**PRAKTIKUM PENERAPAN MACHINE LEARNING PADA ESP32 MENGGUNAKAN TINYML DALAM MATA KULIAH INTERNET OF THINGS**

**(IOT)**



Muhammad Fa’iz Ramadhan

Fakultas Vokasi, Universitas Brawijaya

[faizramadhan23@student.ub.ac.id](mailto:faizramadhan23@student.ub.ac.id)

**Abstrak**

Tujuan dari latihan ini adalah untuk menerapkan algoritma pembelajaran mesin pada mikrokontroler ESP32 dengan menggunakan pendekatan TinyML. Model yang digunakan adalah klasifikasi bunga iris dengan empat fitur utama input. Setelah menggunakan TensorFlow untuk mempelajari model, format.tflite diubah untuk digunakan secara lokal di ESP32 tanpa internet. Hasil pengujian menunjukkan bahwa ESP32 dapat memprediksi data uji dengan akurasi 100% dan menjalankan model dengan waktu inferensi yang sangat cepat.

**Kata kunci**: Machine Learning, TinyML, ESP32, TensorFlow Lite, Iris Dataset

1. **Pendahuluan**
   1. **Latar belakang**

Dengan munculnya Internet of Things (IoT), ada banyak peluang untuk pemrosesan data secara real-time. Dengan munculnya model pembelajaran mesin yang lebih ringan seperti TinyML, algoritma cerdas sekarang dapat dijalankan secara langsung pada perangkat mikro seperti ESP32. Ini memungkinkan pengambilan keputusan cepat tanpa perlu koneksi cloud.

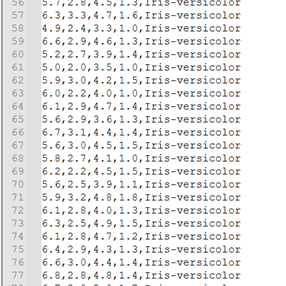
* 1. **Tujuan eksperimen**

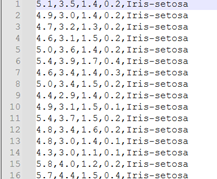
Tujuan dilakukannya praktikum ini adalah :

1. Mengubah model TensorFlow menjadi format .tflite.
2. Menerapkan model klasifikasi Machine Learning menggunakan dataset Iris.
3. Menjalankan model pada ESP32 dengan pendekatan TinyML.
4. Menganalisis performa dan akurasi sistem dalam kondisi simulasi.
5. **Metodologi**
   1. **Alat dan Bahan**
6. Mikrokontroler ESP32
7. File model Machine Learning (format .tflite)
8. Library Arduino: eloquent\_tinyml, tflm\_esp32
9. Dataset Iris (training dan testing dari TensorFlow)
10. Library TensorFlow, NumPy, Pandas, sklearn
    1. **Langkah Implempentasi**
11. Impor dataset Iris menggunakan library pandas atau sklearn.datasets.
12. Lakukan preprocessing data
13. Bangun dan latih model klasifikasi dengan TensorFlow.
14. Gunakan skrip tambahan untuk mengubah .tflite ke .h agar bisa dipakai di Arduino.
15. Buat file iris\_model.h yang berisi hasil konversi model .tflite
16. Tulis kode program menggunakan library eloquent\_tinyml untuk menjalankan inferensi.
17. Upload program ke ESP32
18. Jalankan inferensi terhadap input x0, x1, dan x2 yang mewakili tiga spesies bunga iris.
19. Melakukan debugging jika terjadi kesalahan dalam program atau koneksi perangkat keras.
20. Mengamati hasil dan mencatat performa sistem selama siklus operasional berlangsung.
21. **Hasil dan Pembahasan**

Hasil praktikum menunjukkan bahwa integrasi model Machine Learning sederhana dengan mikrokontroler ESP32 dapat berjalan dengan baik. Menggunakan dua lapisan (Dense dan Softmax), model berhasil dikompresi agar dapat dijalankan secara lokal tanpa koneksi internet. Dengan waktu inferensi kurang dari 1,1 milidetik dan akurasi 100% pada tiga sampel uji, sistem ini ideal untuk aplikasi real-time seperti klasifikasi lingkungan atau perangkat wearable. Penggunaan pustaka EloquentTinyML mempermudah integrasi model `.tflite` ke dalam kode Arduino berkat API yang sederhana. Meskipun masih dasar, sistem ini menunjukkan potensi besar TinyML. Untuk aplikasi nyata, perlu dataset yang lebih besar, validasi silang, serta integrasi sensor dan jaringan seperti MQTT agar prediksi dapat ditindaklanjuti otomatis oleh sistem lain.

* 1. **Hasil Eksperimen**





* 1. **Kode Program**

#include <Arduino.h>

/\*\*

\* Run a TensorFlow model to predict the IRIS dataset

\* For a complete guide, visit

\* https://eloquentarduino.com/tensorflow-lite-esp32

\*/

// replace with your own model

// include BEFORE <eloquent\_tinyml.h>!

#include "iris\_model.h"

// include the runtime specific for your board

// either tflm\_esp32 or tflm\_cortexm

#include <tflm\_esp32.h>

// now you can include the eloquent tinyml wrapper

#include <eloquent\_tinyml.h>

// this is trial-and-error process

// when developing a new model, start with a high value

// (e.g. 10000), then decrease until the model stops

// working as expected

#define ARENA\_SIZE 2000

Eloquent::TF::Sequential<TF\_NUM\_OPS, ARENA\_SIZE> tf;

//Eloquent::TinyML::TfLite<4,3,ARENA\_SIZE> tf;

/\*\*

\*

\*/

void setup() {

Serial.begin(115200);

delay(3000);

Serial.println("\_\_TENSORFLOW IRIS\_\_");

// configure input/output

// (not mandatory if you generated the .h model

// using the everywhereml Python package)

tf.setNumInputs(4);

tf.setNumOutputs(3);

// add required ops

// (not mandatory if you generated the .h model

// using the everywhereml Python package)

tf.resolver.AddFullyConnected();

tf.resolver.AddSoftmax();

while (!tf.begin(irisModel).isOk())

Serial.println(tf.exception.toString());

}

void loop() {

// x0, x1, x2 are defined in the irisModel.h file

// https://github.com/eloquentarduino/EloquentTinyML/tree/main/examples/IrisExample/irisModel.h

// classify sample from class 0

if (!tf.predict(x0).isOk()) {

Serial.println(tf.exception.toString());

return;

}

Serial.print("expcted class 0, predicted class ");

Serial.println(tf.classification);

// classify sample from class 1

if (!tf.predict(x1).isOk()) {

Serial.println(tf.exception.toString());

return;

}

Serial.print("expcted class 1, predicted class ");

Serial.println(tf.classification);

// classify sample from class 2

if (!tf.predict(x2).isOk()) {

Serial.println(tf.exception.toString());

return;

}

Serial.print("expcted class 2, predicted class ");

Serial.println(tf.classification);

// how long does it take to run a single prediction?

Serial.print("It takes ");

Serial.print(tf.benchmark.microseconds());

Serial.println("us for a single prediction");

delay(1000);

}