Nama : Firman Ramadhan Saputra

NIM : 231011400891

Kelas : 05TPLE015

Mata Kuliah : Machine Learning

LAPORAN LEMBAR KERJA PERTEMUAN 7 — ARTIFICIAL NEURAL NETWORK (ANN) UNTUK KLASIFIKASI

- Langkah 1 – Persiapan Data

Pada langkah ini, program mulai dengan memuat data dari file bernama *processed_kelulusan.csv* menggunakan pustaka pandas. Kolom "Lulus" dipilih sebagai target (hasil yang ingin diprediksi), sedangkan kolom lainnya digunakan sebagai fitur (data pendukung untuk prediksi). Kemudian data fitur distandarisasi menggunakan StandardScaler agar semua nilai berada pada skala yang sama — ini penting supaya model lebih mudah belajar. Setelah itu, data dibagi menjadi tiga bagian: data *train* (70%) untuk melatih model, *validation* (15%) untuk mengevaluasi selama proses pelatihan, dan *test* (15%) untuk menguji hasil akhir model.

Berikut Screenshoot codenya:

```
# Pertemuan 7
# Langkah 1 – Siapkan Data
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("processed_kelulusan.csv")
x = df.drop("Lulus", axis=1)
y = df['Lulus']

sc = StandardScaler()
Xs =sc.fit_transform(x)

x_train, x_temp, y_train, y_temp = train_test_split(
    Xs, y, test_size=0.3, stratify=y, random_state=42)
x_val, x_test, y_val, y_test = train_test_split(
    x_temp, y_temp, test_size=0.5, stratify=y_temp, random_state=42)

print(x_train.shape, x_val.shape, x_test.shape)
print("\n==========Batas antar Output==========\n")
```

(parameter) test_size:

Berikut output dari codenya:

```
(53, 5) (12, 5) (12, 5)
```

Berikut Penjelasannya:

- **(53, 5)** → berarti data *training* (`x_train`) memiliki **53 baris** dan **5 kolom**.
  - 53 baris = jumlah data yang digunakan untuk melatih model.
  - 5 kolom = jumlah fitur (variabel input) yang digunakan untuk prediksi.

- **(12, 5)** → data *validation* (`x_val`) memiliki **12 baris** dan **5 kolom**.
  - 12 baris = jumlah data yang digunakan untuk menguji model selama pelatihan.
  - 5 kolom = jumlah fitur yang sama seperti data *train*.

- **(12, 5)** → data *testing* (`x_test`) juga memiliki **12 baris** dan **5 kolom**.
  - 12 baris = jumlah data yang digunakan untuk menguji performa akhir model.
  - 5 kolom = jumlah fitur tetap sama.

- Langkah 2 - Bangun Model ANN (Artificial Neural Network)

Pada tahap ini, dibuat sebuah model jaringan saraf tiruan (ANN) menggunakan *library* TensorFlow dan Keras. Model disusun berlapis-lapis: lapisan pertama menerima input, kemudian ada dua lapisan tersembunyi (hidden layer) dengan jumlah neuron 32 dan 16 yang menggunakan fungsi aktivasi ReLU untuk memproses data, serta lapisan dropout untuk mencegah model terlalu meniru data latihan (*overfitting*). Lapisan terakhir menggunakan aktivasi *sigmoid* karena tujuan akhirnya adalah klasifikasi biner (misalnya "lulus" atau "tidak lulus"). Model kemudian dikompilasi dengan *optimizer* Adam, *loss function* binary crossentropy, dan metrik evaluasi berupa akurasi dan AUC. Berikut contoh codenya:

```
# Langkah 2 - Bangun model ANN
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers # type: ignore

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()
```

Berikut outputnya:

```
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)
```

Penjelasan:

Output tersebut menunjukkan **rangkuman struktur model ANN (Artificial Neural Network)**. Model terdiri dari beberapa lapisan:

- **Dense (32 neuron)**: menerima 5 input dan menghasilkan 32 keluaran.
- **Dropout (0.3)**: mencegah overfitting dengan menonaktifkan sebagian neuron.
- **Dense_1 (16 neuron)**: memproses hasil dari lapisan sebelumnya.
- **Dense_2 (1 neuron)**: lapisan output untuk klasifikasi biner (misalnya lulus/tidak lulus).

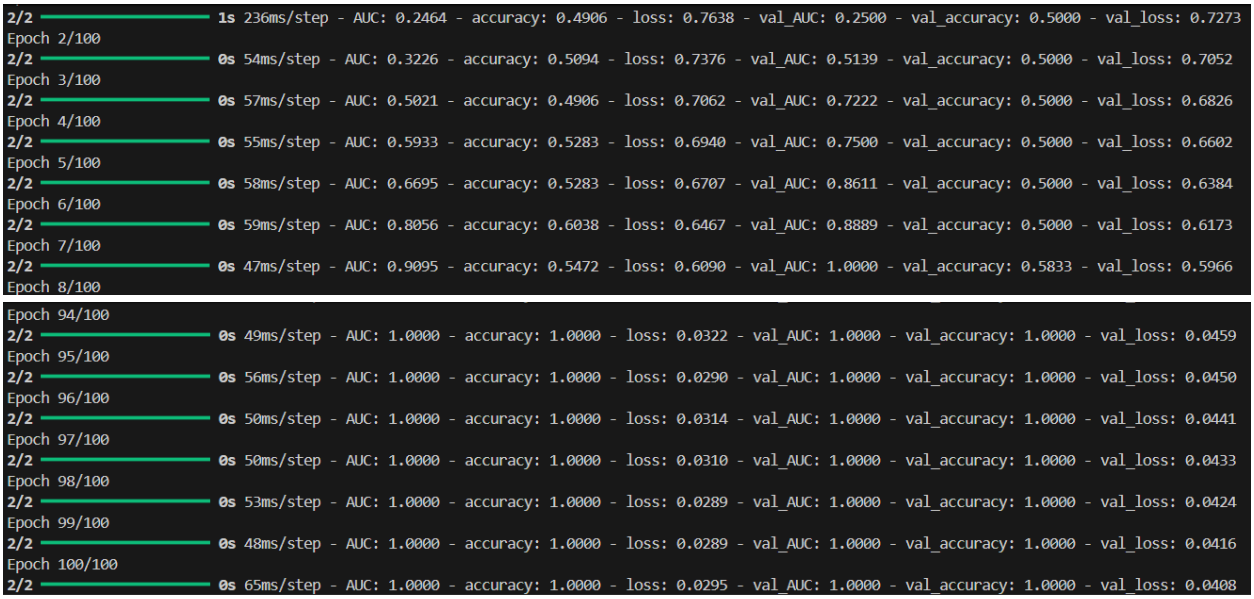
- Langkah 3 - Training dengan Early Stopping

Setelah model siap, proses pelatihan dimulai menggunakan data *train* dan *validation*. Model berusaha mempelajari pola dari data agar bisa memprediksi hasil dengan baik. Digunakan juga fitur *EarlyStopping* agar pelatihan otomatis berhenti jika model tidak lagi membaik setelah beberapa kali percobaan (*epoch*). Hal ini mencegah model bekerja terlalu lama atau menjadi tidak efisien. Proses ini menghasilkan *history* yang mencatat perkembangan nilai *loss* dan akurasi selama pelatihan berlangsung. Berikut dengan codenya:

```python
# langkah 3
es = keras.callbacks.EarlyStopping(
    monitor="val_loss", patience=10, restore_best_weights=True
)

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

Berikut outputnya:



Penjelasanya:

Output tersebut menunjukkan hasil proses **pelatihan model (training)** dari *epoch* ke-1 sampai ke-100.

Secara ringkas:

- Nilai **accuracy** dan **val_accuracy** sama-sama **1.0000 (100%)**, artinya model memprediksi semua data dengan benar, baik pada data latihan maupun validasi.
- Nilai **loss** dan **val_loss** sangat kecil (sekitar 0.03–0.04), menandakan kesalahan prediksi model sangat rendah.
- Nilai **AUC dan val_AUC = 1.0000**, menunjukkan model mampu membedakan kelas positif dan negatif dengan sempurna.

  Kesimpulan: model belajar sangat baik, namun hasil sempurna seperti ini juga bisa menandakan **overfitting** jika diuji pada data baru performanya menurun.

- Langkah 4 - Evaluasi di Test Set

Pada tahap ini, model yang sudah dilatih diuji menggunakan data *test* (data yang belum pernah dilihat oleh model sebelumnya). Dari hasil pengujian, didapatkan nilai akurasi dan AUC yang menunjukkan seberapa baik model mengenali pola. Kemudian, model memprediksi hasil kelulusan dalam bentuk probabilitas, yang dikonversi menjadi 0 dan 1 (tidak lulus/lulus). Hasil prediksi ini dibandingkan dengan data sebenarnya menggunakan *confusion matrix* dan *classification report* untuk melihat seberapa banyak prediksi yang benar atau salah, serta seberapa baik model dalam mengklasifikasikan data.

```
# langkah 4
from sklearn.metrics import classification_report, confusion_matrix

loss, acc, auc = model.evaluate(x_test, y_test, verbose=0)
print("Test Acc:", acc, "AUC:", auc)

y_proba = model.predict(x_test).ravel()
y_pred = (y_proba >= 0.5).astype(int)

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred, digits=3))
print("\n==========Batas antar Output==========\n")
```

Outputnya:

```
Test Acc: 1.0 AUC: 1.0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
[[6 0]
 [0 6]]
              precision    recall  f1-score   support

           0      1.000     1.000     1.000         6
           1      1.000     1.000     1.000         6

    accuracy                          1.000        12
   macro avg      1.000     1.000     1.000        12
weighted avg      1.000     1.000     1.000        12
```

Penjelasan:

- Test Acc (Akurasi Uji): 1.0 dan AUC (Area Under the Curve): 1.0: Model mencapai akurasi 100% dan kinerja Area Under the ROC Curve 100%. Ini berarti model memprediksi semua sampel di set pengujian dengan benar.
- Waktu Pemrosesan: Model dievaluasi dalam 0 detik (0s) dengan waktu 45 milidetik per langkah.

- Total Sampel Uji: Terdapat total 12 sampel, dibagi rata: 6 untuk Kelas 0 dan 6 untuk Kelas 1 (terlihat dari kolom support dan total accuracy).

Confusion Matrix

- **6 (baris 0, kolom 0)**: 6 sampel **Kelas 0** diprediksi dengan benar sebagai Kelas 0 (True Negatives/True Positives, tergantung penamaan).
- **0 (baris 0, kolom 1)**: 0 sampel Kelas 0 diprediksi salah sebagai Kelas 1 (False Positives).
- **0 (baris 1, kolom 0)**: 0 sampel **Kelas 1** diprediksi salah sebagai Kelas 0 (False Negatives).
- **6 (baris 1, kolom 1)**: 6 sampel Kelas 1 diprediksi dengan benar sebagai Kelas 1 (True Positives/True Negatives).

Classification Report (Per Kelas dan Rata-rata)
Semua metrik untuk Kelas 0 dan Kelas 1 adalah 1.000 (100%):
- Precision (Presisi): Dari semua yang diprediksi sebagai kelas tersebut, 100% benar.
- Recall (Sensitivitas/Daya Ingat): Dari semua sampel yang benar-benar termasuk kelas tersebut, 100% berhasil ditangkap.
- f1-score: Rata-rata harmonik dari Precision dan Recall, juga 100%.
- Support: Jumlah sampel untuk kelas tersebut (masing-masing 6).

Rata-rata
- accuracy: Akurasi keseluruhan, 1.000.
- macro avg: Rata-rata sederhana dari metrik per kelas, 1.000.
- weighted avg: Rata-rata dari metrik per kelas, ditimbang berdasarkan *support*, 1.000.

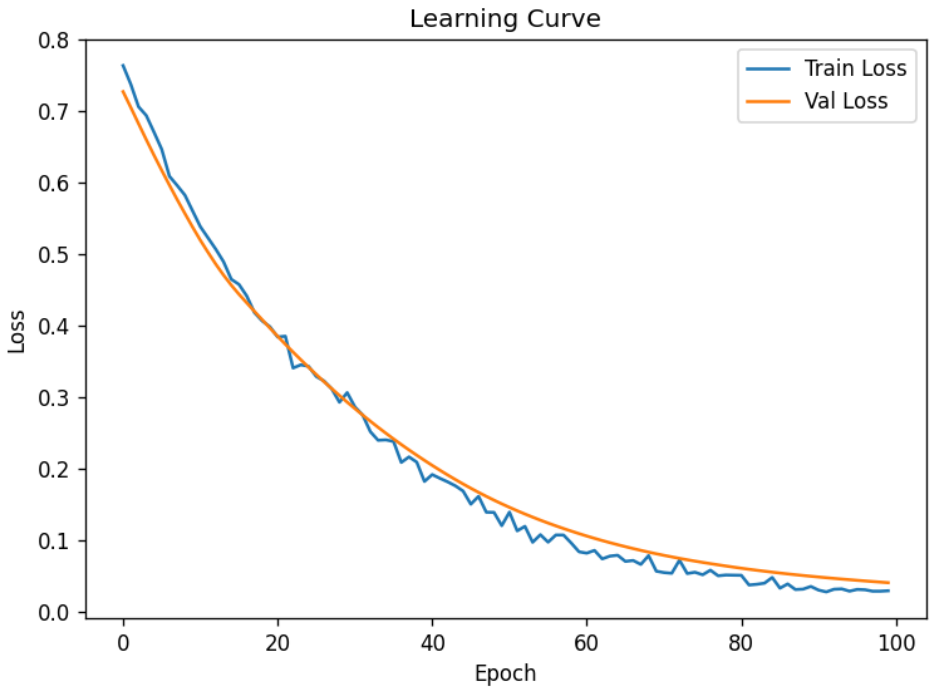
- Langkah 5 - Visualisasi Learning Curve

Langkah terakhir adalah menampilkan grafik *learning curve* yang memperlihatkan perubahan nilai *loss* (kesalahan) selama proses pelatihan dan validasi. Grafik ini membantu melihat apakah model sudah belajar dengan baik atau justru *overfitting*. Hasilnya kemudian disimpan dalam bentuk gambar dengan nama “learning_curve.png”.

```
# langkah 5
import matplotlib.pyplot as plt

plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Val Loss")
plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.legend()
plt.title("Learning Curve")
plt.tight_layout(); plt.savefig("learning_curve.png", dpi=120)
```

Berikut hasilnya:



Grafik ini memvisualisasikan bagaimana kinerja model (dalam hal *Loss* atau Kerugian) berubah seiring berjalannya pelatihan (*Epoch*).

Langkah 6 – Eksperimen

- Ubah jumlah neuron (32/64/128) dan catat efeknya.
- Bandingkan Adam vs SGD+momentum (learning rate berbeda).
- Tambahkan regulasi lain: L2, Dropout lebih besar, atau Batch Normalization.
- Laporkan metrik F1 dan AUC selain akurasi.

Pada langkah ini, dilakukan serangkaian eksperimen untuk mengevaluasi pengaruh arsitektur jaringan, optimizer, dan teknik regulasi terhadap performa model. Variasi jumlah neuron, jenis optimizer, serta penambahan dropout, L2 regularization, dan batch normalization akan diuji. Hasil dari masing-masing eksperimen akan dibandingkan menggunakan metrik akurasi, AUC, dan F1-score untuk mendapatkan pemahaman yang lebih lengkap mengenai model terbaik.

Berikut codenya:

```python
# langkah 6
from tensorflow.keras import regularizers # type: ignore
from tensorflow.keras.optimizers import SGD # type: ignore
import pandas as pd

# Variasi jumlah neuron, optimizer, dan regularisasi
experiments = [
    {"neurons": 32, "optimizer": "adam", "dropout": 0.3, "l2": 0.0, "batch_norm": False},
    {"neurons": 64, "optimizer": "adam", "dropout": 0.3, "l2": 0.0, "batch_norm": False},
    {"neurons": 128, "optimizer": "adam", "dropout": 0.3, "l2": 0.0, "batch_norm": False},
    {"neurons": 64, "optimizer": SGD(learning_rate=0.01, momentum=0.9), "dropout": 0.3, "l2": 0.0, "batch_norm": False},
    {"neurons": 64, "optimizer": "adam", "dropout": 0.5, "l2": 0.01, "batch_norm": True},
]

results = []

for i, exp in enumerate(experiments, 1):
    print(f"\n--- Eksperimen {i}: {exp} ---\n")

    model = keras.Sequential()
    model.add(layers.Input(shape=(x_train.shape[1],)))

    if exp["batch_norm"]:
        model.add(layers.BatchNormalization())

    model.add(layers.Dense(exp["neurons"], activation="relu",
                           kernel_regularizer=regularizers.l2(exp["l2"])))
    model.add(layers.Dropout(exp["dropout"]))

    if exp["batch_norm"]:
        model.add(layers.BatchNormalization())

    model.add(layers.Dense(exp["neurons"]//2, activation="relu",
                           kernel_regularizer=regularizers.l2(exp["l2"])))
    model.add(layers.Dense(1, activation="sigmoid"))

    model.compile(
        optimizer=exp["optimizer"],
        loss="binary_crossentropy",
        metrics=["accuracy", "AUC"]
    )

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

    history = model.fit(
        x_train, y_train,
        validation_data=(x_val, y_val),
        epochs=100, batch_size=32,
        callbacks=[es], verbose=0
    )

    # Evaluasi
    loss, acc, auc = model.evaluate(x_test, y_test, verbose=0)
    y_proba = model.predict(x_test).ravel()
    y_pred = (y_proba >= 0.5).astype(int)

    from sklearn.metrics import f1_score
    f1 = f1_score(y_test, y_pred)

    results.append({
        "Experiment": i,
        "Neurons": exp["neurons"],
        "Optimizer": type(exp["optimizer"]).__name__ if not isinstance(exp["optimizer"], str) else exp["optimizer"],
        "Dropout": exp["dropout"],
        "L2": exp["l2"],
        "BatchNorm": exp["batch_norm"],
        "Accuracy": acc,
        "AUC": auc,
        "F1": f1
    })
```

```python
# Tampilkan hasil eksperimen
results_df = pd.DataFrame(results)
print("\nHasil Eksperimen:\n")
print(results_df)
```

Berikut Outputnya:

```
--- Eksperimen 1: {'neurons': 32, 'optimizer': 'adam', 'dropout': 0.3, 'l2': 0.0, 'batch_norm': False} ---

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step

--- Eksperimen 2: {'neurons': 64, 'optimizer': 'adam', 'dropout': 0.3, 'l2': 0.0, 'batch_norm': False} ---

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step

--- Eksperimen 3: {'neurons': 128, 'optimizer': 'adam', 'dropout': 0.3, 'l2': 0.0, 'batch_norm': False} ---

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step

--- Eksperimen 4: {'neurons': 64, 'optimizer': <keras.src.optimizers.sgd.SGD object at 0x0000018837651450>, 'dropout': 0.3, 'l2': 0.0, 'batch_norm': False} ---

WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x000001883353BC70> t
riggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) pa
ssing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.f
unction has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_re
tracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step

--- Eksperimen 5: {'neurons': 64, 'optimizer': 'adam', 'dropout': 0.5, 'l2': 0.01, 'batch_norm': True} ---

WARNING:tensorflow:6 out of the last 6 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x000001883B76440> t
riggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) pa
ssing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.f
unction has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_re
tracing and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
```

```
Hasil Eksperimen:

   Experiment  Neurons Optimizer  Dropout    L2  BatchNorm  Accuracy  AUC   F1
0           1       32      adam      0.3  0.00      False       1.0  1.0  1.0
1           2       64      adam      0.3  0.00      False       1.0  1.0  1.0
2           3      128      adam      0.3  0.00      False       1.0  1.0  1.0
3           4       64       SGD      0.3  0.00      False       1.0  1.0  1.0
4           5       64      adam      0.5  0.01       True       1.0  1.0  1.0
```

Penjelasan singkat eksperimen:

1. Neuron: 32, 64, 128 → lihat pengaruh ukuran hidden layer.
2. Optimizer: Adam vs SGD + momentum → lihat pengaruh optimizer dan learning rate.
3. Regularisasi: Dropout lebih tinggi, L2, BatchNorm → mencegah overfitting.
4. Metrik: Akurasi, AUC, F1 → memberikan gambaran performa lebih lengkap, terutama untuk dataset tidak seimbang.