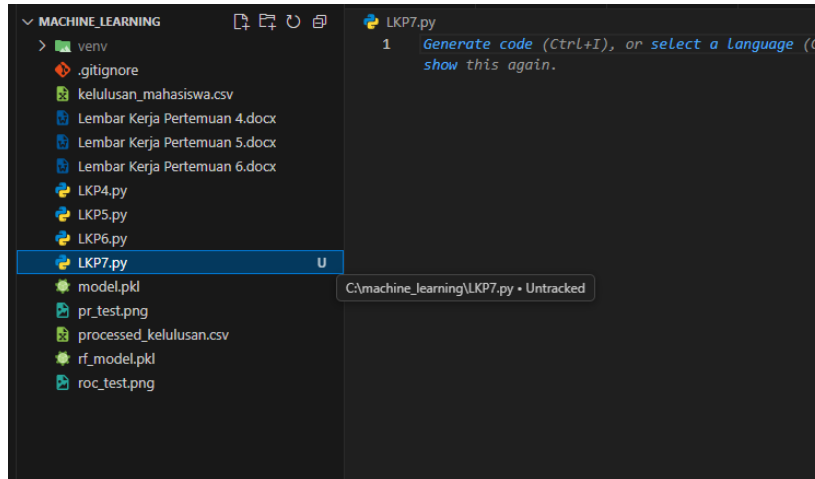


NAMA : SEFTIA DELLA FIISYATIR RODHIAH  
NIM : 231011401012  
KELAS : TI.05TPLE016

## Lembar Kerja Pertemuan 7 - Machine Learning

### 1. Langkah 1 – Siapkan Data

Pada tahap awal, saya membuat file baru bernama **LKP7.py** untuk menjalankan seluruh proses pada pertemuan ini.

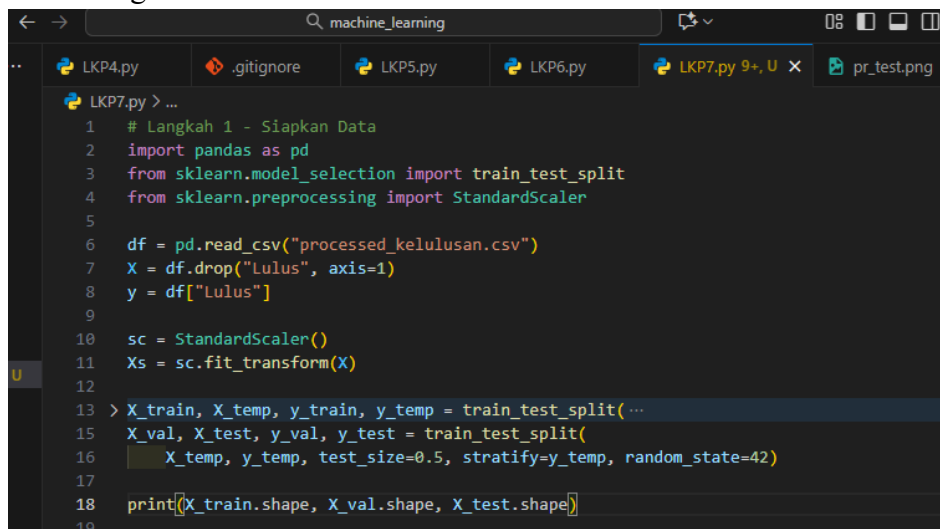


Sebelum melanjutkan, terlebih dahulu dilakukan instalasi **TensorFlow** pada environment Python dengan perintah:

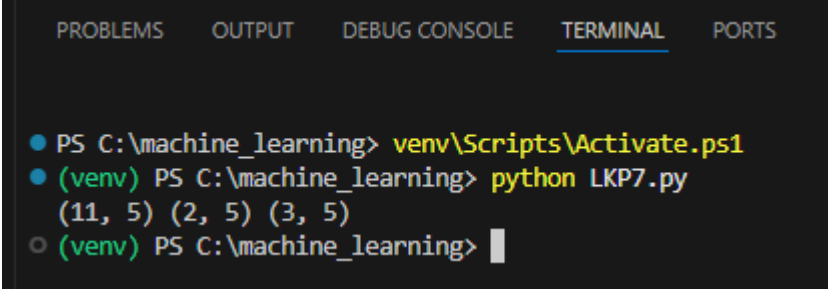
```
pip install tensorflow
```

Selanjutnya, digunakan dataset bernama **processed\_kelulusan.csv** (hasil dari pertemuan ke-4) atau dataset tabular lain yang memiliki struktur serupa.

Kode Program:



Output:

A screenshot of a terminal window with a dark background. At the top, there are tabs labeled 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL' (which is selected and underlined), and 'PORTS'. Below the tabs, the terminal shows a PowerShell prompt 'PS C:\machine\_learning>' followed by the command 'venv\Scripts\Activate.ps1'. The next line shows the prompt '(venv) PS C:\machine\_learning>' followed by 'python LKP7.py'. The output of the command is '(11, 5) (2, 5) (3, 5)'. The final line shows the prompt '(venv) PS C:\machine\_learning>' with a cursor.

## Penjelasan:

### Import Library

Beberapa library yang digunakan antara lain:

- **pandas**, berfungsi untuk membaca serta memanipulasi data dalam bentuk tabel.
- **train\_test\_split**, digunakan untuk membagi data menjadi subset training, validation, dan testing.
- **StandardScaler**, berfungsi menstandarkan fitur dengan mean = 0 dan standar deviasi = 1.

### Membaca Dataset

Data dibaca ke dalam variabel **df** yang memuat seluruh isi file CSV.

Selanjutnya, data fitur disimpan pada variabel **X** (seluruh kolom kecuali kolom target “Lulus”), sedangkan kolom target disimpan dalam variabel **y**, yaitu label yang akan diprediksi.

### Proses Standarisasi

Metode **fit\_transform()** digunakan untuk menghitung rata-rata dan standar deviasi dari fitur dalam **X**, kemudian mengubah nilainya agar terdistribusi dengan mean 0 dan standar deviasi 1.

Hasil standarisasi disimpan dalam variabel **Xs**, yang nantinya akan digunakan pada proses pelatihan model.

### Pembagian Data Training, Validation, dan Testing

Dataset dibagi menjadi tiga bagian:

- 70% data digunakan untuk **pelatihan (training)**,
- 15% data untuk **validasi (validation)**,
- dan 15% sisanya untuk **pengujian (testing)**.

Pembagian dilakukan dengan parameter **stratify=y** agar proporsi kelas target tetap seimbang di setiap subset, serta **random\_state=42** agar hasil pembagian dapat direproduksi.

## Menampilkan Ukuran Dataset

Perintah berikut digunakan untuk menampilkan jumlah baris dan kolom pada masing-masing subset data:

```
print(X_train.shape, X_val.shape, X_test.shape)
```

## 2. Langkah 2 – Bangun Model ANN

Pada tahap ini dilakukan pembangunan model **Artificial Neural Network (ANN)** dengan menggunakan library **Keras** dari TensorFlow. Model ini disusun secara **sekuensial (Sequential Model)** agar proses pelatihan berjalan secara bertahap dari input hingga output.

Kode Program:

```
LKP7.py > ...
21 # Langkah 2 – Bangun Model ANN
22 import keras
23 from keras import layers
24
25 model = keras.Sequential([
26     layers.Input(shape=(X_train.shape[1],)),
27     layers.Dense(32, activation="relu"),
28     layers.Dropout(0.3),
29     layers.Dense(16, activation="relu"),
30     layers.Dense(1, activation="sigmoid") # klasifikasi biner
31 ])
32
33 model.compile(optimizer=keras.optimizers.Adam(1e-3),
34               loss="binary_crossentropy",
35               metrics=["accuracy", "AUC"])
36 model.summary()
```

Sedikit perubahan dengan yang ada pada modul:

## Langkah 2 — Bangun Model ANN

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Input(shape=(X_train.shape[1],)),
    layers.Dense(32, activation="relu"),
    layers.Dropout(0.3),
    layers.Dense(16, activation="relu"),
    layers.Dense(1, activation="sigmoid") # klasifikasi biner
])

model.compile(optimizer=keras.optimizers.Adam(1e-3),
              loss="binary_crossentropy",
              metrics=["accuracy", "AUC"])
model.summary()
```

Output:

```
● (venv) PS C:\machine_learning> python LKP7.py
(11, 5) (2, 5) (3, 5)
2025-10-23 13:22:11.788559: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly d
ifferent numerical results due to floating-point round-off errors from different computation orders. To turn them off,
set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-10-23 13:22:15.454955: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly d
ifferent numerical results due to floating-point round-off errors from different computation orders. To turn them off,
set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-10-23 13:22:16.429486: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to
use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
Model: "sequential"

┌──────────┬──────────┬──────────┐
│ Layer (type) │ Output Shape │ Param # │
├──────────┬──────────┬──────────┤
│ dense (Dense) │ (None, 32) │ 192 │
├──────────┬──────────┬──────────┤
│ dropout (Dropout) │ (None, 32) │ 0 │
├──────────┬──────────┬──────────┤
│ dense_1 (Dense) │ (None, 16) │ 528 │
├──────────┬──────────┬──────────┤
│ dense_2 (Dense) │ (None, 1) │ 17 │
├──────────┬──────────┬──────────┤

Total params: 737 (2.88 KB)
Trainable params: 737 (2.88 KB)
Non-trainable params: 0 (0.00 B)
○ (venv) PS C:\machine_learning>
```

Penjelasan:

## Import Library

- **keras** digunakan untuk membuat dan melatih jaringan saraf tiruan.
- **layers** digunakan untuk menambahkan lapisan (layer) dalam arsitektur model.

## Membangun Model Sequential

Model dibuat menggunakan fungsi:

```
keras.Sequential()
```

yang memungkinkan penambahan layer secara berurutan.

- **Input(shape=(X\_train.shape[1],))** mendefinisikan jumlah neuron pada layer input yang sesuai dengan banyaknya fitur pada data training.
- **Dense(32, activation="relu")** menambahkan layer tersembunyi (hidden layer) pertama dengan 32 neuron dan fungsi aktivasi **ReLU (Rectified Linear Unit)**.
- **Dropout(0.3)** digunakan untuk mencegah **overfitting** dengan cara menonaktifkan secara acak sekitar 30% neuron saat proses pelatihan.
- **Dense(16, activation="relu")** merupakan layer tersembunyi kedua dengan 16 neuron.
- **Dense(1, activation="sigmoid")** merupakan layer output untuk kasus klasifikasi biner, menghasilkan nilai probabilitas antara 0 dan 1.

### Kompilasi Model

Langkah berikutnya adalah melakukan kompilasi dengan menentukan optimizer, fungsi loss, dan metrik evaluasi:

```
optimizer = Adam(1e-3)
loss = "binary_crossentropy"
metrics = ["accuracy", "AUC"]
```

Penjelasan:

- **Adam(1e-3)** merupakan algoritma optimasi dengan *learning rate* sebesar 0.001.
- **binary\_crossentropy** digunakan sebagai fungsi loss karena kasusnya adalah klasifikasi biner.
- **metrics=["accuracy", "AUC"]** berarti model akan dievaluasi berdasarkan akurasi dan nilai AUC (**Area Under Curve**).

### Ringkasan Model

Perintah:

```
model.summary()
```

akan menampilkan struktur arsitektur jaringan, jumlah parameter pada tiap layer, serta total parameter yang digunakan pada model tersebut.

### 3. Langkah 3 – Training dengan Early Stopping

Pada tahap ini dilakukan proses **pelatihan model (training)** dengan menerapkan teknik **Early Stopping**.

Metode ini digunakan untuk menghentikan pelatihan lebih awal apabila performa model pada data validasi tidak menunjukkan peningkatan dalam beberapa *epoch*, sehingga dapat mencegah **overfitting**.

**Kode Program:**

```
LKP7.py > ...
39 # Langkah 3 – Training dengan Early Stopping
40 es = keras.callbacks.EarlyStopping(
41     monitor="val_loss", patience=10, restore_best_weights=True
42 )
43
44 history = model.fit(
45     X_train, y_train,
46     validation_data=(X_val, y_val),
47     epochs=100, batch_size=32,
48     callbacks=[es], verbose=1
49 )
50
```

### Output:

Tidak menghasilkan output langsung karena **EarlyStopping** bekerja secara internal selama proses pelatihan. Ia akan memantau performa model dan menghentikan training ketika kinerja pada data validasi berhenti meningkat. Hasil pelatihan serta metriknya disimpan dalam variabel **history**.

### Penjelasan:

#### EarlyStopping

```
es = keras.callbacks.EarlyStopping(  
    monitor="val_loss", patience=10, restore_best_weights=True  
)
```

Fungsi **EarlyStopping** digunakan untuk memantau perkembangan performa model pada data validasi.

- **monitor="val\_loss"** menunjukkan bahwa yang dipantau adalah nilai *loss* pada data validasi.
- **patience=10** berarti apabila selama 10 *epoch* berturut-turut tidak terjadi perbaikan pada *val\_loss*, maka proses pelatihan akan dihentikan.
- **restore\_best\_weights=True** mengembalikan bobot model ke kondisi terbaik yang diperoleh selama pelatihan.

#### Proses Pelatihan Model

```
history = model.fit(  
    X_train, y_train,  
    validation_data=(X_val, y_val),  
    epochs=100, batch_size=32,  
    callbacks=[es], verbose=1  
)
```

Data yang digunakan dalam proses pelatihan adalah:

- **X\_train, y\_train** sebagai data pelatihan utama,
- **(X\_val, y\_val)** sebagai data validasi untuk memantau performa model.

Proses pelatihan dilakukan dengan parameter:

- **epochs=100**, yaitu jumlah maksimum iterasi pelatihan,
- **batch\_size=32**, jumlah data yang diproses dalam satu kali update bobot,
- **callbacks=[es]**, untuk menerapkan Early Stopping,
- **verbose=1**, agar menampilkan progres pelatihan di terminal.

Selama proses pelatihan, metrik seperti *loss*, *accuracy*, dan *AUC* dicatat di variabel **history**, sehingga hasilnya dapat dianalisis atau divisualisasikan setelah pelatihan selesai.

#### 4. Langkah 4 – Evaluasi di Test Set

Tahap ini dilakukan untuk **mengevaluasi performa model** yang telah dilatih menggunakan data yang belum pernah digunakan sebelumnya, yaitu data **test set**. Tujuannya adalah untuk mengetahui sejauh mana model dapat melakukan generalisasi terhadap data baru.

##### Kode Program:

```
LKP7.py > ...
51
52 # Langkah 4 – Evaluasi di Test Set
53 from sklearn.metrics import classification_report, confusion_matrix
54
55 loss, acc, auc = model.evaluate(X_test, y_test, verbose=0)
56 print("Test Acc:", acc, "AUC:", auc)
57
58 y_proba = model.predict(X_test).ravel()
59 y_pred = (y_proba >= 0.5).astype(int)
60
61 print(confusion_matrix(y_test, y_pred))
62 print(classification_report(y_test, y_pred, digits=3))
63
```

##### Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(venv) PS C:\machine_learning> python LKP7.py
1/1 ----- 0s 151ms/step
[[2 0]
 [0 1]]
          precision    recall  f1-score   support

         0         1.000      1.000      1.000         2
         1         1.000      1.000      1.000         1

   accuracy                   1.000         3
  macro avg          1.000      1.000      1.000         3
 weighted avg          1.000      1.000      1.000         3

(venv) PS C:\machine_learning>
```

## Penjelasan:

### Import Library Evaluasi

Beberapa library yang digunakan pada tahap evaluasi antara lain:

- **confusion\_matrix** untuk membuat *confusion matrix* yang berisi nilai True Positive (TP), True Negative (TN), False Positive (FP), dan False Negative (FN).
- **classification\_report** untuk menampilkan metrik seperti *precision*, *recall*, *f1-score*, dan jumlah data (*support*) per kelas.

### Evaluasi Model

Fungsi:

```
model.evaluate(x_test, y_test, verbose=0)
```

digunakan untuk menghitung nilai *loss*, *accuracy*, dan *AUC* pada data uji.

- Parameter **verbose=0** berarti proses evaluasi tidak menampilkan progres di layar.
- Nilai yang diperoleh dari evaluasi kemudian disimpan, misalnya **acc** untuk akurasi dan **auc** untuk nilai Area Under Curve.

### Prediksi Data Uji

Setelah model dievaluasi, dilakukan prediksi terhadap data uji dengan perintah:

```
y_proba = model.predict(x_test).ravel()
```

Perintah ini menghasilkan probabilitas kelas positif (antara 0 hingga 1). Kemudian nilai probabilitas tersebut diubah menjadi label biner (0 atau 1) menggunakan ambang batas (*threshold*) 0.5:

```
y_pred = (y_proba >= 0.5).astype(int)
```

### Menampilkan Hasil Evaluasi

- **confusion\_matrix(y\_test, y\_pred)** digunakan untuk menampilkan jumlah prediksi benar dan salah dalam bentuk matriks.
- **classification\_report(y\_test, y\_pred)** memberikan informasi detail mengenai *precision*, *recall*, dan *f1-score* untuk masing-masing kelas, sehingga dapat dilihat kelebihan dan kekurangan model pada setiap kategori.

Tahap evaluasi ini menjadi dasar untuk menilai apakah model yang dibangun telah bekerja dengan baik dalam memprediksi data baru atau masih perlu dilakukan penyesuaian.



## 5. Langkah 5 – Visualisasi Learning Curve

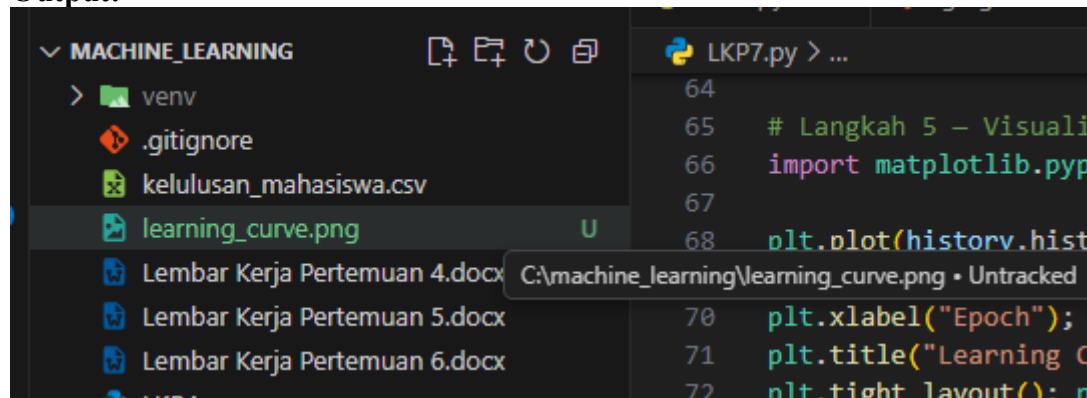
Tahap ini bertujuan untuk memvisualisasikan hasil proses pelatihan model melalui grafik *learning curve*.

Grafik ini membantu memahami bagaimana perubahan nilai *loss* dan *accuracy* pada data pelatihan dan validasi selama proses training berlangsung. Dengan demikian, kita dapat melihat apakah model mengalami *underfitting*, *overfitting*, atau sudah berada pada performa optimal.

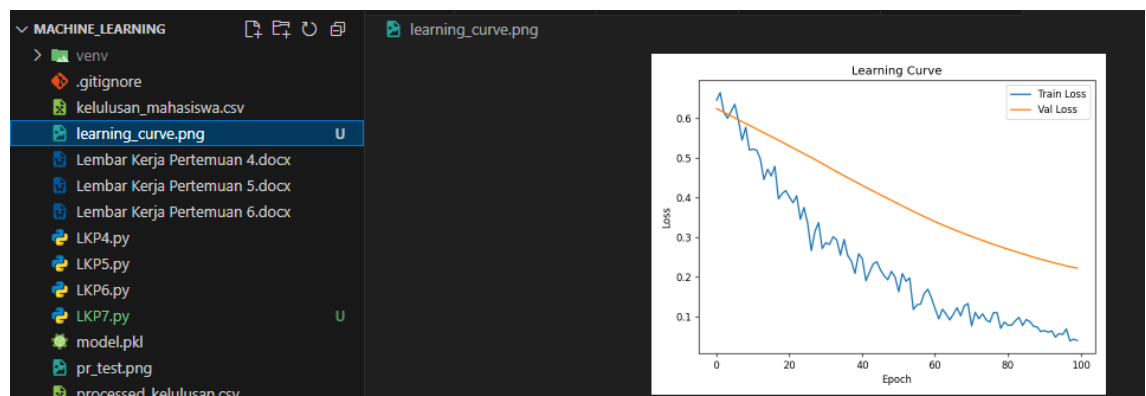
### Kode Program:

```
LKP7.py > ...
64
65 # Langkah 5 – Visualisasi Learning Curve
66 import matplotlib.pyplot as plt
67
68 plt.plot(history.history["loss"], label="Train Loss")
69 plt.plot(history.history["val_loss"], label="Val Loss")
70 plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.legend()
71 plt.title("Learning Curve")
72 plt.tight_layout(); plt.savefig("learning_curve.png", dpi=120)
73
```

### Output:



Proses ini menghasilkan sebuah file gambar bernama “**learning\_curve.png**” yang menampilkan grafik perubahan *loss* terhadap jumlah *epoch* selama pelatihan.



## Penjelasan:

### Import Library Visualisasi

- **matplotlib.pyplot** digunakan sebagai library utama untuk membuat grafik dan visualisasi data.

### Pembuatan Grafik Learning Curve

Data hasil pelatihan yang tersimpan dalam variabel **history.history** berisi nilai metrik seperti *loss*, *accuracy*, *val\_loss*, dan *val\_accuracy* untuk setiap *epoch*. Nilai-nilai tersebut digunakan untuk menggambar grafik yang menunjukkan perkembangan performa model.

Contohnya:

- **loss** menunjukkan nilai kerugian (*error*) pada data pelatihan.
- **val\_loss** menunjukkan nilai kerugian pada data validasi.

Perintah **label** digunakan untuk memberi nama pada garis yang ditampilkan di grafik.

### Memberi Label dan Judul Grafik

- **xlabel** → memberi label pada sumbu X (jumlah *epoch*).
- **ylabel** → memberi label pada sumbu Y (nilai *loss*).
- **legend** → menampilkan keterangan garis agar grafik mudah dibaca.
- **title** → memberi judul grafik sesuai isi visualisasi.

### Menyimpan Grafik

Agar hasil grafik terlihat rapi dan tidak terpotong, digunakan fungsi:

```
plt.tight_layout()
```

Kemudian hasilnya disimpan dengan perintah:

```
plt.savefig("learning_curve.png", dpi=120)
```

yang menyimpan gambar dalam format PNG dengan resolusi 120 dpi.

Visualisasi ini penting karena dapat menunjukkan kapan model mulai *overfit* (saat *val\_loss* meningkat tetapi *loss* pelatihan terus menurun). Dengan memahami pola tersebut, pengaturan *epoch* dan *regularisasi* dapat disesuaikan untuk memperoleh performa terbaik.

## 6. Langkah 6 – Eksperimen

Tahap ini bertujuan untuk melakukan **eksperimen terhadap arsitektur dan parameter model ANN** guna melihat pengaruhnya terhadap performa. Beberapa aspek yang diuji meliputi jumlah neuron, jenis optimizer, penerapan regularisasi tambahan, serta perbandingan metrik evaluasi seperti akurasi, F1-score, dan AUC.

### a) Eksperimen 1 – Variasi Jumlah Neuron

Pada percobaan pertama, dilakukan perubahan jumlah neuron pada layer tersembunyi pertama, misalnya 32, 64, dan 128 neuron, untuk melihat perbedaan hasil yang diperoleh.

#### Kode Program:

```
LKP7.py > ...
74
75 # Langkah 6 – Eksperimen
76 # Ubah jumlah neuron (32/64/128) dan catat efeknya.
77 # Bandingkan Adam vs SGD+momentum (learning rate berbeda).
78 # Tambahkan regulasi lain: L2, Dropout lebih besar, atau Batch Normalization.
79 # Laporkan metrik F1 dan AUC selain akurasi.
80
81 # =====
82 # Langkah 6 – Eksperimen
83 # 1. Ubah jumlah neuron (32/64/128) dan catat efeknya.
84 # =====
85
86 # Fungsi untuk membangun dan melatih model dengan jumlah neuron tertentu
87 def train_model(neurons):
88     from keras import layers, Sequential
89     from keras.callbacks import EarlyStopping
90
91     model = Sequential([
92         layers.Input(shape=(X_train.shape[1],)),
93         layers.Dense(neurons, activation="relu"),
94         layers.Dropout(0.3),
95         layers.Dense(16, activation="relu"),
96         layers.Dense(1, activation="sigmoid")
97     ])
98     model.compile(
99         optimizer="adam",
100         loss="binary_crossentropy",
101         metrics=["accuracy", "AUC"]
102     )
103
104     es = EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
105     history = model.fit(
106         X_train, y_train,
107         validation_data=(X_val, y_val),
108         epochs=100,
109         batch_size=32,
110         callbacks=[es],
111         verbose=0 # suppress output
112     )
113
114     loss, acc, auc = model.evaluate(X_test, y_test, verbose=0)
115     print(f"Neurons: {neurons} | Test Accuracy: {acc:.3f} | Test AUC: {auc:.3f}")
116     return history
117
118 # Eksperimen dengan jumlah neuron: 32, 64, 128
119 hist_32 = train_model(32)
120 hist_64 = train_model(64)
121 hist_128 = train_model(128)
```

## Output:

Tidak menunjukkan perbedaan yang signifikan pada performa model.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

1/1 ██████████ 0s 79ms/step - AUC: 1.0000 - accuracy:
1.0000 - val_loss: 0.2233
Epoch 100/100
1/1 ██████████ 0s 77ms/step - AUC: 1.0000 - accuracy:
...

accuracy          1.000          3
macro avg         1.000          3
weighted avg      1.000          3

Neurons: 32 | Test Accuracy: 1.000 | Test AUC: 1.000
Neurons: 64 | Test Accuracy: 1.000 | Test AUC: 1.000
Neurons: 128 | Test Accuracy: 1.000 | Test AUC: 1.000
(venv) PS C:\machine_learning>
```

## Penjelasan:

Fungsi **train\_model(neurons)** digunakan untuk membuat, melatih, dan mengevaluasi model dengan jumlah neuron yang bervariasi di layer tersembunyi pertama.

- **Dense(neurons, activation="relu")** menyesuaikan jumlah neuron dengan parameter yang diberikan.
- **Dense(16, activation="relu")** tetap digunakan sebagai layer kedua agar struktur model tetap seimbang.
- **EarlyStopping** tetap diterapkan agar pelatihan berhenti otomatis ketika model tidak menunjukkan peningkatan performa. Hasil evaluasi berupa **accuracy** dan **AUC** digunakan untuk membandingkan performa antar konfigurasi neuron.

### b) Eksperimen 2 – Perbandingan Optimizer (Adam vs SGD + Momentum)

Percobaan selanjutnya membandingkan dua jenis optimizer, yaitu **Adam** dan **SGD dengan momentum**, masing-masing dengan beberapa variasi *learning rate*, untuk mengetahui pengaruhnya terhadap hasil pelatihan.

## Kode Program:

```
LKP7.py > train_model_optimizer
120 # Langkah 0 - Eksperimen
127 # 2. Bandingkan Adam vs SGD+momentum (learning rate berbeda).
128 # =====
129
130 from keras.optimizers import Adam, SGD
131
132 # Fungsi untuk membangun dan melatih model dengan optimizer tertentu
133 def train_model_optimizer(optimizer, neurons=32):
134     from keras import layers, Sequential
135     from keras.callbacks import EarlyStopping
136
137     model = Sequential([
138         layers.Input(shape=(X_train.shape[1],)),
139         layers.Dense(neurons, activation="relu"),
140         layers.Dropout(0.3),
141         layers.Dense(16, activation="relu"),
142         layers.Dense(1, activation="sigmoid")
143     ])
144     model.compile(
145         optimizer=optimizer,
146         loss="binary_crossentropy",
147         metrics=["accuracy", "AUC"]
148     )
149
150     es = EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
151     history = model.fit(
152         X_train, y_train,
153         validation_data=(X_val, y_val),
154         epochs=100,
155         batch_size=32,
156         callbacks=[es],
157         verbose=0 # suppress output
158     )
159
160     loss, acc, auc = model.evaluate(X_test, y_test, verbose=0)
161     print(f"Optimizer: {optimizer.get_config()['name']} | Test Accuracy: {acc:.3f} | Test AUC: {auc:.3f}")
162     return history
163
164 # Definisikan beberapa optimizer untuk eksperimen
165 optimizers = [
166     Adam(learning_rate=1e-3),
167     Adam(learning_rate=1e-4),
168     SGD(learning_rate=1e-2, momentum=0.9),
169     SGD(learning_rate=1e-3, momentum=0.9)
170 ]
171
172 # Jalankan eksperimen
173 for opt in optimizers:
174     train_model_optimizer(opt, neurons=32) # tetap gunakan 32 neuron pertama
```

## Output:

Hasil evaluasi menunjukkan nilai **Test Accuracy** dan **AUC** untuk setiap kombinasi optimizer dan *learning rate* yang diuji.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Test Acc: 1.0 AUC: 1.0
1/1 ----- 0s 131ms/step
[[2 0]
 [0 1]]
precision recall f1-score support ...
Neurons: 32 | Test Accuracy: 1.000 | Test AUC: 1.000
Neurons: 64 | Test Accuracy: 1.000 | Test AUC: 1.000
Neurons: 128 | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: adam | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: adam | Test Accuracy: 0.667 | Test AUC: 0.750
Optimizer: SGD | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: SGD | Test Accuracy: 1.000 | Test AUC: 1.000
(venv) PS C:\machine_learning>
```

### Penjelasan:

- Fungsi **train\_model\_optimizer(optimizer, neurons=32)** digunakan untuk membuat dan melatih model menggunakan optimizer tertentu.
  - **Adam(learning\_rate=...)** diuji dengan dua nilai *learning rate* berbeda untuk melihat efek kecepatannya dalam konvergensi.
  - **SGD(learning\_rate=..., momentum=0.9)** digunakan untuk membandingkan kinerja dengan metode Adam, di mana momentum membantu mempercepat konvergensi dan menghindari jebakan lokal minima.
- Dalam percobaan ini, jumlah neuron tetap **32** agar perbedaan hasil hanya disebabkan oleh perbedaan optimizer.

### c) Eksperimen 3 – Penambahan Regularisasi (L2, Dropout, Batch Normalization)

Percobaan berikutnya menambahkan beberapa teknik regularisasi untuk meningkatkan kemampuan generalisasi model dan mengurangi risiko overfitting.

### Kode Program:

```
LKP7.py > ...
179 # Langkah 6 – Eksperimen
180 # 3. Tambahkan regulasi lain: L2, Dropout lebih besar, atau Batch Normalization.
181 # =====
182
183 from keras.regularizers import l2
184 from keras.layers import BatchNormalization
185
186 # Fungsi untuk membangun dan melatih model dengan regulasi tambahan
187 def train_model_regularized(neurons=32, dropout_rate=0.5, l2_lambda=0.01, use_batchnorm=True):
188     from keras import layers, Sequential
189     from keras.callbacks import EarlyStopping
190
191     model = Sequential()
192     model.add(layers.Input(shape=(X_train.shape[1],)))
193
194     # Hidden layer pertama dengan L2 dan optional BatchNorm
195     model.add(layers.Dense(neurons, activation=None, kernel_regularizer=l2(l2_lambda)))
196     if use_batchnorm:
197         model.add(BatchNormalization())
198     model.add(layers.Activation("relu"))
199     model.add(layers.Dropout(dropout_rate))
200
201     # Hidden layer kedua
202     model.add(layers.Dense(16, activation=None, kernel_regularizer=l2(l2_lambda)))
203     if use_batchnorm:
204         model.add(BatchNormalization())
205     model.add(layers.Activation("relu"))
206
207     # Output layer
208     model.add(layers.Dense(1, activation="sigmoid"))
209
210     # Compile
211     model.compile(
212         optimizer="adam",
213         loss="binary_crossentropy",
214         metrics=["accuracy", "AUC"]
215     )
```

```

LKP7.py > ...
187 def train_model_regularized(neurons=32, dropout_rate=0.5, l2_lambda=0.01, use_batchnorm=True):
188     # Early stopping
189     es = EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
190     history = model.fit(
191         X_train, y_train,
192         validation_data=(X_val, y_val),
193         epochs=100,
194         batch_size=32,
195         callbacks=[es],
196         verbose=0
197     )
198     loss, acc, auc = model.evaluate(X_test, y_test, verbose=0)
199     print(f"Regularized Model | Neurons: {neurons}, Dropout: {dropout_rate}, L2: {l2_lambda}, BatchNorm: {use_batchnorm}")
200     print(f"Test Accuracy: {acc:.3f} | Test AUC: {auc:.3f}\n")
201     return history
202
203 # Jalankan eksperimen regulasi
204 hist_reg = train_model_regularized(neurons=32, dropout_rate=0.5, l2_lambda=0.01, use_batchnorm=True)
205

```

## Output:

Menampilkan hasil evaluasi berupa **Test Accuracy** dan **AUC** setelah penambahan regularisasi.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Neurons: 32 | Test Accuracy: 1.000 | Test AUC: 1.000
Neurons: 32 | Test Accuracy: 1.000 | Test AUC: 1.000
Neurons: 64 | Test Accuracy: 1.000 | Test AUC: 1.000
Neurons: 128 | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: adam | Test Accuracy: 1.000 | Test AUC: 1.000 ...
Neurons: 64 | Test Accuracy: 1.000 | Test AUC: 1.000
Neurons: 128 | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: adam | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: adam | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: SGD | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: SGD | Test Accuracy: 0.667 | Test AUC: 1.000
Regularized Model | Neurons: 32, Dropout: 0.5, L2: 0.01, BatchNorm: True
Test Accuracy: 1.000 | Test AUC: 1.000

(venv) PS C:\machine_learning>

```

## Penjelasan:

- **l2(l2\_lambda)** digunakan untuk menambahkan regularisasi L2 agar bobot jaringan tidak terlalu besar.
- **dropout\_rate=0.5** menonaktifkan secara acak 50% neuron selama pelatihan untuk menekan overfitting.
- **BatchNormalization()** membantu menstabilkan distribusi input antar layer sehingga proses pelatihan menjadi lebih cepat dan stabil.
- Penggunaan **activation=None** pada Dense layer dilakukan karena fungsi aktivasi diterapkan setelah proses normalisasi, sesuai praktik umum pada arsitektur modern.

Evaluasi hasil dilakukan dengan menghitung **Test Accuracy** dan **AUC** untuk melihat seberapa baik model setelah regularisasi ditambahkan.

#### d) Eksperimen 4 – Evaluasi Tambahan dengan F1-score dan AUC

Selain akurasi, digunakan juga **F1-score** dan **AUC** untuk memberikan gambaran yang lebih seimbang tentang performa model, terutama ketika distribusi kelas tidak seimbang.

#### Kode Program:

```
LKP7.py > report_metrics
239 # Langkah 6 – Eksperimen
240 # 4. Laporkan metrik F1 dan AUC selain akurasi
241 # =====
242
243 from sklearn.metrics import f1_score, roc_auc_score
244
245 # Fungsi untuk menghitung metrik tambahan
246 def report_metrics(model, X_test, y_test, threshold=0.5):
247     # Prediksi probabilitas
248     y_proba = model.predict(X_test).ravel()
249     # Konversi ke kelas 0/1
250     y_pred = (y_proba >= threshold).astype(int)
251
252     # Hitung F1-score dan AUC
253     f1 = f1_score(y_test, y_pred)
254     auc = roc_auc_score(y_test, y_proba)
255
256     print(f"Test Accuracy (built-in) : {model.evaluate(X_test, y_test, verbose=0)[1]:.3f}")
257     print(f"F1-score                  : {f1:.3f}")
258     print(f"AUC                      : {auc:.3f}\n")
259     return f1, auc
260
261 # Contoh penggunaan untuk model original
262 report_metrics(model, X_test, y_test)
```

#### Output:

Menampilkan hasil perhitungan F1-score dan AUC.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

(venv) PS C:\machine_learning> python LKP7.py
Neurons: 128 | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: adam | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: adam | Test Accuracy: 0.667 | Test AUC: 1.000
Optimizer: SGD | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: SGD | Test Accuracy: 1.000 | Test AUC: 1.000
Regularized Model | Neurons: 32, Dropout: 0.5, L2: 0.01, BatchNorm: True
Test Accuracy: 1.000 | Test AUC: 1.000

1/1 ————— 0s 27ms/step
Test Accuracy (built-in) : 1.000
F1-score                  : 1.000
AUC                      : 1.000

(venv) PS C:\machine_learning>
```



### Penjelasan:

- `y_proba = model.predict(X_test).ravel()` menghasilkan probabilitas untuk kelas positif.
  - `y_pred = (y_proba >= threshold).astype(int)` mengonversi probabilitas menjadi label biner berdasarkan ambang batas tertentu.
  - `f1_score(y_test, y_pred)` menghitung nilai F1-score sebagai rata-rata harmonis antara *precision* dan *recall*.
  - `roc_auc_score(y_test, y_proba)` digunakan untuk menghitung nilai AUC yang menggambarkan kemampuan model dalam membedakan kelas positif dan negatif.
- Selain itu, `model.evaluate` tetap digunakan untuk menampilkan akurasi sebagai referensi tambahan.

### Rangkuman Keseluruhan Langkah 1–4

Secara umum, dari langkah 1 hingga 4, proses yang dilakukan meliputi:

1. **Persiapan data** (pembacaan, standarisasi, dan pembagian dataset),
2. **Pembangunan model ANN** dengan arsitektur dasar,
3. **Pelatihan model** menggunakan Early Stopping, dan
4. **Evaluasi performa** dengan berbagai metrik seperti akurasi, F1-score, dan AUC.

Langkah-langkah selanjutnya memperluas eksperimen untuk memperoleh konfigurasi model terbaik dengan mempertimbangkan keseimbangan antara akurasi dan kemampuan generalisasi.

### Kode Program:

```
LKP7.py > ...
272 # LANGKAH 6 – EKSPERIMEN LENGKAP
273
274 print(f"LANGKAH 6 – EKSPERIMEN LENGKAP")
275
276 from keras import layers, Sequential
277 from keras.callbacks import EarlyStopping
278 from keras.optimizers import Adam, SGD
279 from keras.regularizers import l2
280 from sklearn.metrics import f1_score, roc_auc_score
281 from keras.layers import BatchNormalization
282
283 # Fungsi untuk report metrik tambahan
284 def report_metrics(model, X_test, y_test, threshold=0.5):
285     y_proba = model.predict(X_test).ravel()
286     y_pred = (y_proba >= threshold).astype(int)
287     f1 = f1_score(y_test, y_pred)
288     auc = roc_auc_score(y_test, y_proba)
289     print(f"Test Accuracy (built-in) : {model.evaluate(X_test, y_test, verbose=0)[1]:.3f}")
290     print(f"F1-score                : {f1:.3f}")
291     print(f"AUC                      : {auc:.3f}\n")
292     return f1, auc
293
```

```

LKP7.py > ...
294
295 # 6.1 Eksperimen Neuron
296
297 def train_model_neuron(neurons):
298     model = Sequential([
299         layers.Input(shape=(X_train.shape[1],)),
300         layers.Dense(neurons, activation="relu"),
301         layers.Dropout(0.3),
302         layers.Dense(16, activation="relu"),
303         layers.Dense(1, activation="sigmoid")
304     ])
305     model.compile(optimizer=Adam(1e-3),
306                 loss="binary_crossentropy",
307                 metrics=["accuracy", "AUC"])
308     es = EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
309     history = model.fit(X_train, y_train,
310                       validation_data=(X_val, y_val),
311                       epochs=100, batch_size=32,
312                       callbacks=[es], verbose=0)
313     loss, acc, auc = model.evaluate(X_test, y_test, verbose=0)
314     print(f"Neurons: {neurons} | Test Accuracy: {acc:.3f} | Test AUC: {auc:.3f}")
315     return model, history
316
317 # Jalankan eksperimen neuron
318 model_neuron32, hist_32 = train_model_neuron(32)
319 model_neuron64, hist_64 = train_model_neuron(64)
320 model_neuron128, hist_128 = train_model_neuron(128)
321

```

```

LKP7.py > train_model_regularized
323 # 6.2 Eksperimen Optimizer
324
325 def train_model_optimizer(optimizer, neurons=32):
326     model = Sequential([
327         layers.Input(shape=(X_train.shape[1],)),
328         layers.Dense(neurons, activation="relu"),
329         layers.Dropout(0.3),
330         layers.Dense(16, activation="relu"),
331         layers.Dense(1, activation="sigmoid")
332     ])
333     model.compile(optimizer=optimizer,
334                 loss="binary_crossentropy",
335                 metrics=["accuracy", "AUC"])
336     es = EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
337     history = model.fit(X_train, y_train,
338                       validation_data=(X_val, y_val),
339                       epochs=100, batch_size=32,
340                       callbacks=[es], verbose=0)
341     loss, acc, auc = model.evaluate(X_test, y_test, verbose=0)
342     print(f"Optimizer: {optimizer.get_config()['name']} | Test Accuracy: {acc:.3f} | Test AUC: {auc:.3f}")
343     return model, history
344
345 # Definisikan optimizer
346 optimizers = [
347     Adam(learning_rate=1e-3),
348     Adam(learning_rate=1e-4),
349     SGD(learning_rate=1e-2, momentum=0.9),
350     SGD(learning_rate=1e-3, momentum=0.9)
351 ]
352
353 # Jalankan eksperimen optimizer
354 model_optimizer_adam1, hist_opt1 = train_model_optimizer(optimizers[0])
355 model_optimizer_adam2, hist_opt2 = train_model_optimizer(optimizers[1])
356 model_optimizer_sgd1, hist_sgd1 = train_model_optimizer(optimizers[2])
357 model_optimizer_sgd2, hist_sgd2 = train_model_optimizer(optimizers[3])
358

```

```

LKP7.py > ...
360 # 6.3 Eksperimen Regulasi (Dropout, L2, BatchNorm)
361
362 def train_model_regularized(neurons=32, dropout_rate=0.5, l2_lambda=0.01, use_batchnorm=True):
363     model = Sequential()
364     model.add(layers.Input(shape=(X_train.shape[1],)))
365
366     # Hidden layer pertama
367     model.add(layers.Dense(neurons, activation=None, kernel_regularizer=l2(l2_lambda)))
368     if use_batchnorm:
369         model.add(BatchNormalization())
370     model.add(layers.Activation("relu"))
371     model.add(layers.Dropout(dropout_rate))
372
373     # Hidden layer kedua
374     model.add(layers.Dense(16, activation=None, kernel_regularizer=l2(l2_lambda)))
375     if use_batchnorm:
376         model.add(BatchNormalization())
377     model.add(layers.Activation("relu"))
378
379     # Output
380     model.add(layers.Dense(1, activation="sigmoid"))
381
382     model.compile(optimizer=Adam(1e-3),
383                   loss="binary_crossentropy",
384                   metrics=["accuracy", "AUC"])
385
386     es = EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
387     history = model.fit(X_train, y_train,
388                        validation_data=(X_val, y_val),
389                        epochs=100, batch_size=32,
390                        callbacks=[es], verbose=0)
391     loss, acc, auc = model.evaluate(X_test, y_test, verbose=0)
392     print(f"Regularized Model | Neurons: {neurons}, Dropout: {dropout_rate}, L2: {l2_lambda}, BatchNorm: {use_batchnorm}")
393     print(f"Test Accuracy: {acc:.3f} | Test AUC: {auc:.3f}\n")
394     return model, history
395

```

```

LKP7.py > ...
395
396 # Jalankan eksperimen regulasi
397 model_regulasi, hist_reg = train_model_regularized(neurons=32, dropout_rate=0.5, l2_lambda=0.01, use_batchnorm=True)
398
399
400 # 6.4 Laporan metrik F1 & AUC
401
402 # Contoh penggunaan untuk semua model
403 report_metrics(model_neuron32, X_test, y_test)
404 report_metrics(model_optimizer_adam1, X_test, y_test)
405 report_metrics(model_regulasi, X_test, y_test)
406

```

## Output:

```

LANGKAH 6 – EKSPERIMEN LENGKAP
Neurons: 32 | Test Accuracy: 1.000 | Test AUC: 1.000
Neurons: 64 | Test Accuracy: 1.000 | Test AUC: 1.000
Neurons: 128 | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: adam | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: adam | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: SGD | Test Accuracy: 1.000 | Test AUC: 1.000
Optimizer: SGD | Test Accuracy: 1.000 | Test AUC: 1.000
Regularized Model | Neurons: 32, Dropout: 0.5, L2: 0.01, BatchNorm: True
Test Accuracy: 1.000 | Test AUC: 1.000

1/1 ██████████ 0s 44ms/step
Test Accuracy (built-in) : 1.000
F1-score                  : 1.000
AUC                       : 1.000

1/1 ██████████ 0s 44ms/step
Test Accuracy (built-in) : 1.000
F1-score                  : 1.000
AUC                       : 1.000

1/1 ██████████ 0s 58ms/step
Test Accuracy (built-in) : 1.000
F1-score                  : 1.000
AUC                       : 1.000

(venv) PS C:\machine_learning>

```