

## Demo Aplikasi

```
# =====
# 9) Save trained model ke file .h5
# =====
def save_trained_model():
    keras_model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(len(FEATURE_COLS),)),
        tf.keras.layers.Dense(64, activation="relu"),
        tf.keras.layers.Dense(32, activation="relu"),
        tf.keras.layers.Dense(1, activation="sigmoid"),
    ])

    # Latih model dengan dataset global gabungan
    X_all = pd.concat([X_dinsos, X_dukcapil, X_kemenkes], axis=0).values.astype("float32")
    y_all = np.concatenate([y_dinsos, y_dukcapil, y_kemenkes]).astype("float32")

    keras_model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
        loss=tf.keras.losses.BinaryCrossentropy(from_logits=False),
        metrics=[tf.keras.metrics.BinaryAccuracy()]
    )

    keras_model.fit(X_all, y_all, epochs=10, batch_size=32, verbose=1)

    # Save ke file
    keras_model.save("trained_model.h5")
    print("✅ Model disimpan ke trained_model.h5")

if __name__ == "__main__":
    save_trained_model()
```

- Membuat arsitektur neural network sederhana (2 hidden layer).
- Menggabungkan dataset dari beberapa sumber.
- Melatih model dengan optimizer Adam + loss BinaryCrossentropy.
- Menyimpan model ke file trained\_model.h5 agar bisa digunakan lagi.

```
300 import joblib
301
302 # Simpan preprocessing info
303 joblib.dump({
304     "FEATURE_COLS": FEATURE_COLS,
305     "mins": mins,
306     "rng": rng
307 }, "preprocess.pkl")
308
309 print("✅ Preprocessing info disimpan ke preprocess.pkl")
310
```

Kode ini menyimpan metadata preprocessing ke file preprocess.pkl menggunakan **joblib**. File ini nantinya akan dipakai bersamaan dengan model .h5 supaya pipeline prediksi berjalan mulus:

1. Load preprocess.pkl → terapkan normalisasi dengan mins dan rng.
2. Load trained\_model.h5 → lakukan prediksi pada data baru.

```
grafik disimpan: ctf_serverup_accuracy.png
Epoch 1/10
141/141 [=====] - 1s 2ms/step - loss: 0.1717 - binary_accuracy: 0.9222
Epoch 2/10
141/141 [=====] - 0s 2ms/step - loss: 0.0941 - binary_accuracy: 0.9549
Epoch 3/10
141/141 [=====] - 0s 2ms/step - loss: 0.0877 - binary_accuracy: 0.9596
Epoch 4/10
141/141 [=====] - 0s 2ms/step - loss: 0.0739 - binary_accuracy: 0.9662
Epoch 5/10
141/141 [=====] - 0s 2ms/step - loss: 0.0702 - binary_accuracy: 0.9653
Epoch 6/10
141/141 [=====] - 0s 2ms/step - loss: 0.0644 - binary_accuracy: 0.9691
Epoch 7/10
141/141 [=====] - 0s 2ms/step - loss: 0.0667 - binary_accuracy: 0.9696
Epoch 8/10
141/141 [=====] - 0s 2ms/step - loss: 0.0674 - binary_accuracy: 0.9696
Epoch 9/10
141/141 [=====] - 0s 2ms/step - loss: 0.0623 - binary_accuracy: 0.9722
Epoch 10/10
141/141 [=====] - 0s 2ms/step - loss: 0.0564 - binary_accuracy: 0.9724
/mnt/c/KP/MATERI/4.5. Laporan & Video Demo/venv/lib/python3.11/site-packages/keras/src/engine/trai
considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.
saving_api.save_model(
✓ Model disimpan ke trained_model.h5
✓ Preprocessing info disimpan ke preprocess.pkl
```

1. Model sudah dilatih **10 epoch** dengan hasil akurasi tinggi (**97.24%**).
2. File yang dihasilkan:
  - **trained\_model.h5** → model machine learning yang siap dipakai.
  - **preprocess.pkl** → metadata preprocessing untuk normalisasi data baru.

Menggunakan Flask untuk menjalankan atau mencoba model yang sudah dibuat :

## App.py

1. Import Library

```
app.py > preprocess_input
1 import tensorflow as tf
2 import pandas as pd
3 import numpy as np
4 import joblib
5 from flask import Flask, render_template, request
```

- **TensorFlow** → load model .h5 hasil training.
- **pandas, numpy** → manipulasi data dan preprocessing.
- **joblib** → load preprocessing info (preprocess.pkl).
- **Flask** → framework web Python untuk membuat form input & API prediksi.

## 2. Inisialisasi Flask

```
6
7 app = Flask(__name__)
8
```

- Membuat aplikasi Flask

## 3. Load model dan Pre-Processing Info

```
# === Load trained model ===
keras_model = tf.keras.models.load_model("trained_model.h5", compile=False)

# === Load preprocessing info ===
preproc = joblib.load("preprocess.pkl")
FEATURE_COLS = preproc["FEATURE_COLS"]
mins = preproc["mins"]
rng = preproc["rng"]
```

- **trained\_model.h5** → model hasil training.
- **preprocess.pkl** → menyimpan metadata preprocessing (nama kolom fitur, nilai minimum & range untuk normalisasi).
- **compile=False** → hanya load model untuk inference (tidak butuh compile ulang optimizer/loss).

## 4. Fungsi Pre-Processing Input

```
18 def preprocess_input(form_data):
19     df = pd.DataFrame([form_data])
20
21     # Tambahkan kolom kosong untuk semua FEATURE_COLS yang tidak ada di input
22     for col in FEATURE_COLS:
23         if col not in df:
24             df[col] = 0.0
25
26     # Scale sesuai global
27     df_scaled = (df[FEATURE_COLS] - mins) / rng
28     df_scaled = df_scaled.fillna(0.0)
29     return df_scaled.values.astype("float32")
30
```

- Mengubah dictionary dari form menjadi DataFrame.
- Menambahkan kolom kosong untuk fitur yang hilang (0.0).
- Normalisasi sesuai parameter training (mins dan rng).
- Output berupa array float32 siap diprediksi.

## 5. Routing Halaman Utama

```
30
31 @app.route("/")
32 def index():
33     return render_template("form.html")
34
```

- Saat buka / (homepage), akan menampilkan form.html (form input user).

## 6. Routing Aplikasi

```
35 @app.route("/predict", methods=["POST"])
36 def predict():
37     form = request.form
38
39     # Mulai dengan data numeric
40     input_data = {
41         "jumlah_tanggungan": int(form["jumlah_tanggungan"]),
42         "penghasilan": float(form["penghasilan"]),
43         "umur": int(form["umur"]),
44         "tinggi_cm": float(form["tinggi_cm"]),
45         "berat_kg": float(form["berat_kg"]),
46     }
47
48     # Tambahkan kategori one-hot sesuai pilihan user
49     input_data[f"kondisi_rumah_{form['kondisi_rumah']}"] = 1.0
50     input_data[f"status_pekerjaan_{form['status_pekerjaan']}"] = 1.0
51     input_data[f"status_pernikahan_{form['status_pernikahan']}"] = 1.0
52     input_data[f"riwayat_penakit_{form['riwayat_penakit']}"] = 1.0
53     input_data[f"status_gizi_{form['status_gizi']}"] = 1.0
54
55     X_user = preprocess_input(input_data)
56     pred = keras_model.predict(X_user)[0][0]
57     label = "✅ Layak Subsidi" if pred >= 0.5 else "❌ Tidak Layak Subsidi"
58     return render_template("result.html", result=label, score=pred)
59
```

Alurnya:

1. Ambil **input numerik** (tanggungan, penghasilan, umur, tinggi, berat).
2. Encode **kategori** menjadi one-hot (status, kondisi rumah, gizi, dll).
3. Preprocessing (normalize) → array float32.
4. Prediksi dengan model Keras.
5. Jika hasil  $\geq 0.5$  → Layak Subsidi, kalau  $< 0.5$  → Tidak Layak.
6. Render hasil ke **result.html**.

## 7. Menjalankan Server

```
60 if __name__ == "__main__":
61     app.run(debug=True)
62
```

# Form.html

Cek Kelayakan Subsidi

Jumlah Tanggungan

Penghasilan (Rp)

Kondisi Rumah

Lain

Umur

Status Pekerjaan

Pengangguran

Status Pernikahan

Lajang

Riwayat Penyakit

Sakit

Status Gaji

Normal

Tinggi (cm)

Berat (kg)

Cek Subsidi

# Result.html

Hasil Prediksi

{{ result }}

Skor Model: {{ "%2F" % score }}

Coba Lagi

# UJI COBA

## 1. Kategori Layak

Cek Kelayakan Subsidi

Jumlah Tanggungan: 2

Penghasilan (Rp): 2000000

Kondisi Rumah: Sewa Paman

Umur: 25

Status Pekerjaan: Pengangguran

Status Pernikahan: Lajang

Riwayat Penyakit: Sehat

Status Gizi: Normal

Tinggi (cm): 170

Berat (kg): 65

Cek Subsidi

## Hasil Prediksi

✓ Layak Subsidi

Skor Model: 1.00

Coba Lagi

## 2. Kategori Tidak Layak

Cek Kelayakan Subsidi

Jumlah Tanggungan: 5

Penghasilan (Rp): 2000000

Kondisi Rumah: Layak

Umur: 50

Status Pekerjaan: Pengangguran

Status Pernikahan: Lajang

Riwayat Penyakit: Sehat

Status Gizi: Normal

Tinggi (cm): 150

Berat (kg): 45

Cek Subsidi

## Hasil Prediksi

✗ Tidak Layak Subsidi

Skor Model: 0.00

Coba Lagi