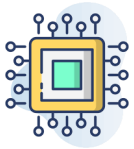
Face Recognition

Implementasi Ringan pada Raspberry Pi dengan Custom TFLite Conversion

Studi Kasus: Konversi Model Keras-MobileNetV2 & Optimasi RTSP Stream

EL5183 IoT System and Application

Latar Belakang



Tantangan: Menjalankan Face Recognition canggih di perangkat kecil (Raspberry Pi 4/5).

Masalah Utama: Model standar (format .h5/SavedModel) sangat besar (90MB+) dan berat.

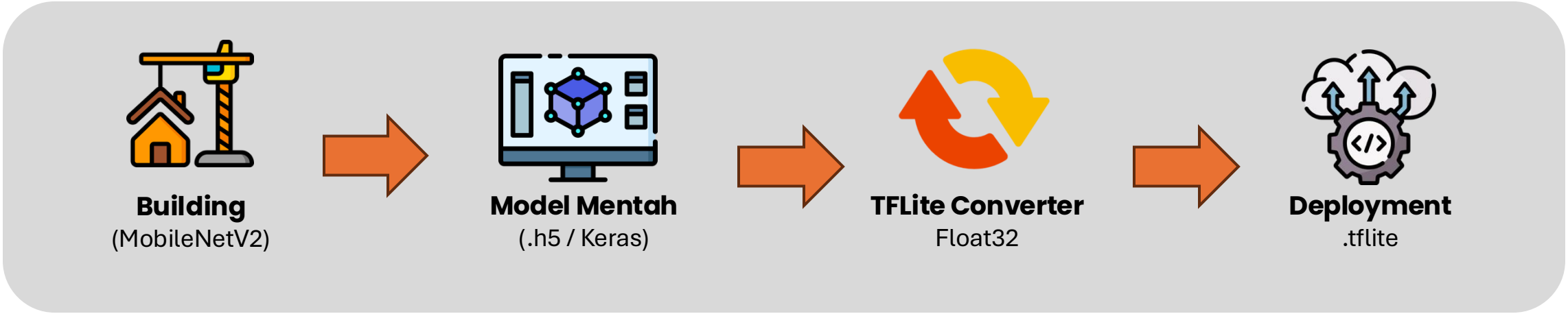
Akibat: FPS rendah (lag), CPU panas, delay tinggi pada CCTV.

Solusi:

Menggunakan MobileNetV2: Output 128 Dimensi. Didesain khusus untuk Mobile/IoT. Ringan dan Stabil.

Alur Konversi Model

Proses mengubah "**Model Raksasa**" menjadi "Model Kecil":



Kode Implementasi

Script untuk membuat model dan konversi ke tflite :

```
import tensorflow as tf
import os
import numpy as np

# Nama file
H5_FILE = "mobilenetv2_128_fresh.h5"
TFLITE_FILE = "facenet_128_float32.tflite"

def main():
print("[1] Membangun Model MobileNetV2 dari Nol...")
# Kita pakai arsitektur MobileNetV2 (ini standar industri buat HP/Raspi)
# Input 160x160 RGB
base_model = tf.keras.applications.MobileNetV2(
input_shape=(160, 160, 3),
include_top=False, # Buang buntut klasifikasinya
weights='imagenet' # Pake otak dasar pengenalan benda (bukan wajah, tapi lumayan)
)
# Tambahkan buntut baru supaya outputnya 128 Dimensi (Kaya FaceNet)
x = base_model.output
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(128, activation=None)(x) # Output Vector 128
# Normalisasi L2 di dalam model (biar outputnya udah mateng)
outputs = tf.keras.layers.Lambda(lambda x: tf.math.l2_normalize(x, axis=1))(x)
model = tf.keras.Model(inputs=base_model.input, outputs=outputs)
```

```
# Simpan jadi .h5 (Ini simulasi lu punya file .h5 sendiri)
print(f" Menyimpan ke '{H5_FILE}'...")
model.save(H5_FILE)

print("\n[2] Proses Belajar: Mengkonversi ke TFLite...")

# Load balik file .h5 yang barusan kita buat (Pasti kompatibel karena satu laptop)
model_loaded = tf.keras.models.load_model(H5_FILE)
converter = tf.lite.TFLiteConverter.from_keras_model(model_loaded)
# -- KUNCI BIAR RASPI GAK OVERFLOW --
# 1. Jangan di-Quantize (Optimizations = []) -> Biar CPU Raspi gak pusing dekompresi
# 2. Model MobileNetV2 ini jauh lebih ringan dari InceptionResnet (90MB kemarin)
converter.optimizations = []
tflite_model = converter.convert()

with open(TFLITE_FILE, "wb") as f:
f.write(tflite_model)
size_mb = os.path.getsize(TFLITE_FILE) / 1024 / 1024
print(f"\n[SUKSES BESAR] File '{TFLITE_FILE}' jadi!")
print(f"Ukuran: {size_mb:.2f} MB")

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

Bagian 1 : Membangun Arsitektur

Di bagian ini, kita menyusun lapisan-lapisan **Neural Network** seperti menyusun **Lego**.

1. Memilih Fondasi (Base Model)

```
base_model = tf.keras.applications.MobileNetV2(
input_shape=(160, 160, 3),
include_top=False,
weights='imagenet')
```

Kita tidak membangun AI dari nol tapi menggunakan model buatan Google yang bernama MobileNetV2.

input_shape=(160, 160, 3): Kita membatasi pintu masuknya. AI ini hanya menerima gambar ukuran 160x160 piksel dengan 3 warna (RGB).

weights='imagenet': Ini konsep Transfer Learning.

- Bayangkan AI itu bayi. Kalau weights=None, otaknya kosong.
- Dengan weights='imagenet', AI ini sudah "lulus SD". Dia sudah belajar mengenali garis, lengkungan, mata, hidung dari dataset ImageNet (jutaan gambar umum). Jadi dia tidak bodoh-bodoh amat walau belum spesifik belajar wajah.

include_top=False : MobileNet asli punya "Kepala" (lapisan akhir) untuk menebak 1000 benda (Kucing, Anjing, Mobil, dll). Kita MEMENGGAL kepalanya (False). Kita cuma butuh "Badan"-nya saja untuk mengekstrak fitur visual, bukan untuk menebak nama benda.

2. Menambah Kepala Baru (Custom Head)

Sekarang kita pasang kepala baru yang spesifik untuk Face Recognition.

```
x = base_model.output
x = tf.keras.layers.GlobalAveragePooling2D()(x)
```

GlobalAveragePooling2D:

- Output dari "Badan" MobileNet itu bentuknya tumpukan data 3D yang tebal (7x7x1280).
- Layer ini "menggepengkan" data tersebut dengan cara dirata-rata, menjadi satu garis lurus (vektor) berisi 1280 angka.

```
x = tf.keras.layers.Dense(128, activation=None)(x)
```

Dense(128): Ini intinya. Kita memadatkan informasi dari 1280 angka menjadi 128 angka saja. 128 angka inilah yang disebut EMBEDDING (Sidik Jari Wajah).

3. Sentuhan Terakhir (L2 Normalization)

```
outputs = tf.keras.layers.Lambda(lambda x: tf.math.l2_normalize(x, axis=1))(x)
```

Apa itu? Teknik matematika untuk mengubah panjang vektor menjadi 1.
Kenapa Wajib?

- Face Recognition bekerja dengan mengukur JARAK (Euclidean Distance).
- Tanpa normalisasi, vektor wajah bisa panjang-pendek sembarangan (skalanya beda-beda).
- Dengan L2 Norm, semua wajah dipaksa berada di "kulit bola" yang sama.
- Ini membuat proses membandingkan "Si A" dan "Si B" jadi adil dan akurat.

Bagian 2 : Konversi

Setelah **model** (Rumah) jadi dan disimpan ke .h5, kita harus **mengubahnya** agar bisa jalan di Raspberry Pi.

```
converter = tf.lite.TFLiteConverter.from_keras_model(model_loaded)
converter.optimizations = [] # <-- KUNCI RAHASIA KITA
tflite_model = converter.convert()
```

Kenapa converter.optimizations = [] itu Penting?

1. Jika optimizations = [tf.lite.Optimize.DEFAULT] **(gagal):**

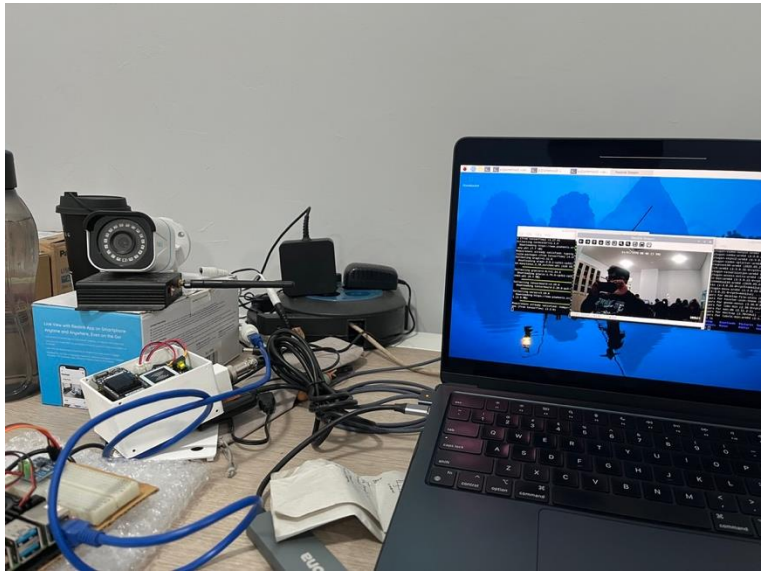
1. Converter akan melakukan **Quantization**.
2. Dia mengubah angka-angka di dalam model dari Float32 (Desimal presisi tinggi, 4 byte) menjadi Int8 (Bilangan bulat, 1 byte).
3. **Masalahnya:** Struktur MobileNet/FaceNet itu rumit. Saat dikompres jadi Int8, TFLite di Raspberry Pi perlu melakukan operasi "Dekompresi" khusus saat mau dipakai.
4. Ternyata, versi library di Raspberry Pi punya *bug* memori saat melakukan dekompresi ini -> **ERROR OVERFLOW**.

2. Jika optimizations = [] **(berhasil):**

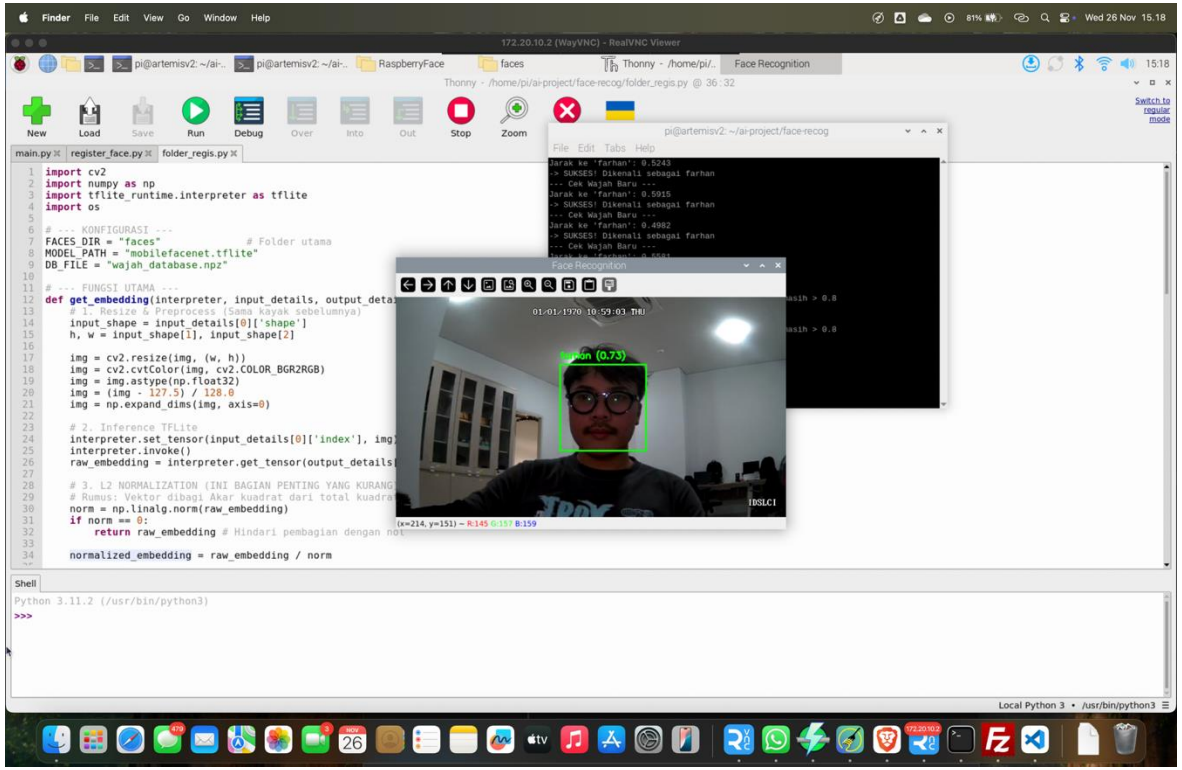
1. Kita bilang ke Converter: *"Jangan diapa-apain. Biarin aja apa adanya."*
2. Model tetap dalam format **Float32**.
3. **Ukuran:** Memang lebih besar (sekitar 9-11 MB).
4. **Kompatibilitas:** Raspberry Pi langsung paham! Tidak perlu dekompresi, tidak perlu hitung alokasi memori aneh-aneh. Langsung telan.
5. Karena kita pakai **MobileNetV2** (yang dasarnya emang arsitektur ringan), meskipun Float32, dia tetap enteng buat Raspberry Pi.

Uji Coba

Menggunakan **Raspberry** Pi 4B dan CCTV Reolink via substream **RSTP**.



Gambar 1. Setup Pengujian



Gambar 2. Pengujian

Terima kasih

https://github.com/inahrafm/face-recog-lite