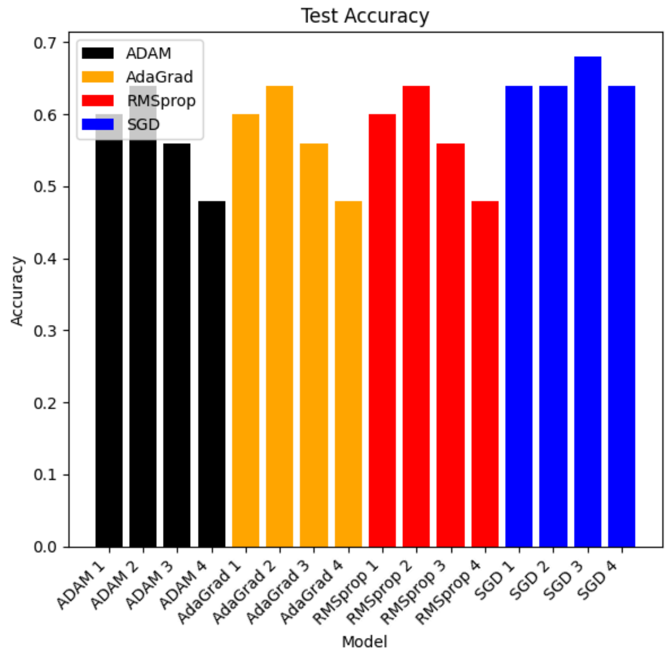
```
update_results('RMSprop', 4, val_acc, test_acc)
print(results_df)
```

```
   Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0        SGD             1                 0.75           0.64
1        SGD             2                 0.75           0.64
2        SGD             3                 0.70           0.68
3        SGD             4                 0.70           0.64
4       ADAM             1                 0.70           0.60
5       ADAM             2                 0.75           0.64
6       ADAM             3                 0.65           0.56
7       ADAM             4                 0.75           0.48
8    AdaGrad             1                 0.70           0.60
9    AdaGrad             2                 0.75           0.64
10   AdaGrad             3                 0.65           0.56
11   AdaGrad             4                 0.75           0.48
12   RMSprop             1                 0.70           0.60
13   RMSprop             2                 0.75           0.64
14   RMSprop             3                 0.65           0.56
15   RMSprop             4                 0.75           0.48
```

2.D Hasil Perbandingan Optimizer SGD, ADAM, AdaGrad, RMSprop

```
import matplotlib.pyplot as plt

# Membuat palet warna yang berbeda untuk setiap optimizer
colors = {'SGD': 'blue', 'Adam': 'green', 'AdaGrad': 'orange', 'RMSprop': 'red'}

# Membuat subplots
fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(12, 6))

# Grafik akurasi validasi
for optimizer, group in results_df.groupby('Optimizer'):
    axs[0].bar(group['Optimizer'] + ' ' + group['Hidden Layers'].astype(str), group['Validation Accuracy'], color=colors.get(optimizer,
axs[0].set_title('Validation Accuracy')
axs[0].set_xlabel('Model', fontsize=10)  # Penyesuaian ukuran font
axs[0].set_ylabel('Accuracy')
axs[0].set_xticklabels(axs[0].get_xticklabels(), rotation=45, ha='right')  # Rotasi label sumbu x
axs[0].legend()

# Grafik akurasi pengujian
for optimizer, group in results_df.groupby('Optimizer'):
    axs[1].bar(group['Optimizer'] + ' ' + group['Hidden Layers'].astype(str), group['Test Accuracy'], color=colors.get(optimizer, 'blac
axs[1].set_title('Test Accuracy')
axs[1].set_xlabel('Model', fontsize=10)  # Penyesuaian ukuran font
axs[1].set_ylabel('Accuracy')
axs[1].set_xticklabels(axs[1].get_xticklabels(), rotation=45, ha='right')  # Rotasi label sumbu x
axs[1].legend()

# Menampilkan grafik
plt.tight_layout()
plt.show()
```






```
from IPython.display import display

# Menampilkan DataFrame
display(results_df)
```

	Optimizer	Hidden Layers	Validation Accuracy	Test Accuracy
0	SGD	1	0.75	0.64
1	SGD	2	0.75	0.64
2	SGD	3	0.70	0.68
3	SGD	4	0.70	0.64
4	ADAM	1	0.70	0.60
5	ADAM	2	0.75	0.64
6	ADAM	3	0.65	0.56
7	ADAM	4	0.75	0.48
8	AdaGrad	1	0.70	0.60
9	AdaGrad	2	0.75	0.64
10	AdaGrad	3	0.65	0.56
11	AdaGrad	4	0.75	0.48
12	RMSprop	1	0.70	0.60
13	RMSprop	2	0.75	0.64
14	RMSprop	3	0.65	0.56
15	RMSprop	4	0.75	0.48

Next steps: View recommended plots

```
# Temukan indeks baris dengan akurasi pengujian tertinggi dan terendah
best_model_idx = results_df['Test Accuracy'].idxmax()
worst_model_idx = results_df['Test Accuracy'].idxmin()

# Dapatkan informasi tentang model dengan akurasi pengujian tertinggi dan terendah
best_model_info = results_df.loc[best_model_idx]
worst_model_info = results_df.loc[worst_model_idx]

print("Model dengan akurasi pengujian tertinggi:")
print(best_model_info)
print("\nModel dengan akurasi pengujian terendah:")
print(worst_model_info)
```

```
Model dengan akurasi pengujian tertinggi:
Optimizer               SGD
Hidden Layers             3
Validation Accuracy     0.7
Test Accuracy          0.68
Name: 2, dtype: object

Model dengan akurasi pengujian terendah:
Optimizer              ADAM
Hidden Layers             4
Validation Accuracy    0.75
Test Accuracy          0.48
Name: 7, dtype: object
```

3. Menambahkan Dropout 50%

3.A Menambahkan Dropout di salah satu layer

```
# Function to build the DNN model with one hidden layer
def build_model_one_hidden_with_dropout():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(1000, activation='relu', input_shape=(X_train_scaled.shape[1],)),  # Hidden layer with 1000 neurons and R
        tf.keras.layers.Dropout(0.5),  # Dropout layer with dropout rate of 50%
        tf.keras.layers.Dense(len(label_encoder.classes_), activation='sigmoid')          # Output layer with sigmoid activation
    ])
    return model

# Function to build the DNN model with two hidden layer
def build_model_two_hidden_with_dropout():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(500, activation='relu', input_shape=(X_train_scaled.shape[1],)),  # 1st hidden layer with 500 neurons and
        tf.keras.layers.Dense(500, activation='relu'),                                   # 2nd hidden layer with 500 neurons and ReLU a
        tf.keras.layers.Dropout(0.5),  # Dropout layer with dropout rate of 50%
        tf.keras.layers.Dense(len(label_encoder.classes_), activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model

# Function to build the DNN model with three hidden layer
def build_model_three_hidden_with_dropout():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(250, activation='relu', input_shape=(X_train_scaled.shape[1],)),  # First hidden layer with 250 neurons a
        tf.keras.layers.Dense(250, activation='relu'),                                   # Second hidden layer with 250 neurons and ReL
        tf.keras.layers.Dense(250, activation='relu'),                                   # Third hidden layer with 250 neurons and ReLU
        tf.keras.layers.Dropout(0.5),  # Dropout layer with dropout rate of 50%
        tf.keras.layers.Dense(len(label_encoder.classes_), activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model

# Function to build the DNN model with four hidden layer
def build_model_four_hidden_with_dropout():
    model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(100, activation='relu', input_shape=(X_train_scaled.shape[1],)),  # First hidden layer with 100 neurons a
        tf.keras.layers.Dense(100, activation='relu'),                                   # Second hidden layer with 100 neurons and ReL
        tf.keras.layers.Dense(100, activation='relu'),                                   # Third hidden layer with 100 neurons and ReLU
        tf.keras.layers.Dense(100, activation='relu'),                                   # Fourth hidden layer with 100 neurons and ReL
        tf.keras.layers.Dropout(0.5),  # Dropout layer with dropout rate of 50%
        tf.keras.layers.Dense(len(label_encoder.classes_), activation='sigmoid')         # Output layer with sigmoid activation
    ])
    return model
```

```
# Build the DNN model
model_one_hidden_layer_with_dropout = build_model_one_hidden_with_dropout()
model_two_hidden_layer_with_dropout = build_model_two_hidden_with_dropout()
model_three_hidden_layer_with_dropout = build_model_three_hidden_with_dropout()
model_four_hidden_layer_with_dropout = build_model_four_hidden_with_dropout()
```

SGD - Train and evaluate the model with 1 hidden Layer

```
# Melatih model dengan data pelatihan
# optimizer_sgd = tf.keras.optimizers.SGD(learning_rate=0.01)  # Define the optimizer with learning rate
model_one_hidden_layer_with_dropout.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_one_hidden_layer_with_dropout.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_
```

```
8/8 [==============================] - 1s 79ms/step - loss: 5.8558e-04 - accuracy: 1.0000 - val_loss: 1.1739 - val_accuracy: 0.70
Epoch 73/100
8/8 [==============================] - 1s 79ms/step - loss: 4.9079e-04 - accuracy: 1.0000 - val_loss: 1.1782 - val_accuracy: 0.70
Epoch 74/100
8/8 [==============================] - 1s 81ms/step - loss: 2.2518e-04 - accuracy: 1.0000 - val_loss: 1.1786 - val_accuracy: 0.70
Epoch 75/100
8/8 [==============================] - 1s 79ms/step - loss: 3.3354e-04 - accuracy: 1.0000 - val_loss: 1.1845 - val_accuracy: 0.70
Epoch 76/100
8/8 [==============================] - 1s 115ms/step - loss: 5.0152e-04 - accuracy: 1.0000 - val_loss: 1.1772 - val_accuracy: 0.7
Epoch 77/100
8/8 [==============================] - 1s 122ms/step - loss: 7.3789e-04 - accuracy: 1.0000 - val_loss: 1.1779 - val_accuracy: 0.7
Epoch 78/100
8/8 [==============================] - 1s 113ms/step - loss: 5.8728e-04 - accuracy: 1.0000 - val_loss: 1.1733 - val_accuracy: 0.7
Epoch 79/100
8/8 [==============================] - 1s 82ms/step - loss: 3.0727e-04 - accuracy: 1.0000 - val_loss: 1.1747 - val_accuracy: 0.70
Epoch 80/100
8/8 [==============================] - 1s 84ms/step - loss: 4.9781e-04 - accuracy: 1.0000 - val_loss: 1.1800 - val_accuracy: 0.70
Epoch 81/100
8/8 [==============================] - 1s 80ms/step - loss: 5.1610e-04 - accuracy: 1.0000 - val_loss: 1.1792 - val_accuracy: 0.70
Epoch 82/100
8/8 [==============================] - 1s 80ms/step - loss: 5.5726e-04 - accuracy: 1.0000 - val_loss: 1.1780 - val_accuracy: 0.70
Epoch 83/100
8/8 [==============================] - 1s 80ms/step - loss: 6.1182e-04 - accuracy: 1.0000 - val_loss: 1.1858 - val_accuracy: 0.70
Epoch 84/100
8/8 [==============================] - 1s 76ms/step - loss: 2.7168e-04 - accuracy: 1.0000 - val_loss: 1.1866 - val_accuracy: 0.70
Epoch 85/100
8/8 [==============================] - 1s 78ms/step - loss: 4.7250e-04 - accuracy: 1.0000 - val_loss: 1.1848 - val_accuracy: 0.70
Epoch 86/100
8/8 [==============================] - 1s 74ms/step - loss: 6.0051e-04 - accuracy: 1.0000 - val_loss: 1.1894 - val_accuracy: 0.70
Epoch 87/100
8/8 [==============================] - 1s 78ms/step - loss: 1.0878e-04 - accuracy: 1.0000 - val_loss: 1.1900 - val_accuracy: 0.70
Epoch 88/100
8/8 [==============================] - 1s 77ms/step - loss: 1.4265e-04 - accuracy: 1.0000 - val_loss: 1.1905 - val_accuracy: 0.70
Epoch 89/100
8/8 [==============================] - 1s 81ms/step - loss: 1.9531e-04 - accuracy: 1.0000 - val_loss: 1.1915 - val_accuracy: 0.70
Epoch 90/100
8/8 [==============================] - 1s 82ms/step - loss: 5.5735e-04 - accuracy: 1.0000 - val_loss: 1.1944 - val_accuracy: 0.70
Epoch 91/100
8/8 [==============================] - 1s 76ms/step - loss: 3.8377e-04 - accuracy: 1.0000 - val_loss: 1.1936 - val_accuracy: 0.70
Epoch 92/100
8/8 [==============================] - 1s 81ms/step - loss: 2.2634e-04 - accuracy: 1.0000 - val_loss: 1.1932 - val_accuracy: 0.70
Epoch 93/100
8/8 [==============================] - 1s 75ms/step - loss: 6.8256e-04 - accuracy: 1.0000 - val_loss: 1.1876 - val_accuracy: 0.70
Epoch 94/100
8/8 [==============================] - 1s 84ms/step - loss: 2.3576e-04 - accuracy: 1.0000 - val_loss: 1.1886 - val_accuracy: 0.70
Epoch 95/100
8/8 [==============================] - 1s 121ms/step - loss: 4.7486e-04 - accuracy: 1.0000 - val_loss: 1.1939 - val_accuracy: 0.7
Epoch 96/100
8/8 [==============================] - 1s 134ms/step - loss: 3.8855e-04 - accuracy: 1.0000 - val_loss: 1.1892 - val_accuracy: 0.7
Epoch 97/100
8/8 [==============================] - 2s 306ms/step - loss: 5.5889e-04 - accuracy: 1.0000 - val_loss: 1.1921 - val_accuracy: 0.7
Epoch 98/100
8/8 [==============================] - 2s 245ms/step - loss: 1.2166e-04 - accuracy: 1.0000 - val_loss: 1.1930 - val_accuracy: 0.7
Epoch 99/100
8/8 [==============================] - 2s 237ms/step - loss: 3.7764e-04 - accuracy: 1.0000 - val_loss: 1.1937 - val_accuracy: 0.7
Epoch 100/100
8/8 [==============================] - 2s 281ms/step - loss: 3.9685e-04 - accuracy: 1.0000 - val_loss: 1.1931 - val_accuracy: 0.7
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_one_hidden_layer_with_dropout.evaluate(X_val_scaled, y_val)
print("Model Validation Accuracy:", val_acc)

# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_one_hidden_layer_with_dropout  # Misalnya, model pertama dianggap sebagai model terbaik
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 155ms/step - loss: 1.1931 - accuracy: 0.7000
Model Validation Accuracy: 0.699999988079071
1/1 [==============================] - 0s 46ms/step - loss: 1.4047 - accuracy: 0.6000
Best Model Test Accuracy: 0.6000000238418579
```

```
update_results('SGD+D', 1, val_acc, test_acc)
print(results_df)
```

```
   Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0        SGD             1                 0.75           0.64
1        SGD             2                 0.75           0.64
2        SGD             3                 0.70           0.68
3        SGD             4                 0.70           0.64
4       ADAM             1                 0.70           0.60
5       ADAM             2                 0.75           0.64
6       ADAM             3                 0.65           0.56
7       ADAM             4                 0.75           0.48
8    AdaGrad             1                 0.70           0.60
9    AdaGrad             2                 0.75           0.64
10   AdaGrad             3                 0.65           0.56
11   AdaGrad             4                 0.75           0.48
12   RMSprop             1                 0.70           0.60
13   RMSprop             2                 0.75           0.64
14   RMSprop             3                 0.65           0.56
15   RMSprop             4                 0.75           0.48
16     SGD+D             1                 0.70           0.60
```

3.B Menambahkan Dropout di semua layer

SGD - Train and evaluate the model with 2 hidden Layer

```
# Melatih model dengan data pelatihan
model_two_hidden_layer_with_dropout.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_two_hidden_layer_with_dropout.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_
```

```
8/8 [==============================] - 0s 35ms/step - loss: 9.5807e-04 - accuracy: 1.0000 - val_loss: 0.8846 - val_accuracy: 0.75
Epoch 72/100
8/8 [==============================] - 0s 36ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 0.8832 - val_accuracy: 0.7500
```

SGD - Train and evaluate the model with 3 hidden Layer

```
# Melatih model dengan data pelatihan
model_three_hidden_layer_with_dropout.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_three_hidden_layer_with_dropout.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y_
```

```
8/8 [==============================] - 0s 30ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.5991 - val_accuracy: 0.7500
Epoch 68/100
8/8 [==============================] - 0s 28ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 0.6068 - val_accuracy: 0.7500
Epoch 69/100
8/8 [==============================] - 0s 30ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.6076 - val_accuracy: 0.7500
Epoch 70/100
8/8 [==============================] - 0s 28ms/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 0.6072 - val_accuracy: 0.7500
Epoch 71/100
8/8 [==============================] - 0s 28ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.6099 - val_accuracy: 0.7500
Epoch 72/100
8/8 [==============================] - 0s 30ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.6122 - val_accuracy: 0.7500
Epoch 73/100
8/8 [==============================] - 0s 30ms/step - loss: 9.9485e-04 - accuracy: 1.0000 - val_loss: 0.6128 - val_accuracy: 0.75
Epoch 74/100
8/8 [==============================] - 0s 31ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.6145 - val_accuracy: 0.7500
Epoch 75/100
8/8 [==============================] - 0s 35ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.6148 - val_accuracy: 0.7500
Epoch 76/100
8/8 [==============================] - 0s 27ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.6167 - val_accuracy: 0.7500
Epoch 77/100
8/8 [==============================] - 0s 20ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 0.6203 - val_accuracy: 0.7500
Epoch 78/100
8/8 [==============================] - 0s 21ms/step - loss: 0.0025 - accuracy: 1.0000 - val_loss: 0.6229 - val_accuracy: 0.7500
Epoch 79/100
8/8 [==============================] - 0s 20ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.6250 - val_accuracy: 0.7500
Epoch 80/100
8/8 [==============================] - 0s 18ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.6224 - val_accuracy: 0.7500
Epoch 81/100
8/8 [==============================] - 0s 19ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.6244 - val_accuracy: 0.7500
Epoch 82/100
8/8 [==============================] - 0s 19ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 0.6281 - val_accuracy: 0.7500
Epoch 83/100
8/8 [==============================] - 0s 18ms/step - loss: 8.9276e-04 - accuracy: 1.0000 - val_loss: 0.6284 - val_accuracy: 0.75
Epoch 84/100
8/8 [==============================] - 0s 18ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.6276 - val_accuracy: 0.7500
Epoch 85/100
8/8 [==============================] - 0s 21ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 0.6264 - val_accuracy: 0.7500
Epoch 86/100
8/8 [==============================] - 0s 19ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 0.6280 - val_accuracy: 0.7500
Epoch 87/100
8/8 [==============================] - 0s 19ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 0.6297 - val_accuracy: 0.7500
Epoch 88/100
8/8 [==============================] - 0s 20ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 0.6326 - val_accuracy: 0.7500
Epoch 89/100
8/8 [==============================] - 0s 18ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.6343 - val_accuracy: 0.7500
Epoch 90/100
8/8 [==============================] - 0s 18ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 0.6375 - val_accuracy: 0.7500
Epoch 91/100
8/8 [==============================] - 0s 25ms/step - loss: 0.0020 - accuracy: 1.0000 - val_loss: 0.6343 - val_accuracy: 0.7500
Epoch 92/100
8/8 [==============================] - 0s 19ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 0.6361 - val_accuracy: 0.7500
Epoch 93/100
8/8 [==============================] - 0s 19ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 0.6367 - val_accuracy: 0.7500
Epoch 94/100
8/8 [==============================] - 0s 21ms/step - loss: 6.3100e-04 - accuracy: 1.0000 - val_loss: 0.6379 - val_accuracy: 0.75
Epoch 95/100
8/8 [==============================] - 0s 18ms/step - loss: 8.1607e-04 - accuracy: 1.0000 - val_loss: 0.6384 - val_accuracy: 0.75
Epoch 96/100
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_two_hidden_layer_with_dropout.evaluate(X_val_scaled, y_val)
print("Model 2 Hidden Layer SGD Optimizer Validation Accuracy:", val_acc)

# 5. Pilih model terbaik berdasarkan kinerja validasi
best_model = model_two_hidden_layer_with_dropout  # Misalnya, model pertama dianggap sebagai model terbaik

# 6. Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 153ms/step - loss: 0.9057 - accuracy: 0.7500
Model 2 Hidden Layer SGD Optimizer Validation Accuracy: 0.75
1/1 [==============================] - 0s 38ms/step - loss: 0.9844 - accuracy: 0.6000
Best Model Test Accuracy: 0.6000000238418579
```

```
update_results('SGD+D', 2, val_acc, test_acc)
print(results_df)
```

```
   Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0        SGD             1                 0.75           0.64
1        SGD             2                 0.75           0.64
2        SGD             3                 0.70           0.68
3        SGD             4                 0.70           0.64
4       ADAM             1                 0.70           0.60
5       ADAM             2                 0.75           0.64
6       ADAM             3                 0.65           0.56
7       ADAM             4                 0.75           0.48
8    AdaGrad             1                 0.70           0.60
9    AdaGrad             2                 0.75           0.64
10   AdaGrad             3                 0.65           0.56
11   AdaGrad             4                 0.75           0.48
12   RMSprop             1                 0.70           0.60
13   RMSprop             2                 0.75           0.64
14   RMSprop             3                 0.65           0.56
15   RMSprop             4                 0.75           0.48
16     SGD+D             1                 0.70           0.60
17     SGD+D             2                 0.75           0.60
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_three_hidden_layer_with_dropout.evaluate(X_val_scaled, y_val)
print("Model 3 Hidden Layer SGD Optimizer Validation Accuracy:", val_acc)

# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_three_hidden_layer_with_dropout  # Misalnya, model pertama dianggap sebagai model terbaik

# Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 152ms/step - loss: 0.6483 - accuracy: 0.7500
Model 3 Hidden Layer SGD Optimizer Validation Accuracy: 0.75
1/1 [==============================] - 0s 31ms/step - loss: 1.0951 - accuracy: 0.5600
Best Model Test Accuracy: 0.5600000023841858
```

```
update_results('SGD+D', 3, val_acc, test_acc)
print(results_df)
```

```
   Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0        SGD             1                 0.75           0.64
1        SGD             2                 0.75           0.64
2        SGD             3                 0.70           0.68
3        SGD             4                 0.70           0.64
4       ADAM             1                 0.70           0.60
5       ADAM             2                 0.75           0.64
6       ADAM             3                 0.65           0.56
7       ADAM             4                 0.75           0.48
8    AdaGrad             1                 0.70           0.60
9    AdaGrad             2                 0.75           0.64
10   AdaGrad             3                 0.65           0.56
11   AdaGrad             4                 0.75           0.48
12   RMSprop             1                 0.70           0.60
13   RMSprop             2                 0.75           0.64
14   RMSprop             3                 0.65           0.56
15   RMSprop             4                 0.75           0.48
16     SGD+D             1                 0.70           0.60
17     SGD+D             2                 0.75           0.60
18     SGD+D             3                 0.75           0.56
```

SGD - Train and evaluate the model with 4 hidden Layer

```
# Melatih model dengan data pelatihan
model_four_hidden_layer_with_dropout.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model_four_hidden_layer_with_dropout.fit(X_train_scaled, y_train, epochs=100, batch_size=10, validation_data=(X_val_scaled, y
```

```
Epoch 80/100
8/8 [==============================] - 0s 11ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 1.0470 - val_accuracy: 0.7500
Epoch 81/100
8/8 [==============================] - 0s 11ms/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 1.0506 - val_accuracy: 0.7500
Epoch 82/100
8/8 [==============================] - 0s 11ms/step - loss: 0.0070 - accuracy: 1.0000 - val_loss: 1.0515 - val_accuracy: 0.8000
Epoch 83/100
8/8 [==============================] - 0s 11ms/step - loss: 0.0029 - accuracy: 1.0000 - val_loss: 1.0564 - val_accuracy: 0.8000
Epoch 84/100
8/8 [==============================] - 0s 13ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 1.0600 - val_accuracy: 0.8000
Epoch 85/100
8/8 [==============================] - 0s 14ms/step - loss: 7.7562e-04 - accuracy: 1.0000 - val_loss: 1.0611 - val_accuracy: 0.80
Epoch 86/100
8/8 [==============================] - 0s 13ms/step - loss: 0.0079 - accuracy: 1.0000 - val_loss: 1.0606 - val_accuracy: 0.8000
Epoch 87/100
8/8 [==============================] - 0s 14ms/step - loss: 0.0011 - accuracy: 1.0000 - val_loss: 1.0627 - val_accuracy: 0.8000
Epoch 88/100
8/8 [==============================] - 0s 11ms/step - loss: 0.0022 - accuracy: 1.0000 - val_loss: 1.0628 - val_accuracy: 0.8000
Epoch 89/100
8/8 [==============================] - 0s 11ms/step - loss: 0.0010 - accuracy: 1.0000 - val_loss: 1.0659 - val_accuracy: 0.8000
Epoch 90/100
8/8 [==============================] - 0s 11ms/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 1.0690 - val_accuracy: 0.8000
Epoch 91/100
8/8 [==============================] - 0s 13ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 1.0719 - val_accuracy: 0.8000
Epoch 92/100
8/8 [==============================] - 0s 11ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 1.0769 - val_accuracy: 0.8000
Epoch 93/100
8/8 [==============================] - 0s 13ms/step - loss: 0.0012 - accuracy: 1.0000 - val_loss: 1.0800 - val_accuracy: 0.8000
Epoch 94/100
8/8 [==============================] - 0s 11ms/step - loss: 0.0019 - accuracy: 1.0000 - val_loss: 1.0837 - val_accuracy: 0.8000
Epoch 95/100
8/8 [==============================] - 0s 11ms/step - loss: 0.0014 - accuracy: 1.0000 - val_loss: 1.0843 - val_accuracy: 0.8000
Epoch 96/100
8/8 [==============================] - 0s 12ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 1.0889 - val_accuracy: 0.8000
Epoch 97/100
8/8 [==============================] - 0s 12ms/step - loss: 0.0028 - accuracy: 1.0000 - val_loss: 1.0913 - val_accuracy: 0.8000
Epoch 98/100
8/8 [==============================] - 0s 14ms/step - loss: 0.0022 - accuracy: 1.0000 - val_loss: 1.0982 - val_accuracy: 0.8000
Epoch 99/100
8/8 [==============================] - 0s 11ms/step - loss: 0.0016 - accuracy: 1.0000 - val_loss: 1.1022 - val_accuracy: 0.8000
Epoch 100/100
8/8 [==============================] - 0s 11ms/step - loss: 0.0013 - accuracy: 1.0000 - val_loss: 1.1056 - val_accuracy: 0.8000
```

```
# Mengevaluasi kinerja model menggunakan data validasi
val_loss, val_acc = model_four_hidden_layer_with_dropout.evaluate(X_val_scaled, y_val)
print("Model 3 Hidden Layer SGD Optimizer Validation Accuracy:", val_acc)

# Pilih model terbaik berdasarkan kinerja validasi
best_model = model_four_hidden_layer_with_dropout  # Misalnya, model pertama dianggap sebagai model terbaik

# Evaluasi akhir menggunakan data pengujian
test_loss, test_acc = best_model.evaluate(X_test, y_test)
print("Best Model Test Accuracy:", test_acc)
```

```
1/1 [==============================] - 0s 154ms/step - loss: 1.1056 - accuracy: 0.8000
Model 3 Hidden Layer SGD Optimizer Validation Accuracy: 0.800000011920929
1/1 [==============================] - 0s 29ms/step - loss: 1.4271 - accuracy: 0.5200
Best Model Test Accuracy: 0.5199999809265137
```

```
update_results('SGD+D', 4, val_acc, test_acc)
print(results_df)
```

```
   Optimizer Hidden Layers  Validation Accuracy  Test Accuracy
0        SGD             1                 0.75           0.64
1        SGD             2                 0.75           0.64
2        SGD             3                 0.70           0.68
3        SGD             4                 0.70           0.64
4       ADAM             1                 0.70           0.60
5       ADAM             2                 0.75           0.64
6       ADAM             3                 0.65           0.56
7       ADAM             4                 0.75           0.48
8    AdaGrad             1                 0.70           0.60
9    AdaGrad             2                 0.75           0.64
10   AdaGrad             3                 0.65           0.56
11   AdaGrad             4                 0.75           0.48
12   RMSprop             1                 0.70           0.60
13   RMSprop             2                 0.75           0.64
14   RMSprop             3                 0.65           0.56
15   RMSprop             4                 0.75           0.48
16     SGD+D             1                 0.70           0.60
17     SGD+D             2                 0.75           0.60
18     SGD+D             3                 0.75           0.56
19     SGD+D             4                 0.80           0.52
```

3.C Hasil Penambahan Dropout 50%

```
from IPython.display import display

# Menampilkan DataFrame
display(results_df)
```

	Optimizer	Hidden Layers	Validation Accuracy	Test Accuracy
0	SGD	1	0.75	0.64