

Machine Learning Systems

Principi ed Esercizi di Ingegneria dei Sistemi di Intelligenza Artificiale
(Parte 2: Laboratori)

Vijay Janapa Reddi

Traduzione italiana: Baldassarre Cesarano
27 novembre 2024

Indice

LABORATORI	5
Panoramica	5
Obiettivi dell'Apprendimento	5
Pubblico Target	5
Dispositivi Supportati	5
Struttura del Lab	6
Sequenza dei Laboratori Consigliata	6
Risoluzione dei Problemi e Supporto	6
Crediti	6
Guida Introduttiva	6
Requisiti Hardware	6
Requisiti Software	7
Connettività di Rete	8
Conclusione	8
Nicla Vision	9
Setup	9
Panoramica	9
Hardware	10
Installazione di Arduino IDE	11
Installazione di OpenMV IDE	16
Collegamento di Nicla Vision a Edge Impulse Studio	22
Espansione della Nicla Vision Board (opzionale)	24
Conclusione	28
Risorse	28
Classificazione delle Immagini	28
Panoramica	29
Visione Artificiale	30
Obiettivo del Progetto di Classificazione delle Immagini	30
Raccolta Dati	31
Addestramento del modello con Edge Impulse Studio	33
Il Dataset	34
Impulse Design	37
Addestramento del Modello	41
Test del Modello	43
Distribuzione del modello	44
Classificazione delle immagini Benchmark (non ufficiale)	53
Conclusione	54
Risorse	55
Rilevamento degli Oggetti	56
Panoramica	56
Obiettivo del Progetto di Object Detection	59
Raccolta Dati	60
Edge Impulse Studio	61
Impulse Design	65
Progettazione, Addestramento e Test del Modello	67
Distribuzione del Modello	71
Conclusione	74
Risorse	75
Keyword Spotting (KWS)	76
Panoramica	76
Come funziona un assistente vocale?	77
Il Progetto Pratico KWS	78
Il Dataset	78
Creazione di Impulse (Pre-Process / Definizione del Modello)	83
Progettazione e Addestramento del Modello	85
Test	87
Distribuzione e Inferenza	88
Post-elaborazione	89
Conclusione	91
Risorse	91
Classificazione del Movimento e Rilevamento delle Anomalie	92
Panoramica	92
Installazione e test dell'IMU	93
Il Caso di Studio: Trasporto Simulato di Container	96
Raccolta Dati	97

Impulse Design	103
Addestramento dei Modelli.....	107
Test.....	108
Distribuzione.....	109
Conclusione.....	112
Risorse.....	113
XIAO ESP32S3.....	115
Setup	115
Panoramica.....	115
Installazione di XIAO ESP32S3 Sense su Arduino IDE	117
Test della scheda con BLINK.....	118
Collegamento del Modulo Sense (Scheda di Espansione)	119
Test del Microfono.....	119
Test della Fotocamera.....	123
Test del WiFi.....	124
Conclusione.....	130
Risorse.....	130
Classificazione delle Immagini	130
Panoramica.....	131
Un progetto TinyML di Motion Classification: Frutta contro Verdura.....	132
Addestramento del modello con Edge Impulse Studio	132
Test del modello (inferenza).....	143
Test con un Modello Più Grande	144
Esecuzione dell'inferenza su SenseCraft-Web-Toolkit	146
Conclusione.....	150
Risorse.....	150
Rilevamento degli Oggetti.....	151
Panoramica.....	151
Obiettivo del Progetto di Object Detection	153
Raccolta Dati	155
Edge Impulse Studio.....	156
Impulse Design	161
Progettazione, Addestramento e Test del Modello.....	163
Deploying del Modello (Arduino IDE)	166
Distribuzione del Modello (SenseCraft-Web-Toolkit)	170
Conclusione.....	173
Risorse.....	173
Keyword Spotting (KWS)	173
Panoramica.....	173
Il Dataset	177
Modello di training con Edge Impulse Studio	186
Test.....	194
Distribuzione e Inferenza	196
Post-elaborazione	199
Conclusione.....	200
Risorse.....	200
Classificazione del Movimento e Rilevamento delle Anomalie	201
Panoramica.....	201
Installazione dell'IMU	202
Il Progetto TinyML Motion Classification.....	208
Collegamento del dispositivo a Edge Impulse	209
Raccolta Dati	211
Pre-elaborazione dei Dati	213
Progettazione del Modello	214
Impulse Design	215
Generazione di feature	216
Training	217
Test.....	219
Distribuzione	220
Inferenza	221
Conclusione.....	224
Risorse.....	225
Raspberry Pi	226
Setup	226
Panoramica.....	226
Panoramica Hardware	228
Installazione del Sistema Operativo	229
Accesso Remoto	232

Aumentare la memoria SWAP	235
Installazione di una Fotocamera	237
Esecuzione del desktop Raspi da remoto	243
Aggiornamento e Installazione del Software	247
Considerazioni Specifiche del Modello.....	247
Classificazione delle Immagini	248
Panoramica	248
Impostazione dell'Ambiente	249
Fare inferenze con Mobilenet V2	253
Progetto di Classificazione delle Immagini	260
Classificazione delle Immagini in Tempo Reale	272
Conclusione:	277
Risorse	277
Rilevamento degli Oggetti	278
Panoramica	278
Panoramica dei Modelli di Rilevamento degli Oggetti Pre-Addestrati.....	281
Progetto di Rilevamento di Oggetti	286
Addestramento di un Modello SSD MobileNet su Edge Impulse Studio.....	293
Training di un modello FOMO su Edge Impulse Studio	305
Esplorazione di un Modello YOLO tramite Ultralytics.....	312
Rilevamento di Oggetti su un Live Streaming	324
Conclusione	327
Risorse	328
Small Language Models (SLM).....	329
Panoramica	329
Setup.....	330
Generative AI (GenAI).....	332
Ollama	335
Libreria Python Ollama.....	345
SLM: Tecniche di Ottimizzazione	356
Implementazione del RAG.....	356
Conclusione	361
Risorse	362
Lab Condivisi.....	363
KWS Feature Engineering	363
Panoramica	363
Il KWS	364
Panoramica sui Segnali Audio	365
Panoramica sugli MFCC	367
Pratica con Python	370
Conclusione	370
Risorse	371
Blocco delle Feature Spettrali DSP	371
Panoramica	371
Estrazione delle Feature di Revisione	372
Un progetto TinyML di Motion Classification	373
Pre-elaborazione dei Dati	374
Feature Statistiche del Dominio del Tempo	379
Feature spettrali	381
Dominio tempo-frequenza	383
Conclusione	388

LABORATORI

Panoramica

Benvienuti alla sezione dei laboratori pratici in cui esplorerete l'implementazione di modelli ML su dispositivi embedded reali, che offriranno un'introduzione pratica ai sistemi ML. A differenza degli approcci tradizionali con modelli su larga scala, questi laboratori si concentrano sull'interazione diretta sia con l'hardware che con il software. Ci aiutano a mostrare varie modalità di sensori in diversi casi d'uso delle applicazioni. Questo approccio fornisce preziose informazioni sulle sfide e le opportunità dell'implementazione dell'IA su sistemi fisici reali.

Obiettivi dell'Apprendimento

Completando questi "lab", speriamo che gli studenti:

💡 Consiglio

- Acquisiscano competenza nell'impostazione e distribuzione di modelli ML su dispositivi supportati, consentendo di affrontare scenari di distribuzione ML nel mondo reale con sicurezza.
- Comprendano i passaggi coinvolti nell'adattamento e nella sperimentazione di modelli ML per diverse applicazioni, consentendo di ottimizzare prestazioni ed efficienza.
- Apprendano tecniche di risoluzione dei problemi specifiche per le distribuzioni ML embedded, dotandoli delle competenze per superare insidie e sfide comuni.
- Acquisiscano esperienza pratica nella distribuzione di modelli TinyML su dispositivi embedded, colmando il divario tra teoria e pratica.
- Esplorino varie modalità di sensori e le loro applicazioni, ampliando la comprensione di come ML può essere sfruttato in diversi domini.
- Favoriscano una comprensione delle implicazioni e delle sfide del mondo reale associate alle distribuzioni di sistemi ML, preparandoli per progetti futuri.

Pubblico Target

Questi lab sono progettati per:

- **Principianti** nel campo dell'apprendimento automatico che hanno un vivo interesse nell'esplorare l'intersezione tra ML e sistemi embedded.
- **Sviluppatori e ingegneri** che desiderano applicare modelli ML ad applicazioni del mondo reale utilizzando dispositivi a basso consumo e risorse limitate.
- **Appassionati e ricercatori** che desiderano acquisire esperienza pratica nell'implementazione dell'IA su dispositivi edge e comprendere le sfide uniche coinvolte.

Dispositivi Supportati

Abbiamo incluso materiali di laboratorio per tre dispositivi chiave che rappresentano diversi profili hardware e capacità.

- **Nicla Vision:** Ottimizzato per applicazioni basate sulla visione come la classificazione delle immagini e il rilevamento di oggetti, ideale per casi d'uso compatti e a basso consumo.
- **XIAO ESP32S3:** Una scheda versatile e compatta adatta per attività di individuazione di parole chiave e rilevamento del movimento.
- **Raspberry Pi:** Una piattaforma flessibile per attività più intensive dal punto di vista computazionale, inclusi piccoli modelli linguistici e varie applicazioni di classificazione e rilevamento.

Esercizio	Nicla Vision	XIAO ESP32S3	Raspberry Pi
Installation & Setup	✓	✓	✓
Keyword Spotting (KWS)	✓	✓	
Image Classification	✓	✓	✓
Object Detection	✓	✓	✓
Motion Detection	✓	✓	
Small Language Models (SLM)			✓

Struttura del Lab

Ogni lab segue un approccio strutturato:

- Introduzione:** Esplora l'applicazione e la sua importanza in scenari reali.
- Configurazione:** Istruzioni dettagliate per configurare l'ambiente hardware e software.
- Distribuzione:** Guida al training e alla distribuzione dei modelli ML pre-addestrati sui dispositivi supportati.
- Esercizi:** Attività pratiche per modificare e sperimentare con i parametri del modello.
- Discussione:** Analisi dei risultati, potenziali miglioramenti e approfondimenti pratici.

Sequenza dei Laboratori Consigliata

Se si è alle prime armi con l'ML embedded, consigliamo di iniziare con la configurazione e l'individuazione delle parole chiave prima di passare alla classificazione delle immagini e al rilevamento degli oggetti. Gli utenti di Raspberry Pi possono esplorare attività più avanzate, come piccoli modelli linguistici, dopo aver familiarizzato con le basi.

Risoluzione dei Problemi e Supporto

Se si riscontrano problemi durante i laboratori, consultare i commenti sui "troubleshooting" [risoluzione dei problemi] o controllare le FAQ all'interno di ogni laboratorio. Per ulteriore assistenza, non esitate a contattare il nostro team di supporto o a interagire con i forum della community.

Crediti

Un ringraziamento speciale e un ringraziamento al [Prof. Marcelo Rovai](#) per il suo prezioso contributo allo sviluppo e al continuo perfezionamento di questi laboratori.

Guida Introduttiva

Benvenuti nell'entusiasmante mondo dell'apprendimento automatico embedded e di TinyML! In questa serie di laboratori pratici, esploreremo vari progetti che dimostrano la potenza dell'esecuzione di modelli di apprendimento automatico su dispositivi con risorse limitate. Prima di immergersi nei progetti, ci si assicuri di avere l'hardware e il software necessari.

Requisiti Hardware

Per seguire i laboratori pratici, ci sarà bisogno del seguente hardware:

- Scheda Arduino Nicla Vision**

- Arduino Nicla Vision è una scheda potente e compatta progettata per applicazioni audio e di visione artificiale di livello professionale. È dotata di un modulo telecamera di alta qualità, un microfono digitale e un'IMU, che la rendono adatta a progetti impegnativi in settori quali robotica, automazione e sorveglianza.

- Specifiche di Arduino Nicla Vision
- Schema e pinout di Arduino Nicla Vision

2. Scheda XIAO ESP32S3 Sense

- La scheda Seeed Studio XIAO ESP32S3 Sense è una scheda minuscola e ricca di funzionalità, progettata per maker, hobbisti e studenti interessati a esplorare applicazioni IA edge. È dotata di fotocamera, microfono e IMU, rendendo facile iniziare con progetti come classificazione delle immagini, individuazione di parole chiave e rilevamento del movimento.
- Specifiche di XIAO ESP32S3 Sense
- Schema e pinout di XIAO ESP32S3 Sense

3. Raspberry Pi Single Computer board

- Il Raspberry Pi è un potente e versatile computer a scheda singola che è diventato uno strumento essenziale per gli ingegneri di varie discipline. Sviluppati dalla [Raspberry Pi Foundation](#), questi dispositivi compatti offrono una combinazione unica di convenienza, potenza di calcolo e ampie capacità GPIO (General Purpose Input/Output), rendendoli ideali per la prototipazione, lo sviluppo di sistemi embedded e progetti di ingegneria avanzata.
- [Documentazione Hardware di Raspberry Pi](#)
- [Documentazione della Telecamera](#)

4. Accessori aggiuntivi

- Cavo USB-C per la programmazione e l'alimentazione di XIAO
- Cavo micro-USB per la programmazione e l'alimentazione di Nicla
- Alimentatore per Raspberry
- Breadboard e cavi jumper (opzionali, per collegare sensori aggiuntivi)

Arduino Nicla Vision è progettata su misura per applicazioni di livello professionale, offrendo funzionalità avanzate e prestazioni adatte a progetti industriali impegnativi. D'altro canto, Seeed Studio XIAO ESP32S3 Sense è rivolta a maker, hobbisti e studenti che desiderano esplorare applicazioni IA edge in un formato più accessibile e adatto ai principianti. Entrambe le schede hanno i loro punti di forza e il loro pubblico di riferimento, consentendo agli utenti di scegliere la soluzione migliore per le loro esigenze e il loro livello di competenza. Raspberry Pi è destinato a progetti di ingegneria e apprendimento automatico più avanzati.

Requisiti Software

Per programmare le schede e sviluppare progetti di apprendimento automatico embedded, ci sarà bisogno del seguente software:

1. Arduino IDE

- Download e installazione
 - Install di Arduino IDE
 - Seguire la guida all'[installazione](#) per il sistema operativo in uso.
 - [Arduino CLI](#)
 - Configurare l'IDE Arduino per le schede [Arduino Nicla Vision](#) e [XIAO ESP32S3 Sense](#).

2. OpenMV IDE (opzionale)

- Scaricare e installare [OpenMV IDE](#) per il sistema operativo in uso.
- Configurare l'IDE OpenMV per [Arduino Nicla Vision](#).

3. Edge Impulse Studio

- Registrarsi per un account gratuito su [Edge Impulse Studio](#).
- Installare [Edge Impulse CLI](#)

- Seguire le guide per connettere le schede [Arduino Nicla Vision](#) e [XIAO ESP32S3 Sense](#) a Edge Impulse Studio.

4. Raspberry Pi OS

- Scaricare e installare [Raspberry Pi Imager](#)

Connettività di Rete

Alcuni progetti potrebbero richiedere la connettività Internet per la raccolta dati o la distribuzione del modello. Assicurarsi che la connessione dell'ambiente di sviluppo sia stabile tramite Wi-Fi o Ethernet. Per Raspberry Pi, è necessaria una connessione Wi-Fi o Ethernet per il funzionamento remoto senza la necessità di collegare un monitor, una tastiera e un mouse.

- Per Arduino Nicla Vision, si può utilizzare il modulo Wi-Fi integrato per connetterti a una rete wireless.
- Per XIAO ESP32S3 Sense, si può utilizzare il modulo Wi-Fi integrato o collegare un modulo Wi-Fi o Ethernet esterno utilizzando i pin disponibili.
- Per Raspberry Pi, si può utilizzare il modulo Wi-Fi integrato per collegare un modulo Wi-Fi o Ethernet esterno utilizzando il connettore disponibile.

Conclusione

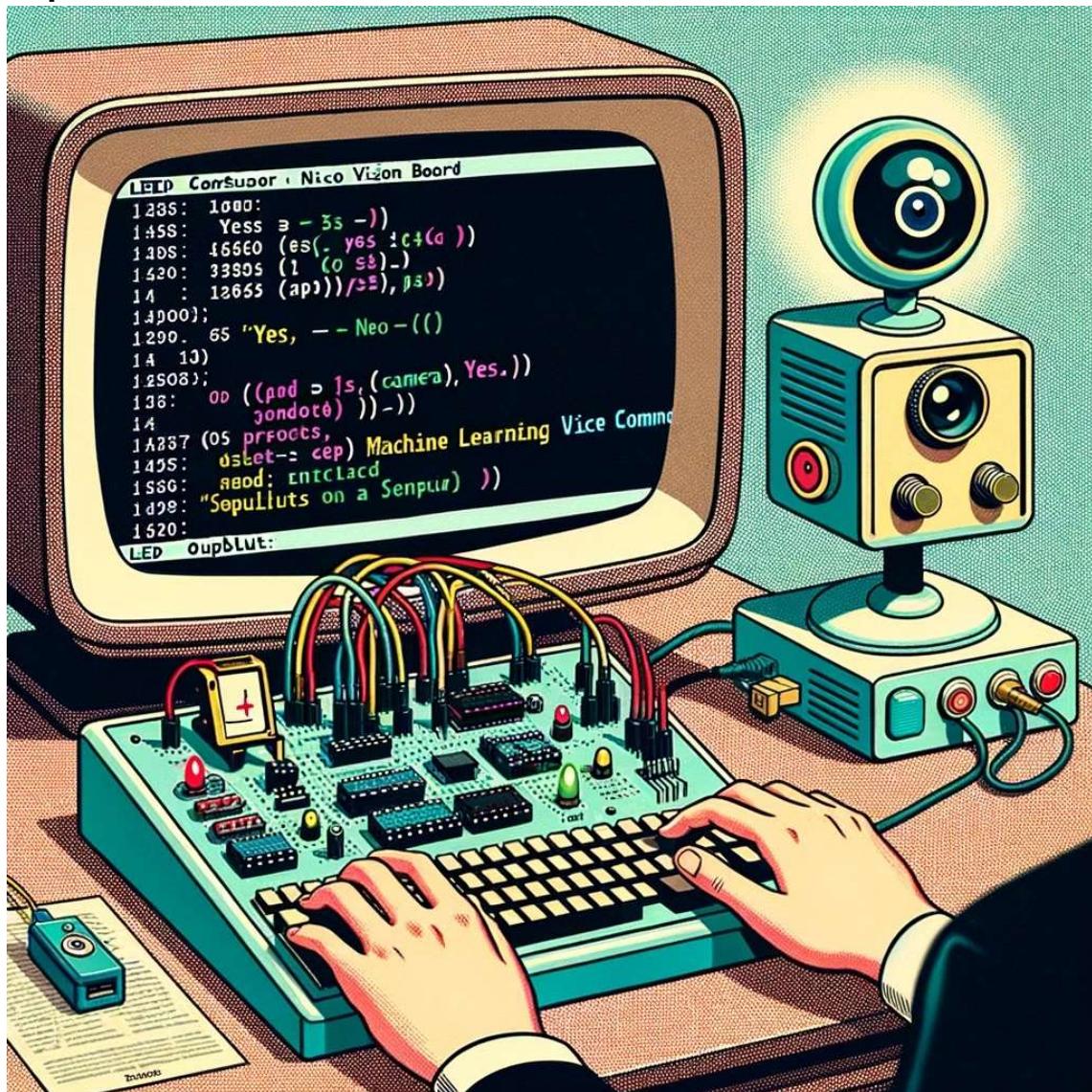
Con l'hardware e il software impostati, siamo pronti per intraprendere il viaggio di apprendimento automatico embedded. I laboratori pratici guideranno attraverso vari progetti, coprendo argomenti come classificazione delle immagini, rilevamento di oggetti, individuazione di parole chiave e classificazione del movimento.

Se si riscontrano problemi o ci sono domande, non esitate a consultare le guide alla risoluzione dei problemi o i forum o a cercare supporto dalla community.

Tuffiamoci e sblocchiamo il potenziale dell'apprendimento automatico su (tiny) sistemi reali!

Nicla Vision

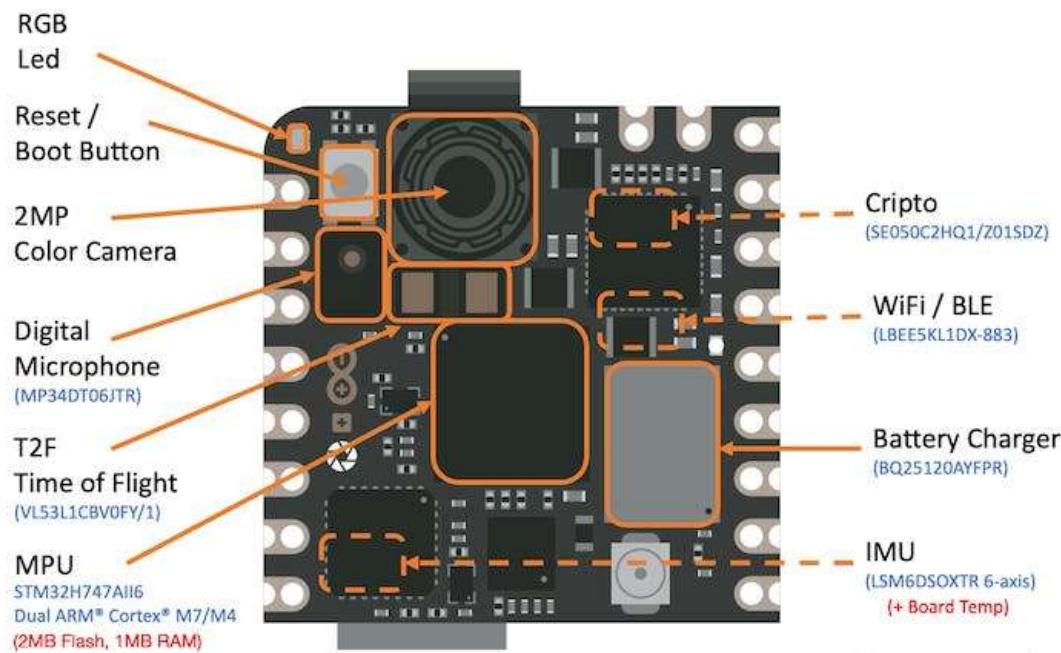
Setup



DALL-E 3 Prompt: Illustrazione che ricorda un cartone animato degli anni '50 in cui la scheda Arduino NICLA VISION, dotata di vari sensori tra cui una telecamera, è il punto focale di una scrivania in stile antico. Sullo sfondo, uno schermo di computer con bordi arrotondati visualizza l'IDE Arduino. Il codice visualizzato è correlato alle configurazioni LED e al rilevamento dei comandi vocali tramite apprendimento automatico. Gli output sul monitor seriale visualizzano esplicitamente le parole 'yes' e 'no'.

Panoramica

La [Arduino Nicla Vision](#) (a volte chiamata *NiclaV*) è una scheda di sviluppo che include due processori in grado di eseguire attività in parallelo. Fa parte di una famiglia di schede di sviluppo con lo stesso fattore di forma ma progettate per attività specifiche, come [Nicla Sense ME](#) e la [Nicla Voice](#). Le *Nicla* possono eseguire in modo efficiente processi creati con TensorFlow Lite. Ad esempio, uno dei core di NiclaV esegue un algoritmo di visione artificiale al volo (inferenza), mentre l'altro esegue operazioni di basso livello come il controllo di un motore e la comunicazione o l'azione come interfaccia utente. Il modulo wireless integrato consente la gestione simultanea della connettività WiFi e Bluetooth Low Energy (BLE).

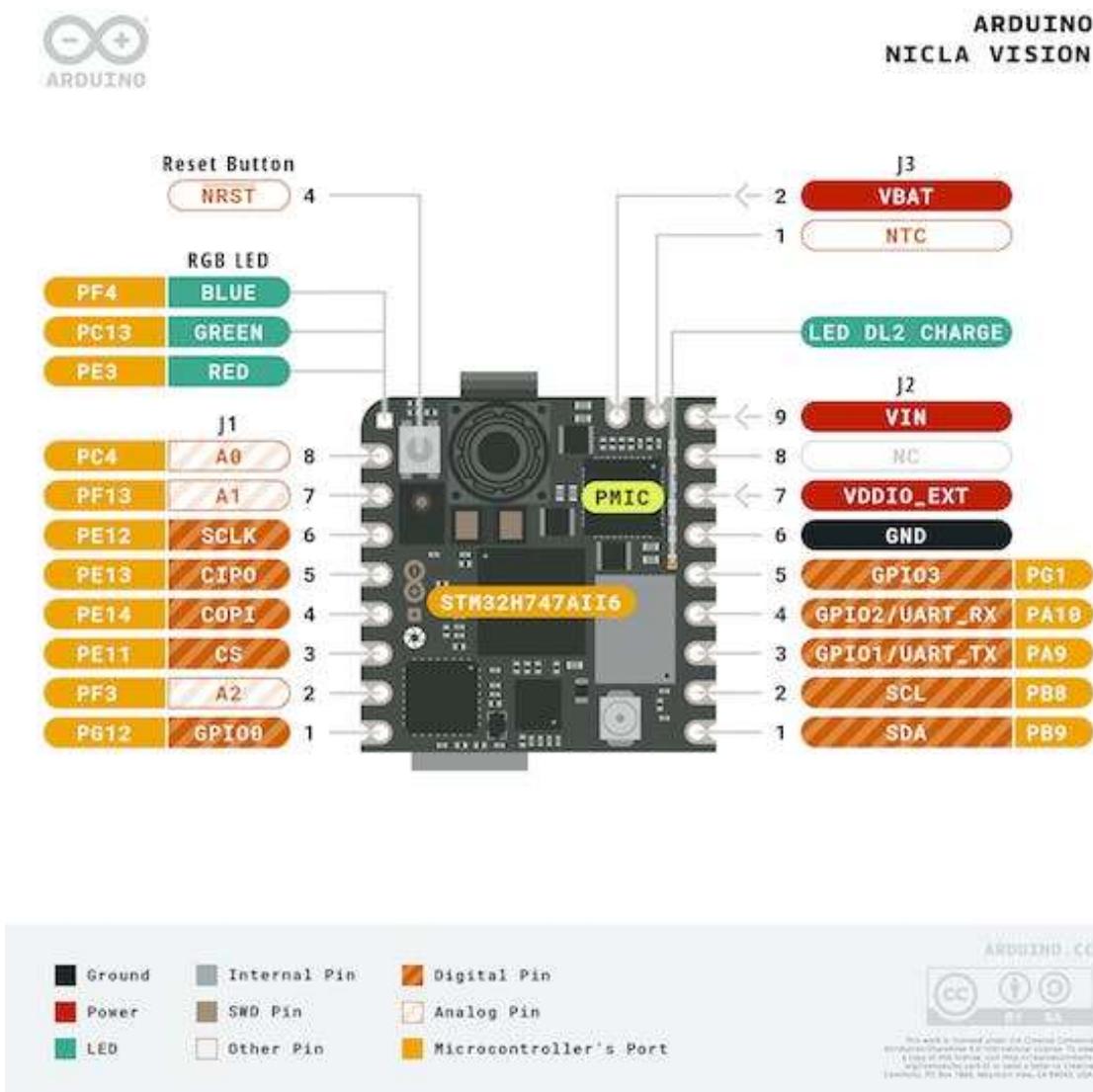


Hardware

Due Core Paralleli

Il processore centrale è il dual-core [STM32H747](#), che include un Cortex M7 at 480 MHz e un Cortex M4 at 240 MHz. I due core comunicano tramite un meccanismo di Remote Procedure Call che consente di richiamare senza problemi le funzioni sull'altro processore. Entrambi i processori condividono tutte le periferiche on-chip e possono eseguire:

- Sketch Arduino su Arm Mbed OS
- Applicazioni Native Mbed
- MicroPython / JavaScript tramite un interprete
- TensorFlow Lite



Memoria

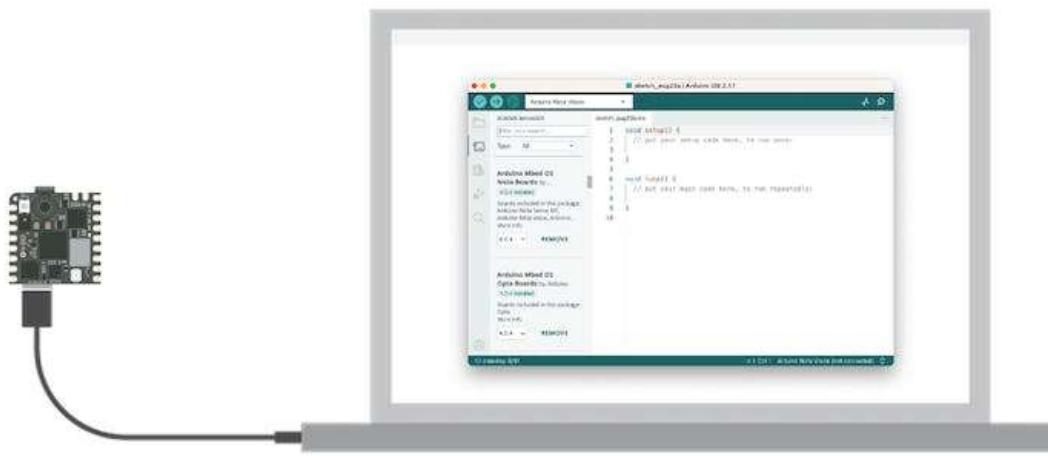
La memoria è fondamentale per i progetti di machine learning [apprendimento automatico] embedded. La scheda NiclaV può ospitare fino a 16 MB di QSPI Flash per l'archiviazione. Tuttavia, è essenziale considerare che la SRAM MCU è quella da utilizzare con le inferenze di machine learning; l'STM32H747 è di soli 1 MB, condiviso da entrambi i processori. Questa MCU ha anche incluso 2 MB di FLASH, principalmente per l'archiviazione del codice.

Sensori

- Fotocamera:** Una fotocamera CMOS a colori GC2145 da 2 MP.
- Microfono:** L'MP34DT05 è un microfono digitale MEMS omnidirezionale, ultracompatto, a basso consumo, costruito con un elemento di rilevamento capacitivo e l'interfaccia IC.
- IMU a 6 Assi:** Dati del giroscopio 3D e dell'accelerometro 3D dall'IMU a 6 assi LSM6DSOX.
- Sensore del Time of Flight:** Il sensore del tempo di volo VL53L1CBV0FY aggiunge capacità di misurazione precise e a bassa potenza alla Nicla Vision. Il laser invisibile VCSEL vicino all'infrarosso (incluso il driver analogico) è incapsulato con ottica ricevente in un piccolo modulo, tutto in uno, sotto la telecamera.

Installazione di Arduino IDE

Si inizia collegando la scheda (*microUSB*) al computer:



Si installa il core Mbed OS per le schede Nicla nell’IDE Arduino. Con l’IDE aperto, si va su Tools > Board > Board Manager, si cerca Arduino Nicla Vision nella finestra di ricerca e si installa la scheda.



Arduino IDE 1.18



Arduino IDE 2.X

Poi, si va su Tools > Board > Arduino Mbed OS Nicla Boards e si seleziona Arduino Nicla Vision. Con la scheda collegata alla porta USB, si dovrebbe vedere “Nicla on Port” e selezionarla.

Si apre lo sketch Blink su Examples/Basic ed lo si esegue usando il pulsante “IDE Upload”. Si dovrebbe vedere il LED integrato (RGB verde) lampeggiare, il che significa che la scheda Nicla è installata correttamente e funzionante!

Test del Microfono

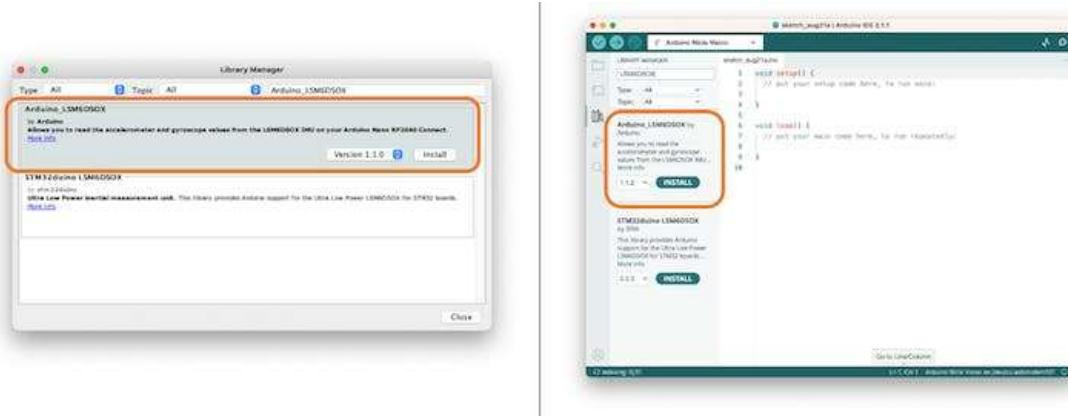
Su Arduino IDE, si va su Examples > PDM > PDMSerialPlotter, si apre e si esegue lo sketch. Si apre il Plotter e si guarda la rappresentazione audio dal microfono:



Variare la frequenza del suono generato e verificare che il microfono funzioni correttamente.

Test dell'IMU

Prima di testare l'IMU, sarà necessario installare la libreria LSM6DSOX. Per farlo, si vai su Library Manager e si cerca LSM6DSOX. Si installa la libreria fornita da Arduino:

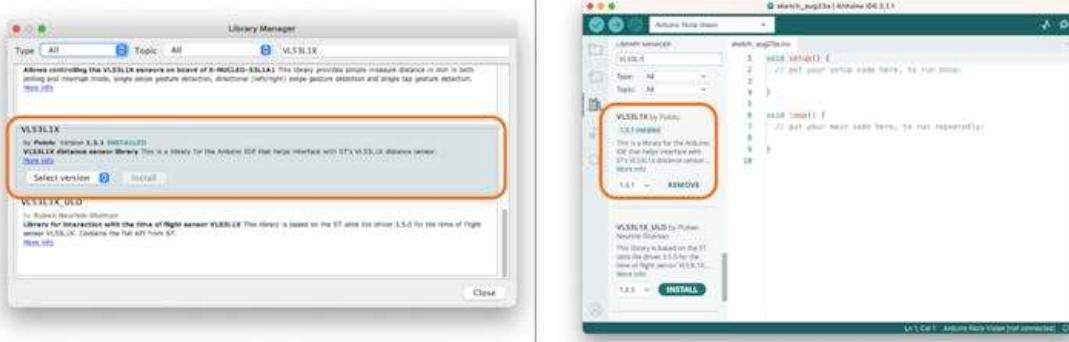


Poi, si va su `Examples > Arduino_LSM6DSOX > SimpleAccelerometer` e si esegue il test dell'accelerometro (si può anche eseguire Gyro e temperatura della scheda):



Test del sensore ToF (Time of Flight) [tempo di volo]

Come fatto con l'IMU, è necessario installare la libreria ToF VL53L1X. Per farlo, si va su Library Manager e si cerca VL53L1X. Si installa la libreria fornita da Pololu:



Poi, si esegue lo sketch `proximity_detection.ino`:

The screenshot shows the Arduino IDE interface. The main window displays the code for 'proximity_detection.ino'. The code initializes a VL53L1X sensor, sets up serial communication at 115200 baud, and configures the sensor for continuous distance measurement. It also sets up an LED pin (D0) for blinking based on the sensor's reading. The 'Serial Monitor' window is open below, with a red box highlighting the message input field and the message history area. The message history shows three messages: '14:54:25.691 => 1434', '14:54:25.726 => 1269', and '14:54:25.726 => 1150'. The status bar at the bottom indicates 'Line 1, Col 1' and 'Arduino Nicla Vision on /dev/cu.usbmodem101'.

```
proximity_detection.ino
1 #include "VL53L1X.h"
2 VL53L1X proximity;
3
4 bool blinkState = false;
5 int reading = 0;
6 int timeStart = 0;
7 int blinkTime = 2000;
8
9 void setup() {
10   Serial.begin(115200);
11   Wire1.begin();
12   Wire1.setClock(400000); // use 400 kHz I2C
13   proximity.setBus(&Wire1);
14
15   pinMode(LED0, OUTPUT);
16   digitalWrite(LED0, blinkState);
17
18   if (!proximity.init()) {
19     Serial.println("Failed to detect and initialize sensor!");
20     while (1);
21   }
22
23   proximity.setDistanceMode(VL53L1X::Long);
24   proximity.setMeasurementTimingBudget(10000);
25   proximity.startContinuous(10);
26 }
27
28 void loop() {
29   reading = proximity.read();
30   Serial.println(reading);
31
32   if (millis() - timeStart >= reading) {
33     digitalWrite(LED0, blinkState);
34     timeStart = millis();
35
36     blinkState = !blinkState;
37   }
38 }
```

Sul monitor seriale, si vedrà la distanza dalla telecamera di un oggetto di fronte ad essa (max 4 m).



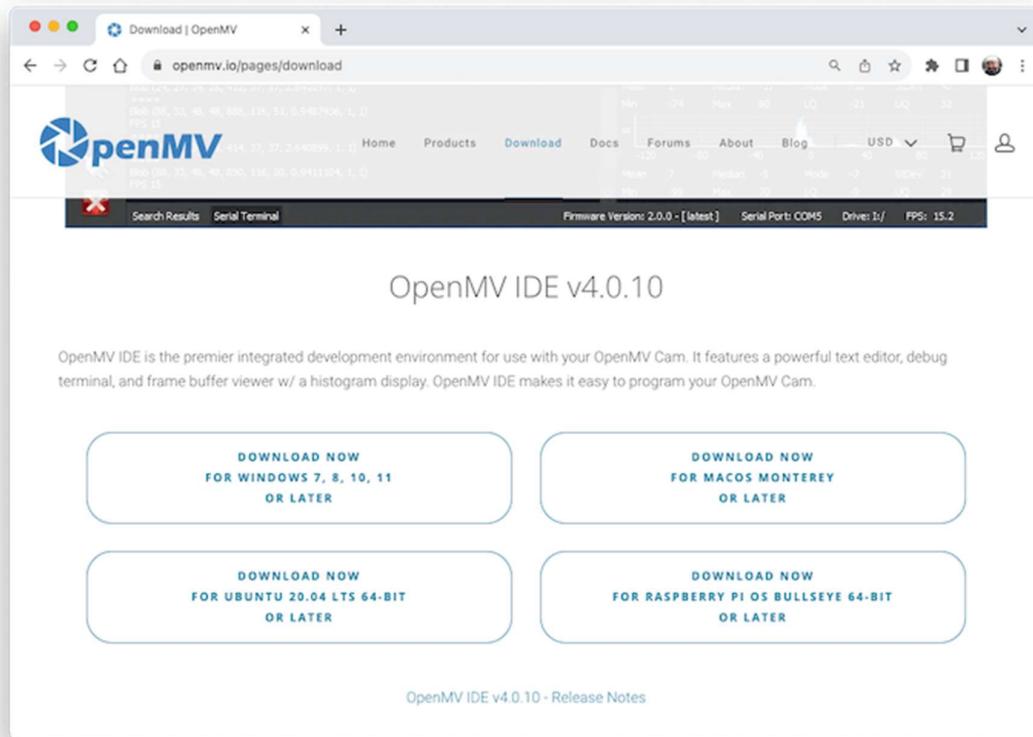
Test della Fotocamera

Possiamo anche testare la fotocamera utilizzando, ad esempio, il codice fornito in [Examples > Camera > CameraCaptureRawBytes](#). Non possiamo vedere l'immagine direttamente, ma è possibile ottenere i dati “crudi” dell'immagine generati dalla telecamera. In ogni caso, il test migliore con la telecamera è vedere un'immagine dal vivo. Per questo, useremo un altro IDE, OpenMV.

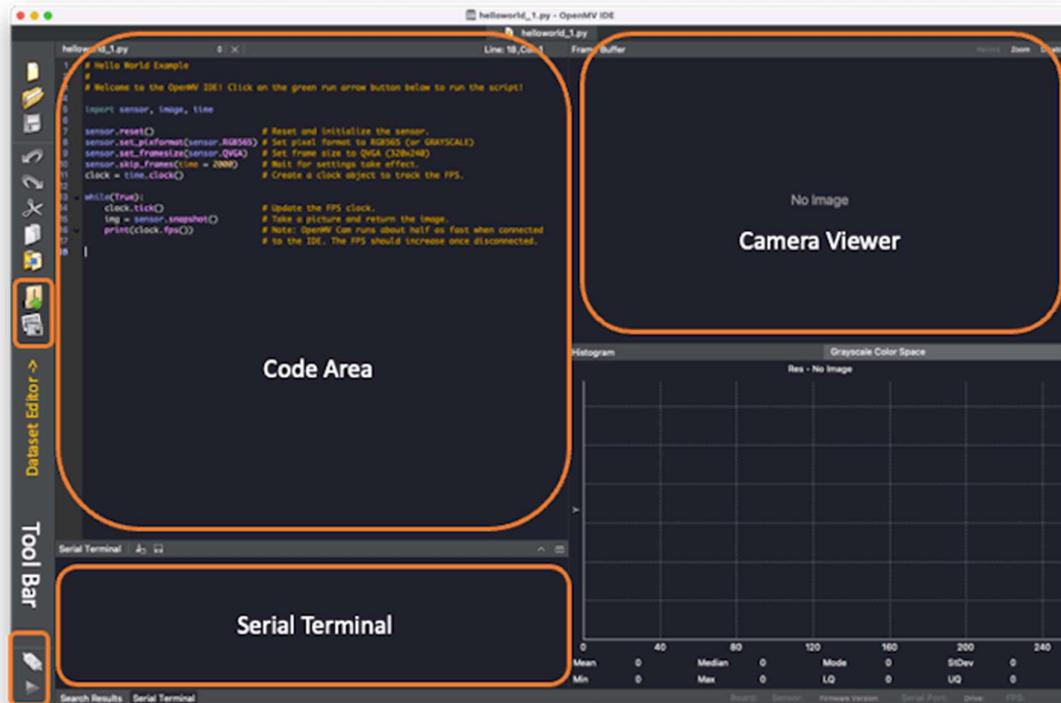
Installazione di OpenMV IDE

OpenMV IDE è il principale ambiente di sviluppo integrato con le telecamere OpenMV come quella su Nicla Vision. È dotato di un potente editor di testo, terminale di debug e visualizzatore di frame buffer con visualizzazione di istogrammi. Utilizzeremo MicroPython per programmare la telecamera.

Si va alla pagina di [OpenMV IDE](#), si scarica la versione corretta per il proprio sistema operativo e si seguono le istruzioni per l'installazione sul computer.



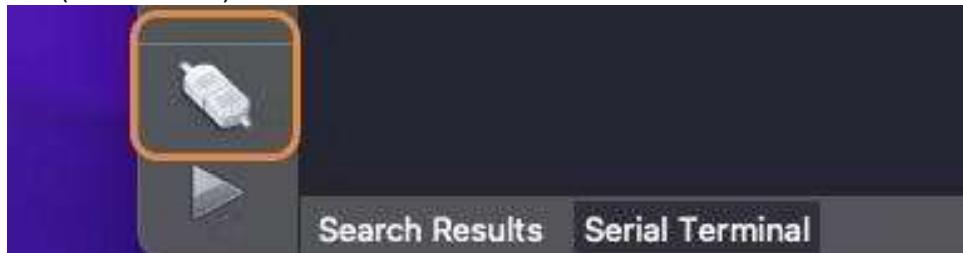
L'IDE dovrebbe aprirsi, con il codice `helloworld_1.py` predefinito nella sua “Code Area”. In caso contrario, lo si può aprire da `Files > Examples > HelloWord > helloworld.py`



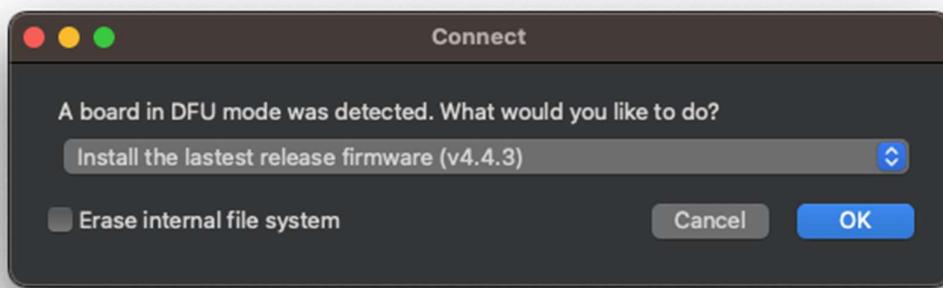
Tutti i messaggi inviati tramite una connessione seriale (utilizzando `print()` o i messaggi di errore) verranno visualizzati sul **Serial Terminal** durante l'esecuzione. L'immagine catturata da una telecamera verrà visualizzata nell'area **Camera Viewer** (o Frame Buffer) e nell'area Histogram immediatamente sotto Camera Viewer.

Prima di collegare la Nicla all'IDE OpenMV, si deve avere la versione più recente del bootloader. Si va all'IDE Arduino, si seleziona la scheda Nicla e si apre lo sketch su Examples > STM_32H747_System STM32H747_manageBootloader. Upload-are il codice sulla board. Il Serial Monitor vi guiderà.

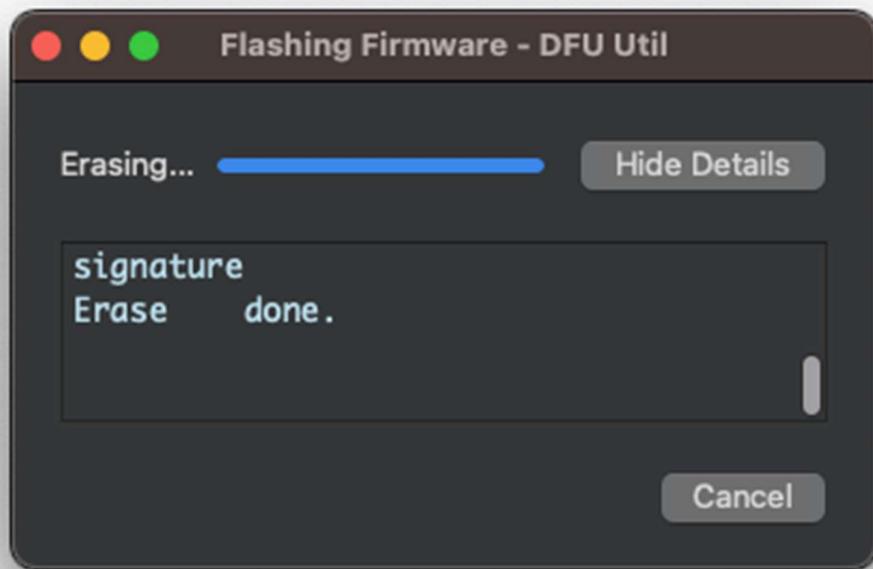
Dopo aver aggiornato il bootloader, si mette la Nicla Vision in modalità bootloader premendo due volte il pulsante di reset sulla scheda. Il LED verde integrato inizierà a spegnersi e accendersi in dissolvenza [fading]. Ora si torna all'IDE OpenMV e si clicca sull'icona di connessione (Left ToolBar):



Un pop-up informerà che è stata rilevata una scheda in modalità DFU e chiederà come si desidera procedere. Per prima cosa, si seleziona Install the latest release firmware (vX.Y.Z). Questa azione installerà l'ultimo firmware OpenMV su Nicla Vision.



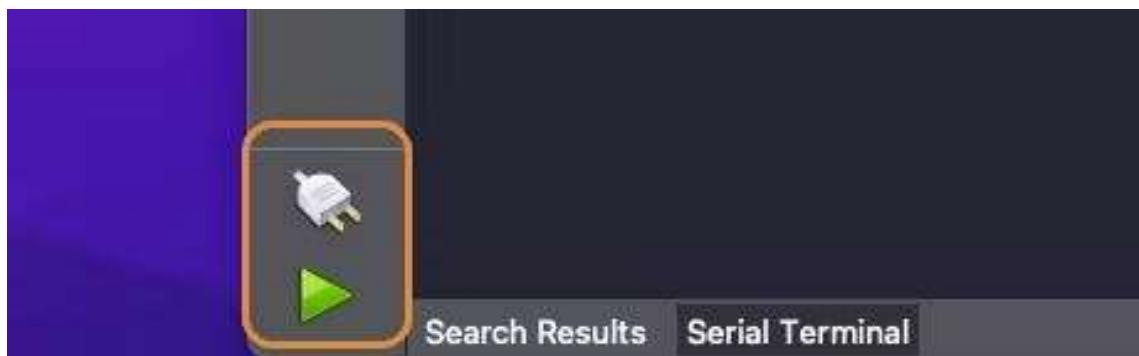
Si può lasciare l'opzione Erase internal file system deselezionata e cliccare su [OK]. Il LED verde di Nicla inizierà a lampeggiare mentre il firmware OpenMV viene caricato sulla scheda e si aprirà una finestra del terminale che mostrerà l'avanzamento del flashing.



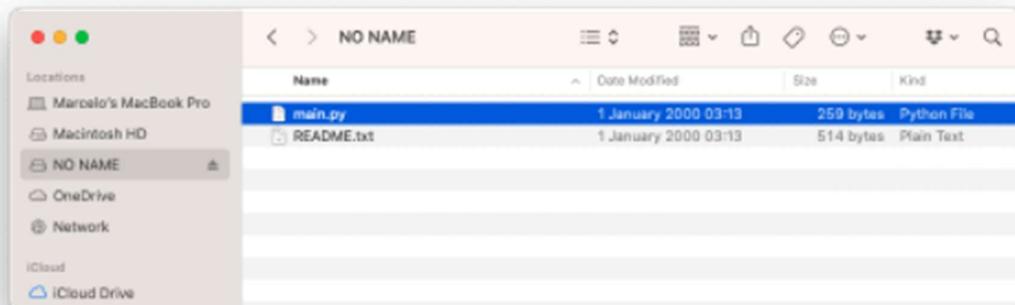
Attendere che il LED verde smetta di lampeggiare in dissolvenza. Quando il processo termina, si vedrà un messaggio che dice “DFU firmware update complete!”. Premere [OK].



Quando Nicla Vison si collega alla Barra degli Strumenti, compare un pulsante verde per la riproduzione.



Notare che sul computer apparirà un'unità denominata “NO NAME”:



Ogni volta che si preme il pulsante [RESET] sulla board, viene automaticamente eseguito lo script *main.py* che vi è memorizzato. Si può caricare il codice di *main.py* sull’IDE (*File > Open File...*).

```

1 # main.py -- put your code here!
2 import pyb, time
3 led = pyb.LED(3)  <-- Blue LED *
4 usb = pyb.USB_VCP()
5 while (usb.isconnected() == False):
6     led.on()
7     time.sleep_ms(150)
8     led.off()
9     time.sleep_ms(100)
10    led.on()
11    time.sleep_ms(150)
12    led.off()
13    time.sleep_ms(600)
14

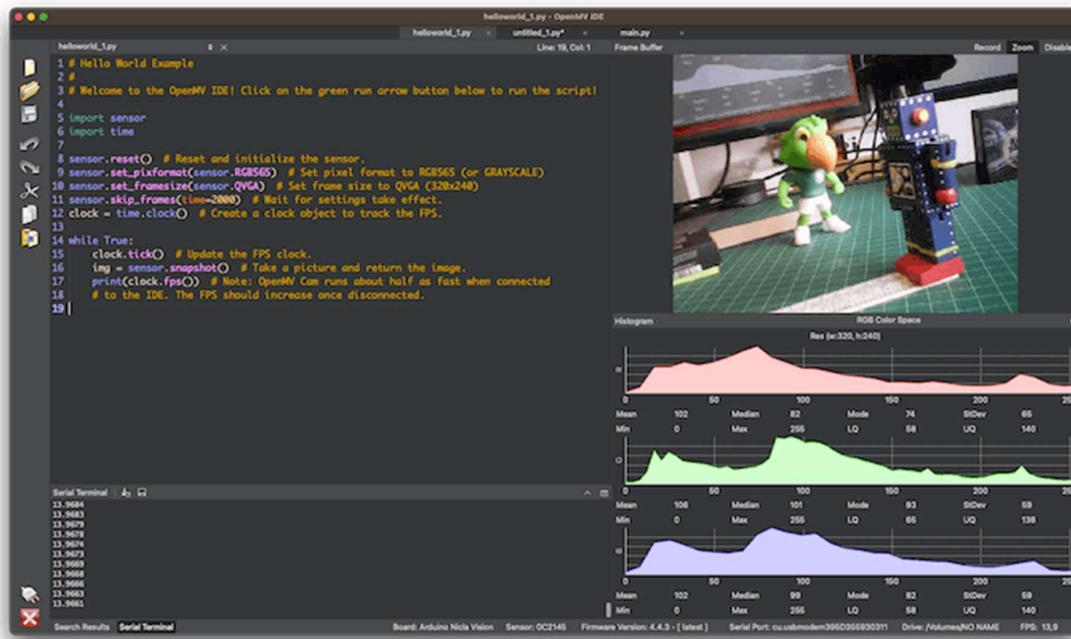
```

* LED(1) : Red
 LED(2) : Green
 LED(3) : Blue

Questo è il codice di “Blink”, che conferma che l’HW è OK.

Per testare la telecamera, eseguiamo *helloworld_1.py*. Per farlo, si seleziona lo script su [File > Examples > HelloWorld > helloworld.py](#),

Quando si clicca sul pulsante di riproduzione verde, lo script MicroPython (*helloworld.py*) nella “Code Area” verrà caricato ed eseguito su Nicla Vision. Sul “Camera Viewer”, si vedrà lo streaming video. Il “Serial Monitor” ci mostrerà gli FPS (fotogrammi al secondo), che dovrebbero essere circa 14 fps.



Ecco lo script `helloworld.py`:

```

# Hello World Example 2
#
# Welcome to the OpenMV IDE! Click on the green run arrow button below to run the script!

import sensor, image, time

sensor.reset()                      # Reset and initialize the sensor.
sensor.set_pixformat(sensor.RGB565)   # Set pixel format to RGB565 (or GRayscale)
sensor.set_framesize(sensor.QVGA)     # Set frame size to QVGA (320x240)
sensor.skip_frames(time = 2000)       # Wait for settings take effect.
clock = time.clock()                # Create a clock object to track the FPS.

while(True):
    clock.tick()                    # Update the FPS clock.
    img = sensor.snapshot()         # Take a picture and return the image.
    print(clock.fps())

```

In [GitHub](#), si trovano gli script Python utilizzati qui.

Il codice può essere suddiviso in due parti:

- **Setup:** Dove le librerie vengono importate, inizializzate e le variabili vengono definite e inizializzate.
- **Loop:** (ciclo while) parte del codice che viene eseguita continuamente. L’immagine (la variabile `img`) viene catturata (un frame). Ognuno di questi frame può essere utilizzato per l’inferenza nelle Applicazioni di Apprendimento Automatico.

Per interrompere l’esecuzione del programma, premere il pulsante rosso [X].

Nota: OpenMV Cam funziona a circa la metà della velocità quando è connesso all’IDE. Gli FPS dovrebbero aumentare una volta disconnessi.

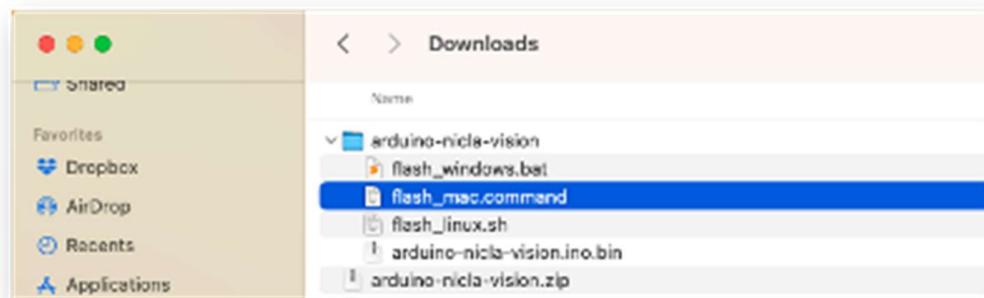
In [GitHub](#), si trovano altri script Python. Provare a testare i sensori di bordo.

Collegamento di Nicla Vision a Edge Impulse Studio

Avremo bisogno di Edge Impulse Studio più avanti in altri esercizi. [Edge Impulse](#) è una piattaforma di sviluppo leader per il machine learning [apprendimento automatico] su dispositivi edge.

Edge Impulse supporta ufficialmente Nicla Vision. Quindi, per iniziare, si crea un nuovo progetto su Studio e vi si collega Nicla. Per farlo, si seguono i passaggi:

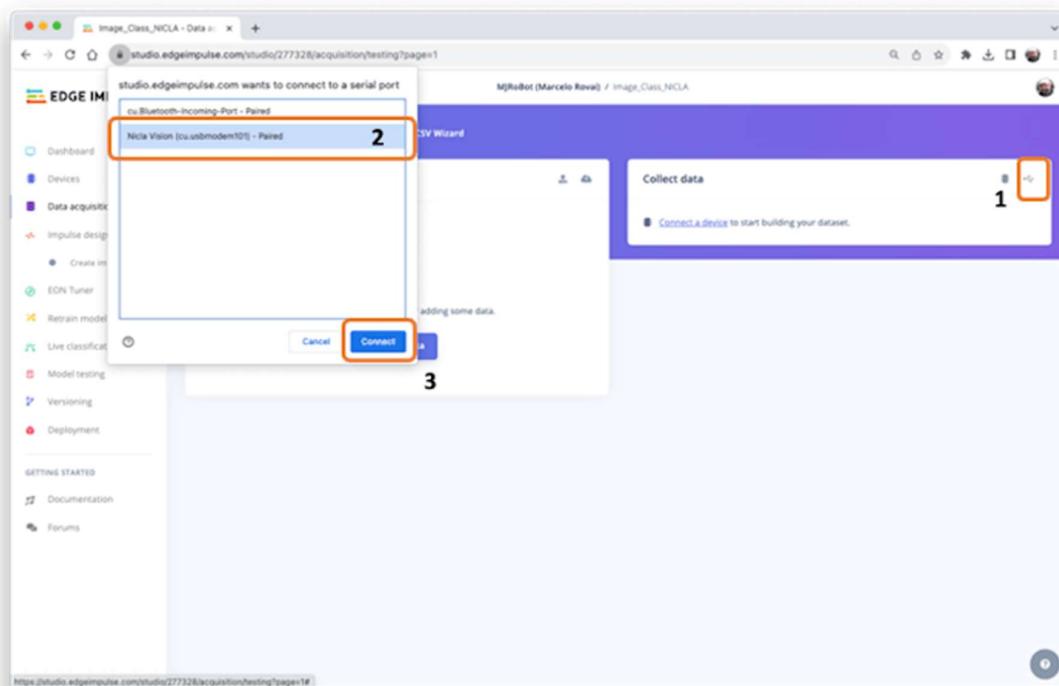
- Scaricare l'[EI Firmware](#) più aggiornato e decomprimere lo.
- Aprire il file zip sul computer e selezionare l'uploader corrispondente al sistema operativo:



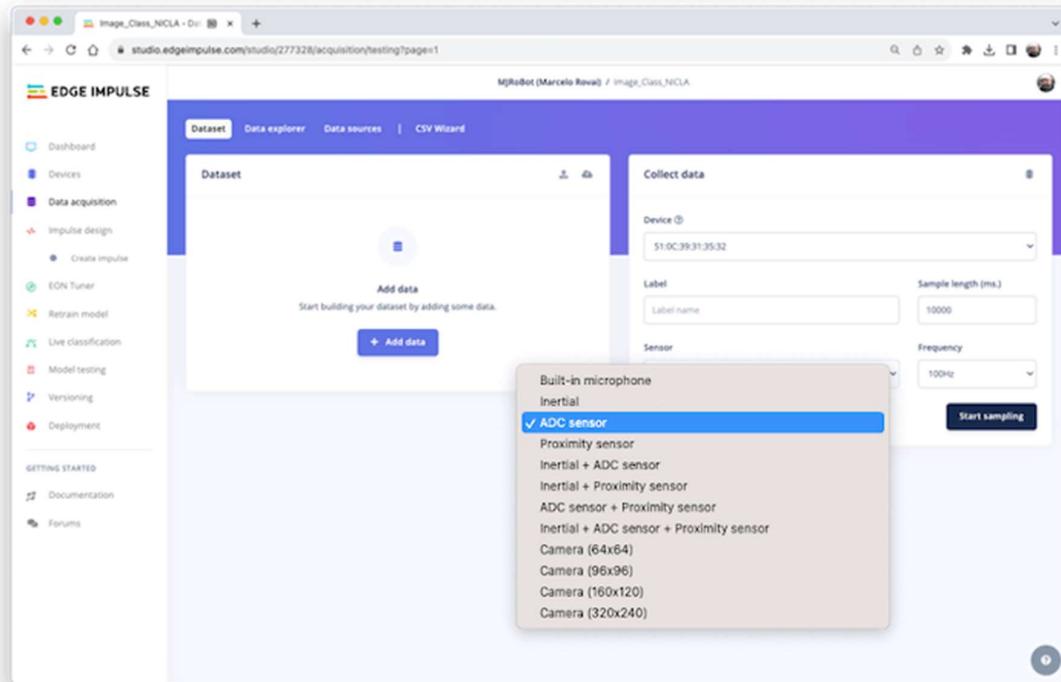
- Mettere Nicla-Vision in “Boot Mode”, premendo due volte il pulsante di reset.
- Eseguire il codice batch specifico per il proprio sistema operativo per caricare il binario `arduino-nicla-vision.bin` sulla board.

Andare sul progetto sullo Studio e nella scheda `Data Acquisition tab`, selezionare `WebUSB` (1).

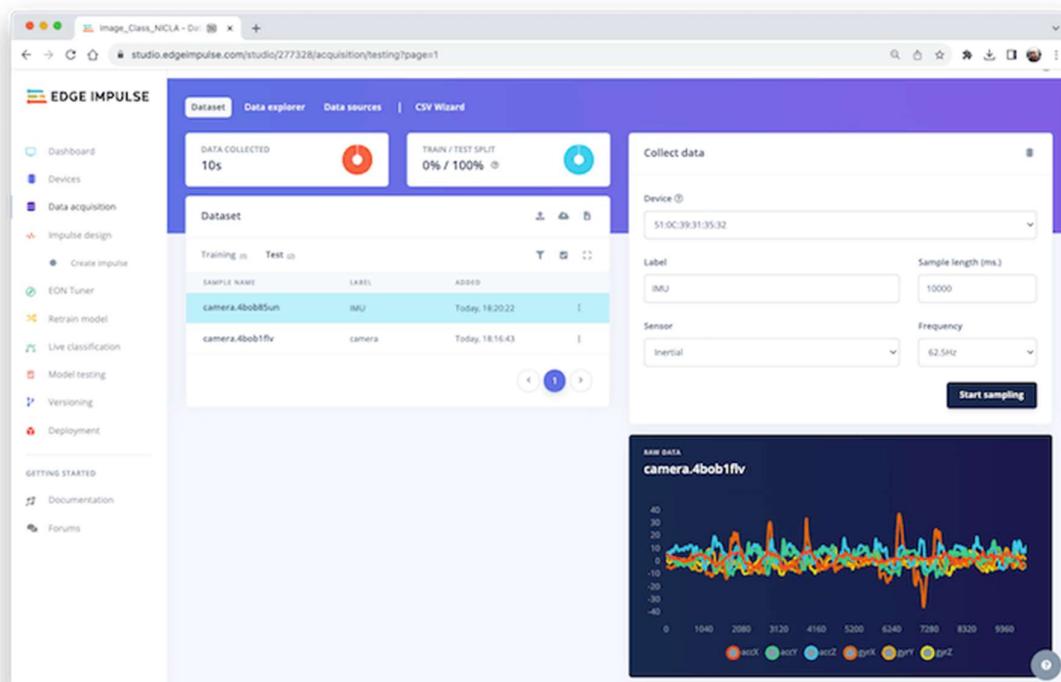
Si aprirà una finestra; scegliere l'opzione che mostra che `Nicla is paired` (2) e premere `[Connect]` (3).



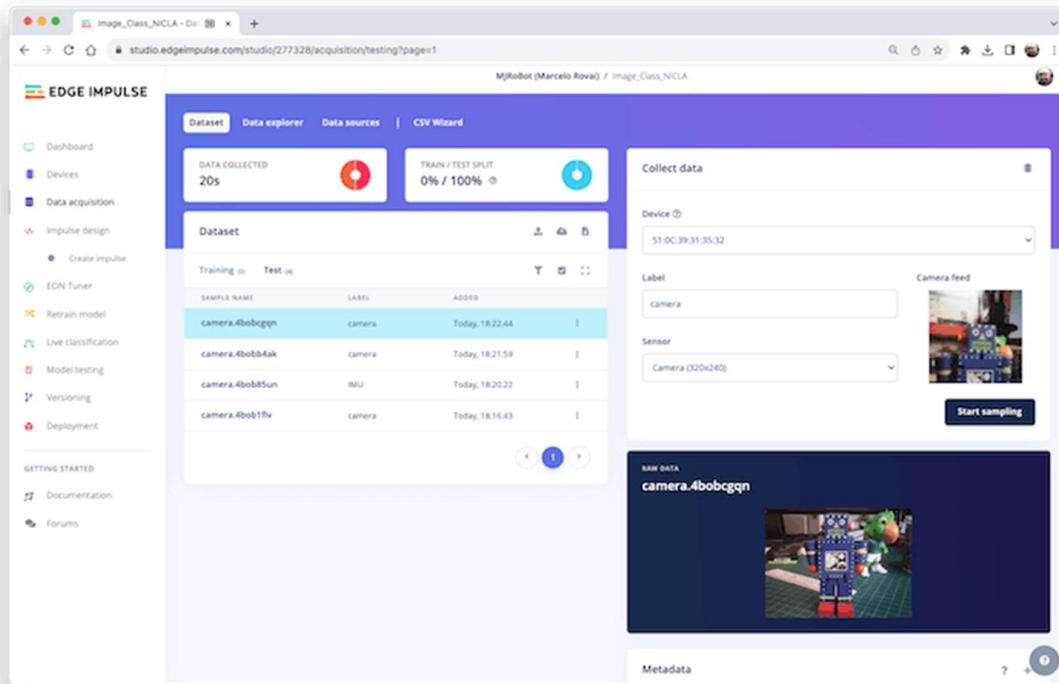
Nella sezione `Collect Data` della scheda `Data Acquisition`, si possono scegliere quali dati del sensore raccogliere.



Per esempio. IMU data:



O Image (Camera):



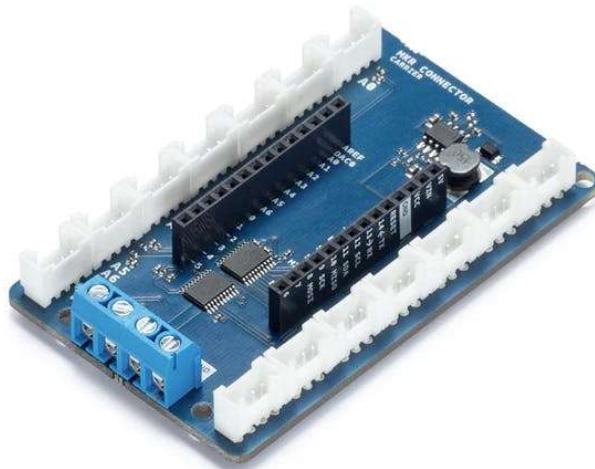
E così via. Si può anche testare un sensore esterno collegato all'[ADC](#) (Nicla pin 0) e agli altri sensori a bordo, come il microfono e il ToF.

Espansione della Nicla Vision Board (opzionale)

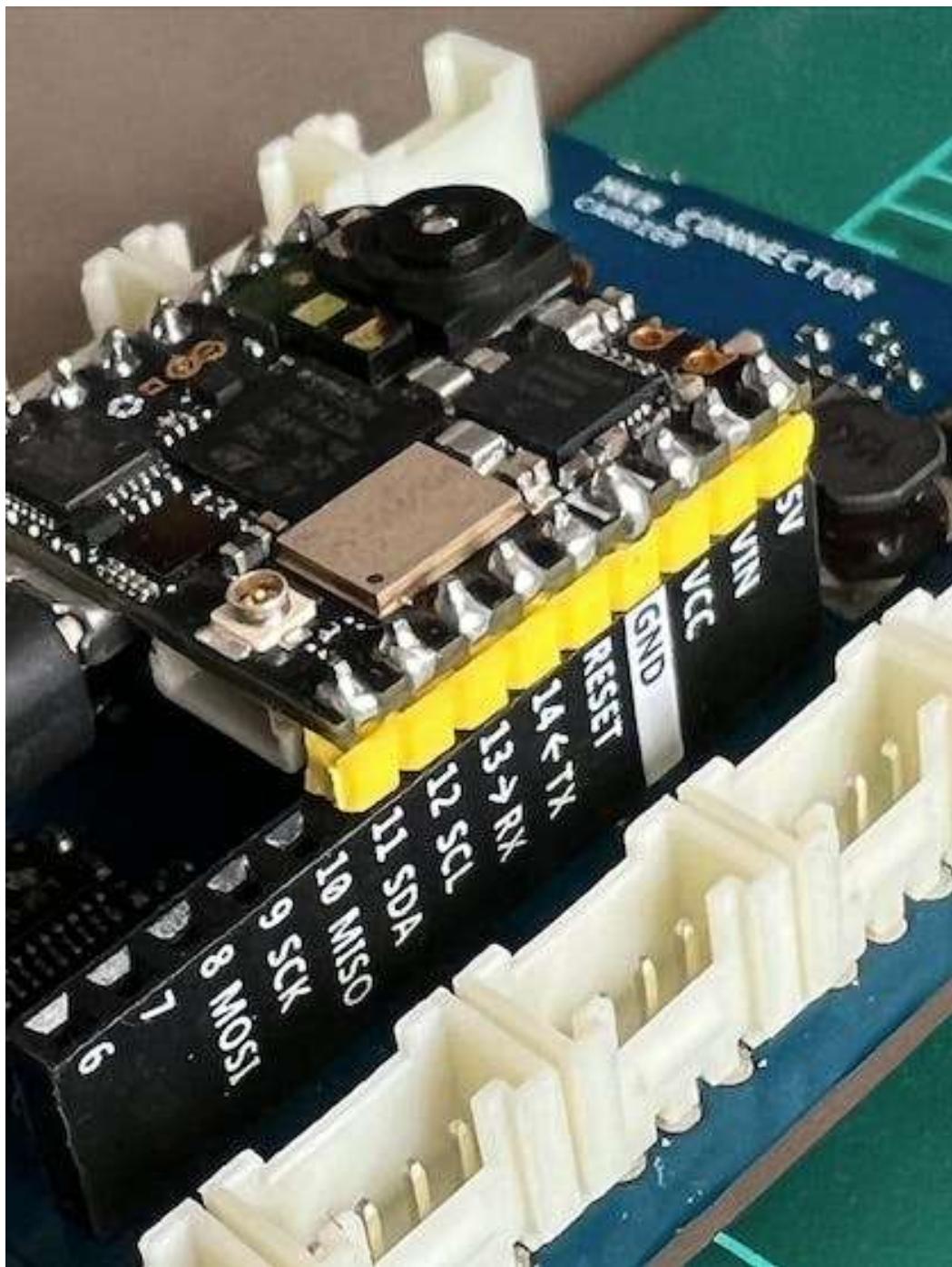
Un ultimo elemento da esplorare è che a volte, durante la prototipazione, è essenziale sperimentare con sensori e dispositivi esterni, e un'eccellente espansione per Nicla è l'[Arduino MKR Connector Carrier \(Grove compatible\)](#).

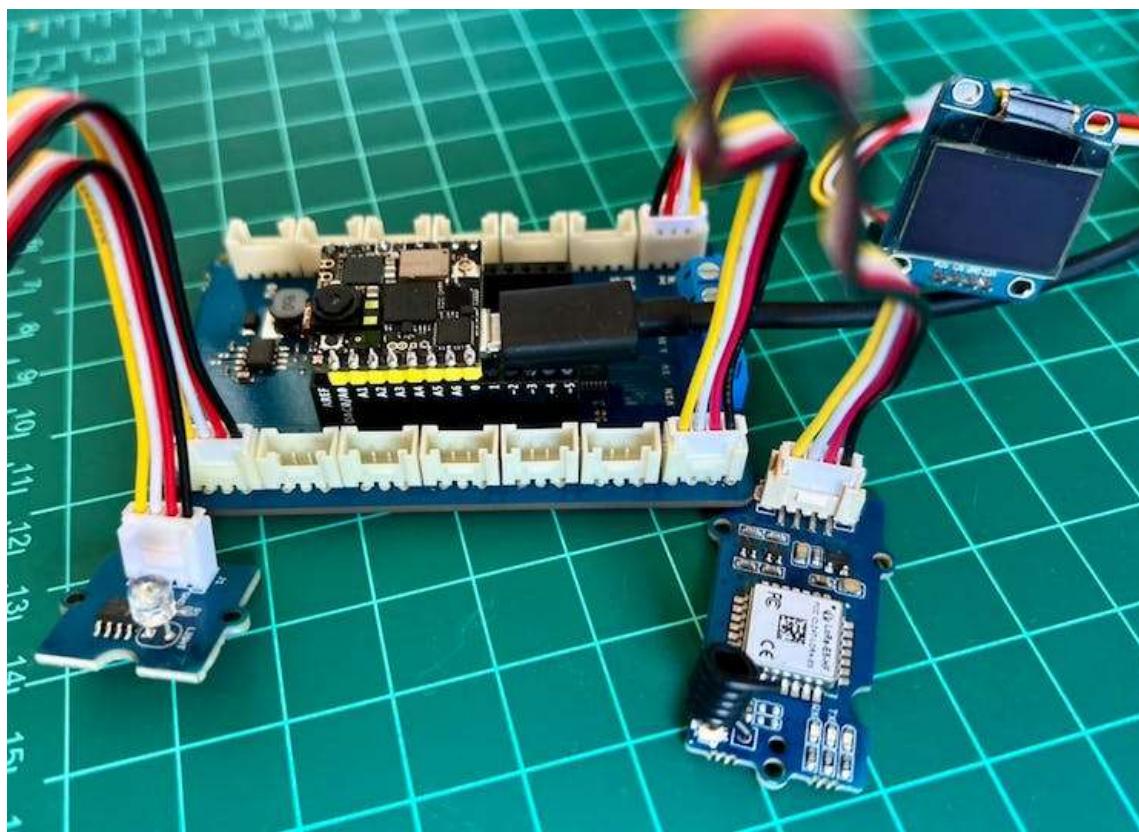
Lo shield ha 14 connettori Grove: cinque ingressi analogici singoli (A0-A5), un ingresso analogico doppio (A5/A6), cinque I/O digitali singoli (D0-D4), un I/O digitale doppio (D5/D6), una I2C (TWI) e una UART (Seriale). Tutti i connettori sono compatibili con 5 V.

Notare che tutti i 17 pin Nicla Vision saranno collegati agli Shield Grove, ma alcune connessioni Grove rimangono scollegate.



Questo shield è compatibile con MKR e può essere utilizzato con Nicla Vision e Portenta.





Il Grove Light Sensor è collegato a uno dei singoli pin analogici (A0/PC4), il LoRaWAN device alla UART e l'OLED al connettore della I2C.

I pin Nicla 3 (Tx) e 4 (Rx) sono collegati al connettore Serial Shield. La comunicazione UART è utilizzata con il dispositivo LoRaWan. Ecco un semplice codice per utilizzare l'UART:

```
# UART Test - By: marcelo_rovai - Sat Sep 23 2023

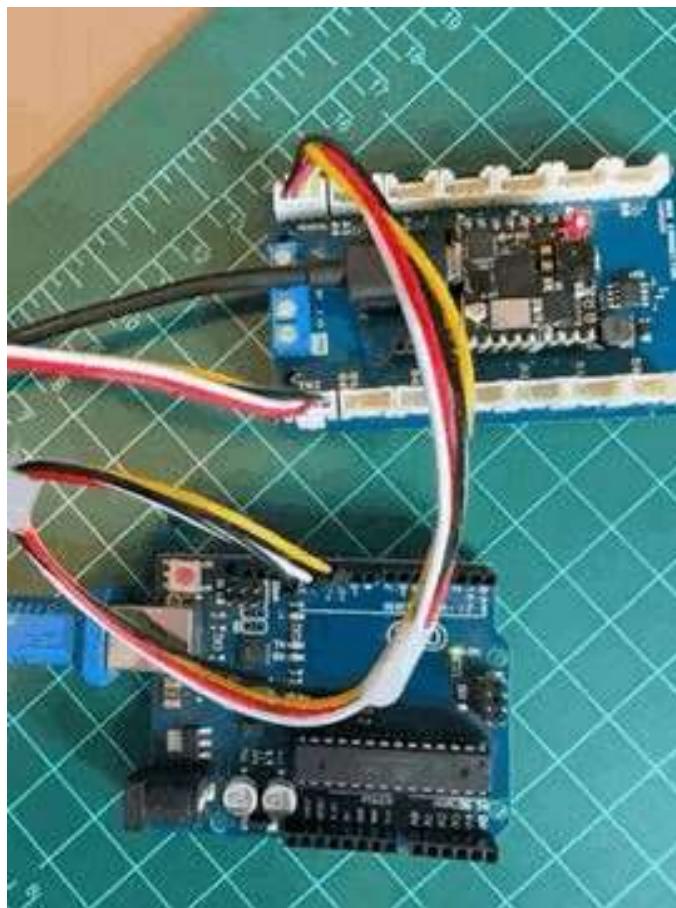
import time
from pyb import UART
from pyb import LED

redLED = LED(1) # built-in red LED

# Init UART object.
# Nicla Vision's UART (TX/RX pins) is on "LP1"
uart = UART("LP1", 9600)

while(True):
    uart.write("Hello World!\r\n")
    redLED.toggle()
    time.sleep_ms(1000)
```

Per verificare che l'UART funzioni, si deve, ad esempio, collegare un altro dispositivo come Arduino UNO, visualizzando “Hello Word” sul Serial Monitor. Ecco il codice.



Di seguito è riportato il codice *Hello World* da utilizzare con l'OLED I2C. Il driver MicroPython SSD1306 OLED (ssd1306.py), creato da Adafruit, dovrebbe essere caricato anche su Nicla (lo script ssd1306.py si trova su [GitHub](#)).

```
# Nicla_OLED_Hello_World - By: marcelo_rovai - Sat Sep 30 2023

#Save on device: MicroPython SSD1306 OLED driver, I2C and SPI interfaces created by Adafruit
import ssd1306

from machine import I2C
i2c = I2C(1)

oled_width = 128
oled_height = 64
oled = ssd1306.SSD1306_I2C(oled_width, oled_height, i2c)

oled.text('Hello, World', 10, 10)
oled.show()

Infine, ecco uno script semplice per leggere il valore ADC sul pin "PC4" (pin A0 di Nicla):
# Light Sensor (A0) - By: marcelo_rovai - Wed Oct 4 2023

import pyb
from time import sleep

adc = pyb.ADC(pyb.Pin("PC4"))      # create an analog object from a pin
val = adc.read()                  # read an analog value

while (True):

    val = adc.read()
    print ("Light={}".format (val))
    sleep (1)
```

L'ADC può essere utilizzato per altre variabili del sensore, come la Temperatura.

Notare che gli script sopra ([scaricati da Github](#)) presentano solo come collegare dispositivi esterni alla scheda Nicla Vision utilizzando MicroPython.

Conclusione

Arduino Nicla Vision è un eccellente *piccolo dispositivo* per usi industriali e professionali! Tuttavia, è potente, affidabile, a basso consumo e ha sensori adatti per le applicazioni di machine learning embedded più comuni come visione, movimento, “fusion” di sensori e suono.

Nel [repository GitHub](#), si trova l’ultima versione di tutti i codici utilizzati o commentati in questo esercizio pratico.

Risorse

- [Codici Micropython](#)
- [Codici Arduino](#)

Classificazione delle Immagini



DALL-E 3 Prompt: Fumetto in stile anni '50 raffigurante un dispositivo elettronico compatto con un modulo telecamera posizionato su un tavolo di legno. Lo schermo mostra un robot blu da un lato e dei pappagallini verdi dall'altro. Le luci LED sul dispositivo indicano le classificazioni, mentre i personaggi in abiti retrò osservano con interesse.

Panoramica

Mentre iniziamo i nostri studi sul machine learning embedded o TinyML, è impossibile ignorare l'impatto trasformativo della Computer Vision (CV) e dell'Intelligenza Artificiale (IA) nelle nostre vite. Queste due discipline interconnesse ridefiniscono ciò che le macchine possono percepire e realizzare, dai veicoli autonomi e dalla robotica all'assistenza sanitaria e alla sorveglianza.

Sempre di più, ci troviamo di fronte a una rivoluzione dell'intelligenza artificiale (IA) in cui, come affermato da Gartner, **Edge AI** ha un potenziale di impatto molto elevato, ed è ora!



Nel "centro" del Radar c'è la *Edge Computer Vision*, e quando parliamo di Machine Learning (ML) applicato alla visione, la prima cosa che viene in mente è la **Classificazione delle immagini**, una specie di "Hello World" di ML!

Questo esercizio esplorerà un progetto di computer vision che utilizza Convolutional Neural Network (CNN) [Reti Neurali Convoluzionali] per la classificazione delle immagini in tempo reale. Sfruttando il robusto ecosistema di TensorFlow, implementeremo un modello MobileNet pre-addestrato e lo adatteremo per il "deployment" edge. L'attenzione sarà rivolta all'ottimizzazione del modello per un'esecuzione efficiente su hardware con risorse limitate senza sacrificare l'accuratezza.

Utilizzeremo tecniche come la quantizzazione e la potatura per ridurre il carico computazionale. Alla fine di questo tutorial, si avrà un prototipo funzionante in grado di

classificare le immagini in tempo reale, il tutto in esecuzione su un sistema embedded a basso consumo basato sulla scheda Arduino Nicla Vision.

Visione Artificiale

In sostanza, la visione artificiale consente alle macchine di interpretare e prendere decisioni sulla base di dati visivi provenienti dal mondo esterno, imitando sostanzialmente la capacità del sistema ottico umano. Al contrario, l'intelligenza artificiale è un campo più ampio che comprende il machine learning [apprendimento automatico], elaborazione del linguaggio naturale e robotica, tra le altre tecnologie. Quando si introducono algoritmi di IA nei progetti di visione artificiale, si potenzia la capacità del sistema di comprendere, interpretare e reagire agli stimoli visivi.

Quando si parla di progetti di visione artificiale applicati a dispositivi embedded, le applicazioni più comuni che vengono in mente sono *Classificazione delle Immagini* e *Rilevamento degli Oggetti*.

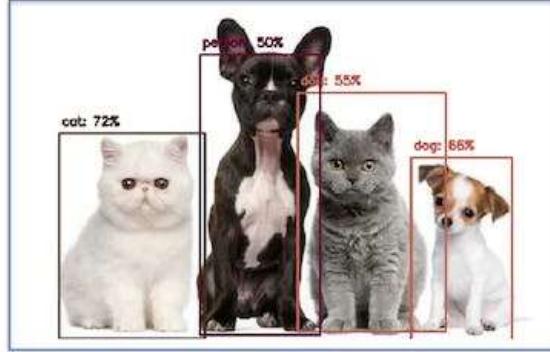
Image Classification
(Multi-Class Classification)



Cat: 70%

Dog: 80%

Object Detection
Multi-Label Classification + Object Localization



Entrambi i modelli possono essere implementati su dispositivi minuscoli come Arduino Nicla Vision e utilizzati in progetti reali. In questo capitolo, parleremo della classificazione delle immagini.

Obiettivo del Progetto di Classificazione delle Immagini

Il primo passo in qualsiasi progetto ML è definire l'obiettivo. In questo caso, è rilevare e classificare due oggetti specifici presenti in un'immagine. Per questo progetto, utilizzeremo due piccoli giocattoli: un *robot* e un piccolo pappagallo brasiliano (chiamato *Periquito*). Inoltre, raccoglieremo immagini di un *background* in cui quei due oggetti sono assenti.

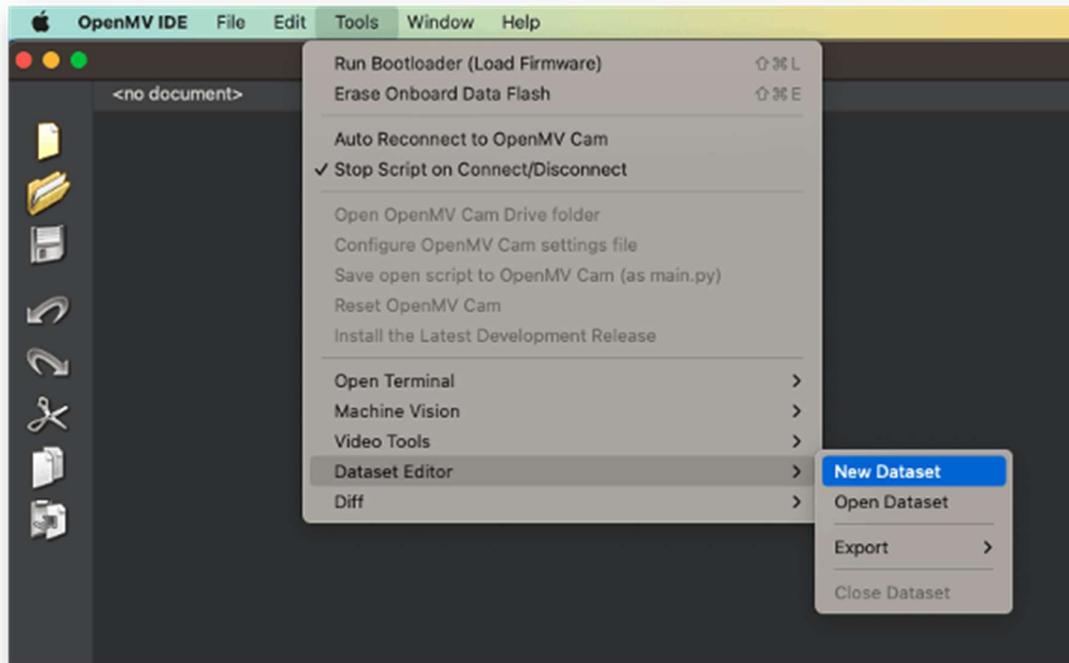


Raccolta Dati

Una volta definito l’obiettivo del progetto di Machine Learning, il passaggio successivo e più cruciale è la raccolta del set di dati. Si può utilizzare Edge Impulse Studio, l’IDE OpenMV che abbiamo installato o persino il proprio telefono per l’acquisizione delle immagini. Qui, utilizzeremo l’IDE OpenMV per questo.

Raccolta del Dataset con OpenMV IDE

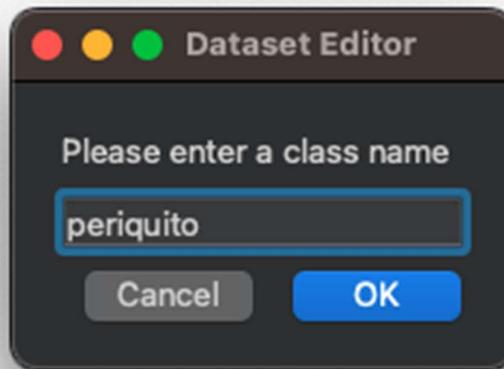
Per prima cosa, si crea sul computer una cartella in cui verranno salvati i dati, ad esempio “data”. Quindi, su OpenMV IDE, si va in Tools > Dataset Editor e si seleziona New Dataset per avviare la raccolta di dati:



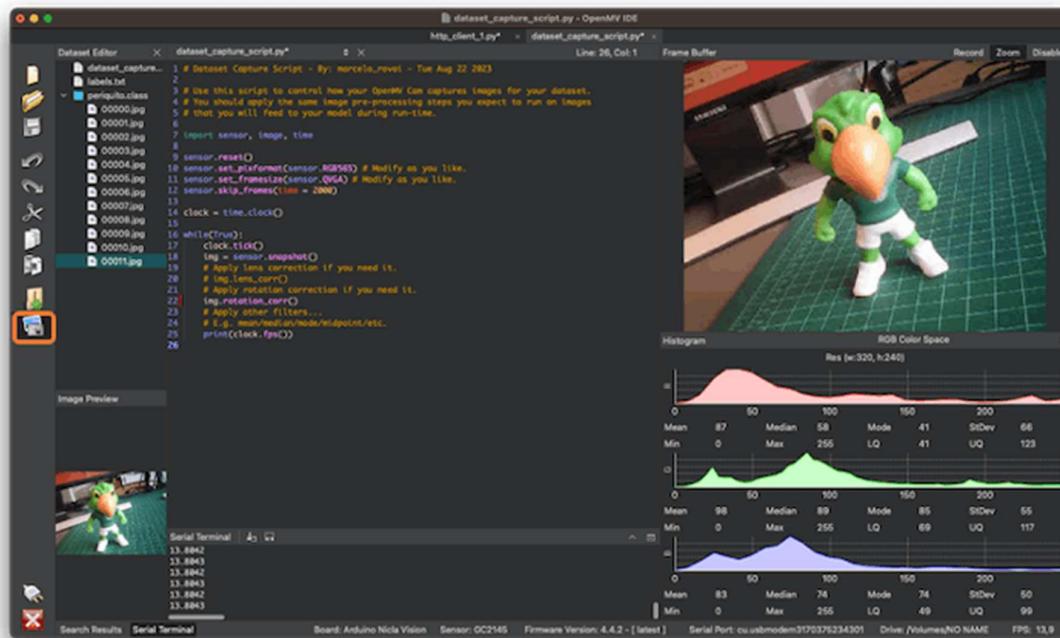
L’IDE chiederà di aprire il file in cui verranno salvati i dati e di scegliere la cartella “data” che è stata creata. Notare che appariranno nuove icone sul pannello di sinistra.



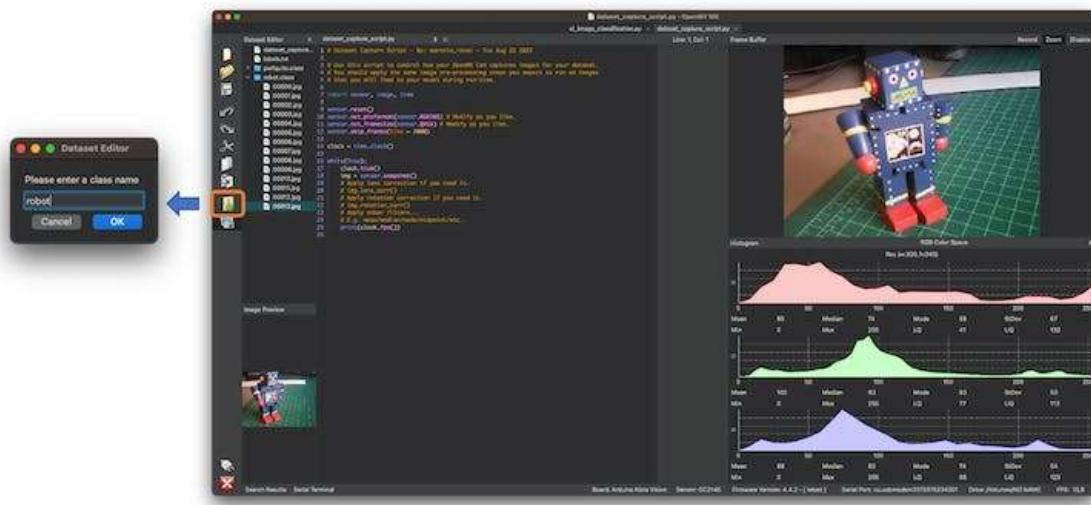
Utilizzando l'icona in alto (1), si inserisce il nome della prima classe, ad esempio “periquito”:



Eseguendo `dataset_capture_script.py` e cliccando sull'icona della fotocamera (2), inizierà l'acquisizione delle immagini:



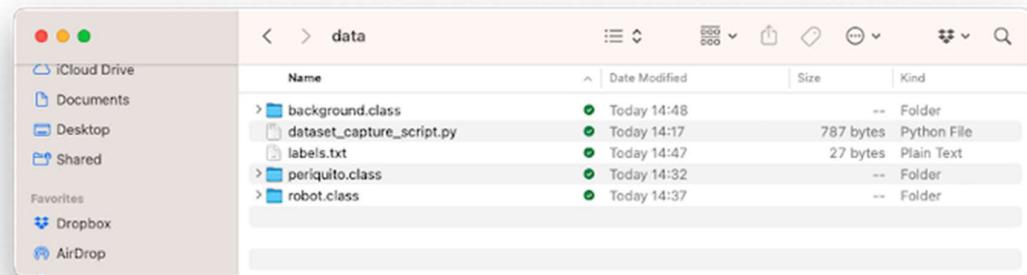
Ripetere la stessa procedura con le altre classi



Suggeriamo circa 60 immagini da ogni categoria. Provare ad acquisire con diverse angolazioni, sfondi e condizioni di luce.

Le immagini archiviate utilizzano una dimensione del fotogramma QVGA di 320x240 e RGB565 (formato pixel colore).

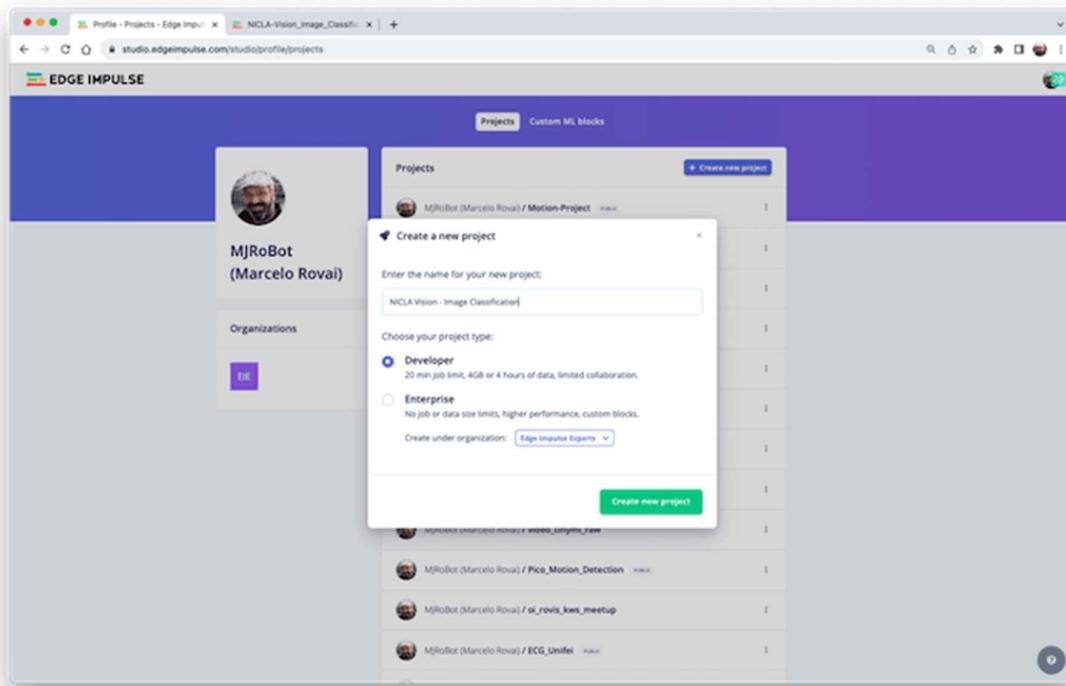
Dopo aver acquisito il dataset, si chiude il Tool Dataset Editor su `Tools > Dataset Editor`. Sul computer, si finirà con un set di dati che contiene tre classi: *periquito*, *robot* e *background*.



Si deve tornare a *Edge Impulse Studio* e caricare il set di dati nel progetto.

Adddestramento del modello con Edge Impulse Studio

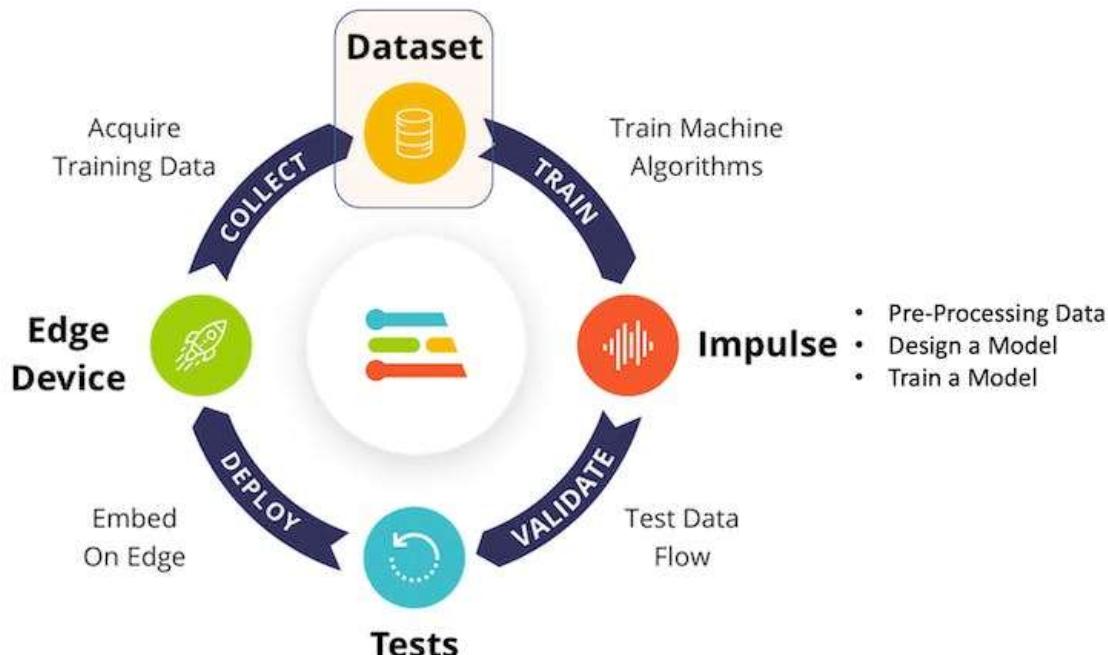
Useremo Edge Impulse Studio per addestrare il nostro modello. Inserire le credenziali del proprio account e creare un nuovo progetto:



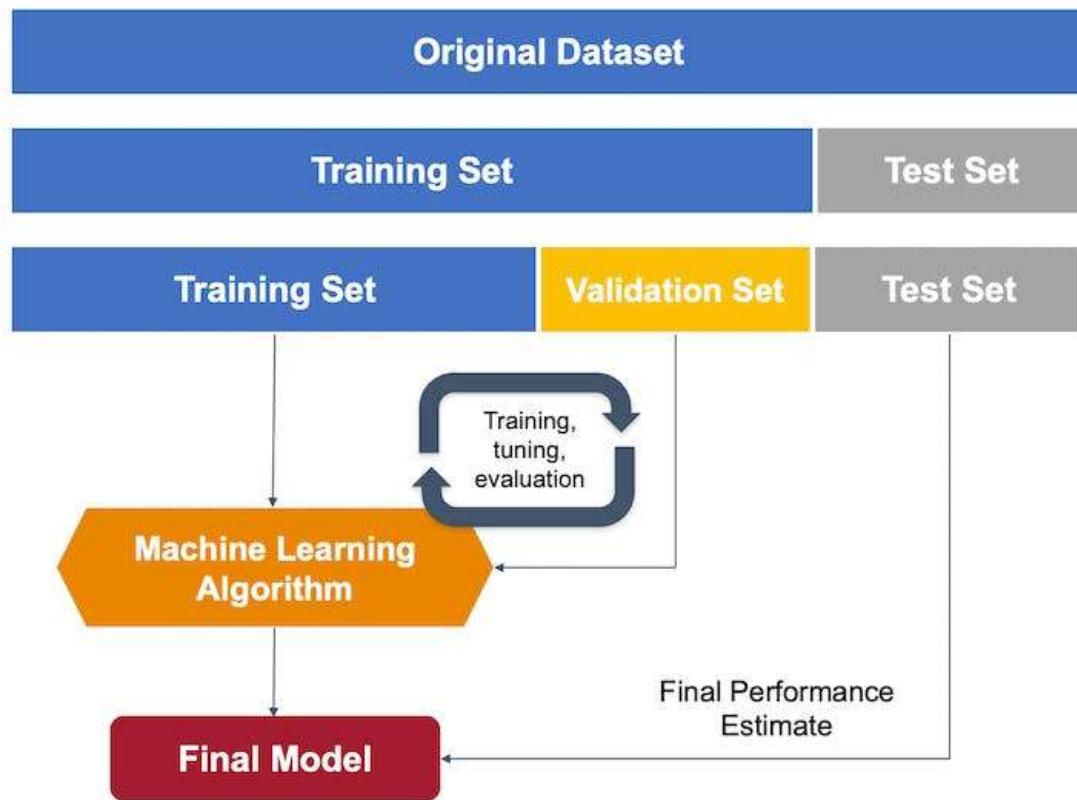
Qui si può clonare un progetto simile: [NICLA-Vision_Image_Classification](#).

Il Dataset

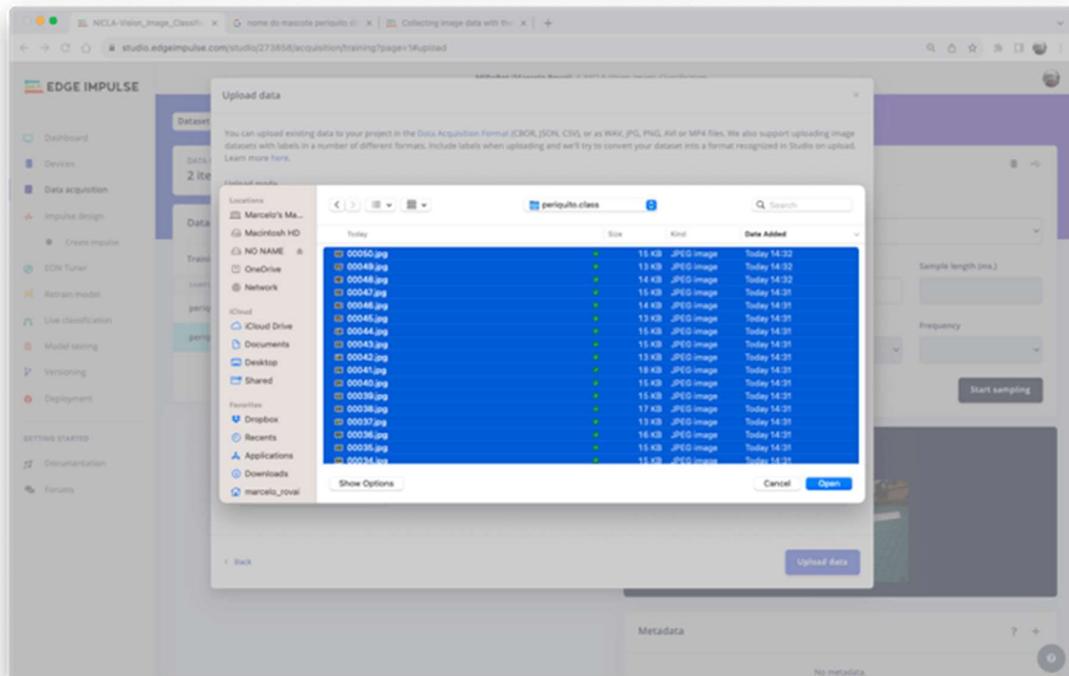
Utilizzando El Studio (o *Studio*), esamineremo quattro passaggi principali per avere il nostro modello pronto per l'uso sulla scheda Nicla Vision: Dataset, Impulse, Tests e Deploy (su Edge Device, in questo caso, la NiclaV).



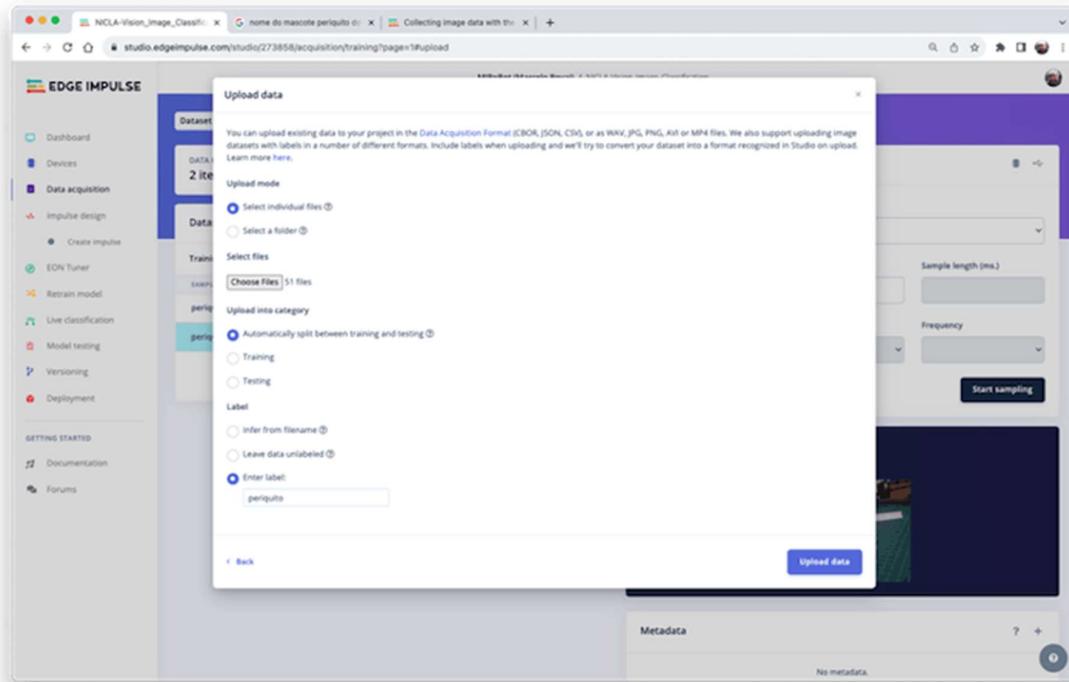
Per quanto riguarda il Dataset, è essenziale sottolineare che il nostro Dataset originale, acquisito con OpenMV IDE, sarà suddiviso in *Training*, *Validation* e *Test*. Il Test Set sarà suddiviso dall'inizio e una parte sarà riservata per essere utilizzata solo nella fase di Test dopo l'addestramento. Il Validation Set sarà utilizzato durante l'addestramento.



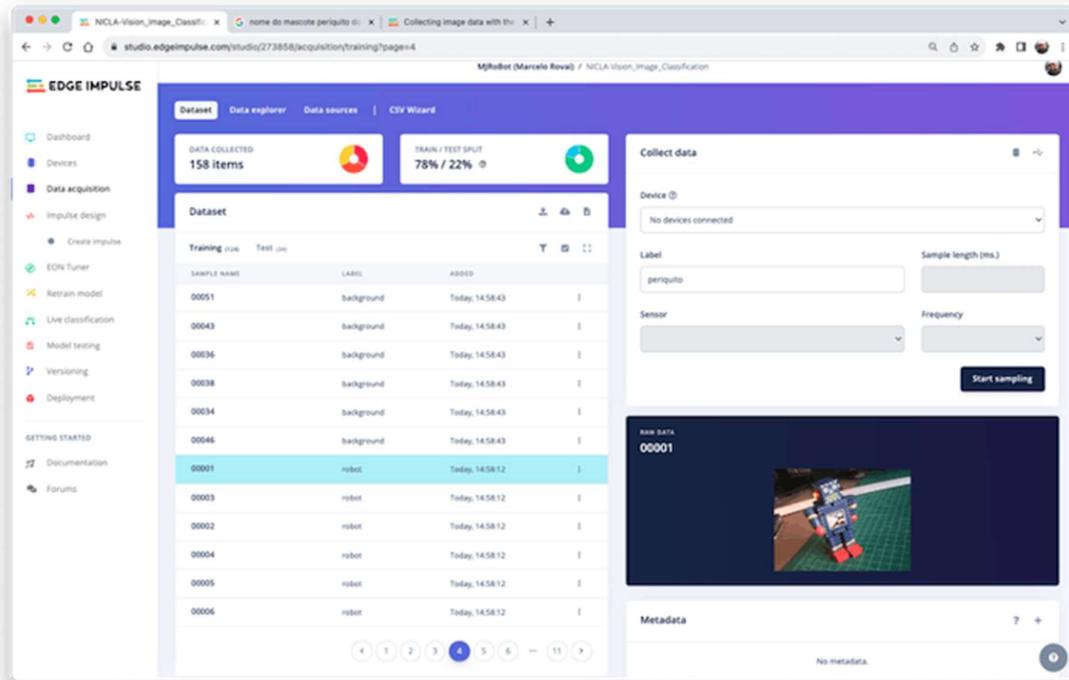
Su Studio, nella scheda Data acquisition e nella sezione UPLOAD DATA, si caricano i file delle categorie scelte dal computer:



Lasciare in Studio la suddivisione del dataset originale in *train and test* e scegliere l'etichetta relativa a quei dati specifici:



Ripetere la procedura per tutte e tre le classi. Alla fine, si vedranno “dati grezzi” in Studio:



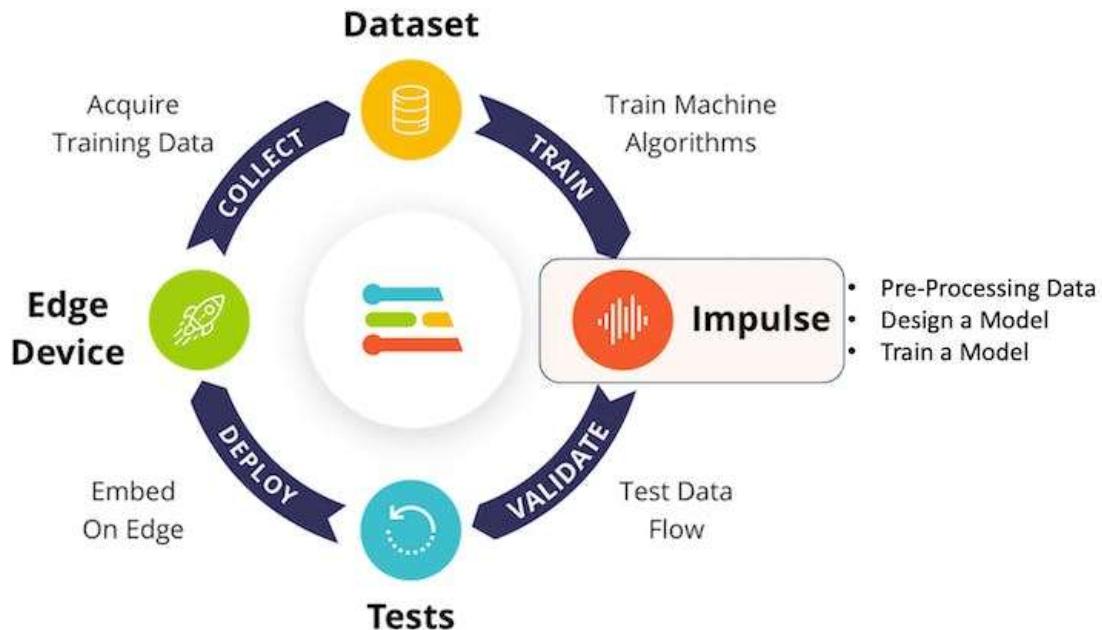
Studio consente di esplorare i dati, mostrando una vista completa di tutti i dati nel progetto. Si possono cancellare, ispezionare o modificare le etichette cliccando sui singoli elementi di dati. Nel nostro caso, un progetto molto semplice, i dati sembrano OK.



Impulse Design

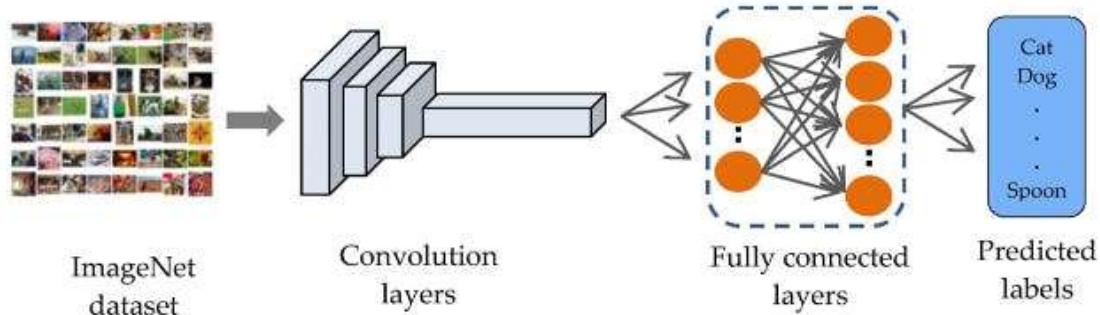
In questa fase, dovremmo definire come:

- Pre-elaborare i nostri dati, il che consiste nel ridimensionare le singole immagini e determinare la profondità di colore da utilizzare (RGB o scala di grigi) e
- Specificare un modello, in questo caso, sarà il Transfer Learning (Images) per mettere a punto un modello di classificazione delle immagini MobileNet V2 pre-addestrato sui nostri dati. Questo metodo funziona bene anche con set di dati di immagini relativamente piccoli (circa 150 immagini nel nostro caso).

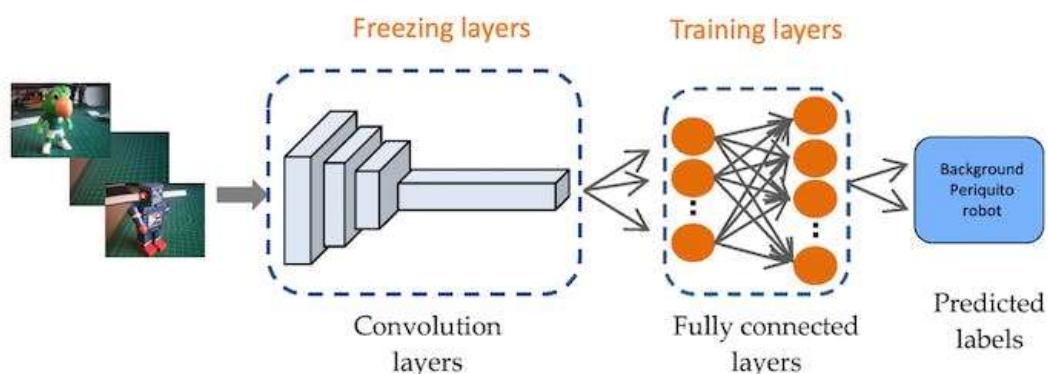


Transfer Learning con MobileNet offre un approccio semplificato all'addestramento del modello, che è particolarmente utile per ambienti con risorse limitate e progetti con dati

etichettati limitati. MobileNet, noto per la sua architettura leggera, è un modello pre-addestrato che ha già appreso funzionalità preziose da un ampio set di dati (ImageNet).

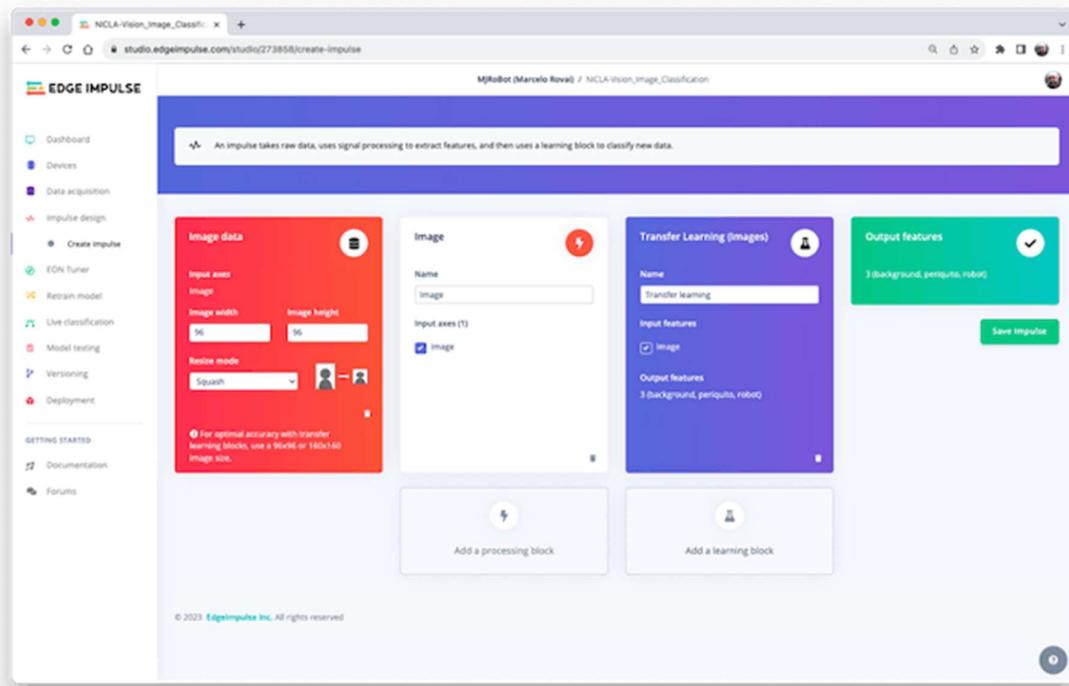


Sfruttando queste funzionalità apprese, si può addestrare un nuovo modello per il compito specifico con meno dati e risorse computazionali e tuttavia ottenere una precisione competitiva.



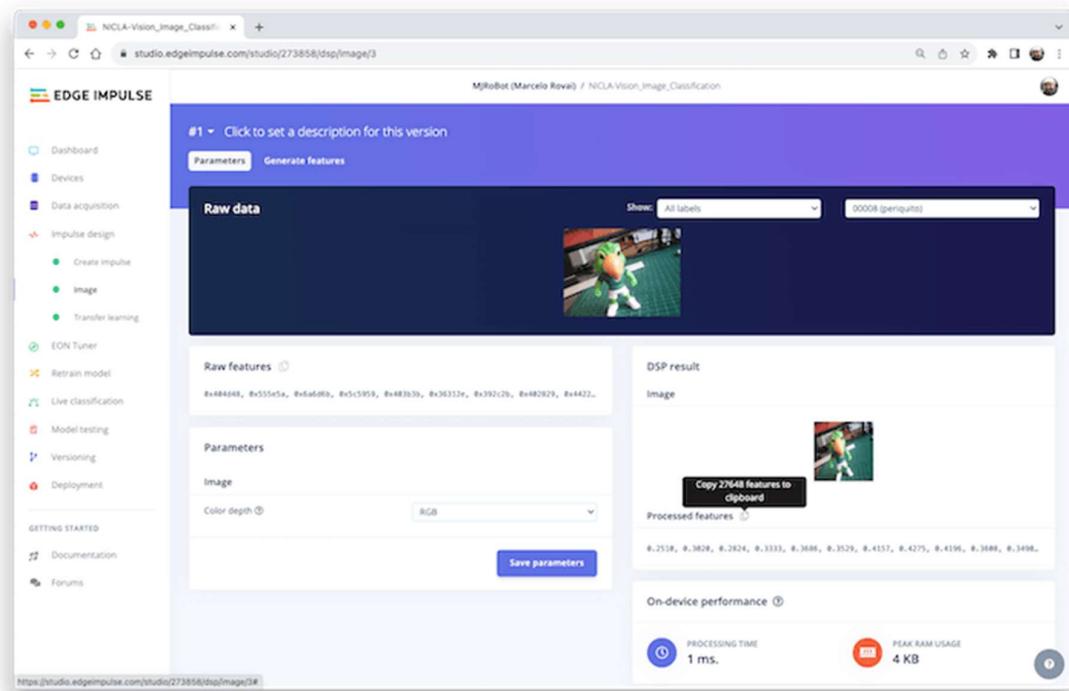
Questo approccio riduce significativamente i tempi di addestramento e i costi computazionali, rendendolo ideale per la prototipazione rapida e l'implementazione su dispositivi embedded in cui l'efficienza è fondamentale.

Si va alla scheda Impulse Design e si crea l'*impulse*, definendo una dimensione dell'immagine di 96x96 e schiacciandola (forma quadrata, senza ritaglio). Si seleziona Image e i blocchi Transfer Learning. Si salva l'Impulse.

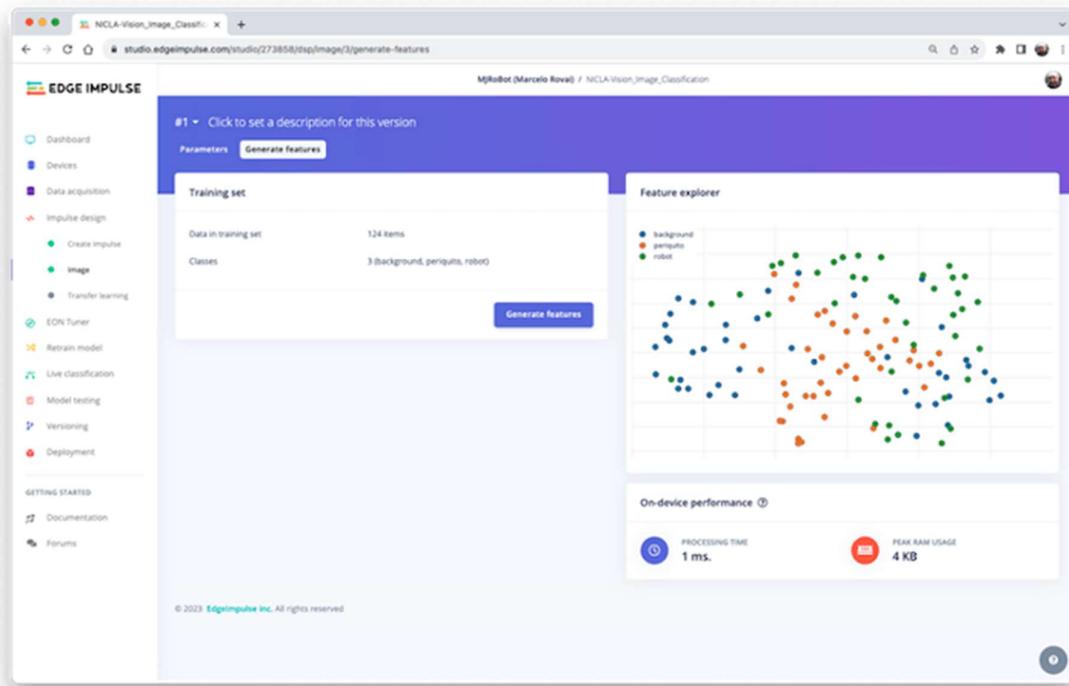


Pre-elaborazione delle Immagini

Tutte le immagini QVGA/RGB565 in ingresso verranno convertite in 27.640 feature (96x96x3).



Si preme [Save parameters] e Generate all features:



Progettazione del Modello

Nel 2007, Google ha introdotto [MobileNetV1](#), una famiglia di reti neurali per la visione artificiale di uso generale progettate pensando ai dispositivi mobili per supportare la classificazione, il rilevamento e altro ancora. Le MobileNet sono modelli piccoli, a bassa latenza e a basso consumo, parametrizzati per soddisfare i vincoli di risorse di vari casi d'uso.

Nel 2018, Google ha lanciato [MobileNetV2: Inverted Residuals and Linear Bottlenecks](#).

MobileNet V1 e MobileNet V2 mirano all'efficienza mobile e alle applicazioni di visione embedded, ma differiscono per complessità architettonica e prestazioni. Mentre entrambi utilizzano convoluzioni separabili in profondità per ridurre i costi computazionali, MobileNet V2 introduce Inverted Residual Blocks e Linear Bottlenecks per migliorare le prestazioni.

Queste nuove funzionalità consentono a V2 di acquisire funzionalità più complesse utilizzando meno parametri, rendendolo più efficiente dal punto di vista computazionale e generalmente più accurato rispetto al suo predecessore. Inoltre, V2 impiega un'attivazione non lineare nello layer di espansione intermedio. Utilizza ancora un'attivazione lineare per il layer del bottleneck [collo di bottiglia], una scelta di progettazione che si è rivelata utile per preservare informazioni importanti attraverso la rete. MobileNet V2 offre un'architettura ottimizzata per una maggiore accuratezza ed efficienza e verrà utilizzata in questo progetto.

Sebbene l'architettura di base di MobileNet sia già minuscola e abbia una bassa latenza, molte volte, un caso d'uso o un'applicazione specifica potrebbe richiedere che il modello sia ancora più piccolo e veloce. MobileNets introduce un parametro semplice α (alfa) chiamato moltiplicatore di larghezza per costruire questi modelli più piccoli e meno costosi dal punto di vista computazionale. Il ruolo del moltiplicatore di larghezza α è quello di assottigliare una rete in modo uniforme a ogni layer.

Edge Impulse Studio può utilizzare sia MobileNetV1 (immagini 96x96) che V2 (immagini 96x96 o 160x160), con diversi valori di α (da 0,05 a 1,0). Ad esempio, si otterrà la massima accuratezza con V2, immagini 160x160 e $\alpha=1,0$. Naturalmente, c'è un compromesso.

Maggiore è la precisione, più memoria (circa 1,3 MB di RAM e 2,6 MB di ROM) sarà necessaria per eseguire il modello, il che implica una maggiore latenza. L'ingombro minore sarà ottenuto all'altro estremo con MobileNetV1 e $\alpha=0,10$ (circa 53,2 K di RAM e 101 K di ROM).

MobileNetV1 96x96 0.1

Uses around 53.2K RAM and 101K ROM with default settings and optimizations. Works best with 96x96 input size. Supports both RGB and grayscale.

Model**MobileNetV2 96x96 0.35**

Uses around 296.8K RAM and 575.2K ROM with default settings and optimizations. Works best with 96x96 input size. Supports both RGB and grayscale.

Image Size**MobileNetV2 96x96 0.1**

Uses around 270.2K RAM and 212.3K ROM with default settings and optimizations. Works best with 96x96 input size. Supports both RGB and grayscale.

Alpha**MobileNetV2 96x96 0.05**

Uses around 265.3K RAM and 162.4K ROM with default settings and optimizations. Works best with 96x96 input size. Supports both RGB and grayscale.

Per questo progetto utilizzeremo **MobileNetV2 96x96 0.1**, con un costo di memoria stimato di 265,3 KB in RAM. Questo modello dovrebbe andare bene per Nicla Vision con 1 MB di SRAM. Nella scheda Transfer Learning, si seleziona questo modello:

The screenshot shows the Edge Impulse studio interface with the URL <https://studio.edgeimpulse.com/studio/273858/learning/keras-transfer-image/>. The left sidebar has sections like Dashboard, Devices, Data acquisition, Impulse design, Image, Transfer learning, EON Tuner, Retrain model, Live classification, Model testing, Versioning, Deployment, Getting Started, Documentation, and Forums. The main area is titled 'Choose a different model' and lists several pre-trained models. The 'MobileNetV2 96x96 0.1' model is highlighted with a red circle. Other models listed include MobileNetV1 96x96 0.25, MobileNetV1 96x96 0.35, MobileNetV2 96x96 0.35, MobileNetV2 96x96 0.05, MobileNetV2 160x160 1.0, and MobileNetV2 160x160 0.75. Each model entry includes a brief description, RAM/ROM requirements, and an 'Add' button.

Adddestramento del Modello

Un'altra tecnica preziosa da utilizzare con il Deep Learning è la **Data Augmentation**. Il “data augmentation” è un metodo per migliorare l’accuratezza dei modelli di apprendimento automatico mediante la creazione di dati artificiali aggiuntivi. Un sistema di Data Augmentation apporta piccole modifiche casuali ai dati di training (ad esempio capovolgendo, ritagliando o ruotando le immagini).

Guardando “sotto il cofano”, qui si può vedere come Edge Impulse implementa una policy di Data Augmentation sui dati:

```
# Implements the data augmentation policy
def augment_image(image, label):
    # Flips the image randomly
    image = tf.image.random_flip_left_right(image)
```

```

# Increase the image size, then randomly crop it down to
# the original dimensions
resize_factor = random.uniform(1, 1.2)
new_height = math.floor(resize_factor * INPUT_SHAPE[0])
new_width = math.floor(resize_factor * INPUT_SHAPE[1])
image = tf.image.resize_with_crop_or_pad(image, new_height, new_width)
image = tf.image.random_crop(image, size=INPUT_SHAPE)

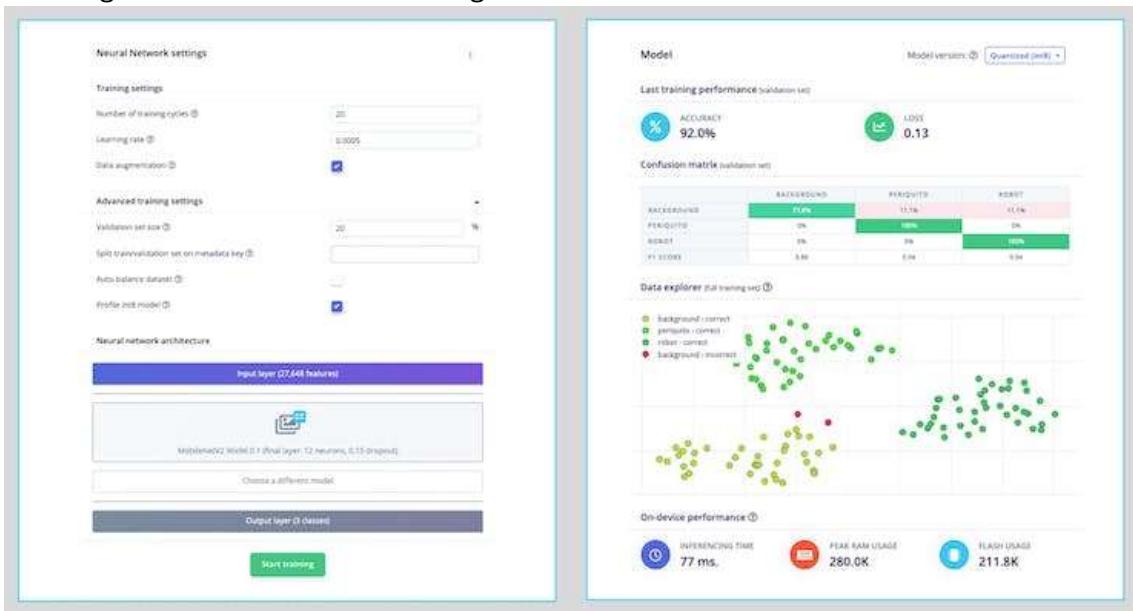
# Vary the brightness of the image
image = tf.image.random_brightness(image, max_delta=0.2)

return image, label

```

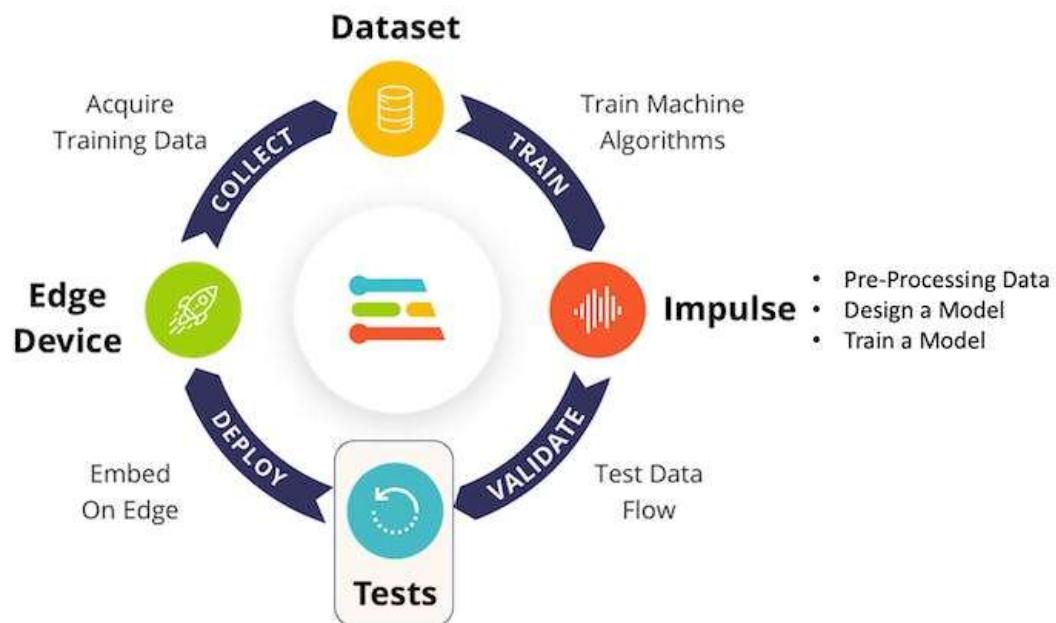
L'esposizione a queste variazioni durante l'addestramento può aiutare a impedire al modello di prendere scorciatoie "memorizzando" indizi superficiali nei dati di addestramento, il che significa che potrebbe riflettere meglio i pattern profondi in esame nel set di dati.

L'ultimo layer del nostro modello avrà 12 neuroni con un dropout del 15% per prevenire l'overfitting. Ecco il risultato del Training:

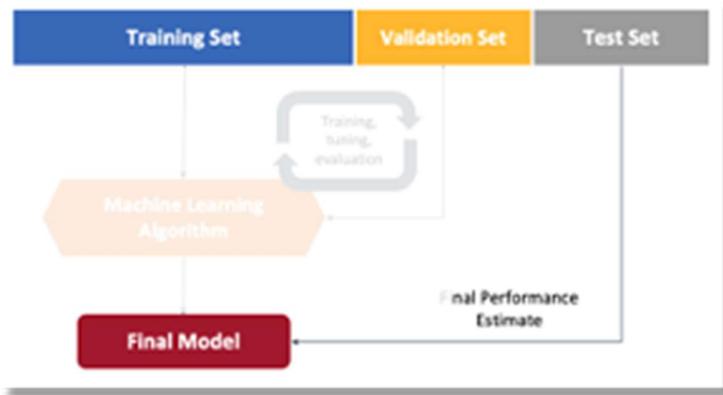


Il risultato è eccellente, con 77 ms di latenza, che dovrebbero tradursi in 13 fps (frame al secondo) durante l'inferenza.

Test del Modello



Ora, si dovrebbe mettere da parte il set di dati all'inizio del progetto ed eseguire il modello addestrato usandolo come input:

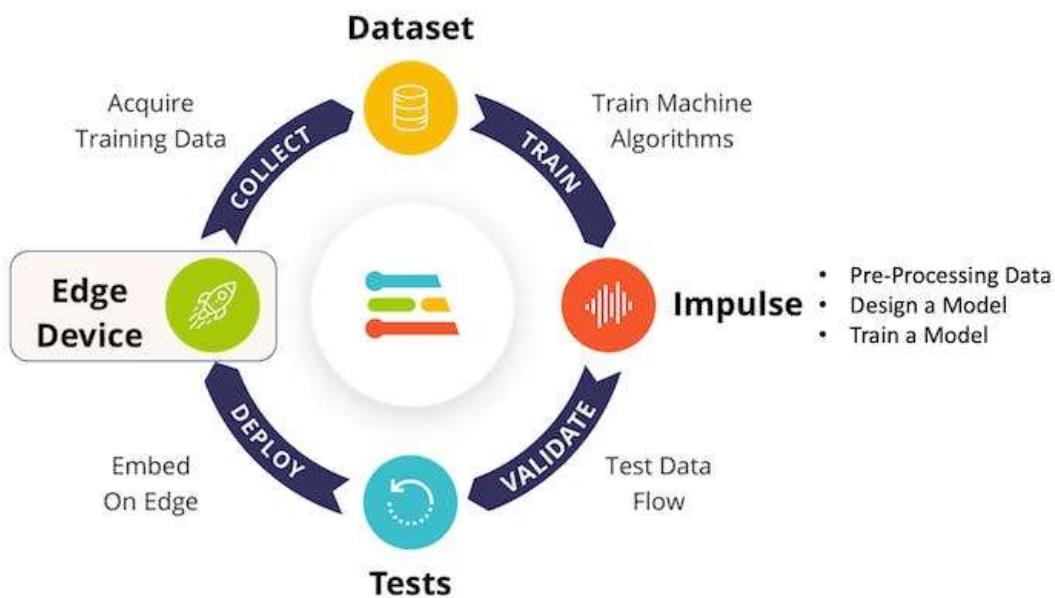


Il risultato è, ancora una volta, eccellente.

The screenshot shows the Edge Impulse Studio interface. On the left is a sidebar with navigation links like Dashboard, Devices, Data acquisition, Impulse design, Model testing, EON Tuner, Retrain model, Live classification, Versioning, Deployment, Documentation, and Forums. The main area has three tabs: 'Test data' (listing 32 samples with labels like 'background', 'robot', 'penquin', and 'periquito'), 'Model testing output' (showing logs of the classification process), and 'Model testing results' (displaying accuracy of 97.06% and a confusion matrix). Below these is a 'Feature explorer' section with a scatter plot.

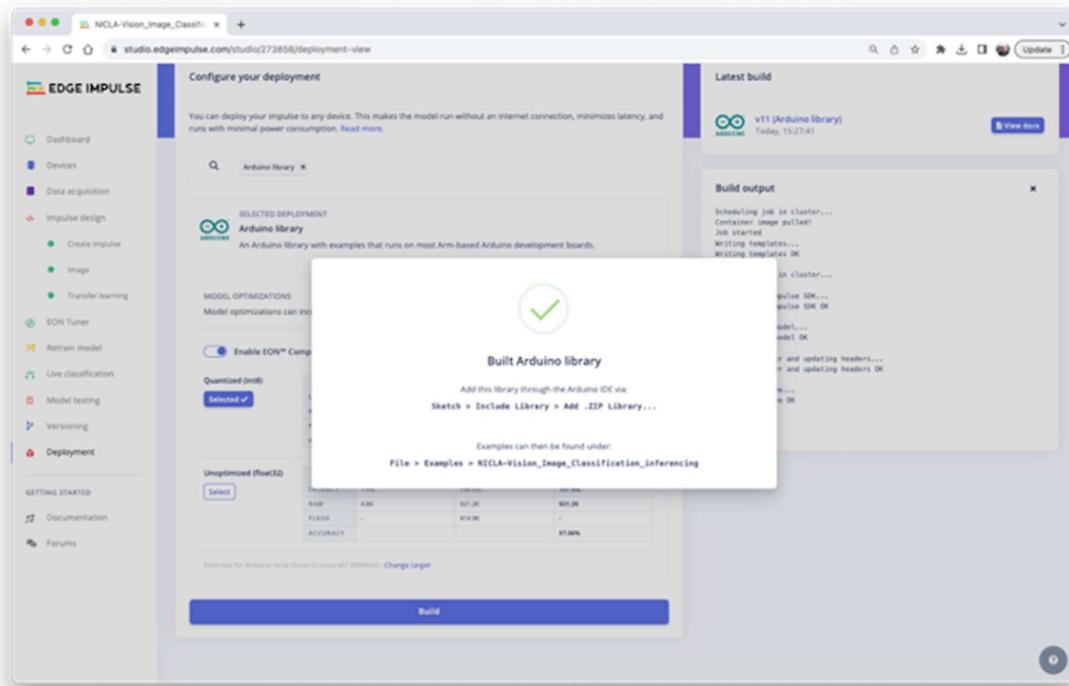
Distribuzione del modello

A questo punto, possiamo distribuire il modello addestrato come .tflite e usare l'IDE OpenMV per eseguirlo usando MicroPython, oppure possiamo distribuirlo come C/C++ o libreria Arduino.



Libreria Arduino

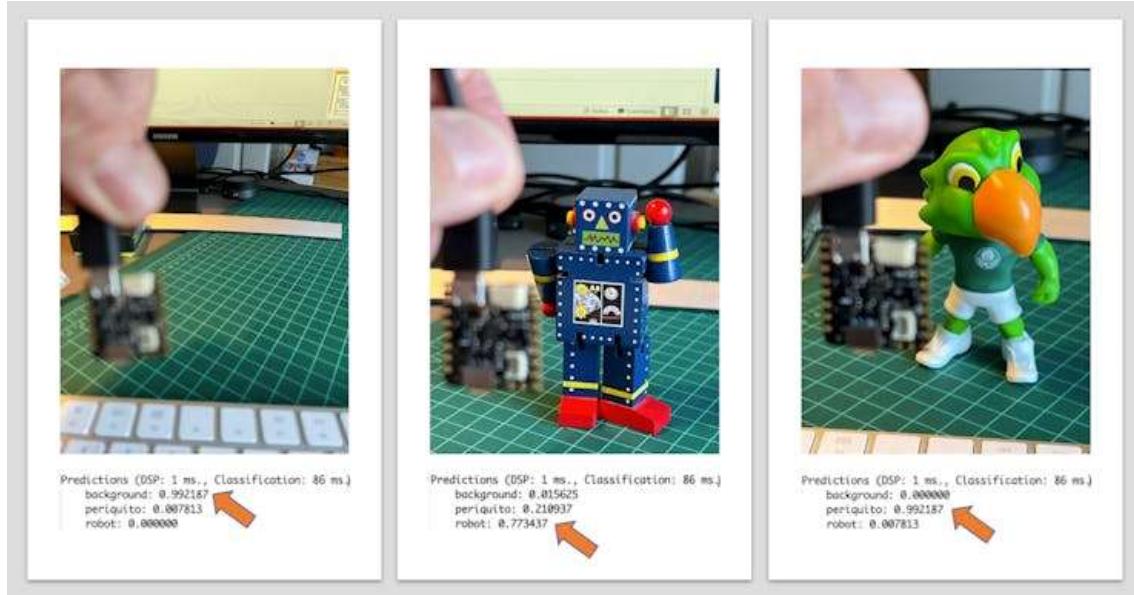
Per prima cosa, distribuiamolo come Libreria Arduino:



Si dovrebbe installare la libreria come .zip sull'IDE Arduino ed eseguire lo sketch *nicla_vision_camera.ino* disponibile in Examples sotto il nome della libreria.

Notare che Arduino Nicla Vision ha, per default, 512KB di RAM allocati per il core M7 e altri 244 KB sullo spazio di indirizzamento dell'M4. Nel codice, questa allocazione è stata modificata in 288 kB per garantire che il modello verrà eseguito sul dispositivo (`malloc_addblock((void*)0x30000000, 288 * 1024);`).

Il risultato è buono, con 86 ms di latenza misurata.



Ecco un breve video che mostra i risultati dell'inferenza: <https://youtu.be/bZPZZJblU-o>

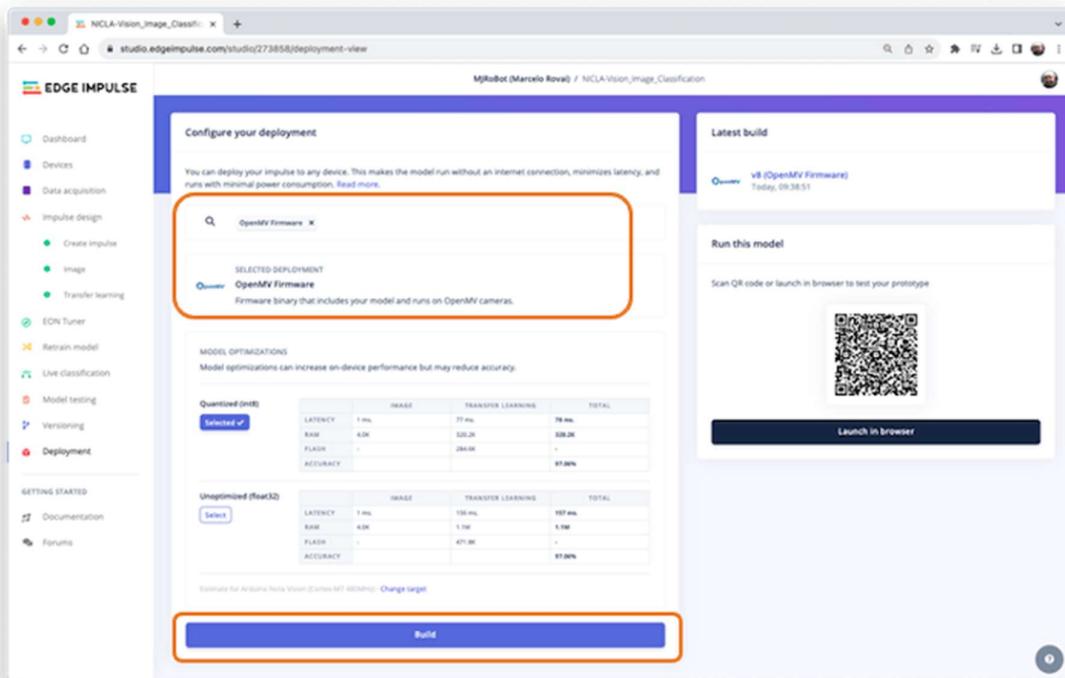
OpenMV

È possibile distribuire il modello addestrato da utilizzare con OpenMV in due modi: come libreria e come firmware.

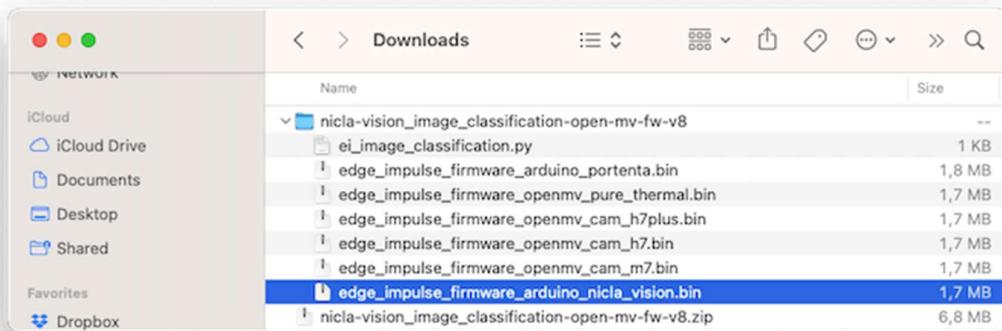
Come libreria vengono generati tre file: il modello .tflite addestrato, un elenco con etichette e un semplice script MicroPython che può effettuare inferenze utilizzando il modello.

Name	Size	Kind	Date Added
ei-nicla-vision_image_classification-openmv-v17	--	Folder	Today 14:59
● trained.tflite	234 KB	TensorFlow Lite Model	Today 14:59
● labels.txt	26 bytes	Plain Text Document	Today 14:59
● ei_image_classification.py	2 KB	Python File	Today 14:59
1 ei-nicla-vision_image_classification-openmv-v17.zip	140 KB	ZIP archive	Today 14:59

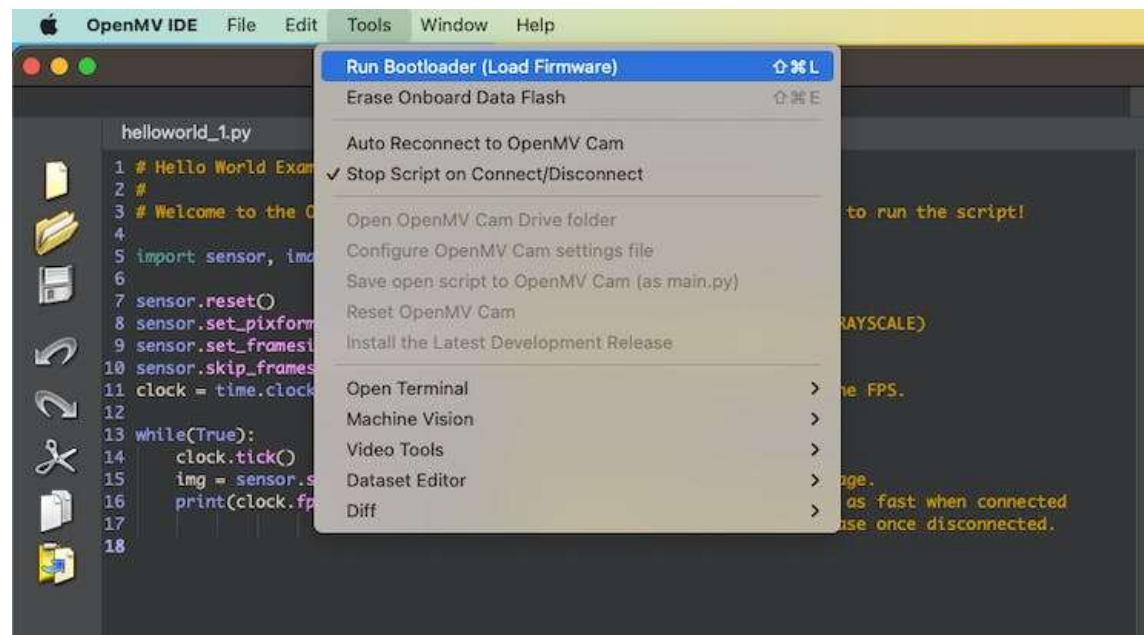
Eseguire questo modello come .tflite direttamente in Nicla era impossibile. Quindi, possiamo sacrificare l'accuratezza utilizzando un modello più piccolo o distribuire il modello come Firmware OpenMV (FW). Scegliendo FW, Edge Impulse Studio genera modelli, librerie e framework ottimizzati necessari per effettuare l'inferenza. Esploriamo questa opzione. Selezionare OpenMV Firmware nella scheda Deploy e premere [Build].



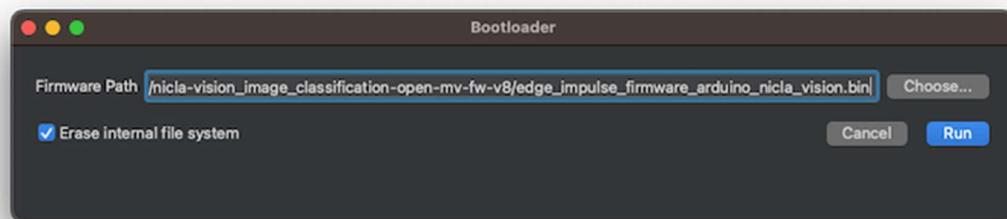
Sul computer si troverà un file ZIP. Lo si apre:



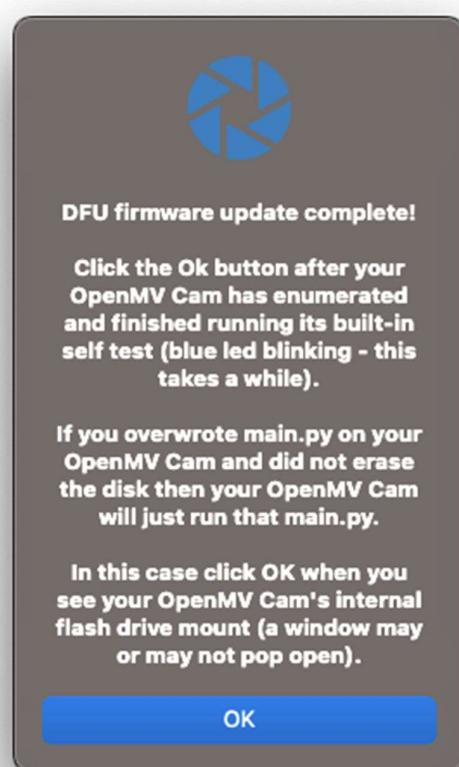
Si usa il tool Bootloader sull'IDE OpenMV per caricare il FW sulla board:



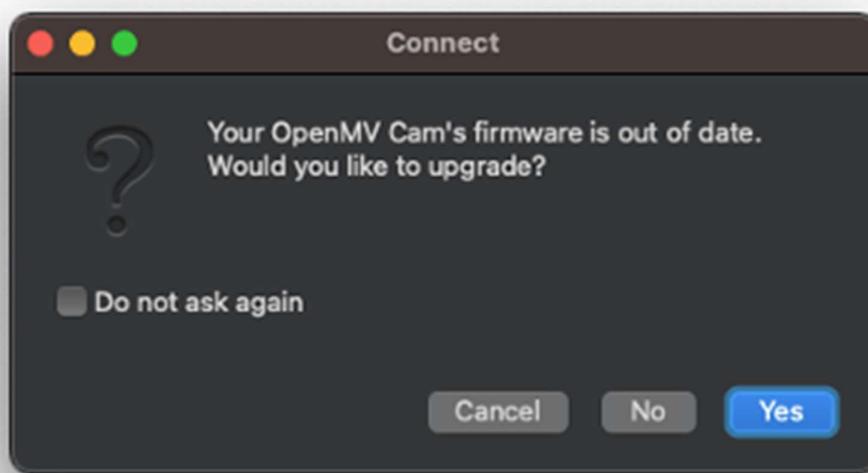
Si seleziona il file appropriato (.bin per Nicla-Vision):



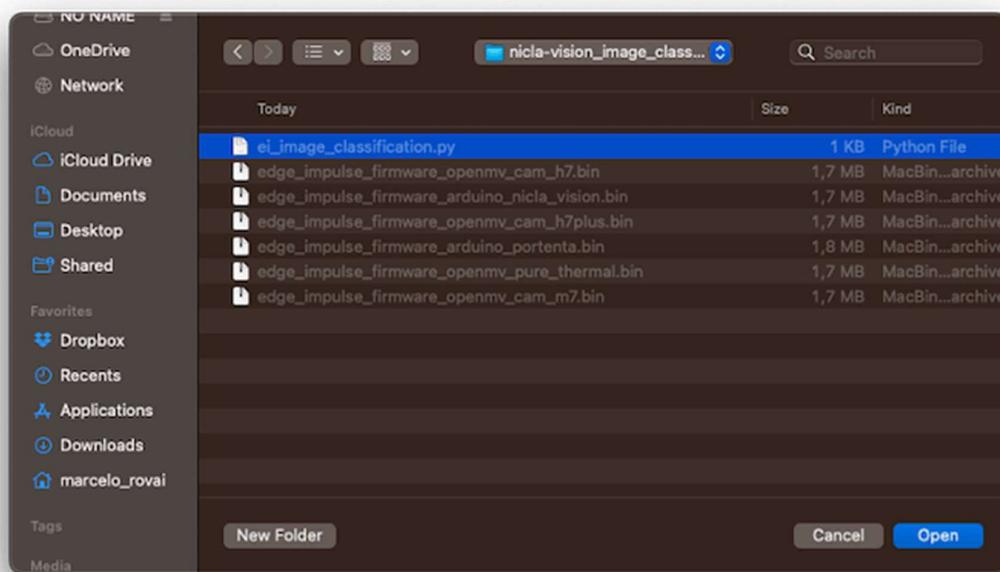
Dopo aver completato il download, si preme OK:



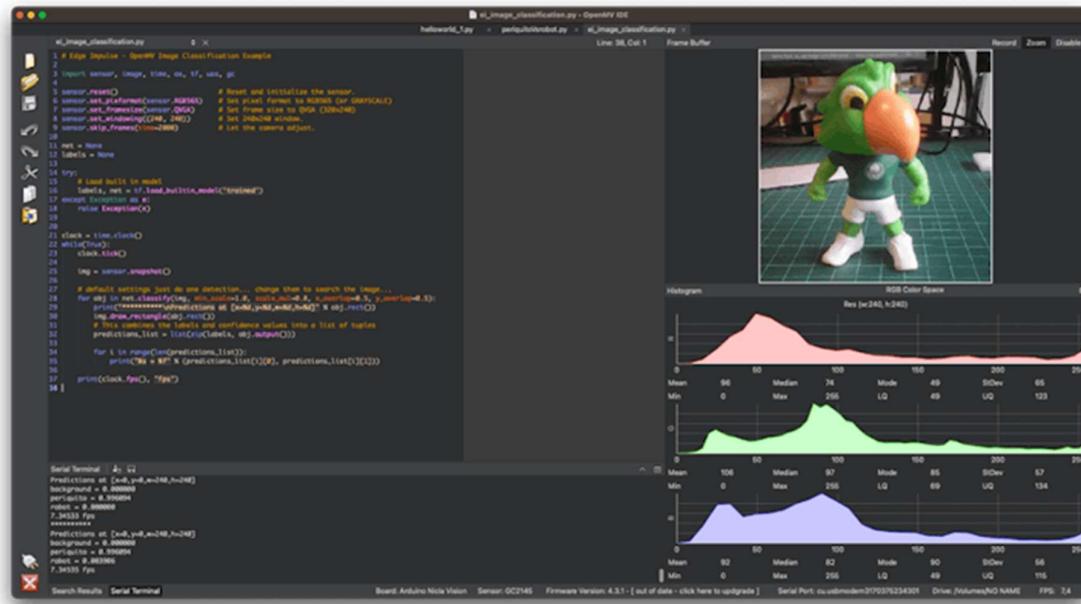
Se un messaggio dice che il FW è “outdated” [obsoleto], NON ESEGUIRE L’AGGIORNAMENTO. Selezionare [NO].



Ora, si apre lo script **ei_image_classification.py** che è stato scaricato da Studio e il file .bin per Nicla.



Eseguirlo. Puntando la telecamera sugli oggetti che vogliamo classificare, il risultato dell’inferenza verrà visualizzato sul Serial Terminal.



Modifica del Codice per Aggiungere Etichette

Il codice fornito da Edge Impulse può essere modificato in modo da poter vedere, per motivi di test, il risultato dell'inferenza direttamente sull'immagine visualizzata sull'IDE OpenMV.

[Caricare il codice da GitHub](#) o modificalo come di seguito:

```
# Marcelo Rovai - NICLA Vision - Image Classification
# Adapted from Edge Impulse - OpenMV Image Classification Example
# @24Aug23

import sensor, image, time, os, tf, uos, gc

sensor.reset()                               # Reset and initialize the sensor.
sensor.set_pixformat(sensor.RGB565)          # Set pxi fmt to RGB565 (or GRayscale)
sensor.set_framesize(sensor.QVGA)             # Set frame size to QVGA (320x240)
sensor.set_windowing((240, 240))              # Set 240x240 window.
sensor.skip_frames(time=2000)                  # Let the camera adjust.

net = None
labels = None

try:
    # Load built in model
    labels, net = tf.load_builtin_model('trained')
except Exception as e:
    raise Exception(e)

clock = time.clock()
while(True):
    clock.tick()    # Starts tracking elapsed time.

    img = sensor.snapshot()

    # default settings just do one detection
    for obj in net.classify(img,
                            min_scale=1.0,
                            scale_mul=0.8,
                            x_overlap=0.5,
                            y_overlap=0.5):
        fps = clock.fps()
        lat = clock.avg()

        print("*****\nPrediction:")
        img.draw_rectangle(obj.rect())
        # This combines the labels and confidence values into a list of tuples
        predictions_list = list(zip(labels, obj.output()))

        max_val = predictions_list[0][1]
```

```

max_lbl = 'background'
for i in range(len(predictions_list)):
    val = predictions_list[i][1]
    lbl = predictions_list[i][0]

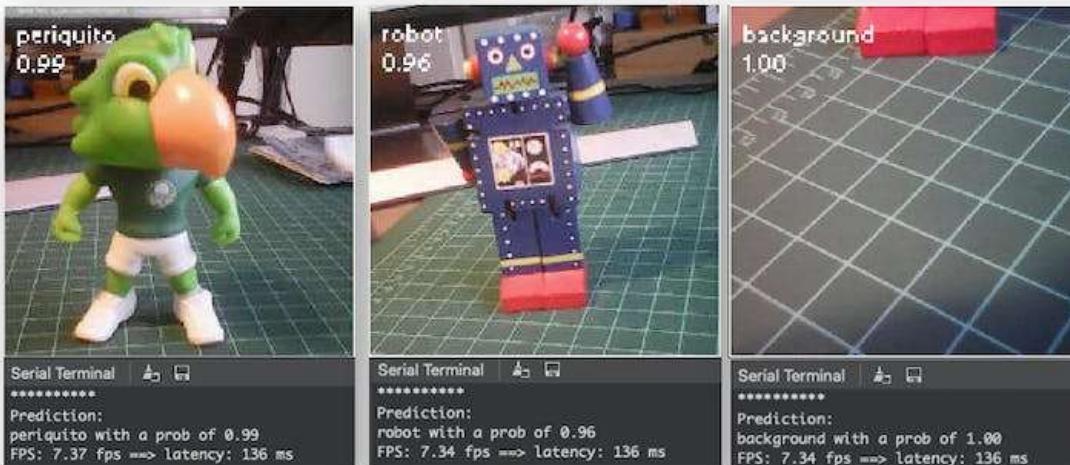
    if val > max_val:
        max_val = val
        max_lbl = lbl

# Print label with the highest probability
if max_val < 0.5:
    max_lbl = 'uncertain'
print("{} with a prob of {:.2f}".format(max_lbl, max_val))
print("FPS: {:.2f} fps ==> latency: {:.0f} ms".format(fps, lat))

# Draw label with highest probability to image viewer
img.draw_string(
    10, 10,
    max_lbl + "\n{:.2f}".format(max_val),
    mono_space = False,
    scale=2
)

```

Qui si può vedere il risultato:



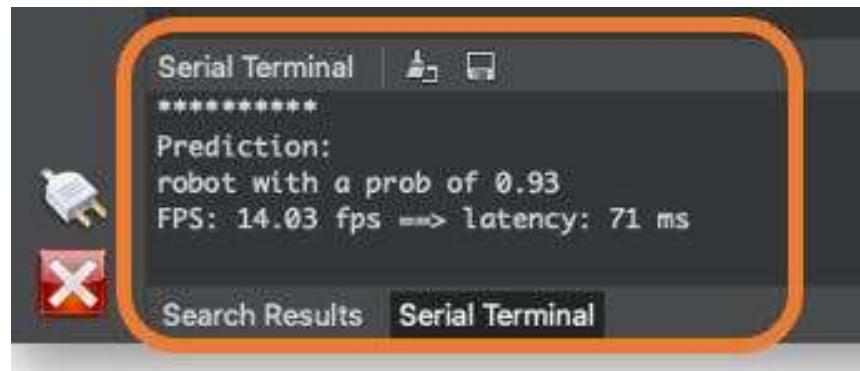
Notare che la latenza (136 ms) è quasi il doppio di quella che ottenuta direttamente con l'IDE Arduino. Questo perché stiamo usando l'IDE come interfaccia e anche il tempo di attesa per la fotocamera per essere pronta. Se avviamo il clock appena prima dell'inferenza:

```

56 while(True):
57
58     img = sensor.snapshot()
59
60     clock.tick() # Starts tracking elapsed time.
61
62     # default settings just do one detection... change them to search the image...
63     for obj in net.classify(img, min_scale=1.0, scale_mul=0.8, x_overlap=0.5, y_overlap=0.5):
64         fps = clock.fps()
65         lat = clock.avg()
66
67         print("*****\nPrediction:")
68         img.draw_rectangle(obj.rect())
69         # This combines the labels and confidence values into a list of tuples
70         predictions_list = list(zip(labels, obj.output()))
71

```

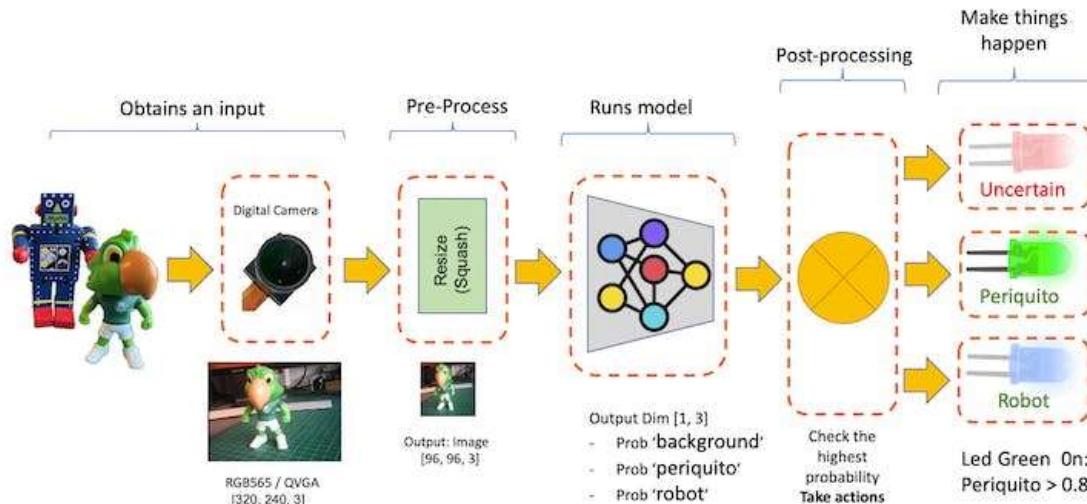
La latenza scenderà a soli 71 ms.



NICLA V funziona a circa la metà della velocità quando è connesso all'IDE. Gli FPS dovrebbero aumentare una volta disconnessi.

Post-elaborazione con i LED

Quando lavoriamo con l'apprendimento automatico embedded, cerchiamo dispositivi che possano procedere continuamente con l'inferenza e il risultato, eseguendo un'azione direttamente sul mondo fisico e non visualizzando il risultato su un computer connesso. Per simulare ciò, accenderemo un LED diverso per ogni possibile risultato dell'inferenza.



Per ottenere ciò, dovremmo [caricare il codice da GitHub](#) o modificare l'ultimo codice per includere i LED:

```
# Marcelo Rovai - NICLA Vision - Image Classification with LEDs
# Adapted from Edge Impulse - OpenMV Image Classification Example
# @24Aug23

import sensor, image, time, os, tf, uos, gc, pyb

ledRed = pyb.LED(1)
ledGre = pyb.LED(2)
ledBlu = pyb.LED(3)

sensor.reset()                                # Reset and initialize the sensor.
sensor.set_pixformat(sensor.RGB565)           # Set pixel fmt to RGB565 (or GRayscale)
sensor.set_framesize(sensor.QVGA)              # Set frame size to QVGA (320x240)
sensor.set_windowing((240, 240))               # Set 240x240 window.
sensor.skip_frames(time=2000)                  # Let the camera adjust.

net = None
labels = None

ledRed.off()
ledGre.off()
ledBlu.off()
```

```

try:
    # Load built in model
    labels, net = tf.load_built_in_model('trained')
except Exception as e:
    raise Exception(e)

clock = time.clock()

def setLEDs(max_lbl):
    if max_lbl == 'uncertain':
        ledRed.on()
        ledGre.off()
        ledBlu.off()

    if max_lbl == 'periquito':
        ledRed.off()
        ledGre.on()
        ledBlu.off()

    if max_lbl == 'robot':
        ledRed.off()
        ledGre.off()
        ledBlu.on()

    if max_lbl == 'background':
        ledRed.off()
        ledGre.off()
        ledBlu.off()

while(True):
    img = sensor.snapshot()
    clock.tick() # Starts tracking elapsed time.

    # default settings just do one detection.
    for obj in net.classify(img,
                             min_scale=1.0,
                             scale_mul=0.8,
                             x_overlap=0.5,
                             y_overlap=0.5):
        fps = clock.fps()
        lat = clock.avg()

        print("*****\nPrediction:")
        img.draw_rectangle(obj.rect())
        # This combines the labels and confidence values into a list of tuples
        predictions_list = list(zip(labels, obj.output()))

        max_val = predictions_list[0][1]
        max_lbl = 'background'
        for i in range(len(predictions_list)):
            val = predictions_list[i][1]
            lbl = predictions_list[i][0]

            if val > max_val:
                max_val = val
                max_lbl = lbl

    # Print label and turn on LED with the highest probability
    if max_val < 0.8:
        max_lbl = 'uncertain'

    setLEDs(max_lbl)

    print("{} with a prob of {:.2f}".format(max_lbl, max_val))
    print("FPS: {:.2f} fps ==> latency: {:.0f} ms".format(fps, lat))

    # Draw label with highest probability to image viewer
    img.draw_string(
        10, 10,
        max_lbl + "\n{:.2f}".format(max_val),
        mono_space = False,

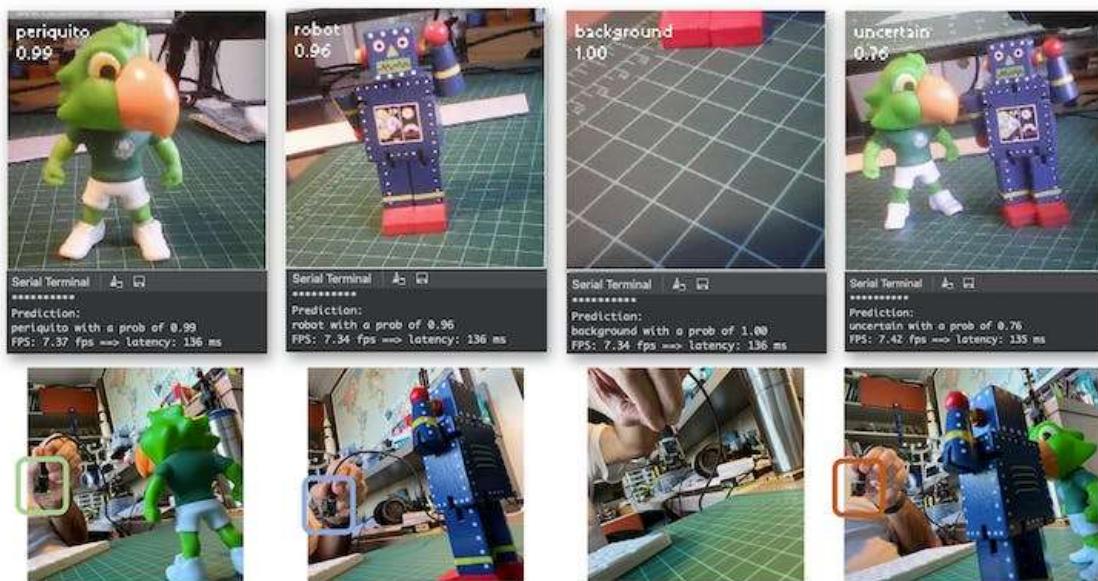
```

```
    scale=2  
)
```

Ora, ogni volta che una classe ottiene un risultato superiore a 0,8, il LED corrispondente si accenderà:

- Led Rosso On: incerto (nessuna classe supera 0,8)
- Led Green On: Periquito > 0.8
- Led Blue On: Robot > 0.8
- Tutti i LED spenti: Sfondo > 0.8

Ecco il risultato:



Più in dettaglio



Classificazione delle immagini Benchmark (non ufficiale)

Diverse schede di sviluppo possono essere utilizzate per l'apprendimento automatico embedded (TinyML) e le più comuni per le applicazioni di Computer Vision (consumo energetico basso) sono ESP32 CAM, Seeed XIAO ESP32S3 Sense, Arduino Nicla Vison e Arduino Portenta.



	ESP 32	Seeed XIAO Sense / ESP32S3	Arduino Pro
32Bits CPU	Xtensa LX6 Dual Core	Arm Cortex-M4F (BLE) Xtensa LX7 Dual Core	Dual Core Arm Cortex M7/M4
CLOCK	240MHz	64 / 240MHz	480/240MHz
RAM	520KB (part available)	256KB / 8MB	1MB
ROM	2MB	2MB / 8MB	2MB
Radio	BLE/WiFi	BLE / WiFi (ESP32S3)	BLE/WiFi
Sensors	Yes (CAM)	Yes (Sense)	Yes (Nicla)
Bat. Power Manag.	No	Yes	Yes
Price	\$	\$\$	\$\$\$\$

Cogliendo l'occasione, lo stesso modello addestrato è stato distribuito su ESP-CAM, XIAO e Portenta (in questo caso, il modello è stato addestrato di nuovo, utilizzando immagini in scala di grigi per essere compatibile con la sua fotocamera). Ecco il risultato, distribuendo i modelli come Libreria di Arduino:



Conclusione

Prima di finire, si tenga presente che la Computer Vision è più di una semplice classificazione delle immagini. Ad esempio, si possono sviluppare progetti Edge Machine Learning sulla visione in diverse aree, come:

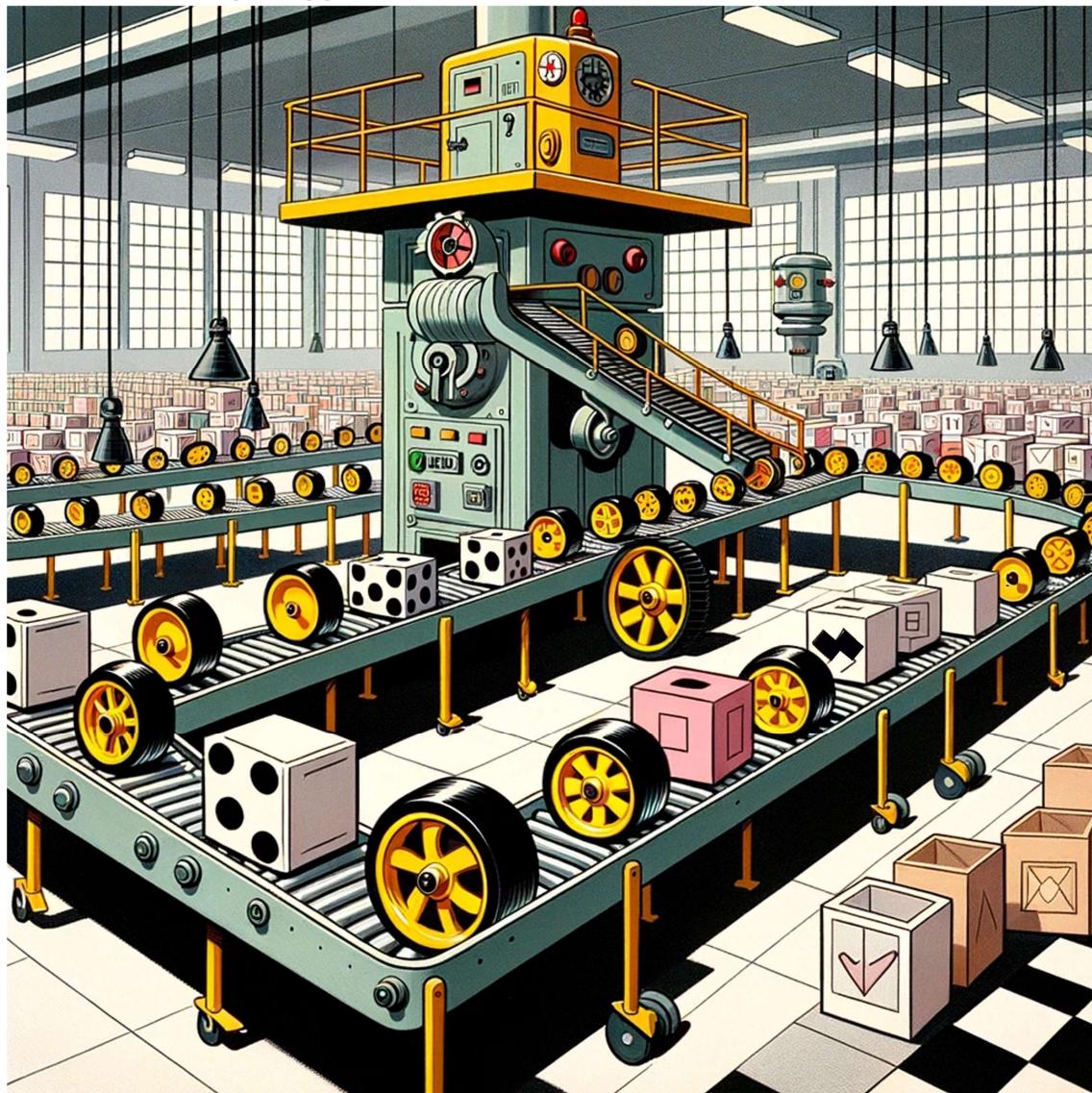
- **Veicoli Autonomi:** Usa un gruppo di sensori, i dati lidar e gli algoritmi di visione artificiale per navigare e prendere decisioni.
- **Sanità:** Diagnosi automatizzata di malattie tramite analisi delle immagini di risonanza magnetica, raggi X e TAC
- **Vendita al Dettaglio:** Sistemi di pagamento automatizzati che identificano i prodotti mentre passano attraverso uno scanner.

- **Sicurezza e Sorveglianza:** Riconoscimento facciale, rilevamento di anomalie e tracciamento di oggetti in video in tempo reale.
- **Realtà Aumentata:** Rilevamento e classificazione di oggetti per sovrapporre informazioni digitali al mondo reale.
- **Automazione Industriale:** Ispezione visiva di prodotti, manutenzione predittiva e guida di robot e droni.
- **Agricoltura;** Monitoraggio delle colture basato su droni e raccolta automatizzata.
- **Elaborazione del Linguaggio Naturale:** Didascalie delle immagini e risposte visive alle domande.
- **Riconoscimento dei Gesti:** Per giochi, traduzione del linguaggio dei segni e interazione uomo-macchina.
- **Raccomandazione dei Contenuti:** Sistemi di raccomandazione basati sulle immagini nell'e-commerce.

Risorse

- [Codici Micropython](#)
- [Dataset](#)
- [Progetto Edge Impulse](#)

Rilevamento degli Oggetti



DALL-E 3 Prompt: umetto in stile anni '40 o '50 che mostra l'interno di un ampio magazzino industriale. Un nastro trasportatore è in primo piano, trasportando un mix di ruote e scatole giocattolo. Le ruote sono distinguibili per i loro cerchioni giallo brillante e pneumatici neri. Le scatole sono cubi bianchi dipinti con pattern alternati in bianco e nero. Alla fine del nastro trasportatore in movimento c'è un robot in stile retrò, dotato di strumenti e sensori, che classifica e conta diligentemente le ruote e le scatole in arrivo. L'estetica generale ricorda l'animazione di metà secolo con linee audaci e una tavolozza di colori classica.

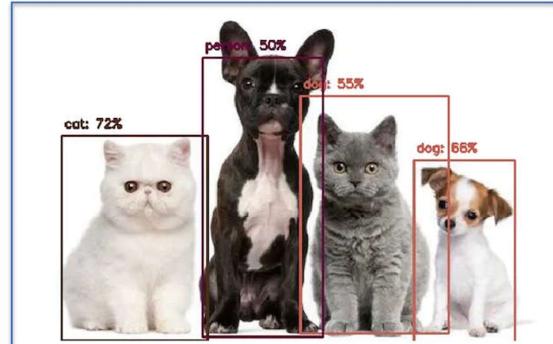
Panoramica

Questo è il seguito di **CV su Nicla Vision**, che ora esplora l'**Object Detection** sui microcontrollori.

Image Classification (Multi-Class Classification)



Object Detection Multi-Label Classification + Object Localization

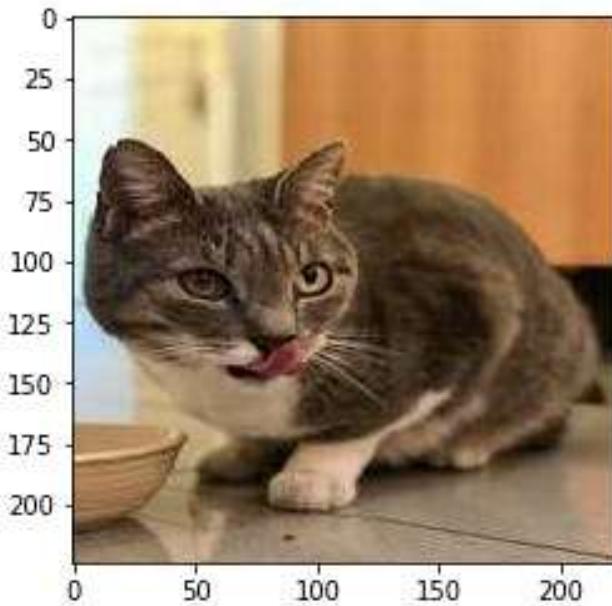


Object Detection e Image Classification

Il compito principale dei modelli di Image Classification [classificazione delle immagini](#) è quello di produrre un elenco delle categorie di oggetti più probabili presenti in un'immagine, ad esempio, per identificare un gatto soriano subito dopo cena:

[PREDICTION] :

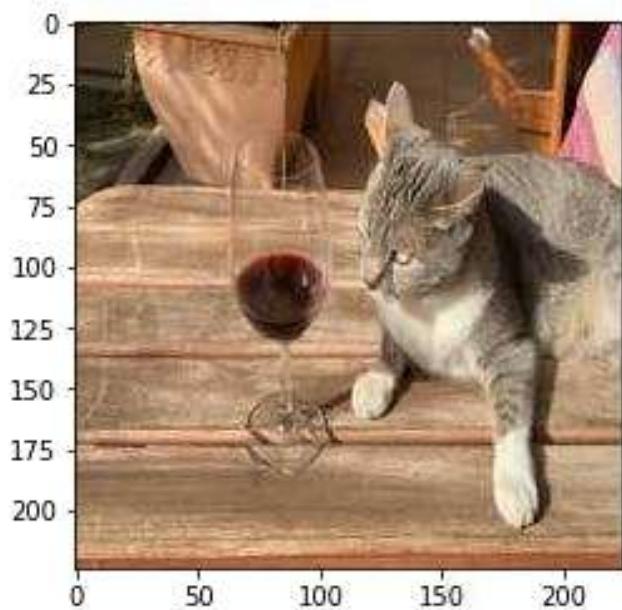
- 1) [tabby] ==> Probability of 30%
- 2) [bow tie] ==> Probability of 11%
- 3) [Egyptian cat] ==> Probability of 18%



Ma cosa succede quando il gatto salta vicino al bicchiere di vino? Il modello riconosce ancora solo la categoria predominante nell'immagine, il gatto soriano:

[PREDICTION]:

- 1) [tabby] ==> Probability of 53%
- 2) [tiger cat] ==> Probability of 23%
- 3) [Egyptian cat] ==> Probability of 10%



E cosa succede se non c'è una categoria dominante nell'immagine?

[PREDICTION]	[Prob]
ashcan	: 27%
Egyptian cat	: 19%
hamper	: 13%

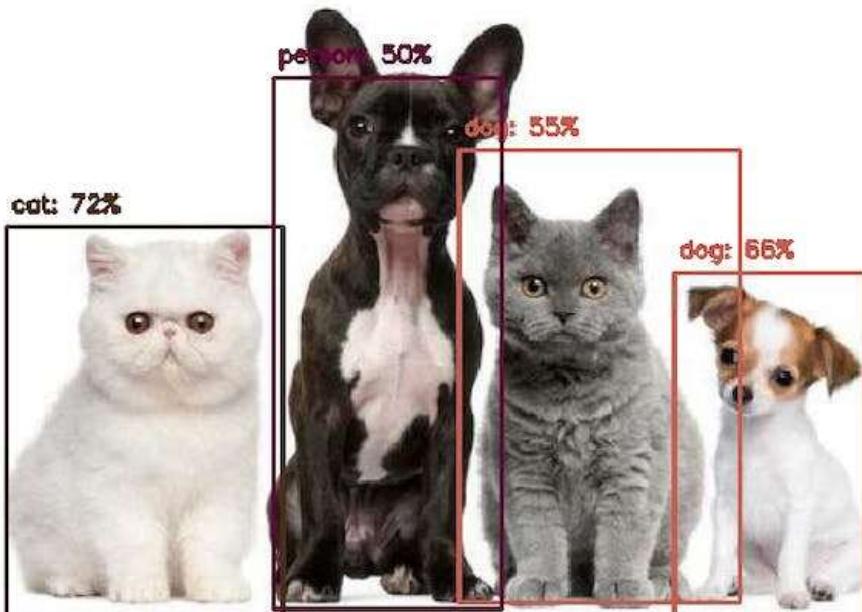


Il modello identifica l'immagine soprastante in modo completamente sbagliato come un "ashcan" [?pattumiera?], probabilmente a causa delle tonalità di colore.

Il modello utilizzato in tutti gli esempi precedenti è *MobileNet*, addestrato con un ampio set di dati, la *ImageNet*.

Per risolvere questo problema, abbiamo bisogno di un altro tipo di modello, in cui non solo possono essere trovate **più categorie** (o etichette), ma anche **dove** si trovano gli oggetti in una determinata immagine.

Come possiamo immaginare, tali modelli sono molto più complicati e più grandi, ad esempio, **MobileNetV2 SSD FPN-Lite 320x320, addestrato con il set di dati COCO**. Questo modello di rilevamento degli oggetti pre-addestrato è progettato per individuare fino a 10 oggetti all'interno di un'immagine, generando un riquadro di delimitazione per ogni oggetto rilevato. L'immagine sottostante è il risultato di un tale modello in esecuzione su un Raspberry Pi:



Quei modelli utilizzati per il rilevamento degli oggetti (come MobileNet SSD o YOLO) hanno solitamente dimensioni di diversi MB, il che è OK per l'uso con Raspberry Pi ma non adatto per l'uso con dispositivi embedded, dove la RAM solitamente è inferiore a 1 M Byte.

Una soluzione innovativa per il rilevamento degli oggetti: FOMO

Edge Impulse ha lanciato nel 2022 **FOMO** (Faster Objects, More Objects), una nuova soluzione per eseguire il rilevamento di oggetti su dispositivi embedded, non solo su Nicla Vision (Cortex M7) ma anche su CPU Cortex M4F (serie Arduino Nano33 e OpenMV M4) e sui dispositivi Espressif ESP32 (ESP-CAM e XIAO ESP32S3 Sense).

In questo esercizio pratico, esploreremo l'uso di FOMO con Object Detection, senza entrare in molti dettagli sul modello stesso. Per saperne di più su come funziona il modello, si può esaminare l'[annuncio ufficiale FOMO](#) di Edge Impulse, dove Louis Moreau e Mat Kelcey spiegano in dettaglio come funziona.

Obiettivo del Progetto di Object Detection

Tutti i progetti di apprendimento automatico devono iniziare con un obiettivo dettagliato. Supponiamo di trovarci in una struttura industriale e di dover ordinare e contare **ruote** e **scatole** speciali.



In altre parole, dovremmo eseguire una classificazione multi-etichetta, in cui ogni immagine può avere tre classi:

- Background [Sfondo] (nessun oggetto)
- Box [Scatola]
- Wheel [Ruota]

Ecco alcuni campioni di immagini non etichettate che dovremmo usare per rilevare gli oggetti (ruote e scatole):



Siamo interessati a quale oggetto è presente nell'immagine, alla sua posizione (centroide) e a quanti ne possiamo trovare su di essa. La dimensione dell'oggetto non viene rilevata con FOMO, come con MobileNet SSD o YOLO, in cui il Bounding Box è uno degli output del modello.

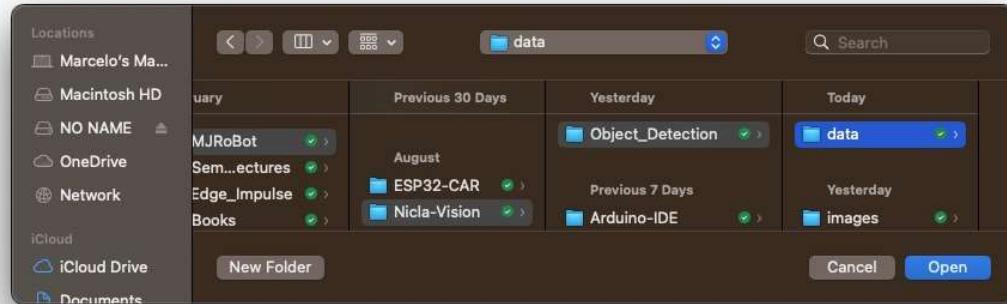
Svilupperemo il progetto utilizzando Nicla Vision per l'acquisizione di immagini e l'inferenza del modello. Il progetto ML verrà sviluppato utilizzando Edge Impulse Studio. Ma prima di iniziare il progetto di "object detection" in Studio, creiamo un *dataset* grezzo (non etichettato) con immagini che contengono gli oggetti da rilevare.

Raccolta Dati

Possiamo utilizzare Edge Impulse Studio, OpenMV IDE, il telefono o altri dispositivi per l'acquisizione delle immagini. Qui, utilizzeremo di nuovo OpenMV IDE per il nostro scopo.

Raccolta del Dataset con OpenMV IDE

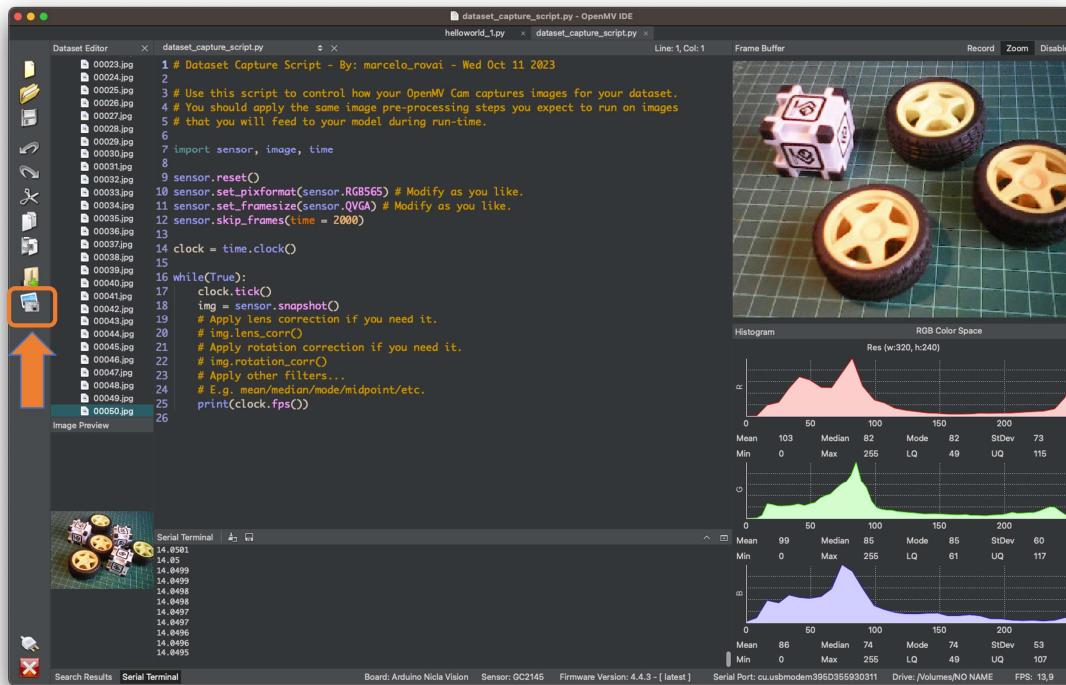
Per prima cosa, si crea sul computer una cartella in cui verranno salvati i dati, ad esempio "data". Quindi, su OpenMV IDE, si va in "Tools > Dataset Editor" e si seleziona "New Dataset" per avviare la raccolta di dati:



Edge impulse suggerisce che gli oggetti dovrebbero essere di dimensioni simili e non sovrapposti per prestazioni migliori. Questo va bene in una struttura industriale, dove la telecamera dovrebbe essere fissa, mantenendo la stessa distanza dagli oggetti da rilevare. Nonostante ciò, proveremo anche con dimensioni e posizioni miste per vedere il risultato.

Non creeremo cartelle separate per le nostre immagini perché ciascuna contiene più etichette.

Si collega Nicla Vision a OpenMV IDE e si esegue `dataset_capture_script.py`. Cliccando sul pulsante “Capture Image” inizierà l’acquisizione delle immagini:



Suggeriamo circa 50 immagini che mescolano gli oggetti e variano il numero di ciascuno che appare sulla scena. Provare ad acquisire con diverse angolazioni, sfondi e condizioni di luce.

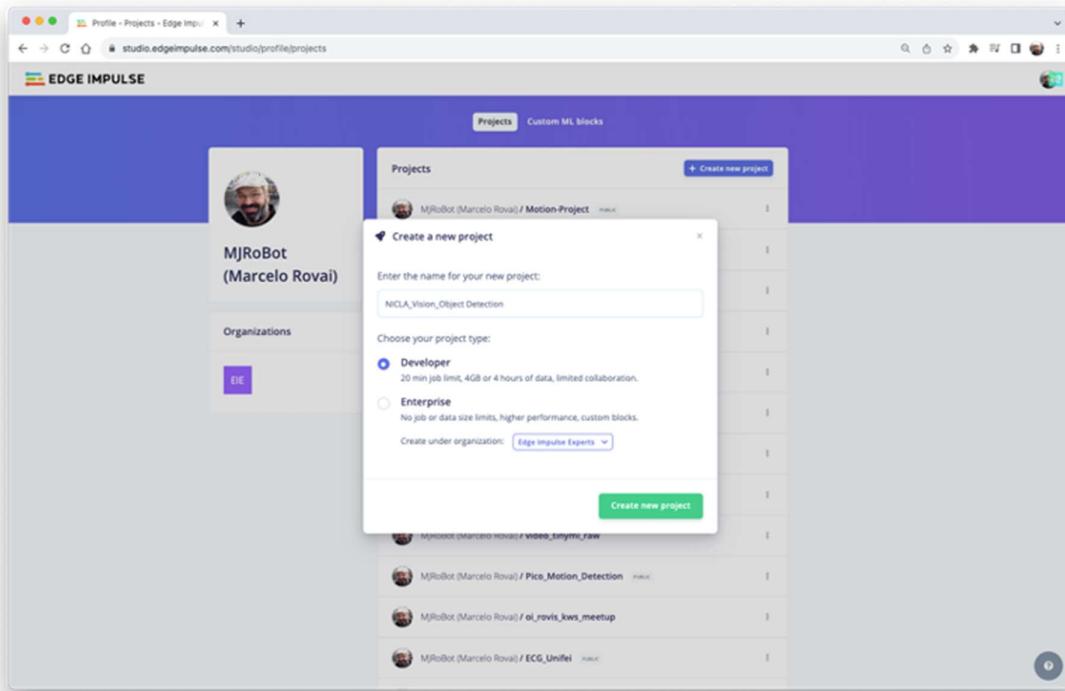
Le immagini memorizzate utilizzano una dimensione del fotogramma QVGA 320x240 e RGB565 (formato pixel a colori).

Dopo aver acquisito il dataset, si chiude il Tool Dataset Editor su `Tools > Dataset Editor`.

Edge Impulse Studio

Setup del progetto

Si va su [Edge Impulse Studio](#), si inseriscono le proprie credenziali in **Login** (o si crea un account) e si avvia un nuovo progetto.

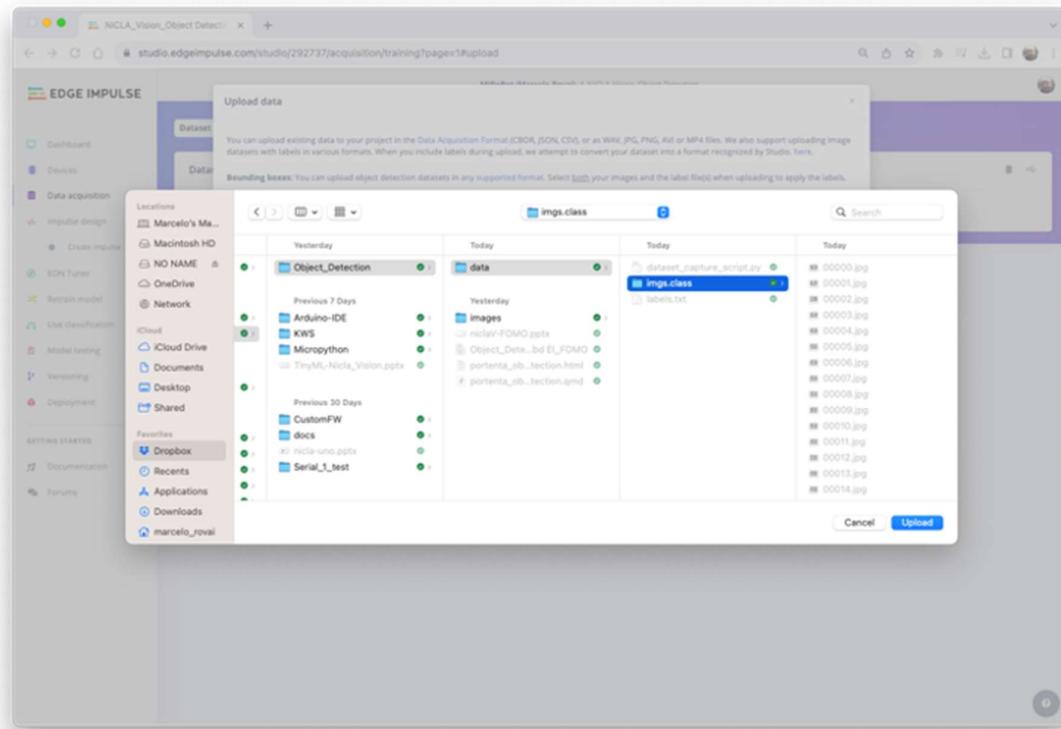


Qui si può clonare il progetto sviluppato per questa esercitazione pratica:
NICLA_Vision_Object_Detection.

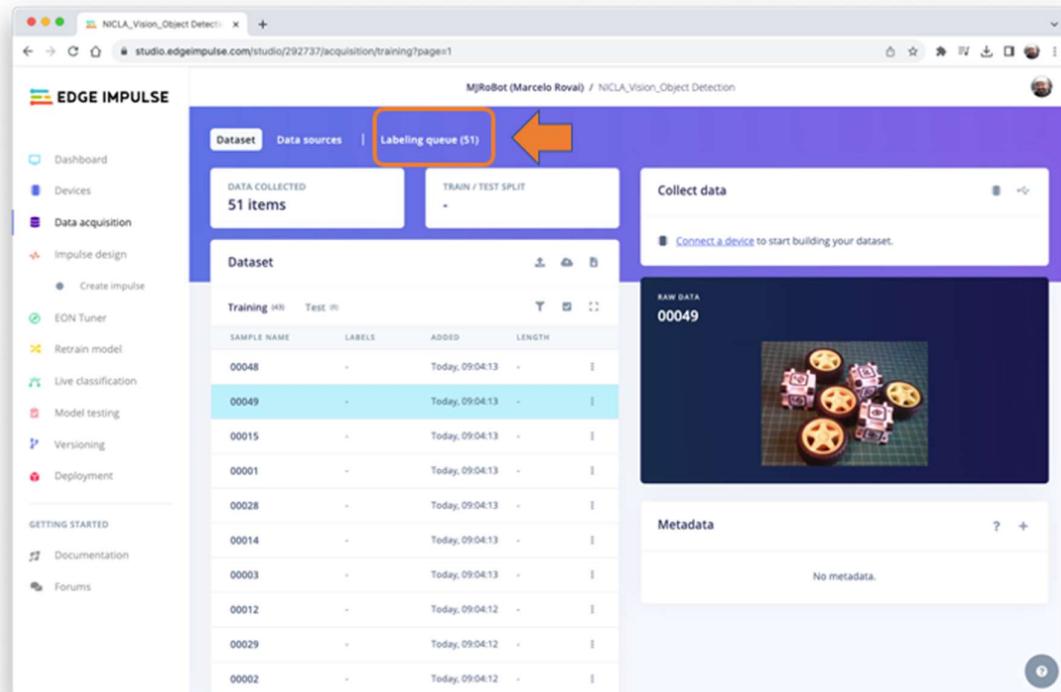
Nella “Project Dashboard”, si va in basso e su **Project info** e si seleziona **Bounding boxes (object detection)** e Nicla Vision come “Target Device”:

Caricamento dei dati non etichettati

Su Studio, si va alla scheda **Data acquisition** e nella sezione **UPLOAD DATA**, si caricano dal computer i file acquisiti.



Si può lasciare che Studio divida automaticamente i dati tra “Train” e “Test” o farlo manualmente.



Tutte le immagini non etichettate (51) sono state caricate, ma devono comunque essere etichettate in modo appropriato prima di utilizzarle come set di dati nel progetto. Lo Studio ha uno strumento per questo scopo, che si trova al link [Labeling queue \(51\)](#). Ci sono due modi per eseguire l’etichettatura assistita dall’IA su Edge Impulse Studio (versione gratuita):

- Utilizzando yolov5

- Tracciando di oggetti tra i frame

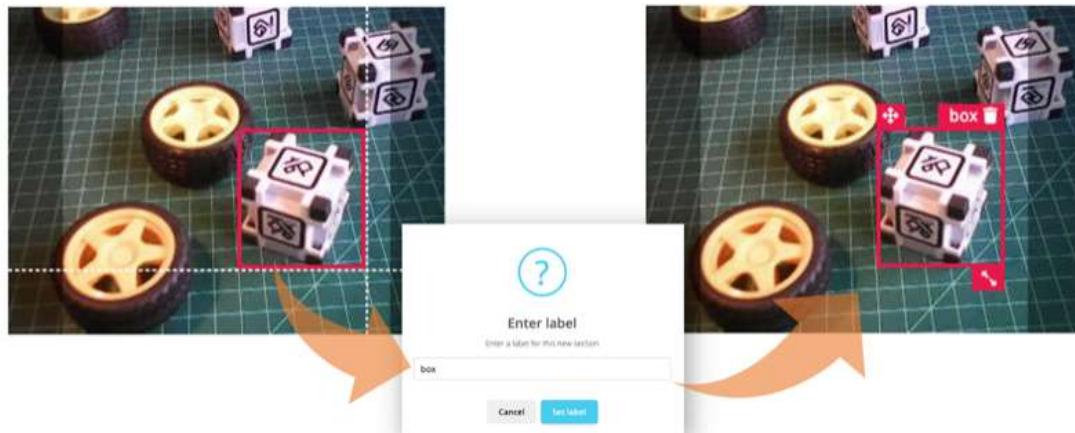
Edge Impulse ha lanciato una [funzione di auto-labeling](#) per i clienti Enterprise, semplificando le attività di etichettatura nei progetti di rilevamento degli oggetti.

Gli oggetti ordinari possono essere rapidamente identificati ed etichettati utilizzando una libreria esistente di modelli di rilevamento degli oggetti pre-addestrati da YOLOv5 (addestrati con il set di dati COCO). Ma poiché, nel nostro caso, gli oggetti non fanno parte dei set di dati COCO, dovremmo selezionare l'opzione di `tracking objects`. Con questa opzione, una volta disegnati i riquadri di delimitazione ed etichettate le immagini in un frame, gli oggetti verranno tracciati automaticamente da un frame all'altro, etichettando *parzialmente* quelli nuovi (non tutti sono etichettati correttamente).

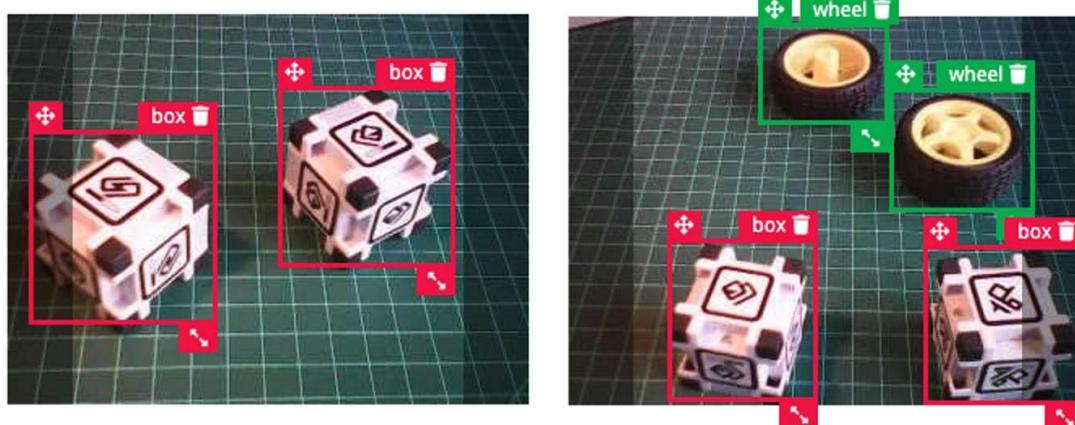
Si può usare [El uploader](#) per importare i dati se si ha già un dataset etichettato contenente dei “bounding box”.

Etichettatura del Dataset

Iniziando dalla prima immagine dei dati non etichettati, si usa il mouse per trascinare una casella attorno a un oggetto per aggiungere un'etichetta. Poi si clicca su **Save labels** per passare all'elemento successivo.



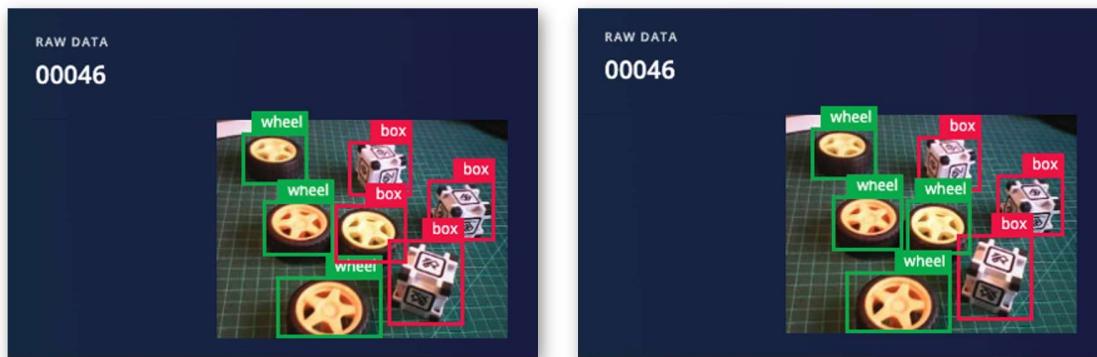
Si continua con questo processo finché la coda non è vuota. Alla fine, tutte le immagini dovrebbero avere gli oggetti etichettati come i campioni sottostanti:



Poi, si esaminano i campioni etichettati nella scheda `Data acquisition`. Se un'etichetta è sbagliata, la si può modificare usando il menù `three dots` dopo il nome del campione:

The screenshot shows the Edge Impulse Studio interface. On the left, a sidebar lists various tools: Dashboard, Devices, Data acquisition (with sub-options like Create impulse, EON Tuner, Retrain model, Live classification, Model testing, Versioning, Deployment), and Getting Started (Documentation, Forums). The main area is titled "Dataset" and shows "DATA COLLECTED 51 items". A progress bar indicates "TRAIN / TEST SPLIT 86% / 14%". Below this, a table lists training samples with their labels: 00037 (wheel, wheel, whe...), 00050 (box, box, box, whe...), 00025 (wheel), 00041 (box, box, box, box, ...), 00046 (box, box, box, box, ...), 00045 (box, box, wheel, w...), and 00043 (box, box, box, whe...). A context menu is open over sample 00046, showing options: Rename, Edit labels, Clear labels, Move to test set, Disable, Download, and Delete. To the right, a "Collect data" section says "Connect a device to start building your dataset." Below it, a "RAW DATA" section shows a grid of images labeled "wheel" and "box". A "Metadata" section below the raw data states "No metadata.".

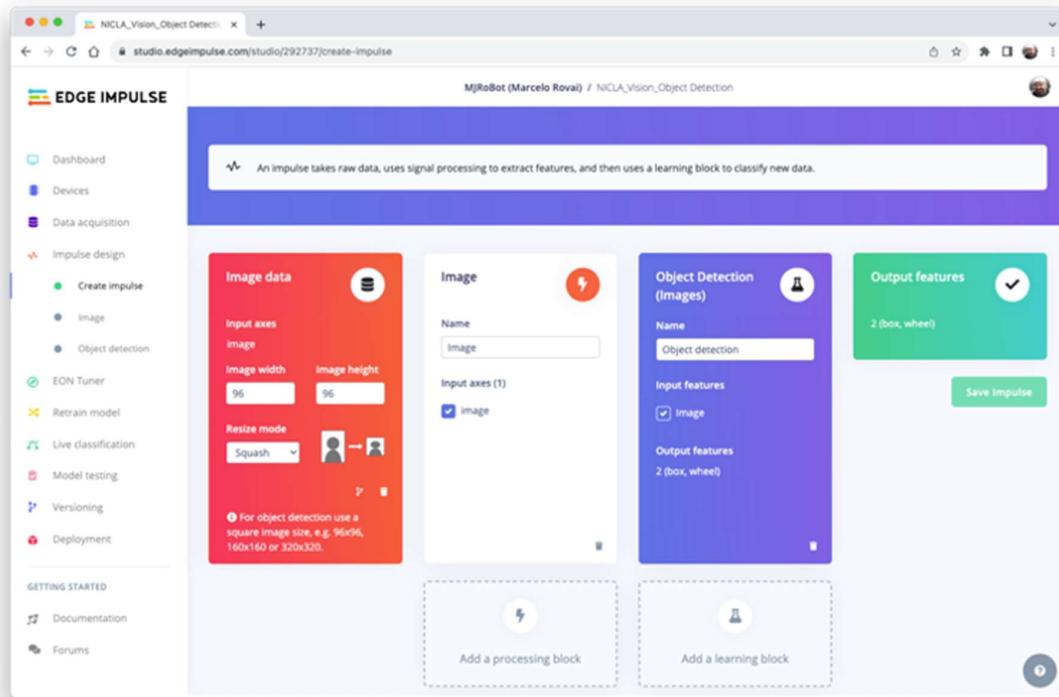
Si verrà guidati a sostituire l'etichetta sbagliata, correggendo il dataset.



Impulse Design

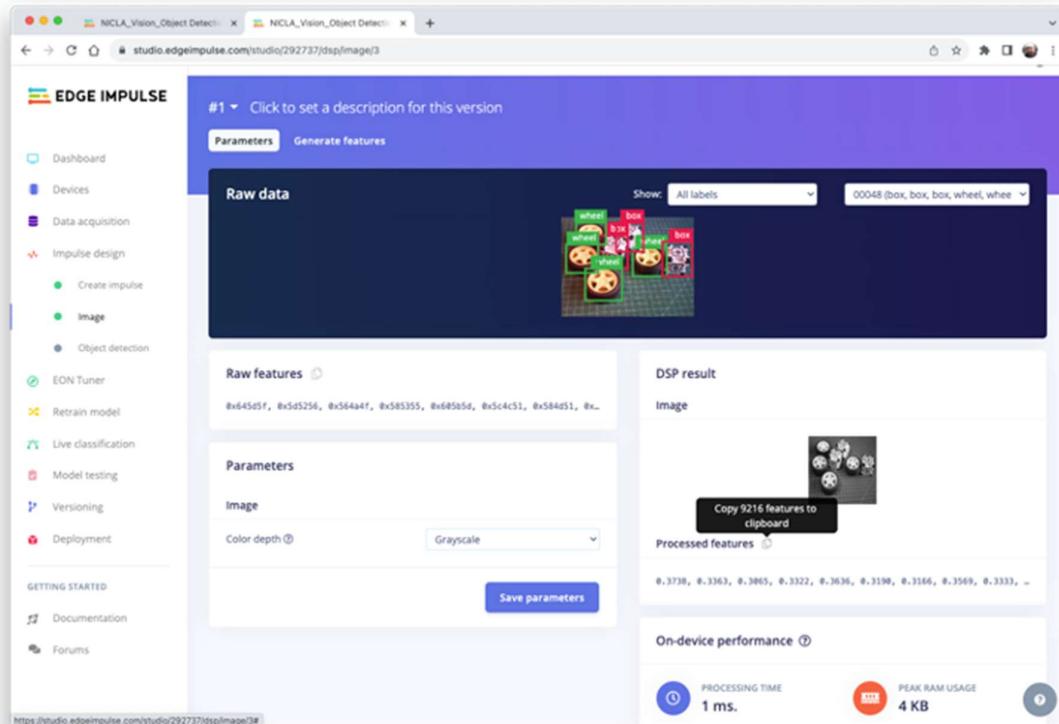
In questa fase, si deve definire come:

- Il **Pre-processing** consiste nel ridimensionare le singole immagini da 320 x 240 a 96 x 96 e nel comprimerle (forma quadrata, senza ritaglio). Successivamente, le immagini vengono convertite da RGB a scala di grigi.
- **Design a Model**, in questo caso, “Object Detection”.

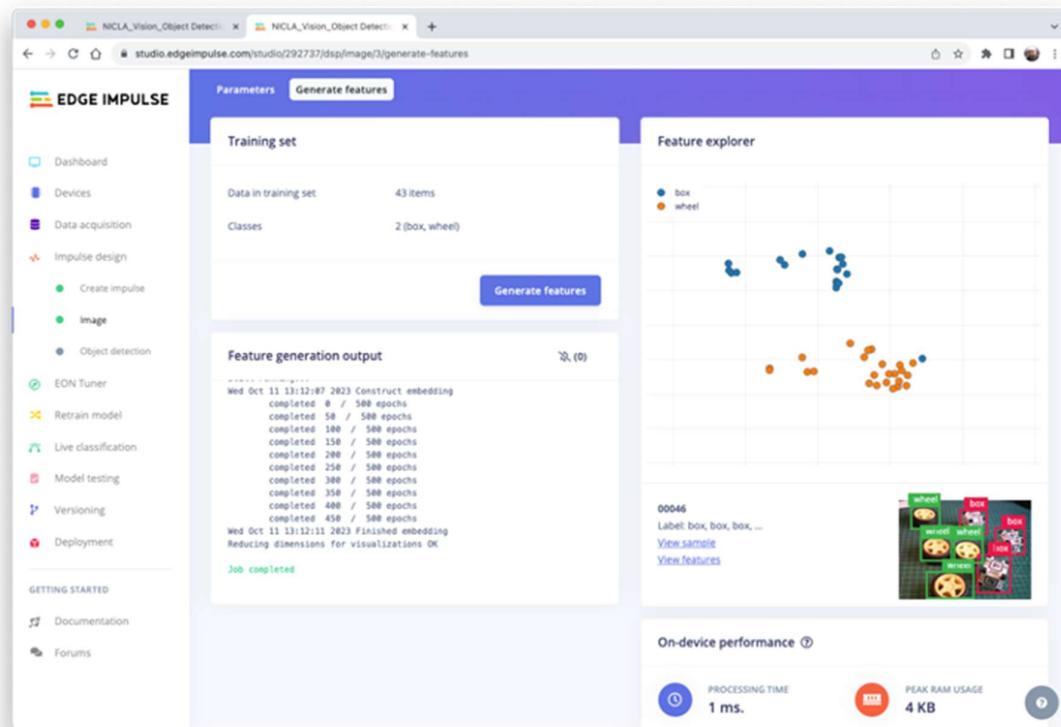


Pre-elaborazione di tutti i dataset

In questa sezione, si seleziona **Color depth** come Grayscale, che è adatta per l'uso con modelli FOMO e Save parameters.



Lo Studio passa automaticamente alla sezione successiva, **Generate features**, dove tutti i campioni saranno pre-elaborati, con conseguente set di dati con singole immagini 96x96x1 o 9.216 "feature".



L'esploratore di feature mostra che tutti i campioni evidenziano una buona separazione dopo la generazione delle feature.

Uno dei campioni (46) apparentemente si trova nello spazio sbagliato, ma cliccandoci sopra si può confermare che l'etichettatura è corretta.

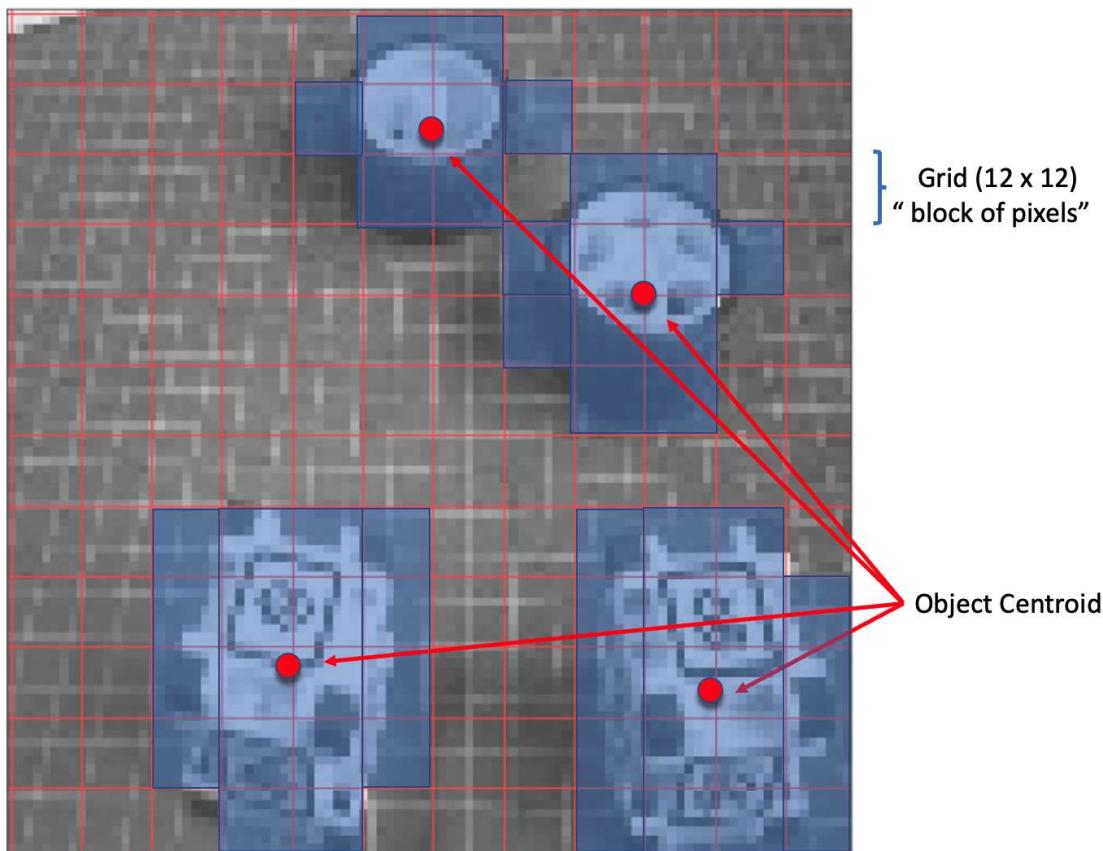
Progettazione, Addestramento e Test del Modello

Useremo FOMO, un modello di rilevamento degli oggetti basato su MobileNetV2 (alpha 0.35) progettato per segmentare grossolanamente un'immagine in una griglia di **background** rispetto a **oggetti di interesse** (in questo caso, scatole e ruote).

FOMO è un modello di apprendimento automatico innovativo per il rilevamento degli oggetti, che può utilizzare fino a 30 volte meno energia e memoria rispetto ai modelli tradizionali come Mobilenet SSD e YOLOv5. FOMO può funzionare su microcontrollori con meno di 200 KB di RAM. Il motivo principale per cui ciò è possibile è che mentre altri modelli calcolano le dimensioni dell'oggetto disegnando un quadrato attorno ad esso (bounding box), FOMO ignora le dimensioni dell'immagine, fornendo solo le informazioni su dove si trova l'oggetto nell'immagine, tramite le coordinate del centroide.

Come funziona FOMO?

FOMO prende l'immagine in scala di grigi e la divide in blocchi di pixel usando un fattore di 8. Per l'input di 96x96, la griglia è 12x12 (96/8=12). Successivamente, FOMO eseguirà un classificatore attraverso ogni blocco di pixel per calcolare la probabilità che ci sia una scatola o una ruota in ognuno di essi e, successivamente, determinerà le regioni che hanno la più alta probabilità di contenere l'oggetto (se un blocco di pixel non ha oggetti, verrà classificato come *background*). Dalla sovrapposizione della regione finale, FOMO fornisce le coordinate (relative alle dimensioni dell'immagine) del centroide di questa regione.



Per l'addestramento, dovremo selezionare un modello pre-addestrato. Usiamo **FOMO** (**Faster Objects, More Objects**) **MobileNetV2 0.35**. Questo modello utilizza circa 250 KB di RAM e 80 KB di ROM (Flash), che si adatta bene alla nostra scheda poiché ha 1 MB di RAM e ROM.

The screenshot shows the Edge Impulse web interface. On the left, there is a sidebar with various options like Dashboard, Devices, Data acquisition, and Model testing. The main area displays a 'Choose a different model' dialog. It lists several pre-trained models:

- MobileNetV2 SSD FPN-Lite 320x320** (OFFICIALLY SUPPORTED): A pre-trained object detection model designed to locate up to 10 objects within an image, outputting a bounding box for each object detected. The model is around 3.7MB in size. It supports an RGB input at 320x320px.
- FOMO (Faster Objects, More Objects) MobileNetV2 0.1** (OFFICIALLY SUPPORTED): An object detection model based on MobileNetV2 (alpha 0.1) designed to coarsely segment an image into a grid of background vs objects of interest. These models are designed to be <100KB in size and support a grayscale input at any resolution.
- FOMO (Faster Objects, More Objects) MobileNetV2 0.35** (OFFICIALLY SUPPORTED): An object detection model based on MobileNetV2 (alpha 0.35) designed to coarsely segment an image into a grid of background vs objects of interest. These models are designed to be <100KB in size and support a grayscale input at any resolution.
- YOLOv5 for Renesas DRP-AI** (COMMUNITY): Transfer learning model using YOLOv5 v5 branch with yolov5s.pt weights. This block is only compatible with Renesas DRP-AI.
- YOLOv5** (COMMUNITY): Transfer learning model based on Ultralytics YOLOv5 using yolov5n.pt weights, supports RGB input at any resolution (square images only).
- YOLOX for TI TDA4VM** (COMMUNITY): TI's EDGEAI YOLOX. <https://github.com/TexasInstruments/edgeai-yolox>. Outputs ONNX v7 model format both with and without final detect layers using PyTorch 1.7.1. See the implementation <https://github.com/edgeimpulse/example-custom-mi-block-s-yolox/tree/onnx-v7>

Per quanto riguarda gli iperparametri di training, il modello verrà addestrato con:

- Epochs: 60,

- Batch size: 32
- Learning Rate: 0.001.

Per la convalida durante l'addestramento, il 20% del set di dati (*validation_dataset*) verrà risparmiato. Per il restante 80% (*train_dataset*), applicheremo il "Data Augmentation", che capovolgerà casualmente, cambierà le dimensioni e la luminosità dell'immagine e le ritaglierà, aumentando artificialmente il numero di campioni sul set di dati per l'addestramento.

Di conseguenza, il modello termina con praticamente 1,00 nel punteggio F1, con un risultato simile quando si utilizzano i dati di test.

Notare che FOMO ha aggiunto automaticamente una terza etichetta di background [sfondo] ai due precedentemente definiti (*box* e *wheel*).

The screenshot shows two panels of the Nicla Vision Studio. The left panel, titled 'Neural Network settings', contains sections for 'Training settings' (Number of training cycles: 60, Learning rate: 0.001, Data augmentation checked), 'Advanced training settings' (Validation set size: 20%, Split train/validation set on metadata key, Batch size: 32, Profile int8 model checked), and 'Neural network architecture' (Input layer (9,216 features) showing 'FOMO (Faster Objects, More Objects) MobileNetV2 0.35', Choose a different model, Output layer (2 classes)). The right panel, titled 'Model', shows 'Last training performance (validation set)' with an F1 score of 93.9% and a confusion matrix:

	BACKGROUND	BOX	WHEEL
BACKGROUND	99.7%	0.3%	0%
BOX	0%	100%	0%
WHEEL	0%	0%	100%
F1 SCORE	1.00	0.86	1.00

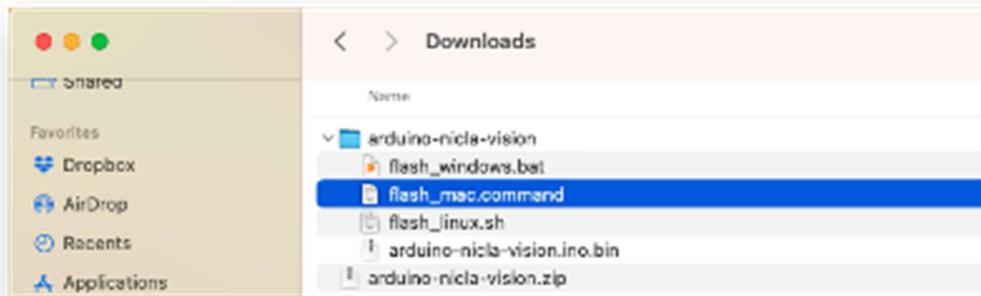
It also displays 'On-device performance' metrics: INFERENCING ... 91 ms., PEAK RAM US... 239.4K, and FLASH USAGE 77.7K.

Nelle attività di rilevamento degli oggetti, l'accuratezza non è generalmente la **evaluation metric** primaria. Il rilevamento degli oggetti comporta la classificazione degli oggetti e la fornitura di riquadri di delimitazione attorno a essi, il che lo rende un problema più complesso della semplice classificazione. Il problema è che non abbiamo il riquadro di delimitazione, solo i centroidi. In breve, usare l'accuratezza come metrifica potrebbe essere fuorviante e potrebbe non fornire una comprensione completa delle prestazioni del modello. Per questo motivo, useremo il punteggio F1.

Modello di test con "Live Classification"

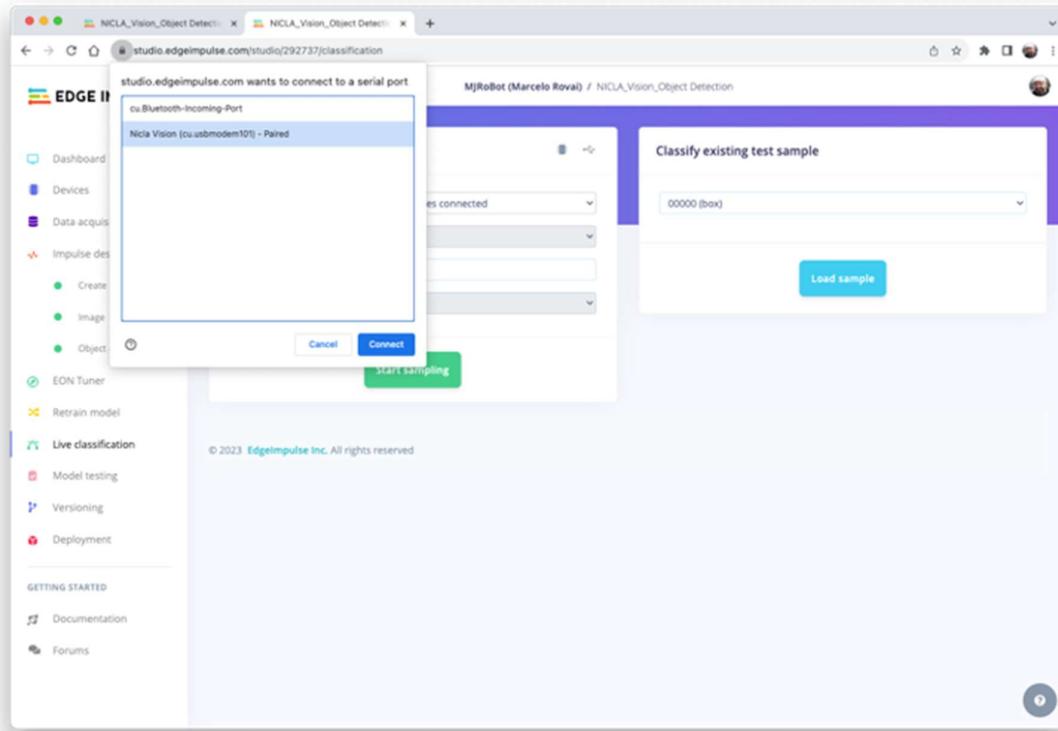
Dato che Edge Impulse supporta ufficialmente Nicla Vision, colleghiamolo allo Studio. Per farlo, si seguono i passaggi:

- Si effettua il download dell'[last EI Firmware](#) e lo si decomprime.
- Si apre il file zip sul computer e si seleziona l'uploader relativo al proprio sistema operativo:

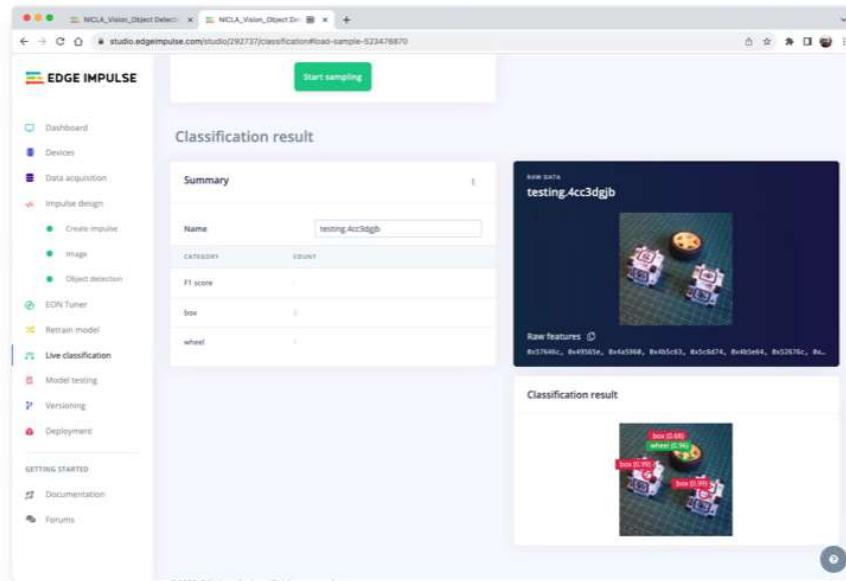


- Mettere Nicla-Vision in “Boot Mode”, premendo due volte il pulsante di reset.
- Eseguire il codice batch specifico per il sistema operativo per caricare il binario (`arduino-nicla-vision.bin`) sulla board.

Si va nella sezione `Live classification` su El Studio e, tramite `webUSB`, si connette la Nicla Vision:



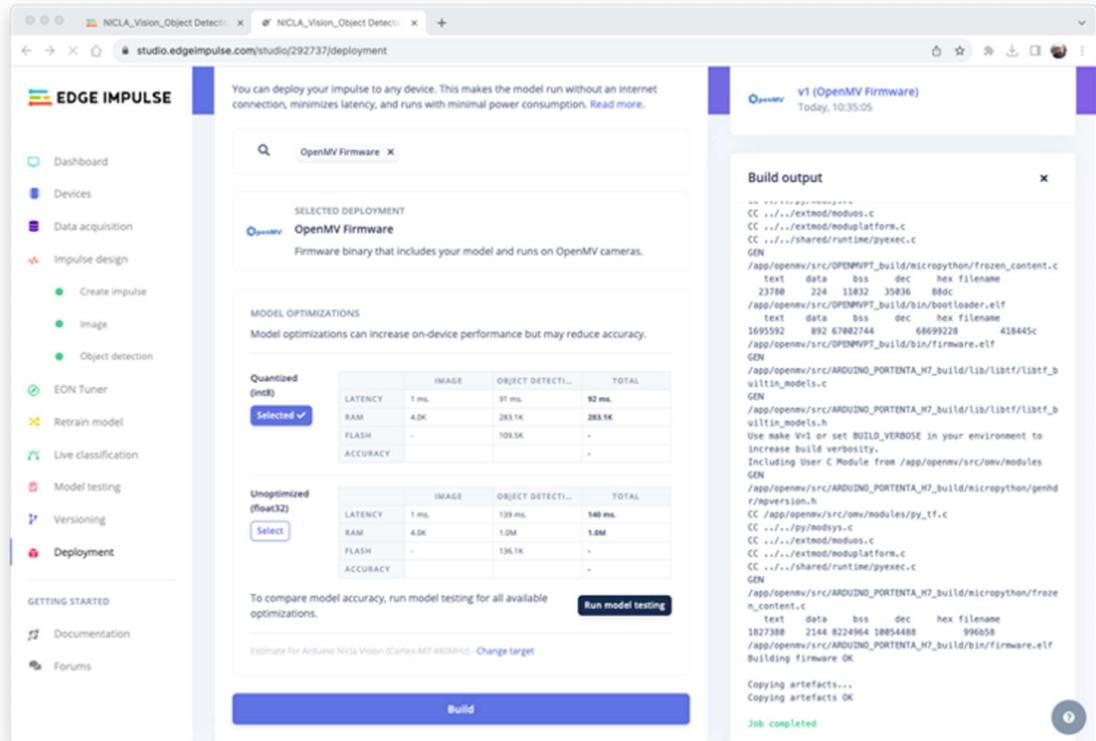
Una volta connessa, si può usare la Nicla per catturare immagini reali da testare col modello addestrato su Edge Impulse Studio.



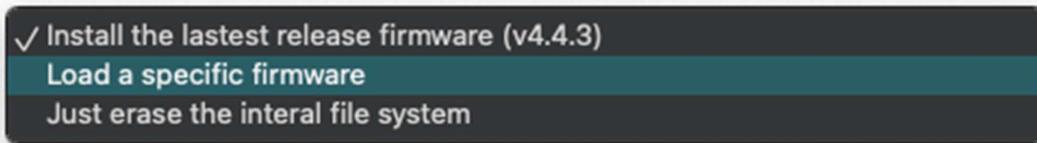
Una cosa da notare è che il modello può produrre falsi positivi e falsi negativi. Questo può essere ridotto al minimo definendo una **Confidence Threshold** [Soglia di confidenza] (si usa il menu **Three dots** [tre-punti] per la configurazione). Provare con 0,8 o più.

Distribuzione del Modello

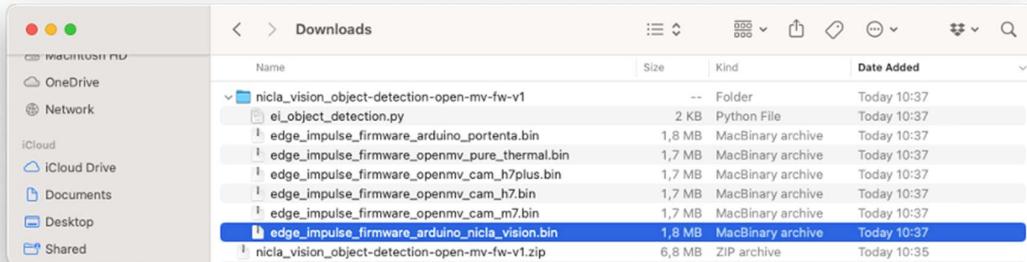
Selezionare OpenMV Firmware nella scheda Deploy e premere [Build].



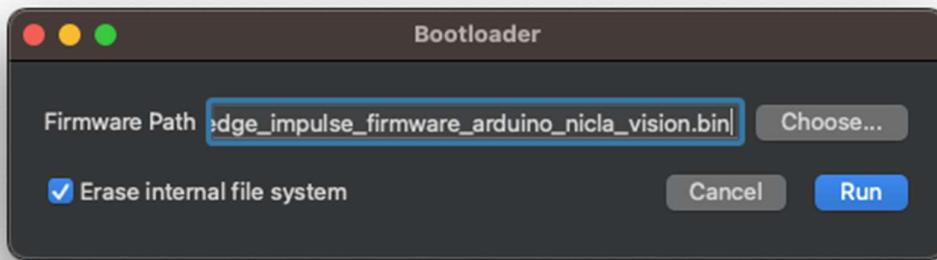
Quando si connette di nuovo Nicla con OpenMV IDE, proverà ad aggiornarne il FW. Scegliere invece l'opzione **Load a specific firmware**.



Si troverà un file ZIP sul computer dallo Studio. Lo si apre:



Caricare il file .bin sulla board:



Una volta terminato il download, verrà visualizzato un messaggio pop-up. Premere OK e aprire lo script **ei_object_detection.py** scaricato da Studio.

Prima di eseguire lo script, modifichiamo alcune righe. Notare che si può lasciare la definizione della finestra come 240 x 240 e la telecamera che cattura le immagini come QVGA/RGB. L'immagine catturata verrà pre-elaborata dal FW distribuito da Edge Impulse

```
# Edge Impulse - OpenMV Object Detection Example

import sensor, image, time, os, tf, math, uos, gc

sensor.reset()                      # Reset and initialize the sensor.
sensor.set_pixformat(sensor.RGB565)   # Set pixel format to RGB565 (or GRayscale)
sensor.set_framesize(sensor.QVGA)      # Set frame size to QVGA (320x240)
sensor.set_windowing((240, 240))       # Set 240x240 window.
sensor.skip_frames(time=2000)          # Let the camera adjust.
```

Ridefinire la confidenza minima, ad esempio, a 0,8 per ridurre al minimo i falsi positivi e negativi.

```
min_confidence = 0.8
```

Se necessario, modificare il colore dei cerchi che saranno utilizzati per visualizzare il centroide dell'oggetto rilevato per un contrasto migliore.

```
try:
    # Load built in model
    labels, net = tf.load_builtin_model('trained')
```

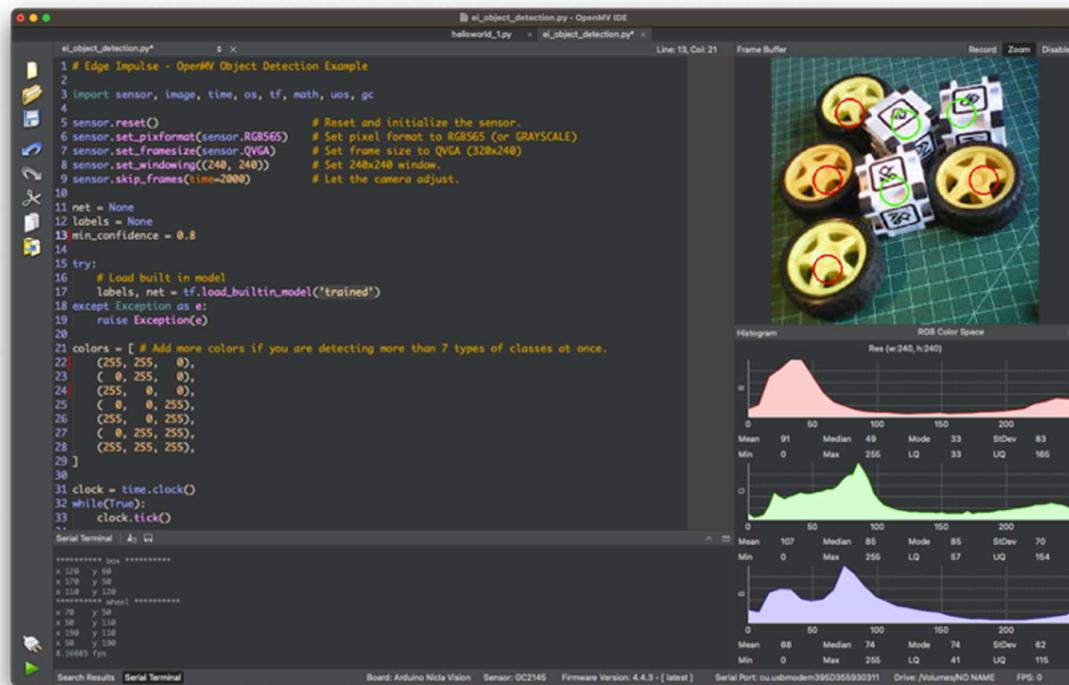
```

except Exception as e:
    raise Exception(e)

colors = [ # Add more colors if you are detecting more than 7 types of classes at once.
    (255, 255, 0), # background: yellow (not used)
    (0, 255, 0), # cube: green
    (255, 0, 0), # wheel: red
    (0, 0, 255), # not used
    (255, 0, 255), # not used
    (0, 255, 255), # not used
    (255, 255, 255), # not used
]

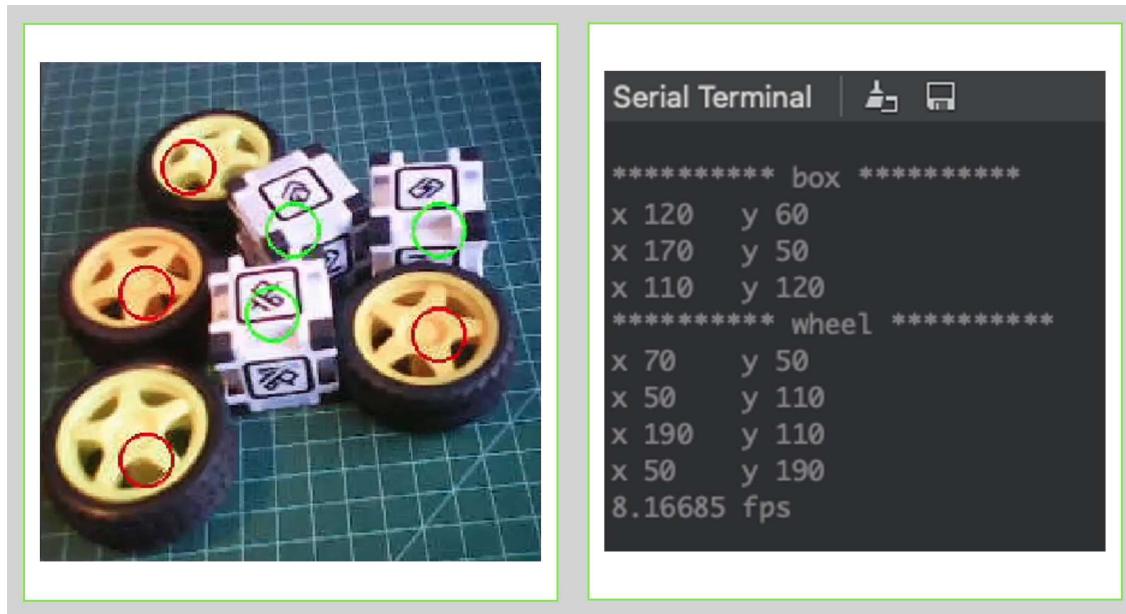
```

Mantenere il codice restante così com'è e premere il pulsante Play verde per eseguire il codice:



Nella vista della telecamera, possiamo vedere gli oggetti con i loro centroidi contrassegnati con 12 cerchi pixel-fixed (ogni cerchio ha un colore distinto, a seconda della sua classe). Sul terminale seriale, il modello mostra le etichette rilevate e la loro posizione sulla finestra dell'immagine (240X240).

Attenzione l'origine delle coordinate è nell'angolo in alto a sinistra.



Notare che la frequenza dei fotogrammi al secondo è di circa 8 fps (simile a quella ottenuta con il progetto Image Classification). Ciò accade perché FOMO è intelligentemente costruito su un modello CNN, non con un modello di rilevamento degli oggetti come SSD MobileNet. Ad esempio, quando si esegue un modello MobileNetV2 SSD FPN-Lite 320x320 su un Raspberry Pi 4, la latenza è circa 5 volte superiore (circa 1,5 fps). Ecco un breve video che mostra i risultati dell'inferenza: <https://youtu.be/JbpoqRp3BbM>

Conclusione

FOMO è un salto significativo nello spazio di elaborazione delle immagini, come hanno affermato Louis Moreau e Mat Kelcey durante il suo lancio nel 2022:

FOMO è un algoritmo rivoluzionario che porta per la prima volta il rilevamento, il tracciamento e il conteggio degli oggetti in tempo reale sui microcontrollori.

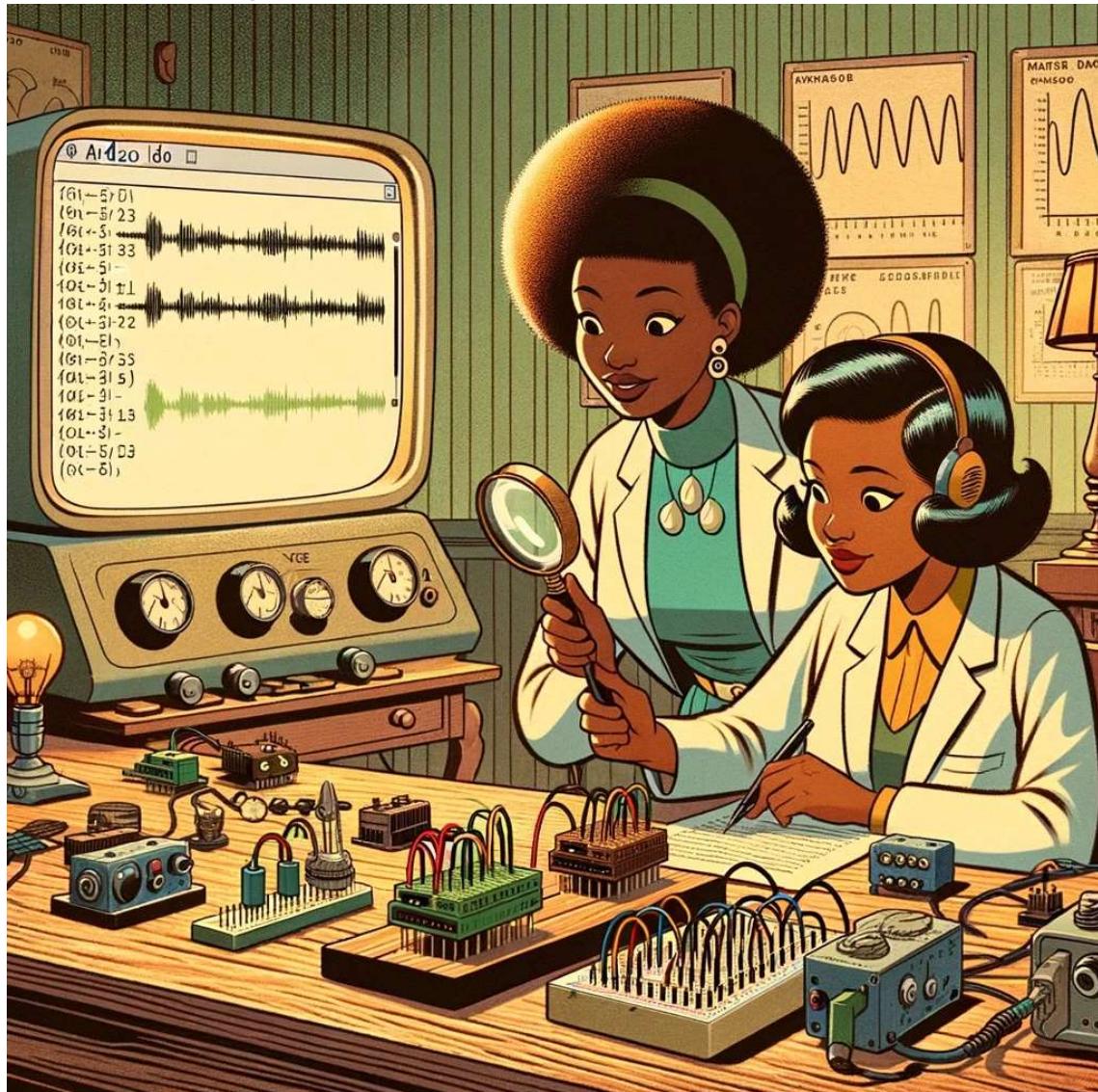
Esistono molteplici possibilità per esplorare il rilevamento degli oggetti (e, più precisamente, il loro conteggio) su dispositivi embedded, ad esempio, per esplorare la il raggruppamento di sensori (telecamera + microfono) e il rilevamento degli oggetti di Nicla. Questo può essere molto utile nei progetti che coinvolgono le api, ad esempio.



Risorse

- [Progetto Edge Impulse](#)

Keyword Spotting (KWS)



DALL·E 3 Prompt: Scena di un cartone animato in stile anni '50 ambientata in una sala di ricerca audio vintage. Al centro ci sono due scienziate afroamericane. Una tiene una lente di ingrandimento, esaminando attentamente circuiti antichi, mentre l'altra prende appunti. Sul loro tavolo di legno ci sono diverse schede con sensori, in particolare un microfono. Dietro queste schede, un computer con un ampio display arrotondato mostra l'IDE Arduino. L'IDE mostra il codice per le assegnazioni dei pin LED e l'inferenza di apprendimento automatico per il rilevamento dei comandi vocali. Una finestra distinta nell'IDE, il monitor seriale, rivela output che indicano i comandi vocali 'yes' e 'no'. L'atmosfera della stanza è nostalgica con lampade vintage, strumenti di analisi audio classici e grafici FFT e curve del dominio del tempo.

Panoramica

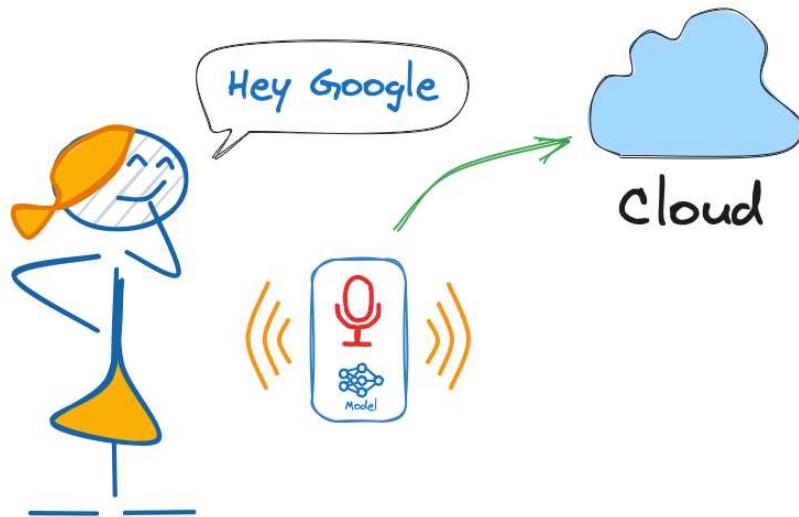
Dopo aver già esplorato la scheda Nicla Vision nelle applicazioni di *Image Classification* e *Object Detection*, stiamo ora spostando la nostra attenzione sulle applicazioni attivate tramite comando vocale con un progetto su Keyword Spotting (KWS).

Come introdotto nel tutorial *Feature Engineering for Audio Classification*, il Keyword Spotting (KWS) è integrato in molti sistemi di riconoscimento vocale, consentendo ai dispositivi di rispondere a parole o frasi specifiche. Sebbene questa tecnologia sia alla base di dispositivi popolari come Google Assistant o Amazon Alexa, è ugualmente applicabile e fattibile su dispositivi più piccoli e a basso consumo. Questo tutorial guiderà nell'implementazione di un sistema KWS utilizzando TinyML sulla scheda di sviluppo Nicla Vision dotata di un microfono digitale.

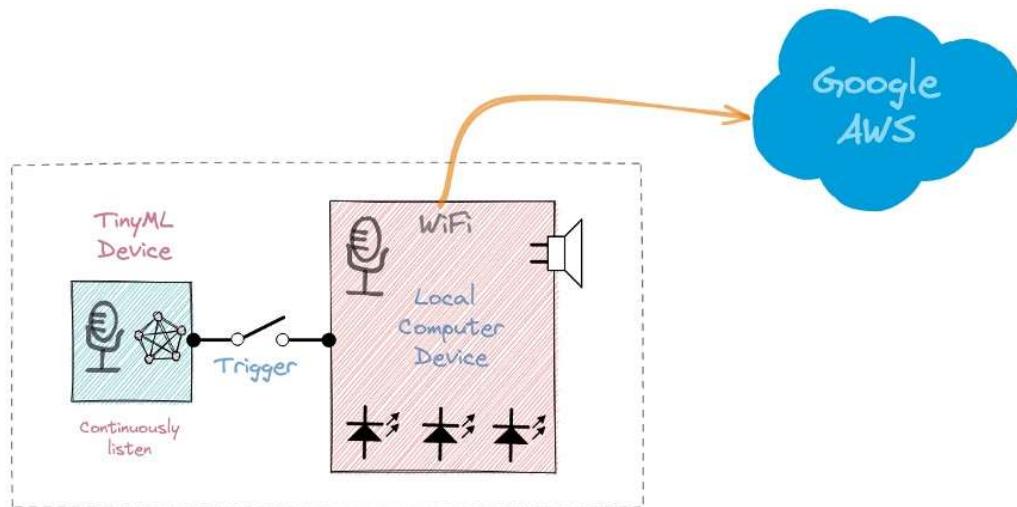
Il nostro modello sarà progettato per riconoscere parole chiave che possono riattivare un dispositivo o azioni specifiche, dando loro vita con comandi vocali.

Come funziona un assistente vocale?

Come detto, gli *assistenti vocali* sul mercato, come Google Home o Amazon Echo-Dot, reagiscono agli umani solo quando vengono “svegliati” da parole chiave particolari come “Hey Google” sul primo e “Alexa” sul secondo.



In altre parole, il riconoscimento dei comandi vocali si basa su un modello multi-fase o Cascade Detection.



Fase 1: Un piccolo microprocessore all'interno di Echo Dot o Google Home ascolta continuamente, in attesa che venga individuata la parola chiave, utilizzando un modello TinyML nel device (applicazione KWS).

Fase 2: Solo quando vengono attivati dall'applicazione KWS nella Fase 1, i dati vengono inviati al cloud ed elaborati su un modello più grande.

Il video qui sotto mostra un esempio di un Google Assistant programmato su un Raspberry Pi (Fase 2), con un Arduino Nano 33 BLE come dispositivo TinyML (Fase 1).

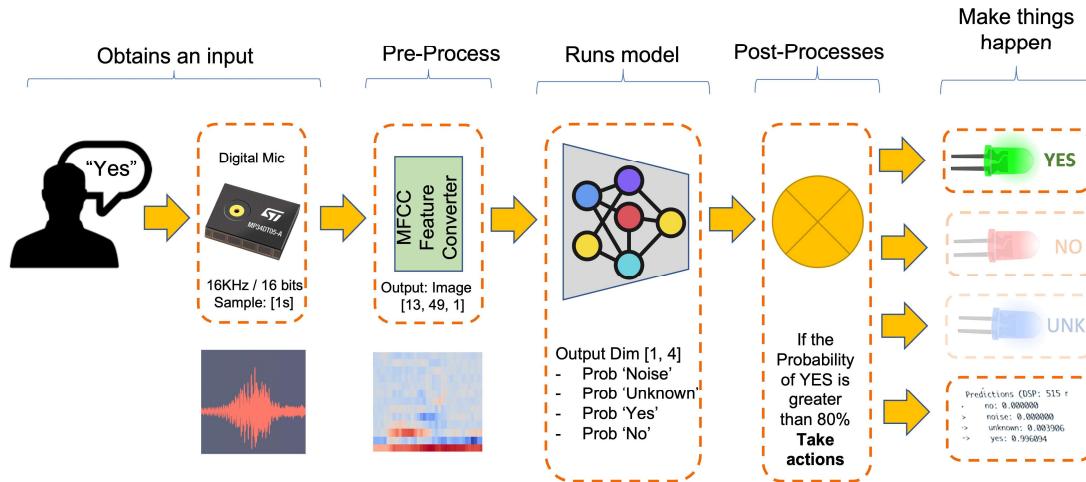
https://youtu.be/e_OPgcnsyvM

Per esplorare il progetto Google Assistant di cui sopra, consultare il tutorial:
[Building an Intelligent Voice Assistant From Scratch](#).

In questo progetto KWS, ci concentreremo sulla Fase 1 (KWS o Keyword Spotting), dove utilizzeremo Nicla Vision, che ha un microfono digitale che verrà utilizzato per individuare la parola chiave.

Il Progetto Pratico KWS

Il diagramma seguente fornisce un'idea di come dovrebbe funzionare l'applicazione KWS finale (durante l'inferenza):



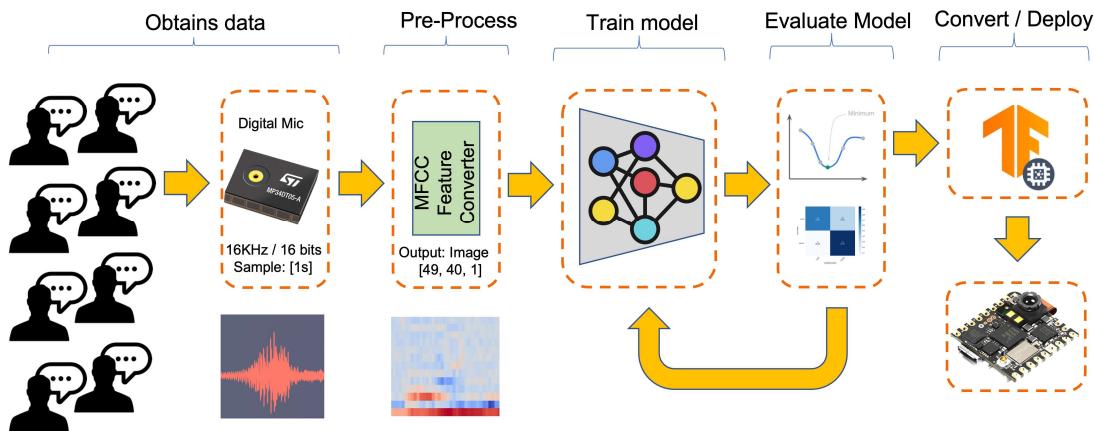
La nostra applicazione KWS riconoscerà quattro classi di suono:

- **YES** (Keyword 1)
- **NO** (Keyword 2)
- **NOISE** (nessuna parola pronunciata; è presente solo rumore di fondo)
- **UNKNOWN** (un mix di parole diverse da YES e NO)

Per progetti reali, è sempre consigliabile includere altri suoni oltre alle parole chiave, come "Noise" (o Background) e "Unknown".

Il Flusso di Lavoro del Machine Learning

Il componente principale dell'applicazione KWS è il suo modello. Quindi, dobbiamo addestrare un modello del genere con le nostre parole chiave specifiche, rumore e altre parole (lo "unknown"):



Il Dataset

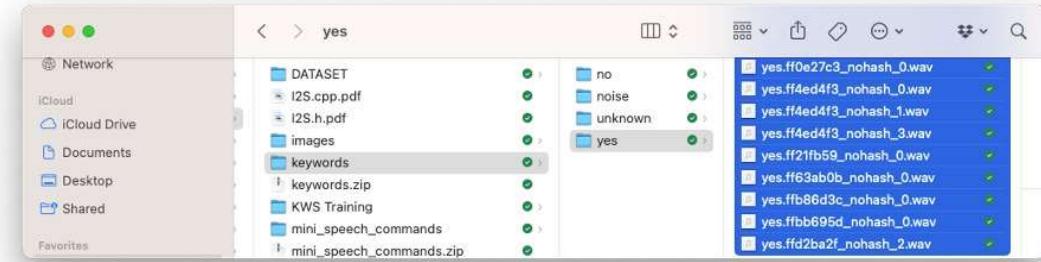
Il componente critico di qualsiasi flusso di lavoro di apprendimento automatico è il **dataset**. Una volta decise le parole chiave specifiche, nel nostro caso (YES e NO), possiamo sfruttare il dataset sviluppato da Pete Warden, ["Speech Commands: A Dataset for Limited-Vocabulary"](#)

Speech Recognition". Questo dataset ha 35 parole chiave (con +1.000 campioni ciascuna), come yes, no, stop e go. In parole come yes e no, possiamo ottenere 1.500 campioni. Si può scaricare una piccola parte del dataset da Edge Studio ([Keyword spotting pre-built dataset](#)), che include campioni dalle quattro classi che utilizzeremo in questo progetto: yes, no, noise e background. Per farlo, si seguono i passaggi seguenti:

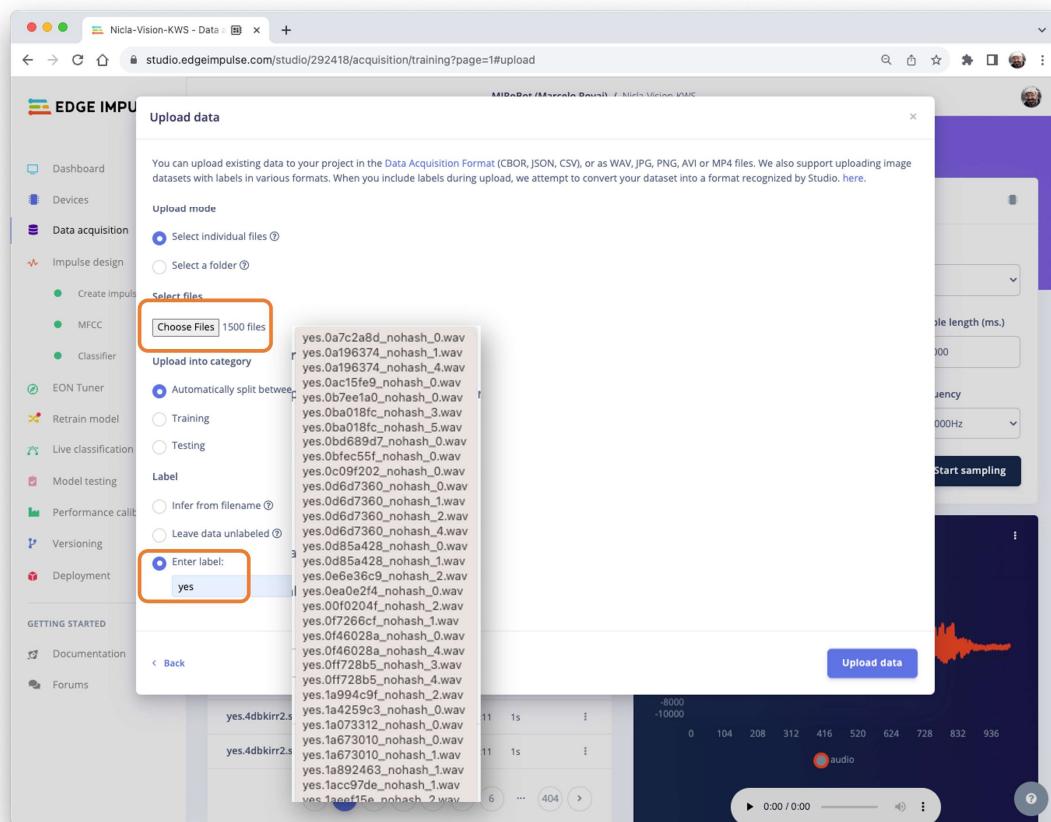
- Download del [dataset delle parole chiave](#).
- Unzip del file in una posizione a scelta.

Caricamento del set di dati su Edge Impulse Studio

Si avvia un nuovo progetto su Edge Impulse Studio (EIS) e si seleziona il tool [Upload Existing Data](#) nella sezione [Data Acquisition](#). Si scelgono i file da caricare:



Si definisce l'etichetta, si seleziona [Automatically split between train and test](#), e poi [Upload data](#) su EIS. Si ripete per tutte le classi.



Il dataset apparirà ora nella sezione [Data acquisition](#). Notare che i circa 6.000 campioni (1.500 per ogni classe) sono suddivisi in set di Train (4.800) e di Test (1.200).

The screenshot shows the Edge Impulse Studio interface. On the left is a sidebar with various options like Dashboard, Devices, Data acquisition, and Model testing. The main area has tabs for Dataset, Data explorer, Data sources, and CSV Wizard. The Dataset tab is active, showing a summary: "DATA COLLECTED 1h 42m 52s" and "TRAIN / TEST SPLIT 79% / 21%". Below this is a table titled "Dataset" with columns for SAMPLE NAME, LABEL, ADDED, and LENGTH. The table lists 404 entries, all labeled "yes". To the right of the table is a "RAW DATA" section showing a waveform for a file named "yes.4dbkt7s6.s8". The waveform has a red line on a dark background, with numerical values from -10000 to 10000 on the y-axis and time markers from 0 to 936 on the x-axis. Below the waveform is a "Metadata" section which is currently empty.

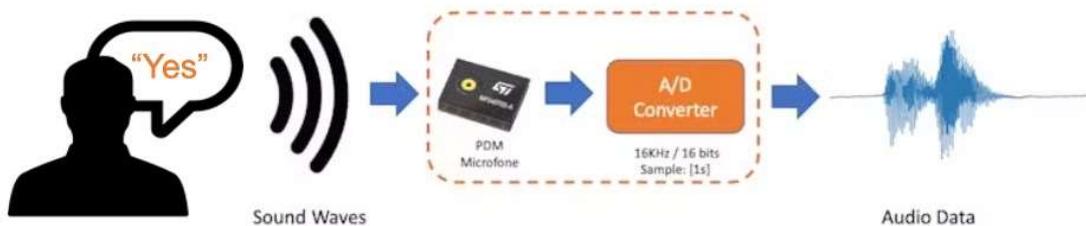
Acquisizione di Dati Audio Aggiuntivi

Sebbene disponiamo di molti dati dal dataset di Pete, è consigliabile raccogliere alcune parole pronunciate da noi. Lavorando con gli accelerometri, è essenziale creare un set di dati con dati acquisiti dallo stesso tipo di sensore. Nel caso del suono, questo è facoltativo perché ciò che classificheremo sono, in realtà, dati *audio*.

La differenza fondamentale tra suono e audio è il tipo di energia. Il suono è una perturbazione meccanica (onde sonore longitudinali) che si propagano attraverso un mezzo, causando variazioni di pressione in esso. L'audio è un segnale elettrico (analogico o digitale) che rappresenta il suono.

Quando pronunciamo una parola chiave, le onde sonore devono essere convertite in dati audio. La conversione deve essere eseguita campionando il segnale generato dal microfono a una frequenza di 16 KHz con ampiezza di 16 bit per campione.

Quindi, qualsiasi dispositivo in grado di generare dati audio con questa specifica di base (16 KHz/16 bit) funzionerà correttamente. Come *device*, possiamo usare NiclaV, un computer o persino il cellulare.

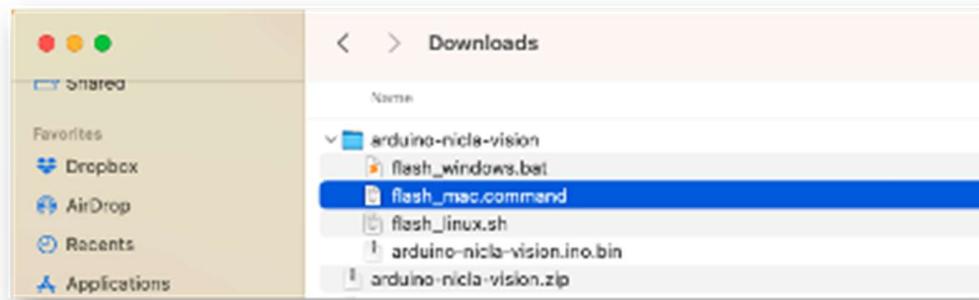


Utilizzo di NiclaV ed Edge Impulse Studio

Come abbiamo appreso nel capitolo *Configurazione di Nicla Vision*, EIS supporta ufficialmente Nicla Vision, semplificando l'acquisizione dei dati dai suoi sensori, incluso il

microfono. Quindi, si crea un nuovo progetto su EIS e vi si collega la Nicla, seguendo questi passaggi:

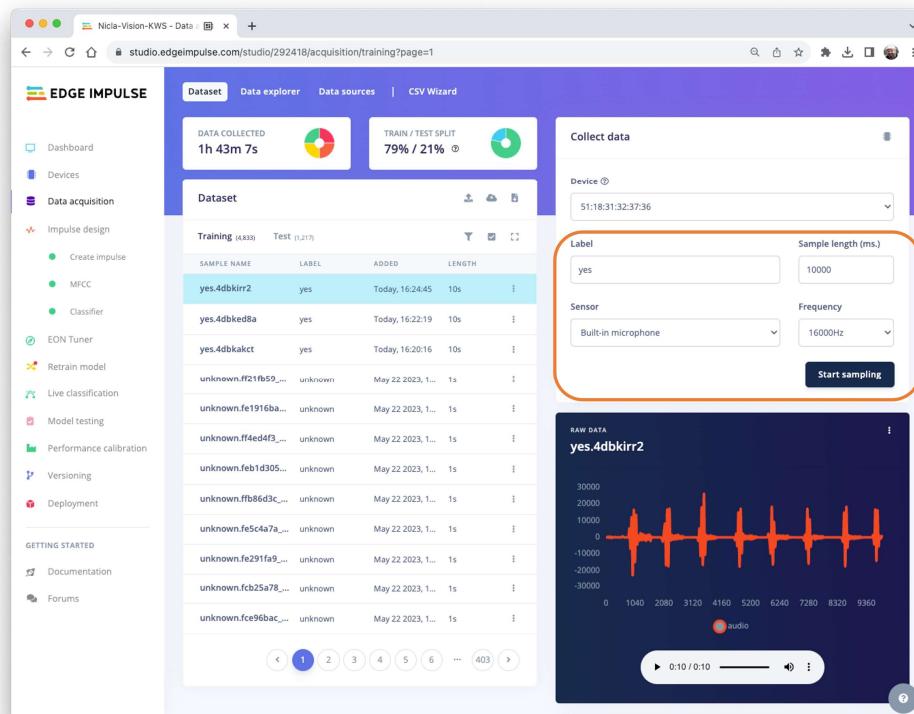
- Download del [Firmware EIS](#) più aggiornato e lo si decomprime.
- Aprire il file zip sul computer e selezionare l'uploader corrispondente al sistema operativo:



- Si mette NiclaV in Boot Mode premendo due volte il pulsante di reset.
- Si carica il binario *arduino-nicla-vision.bin* sulla board eseguendo il codice batch corrispondente al sistema operativo.

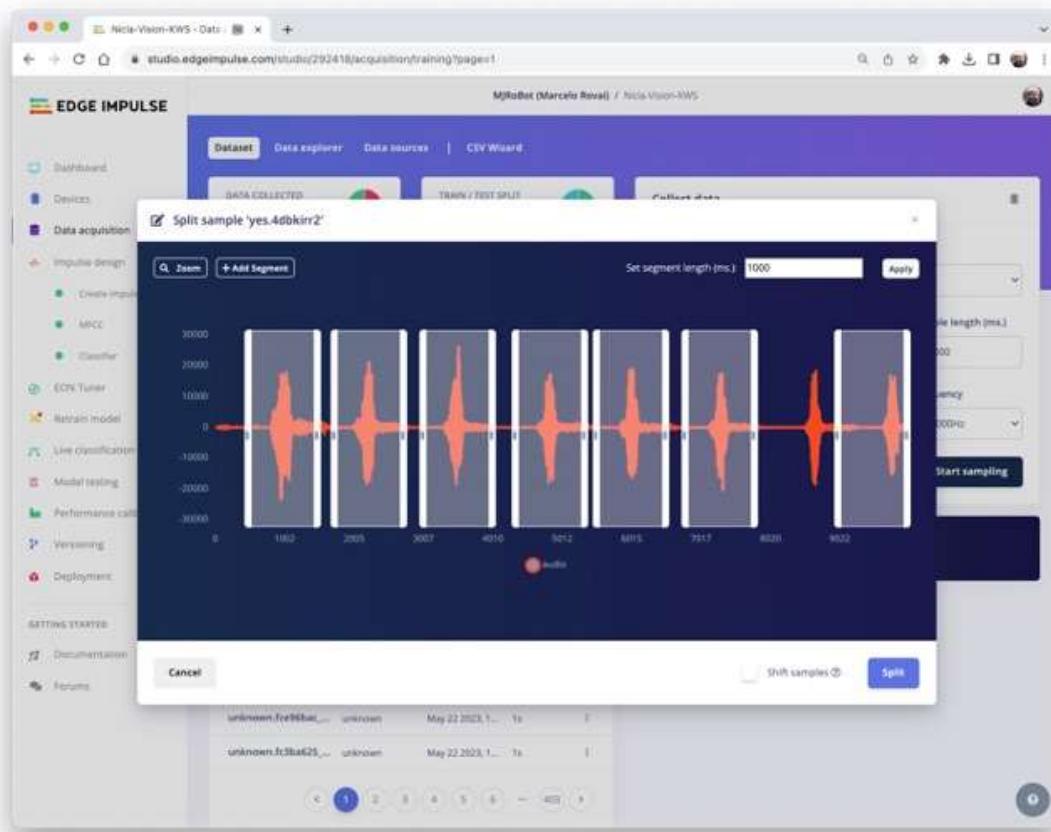
Si va sul proprio progetto EIS e nella scheda [Data Acquisition](#) tab, si seleziona [WebUSB](#). Apparirà una finestra; scegliere l'opzione che mostra che [Nicla is paired](#) e si preme [Connect].

Si possono scegliere quali dati del sensore raccogliere nella sezione [Collect Data](#) nella scheda [Data Acquisition](#). Si seleziona: [Built-in microphone](#), si definisce la [label](#) (per esempio, yes), la [Frequency\[16000Hz\]](#) e la [Sample length \(in milliseconds\)](#), per esempio [10s]. Start sampling.



I dati sul dataset di Pete hanno una lunghezza di 1s, ma i campioni registrati sono lunghi 10s e devono essere suddivisi in campioni da 1s. Cliccare sui [tre puntini](#) dopo il nome del campione e selezionare [Split sample](#).

Si aprirà una finestra col tool Split.

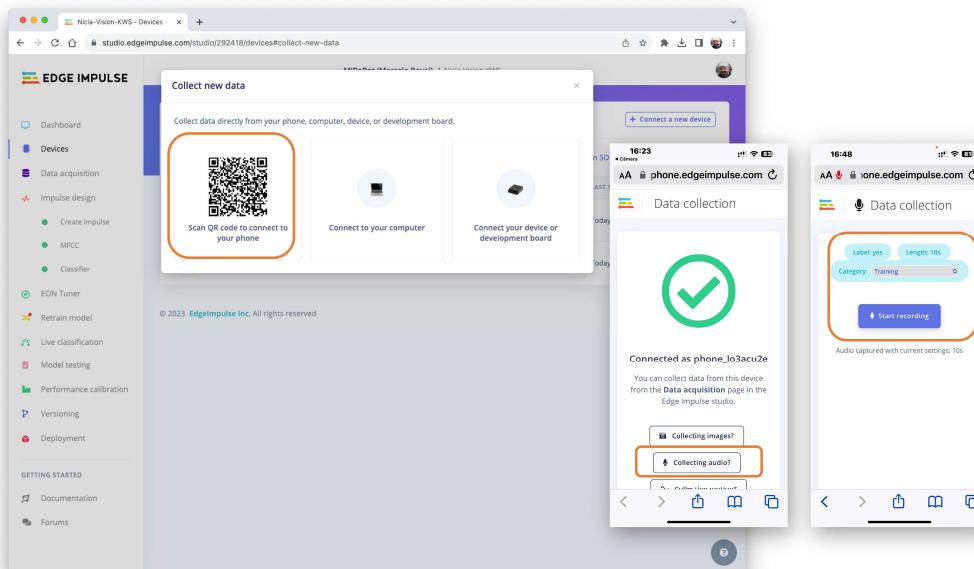


Una volta all'interno dello strumento, si dividono i dati in record da 1 secondo (1000 ms). Se necessario, si aggiungono o rimuovono segmenti. Questa procedura deve essere ripetuta per tutti i nuovi campioni.

Utilizzo di uno smartphone e di El Studio

Si può anche utilizzare il PC o lo smartphone per acquisire dati audio, utilizzando una frequenza di campionamento di 16 KHz e una profondità di 16 bit.

Si va su Devices, si scansione il QR Code col telefono e si clicca sul link. Un'app, Collection, per la raccolta dei dati apparirà nel browser. Si seleziona Collecting Audio e si definisce la Label, la Length [lunghezza] dei dati catturati e la Category.



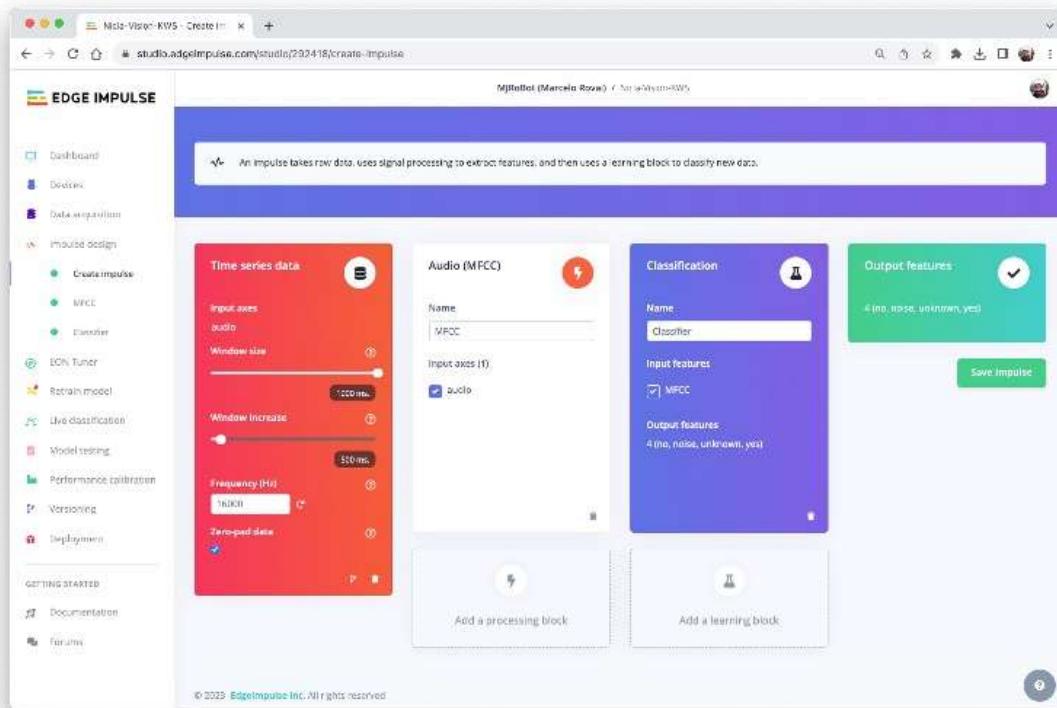
Si ripete la stessa procedura usata con NiclaV.

Notare che qualsiasi app, come [Audacity](#), può essere usata per la registrazione audio, a condizione che si usino campioni di 16KHz/16-bit.

Creazione di Impulse (Pre-Process / Definizione del Modello)

Un impulse prende dati grezzi, usa l'elaborazione del segnale per estrarre le feature e poi usa un blocco di apprendimento per classificare nuovi dati.

Impulse Design



Innanzitutto, prenderemo i dati con una finestra di 1 secondo, aumentando i dati e facendo scorrere quella finestra in intervalli di 500 ms. Notare che è impostata l'opzione Zero-pad data. È essenziale riempire i campioni con degli ‘zeri’ inferiori a 1 secondo (in alcuni casi, alcuni campioni possono risultare inferiori alla finestra di 1000 ms sul tool di suddivisione per evitare rumore e picchi).

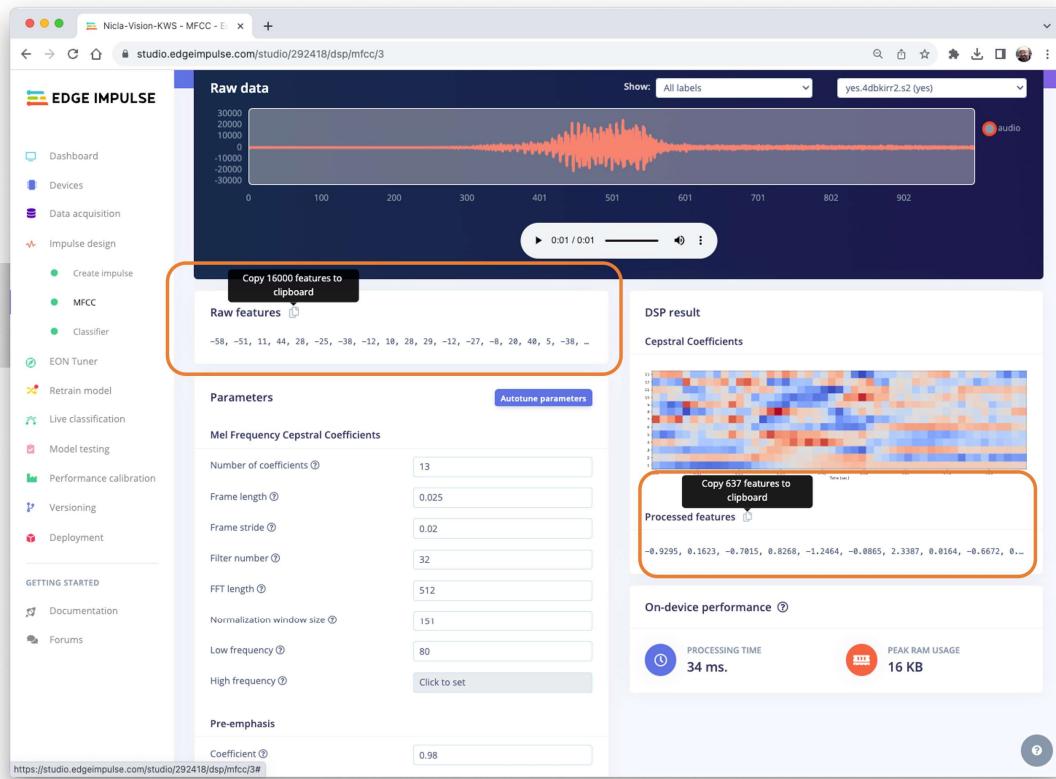
Ogni campione audio di 1 secondo dovrebbe essere pre-elaborato e convertito in un’immagine (ad esempio, $13 \times 49 \times 1$). Come discusso nel tutorial *Feature Engineering for Audio Classification*, utilizzeremo [Audio \(MFCC\)](#), che estrae le feature dai segnali audio utilizzando i [Mel Frequency Cepstral Coefficients](#), che sono adatti alla voce umana, il nostro caso qui.

Successivamente, selezioniamo il blocco [Classification](#) per costruire il nostro modello da zero utilizzando una Convolution Neural Network (CNN).

In alternativa, si può utilizzare il blocco [Transfer Learning \(Keyword Spotting\)](#), che ottimizza un modello di keyword spotting pre-addestrato sui dati. Questo approccio ha buone prestazioni con set di dati di parole chiave relativamente piccoli.

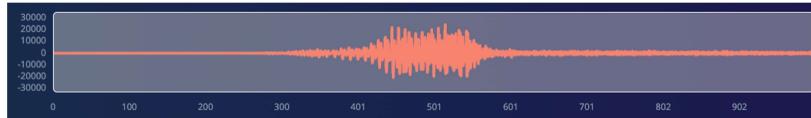
Pre-elaborazione (MFCC)

Il passaggio successivo consiste nel creare le feature da addestrare nella fase successiva: Potremmo mantenere i valori di default dei parametri, ma utilizzeremo l’opzione [DSP Autotune parameters](#).



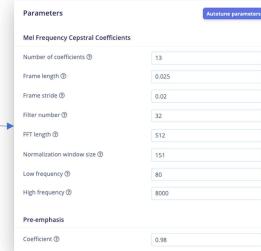
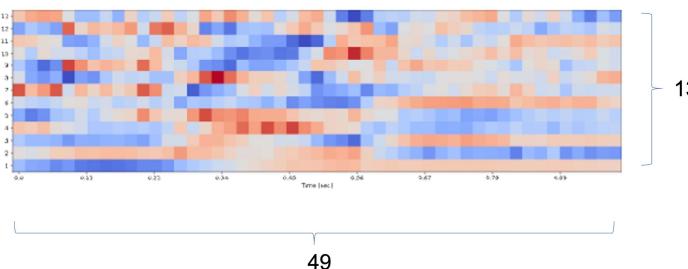
Prenderemo le **Raw features** (i dati audio campionati a 16 KHz e 1 secondo) e utilizzeremo il blocco di elaborazione **MFCC** per calcolare le **Processed features**. Per ogni 16.000 feature grezze (16.000 x 1 secondo), otterremo 637 feature elaborate (13 x 49).

Raw data → 16,000 features (1s @ 16KHz)



MFCC

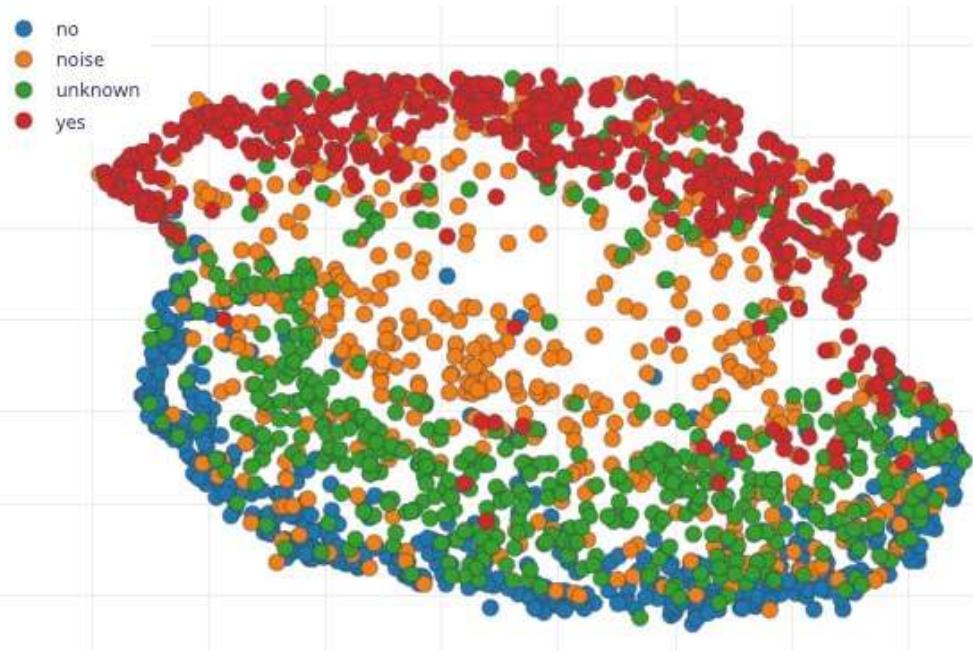
Processed features → 637 features (13 x 49)



Il risultato mostra che abbiamo utilizzato solo una piccola quantità di memoria per pre-elaborare i dati (16 KB) e una latenza di 34 ms, il che è eccellente. Ad esempio, su un Arduino Nano (Cortex-M4f @ 64 MHz), lo stesso pre-processo richiederà circa 480 ms. I parametri scelti, come la FFT length [512], avranno un impatto significativo sulla latenza.

Ora, **Save parameters** e passiamo alla scheda **Generated features**, dove verranno generate le feature effettive. Utilizzando **UMAP**, una tecnica di riduzione delle dimensioni, **Feature explorer** mostra come le feature sono distribuite su un grafico bidimensionale.

Feature explorer ⓘ



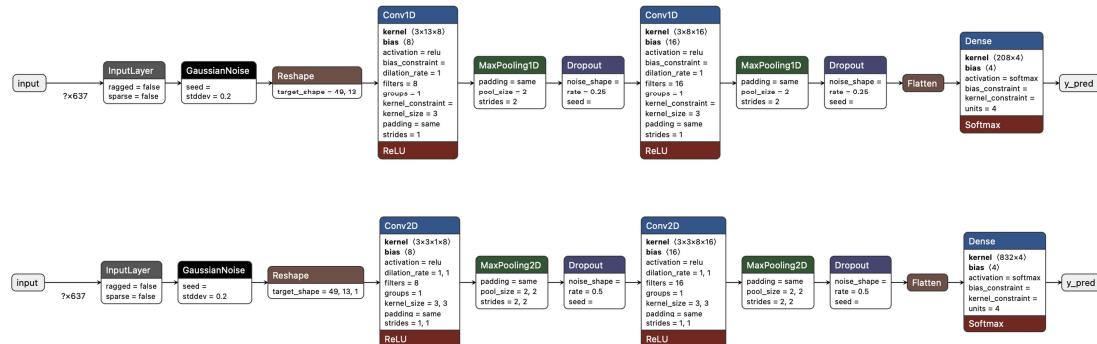
Il risultato sembra OK, con una separazione visivamente netta tra feature yes (in rosso) e no (in blu). Le feature *unknown* [sconosciute] sembrano più vicine allo spazio dei no che a quello degli yes. Questo suggerisce che la parola chiave *no* ha una maggiore propensione ai falsi positivi.

Andiamo sotto il cofano

Per comprendere meglio come viene pre-elaborato il suono grezzo, leggere il capitolo *Feature Engineering for Audio Classification*. Si può sperimentare con la generazione di feature MFCC scaricando questo [notebook](#) da GitHub o [\[Opening it In Colab\]](#)

Progettazione e Addestramento del Modello

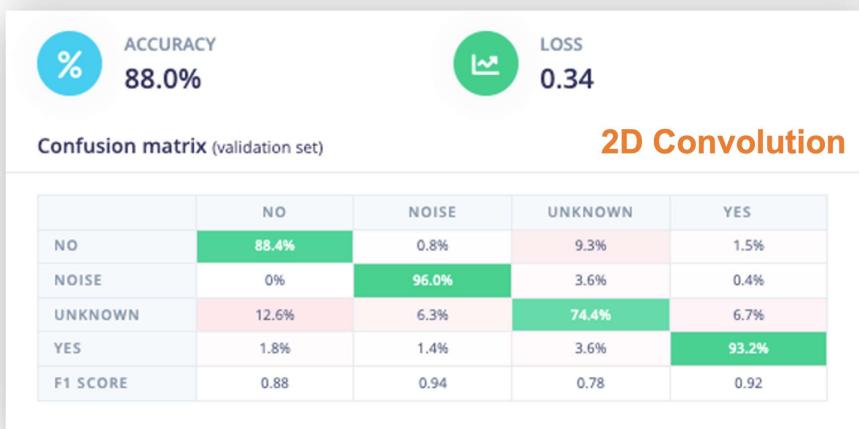
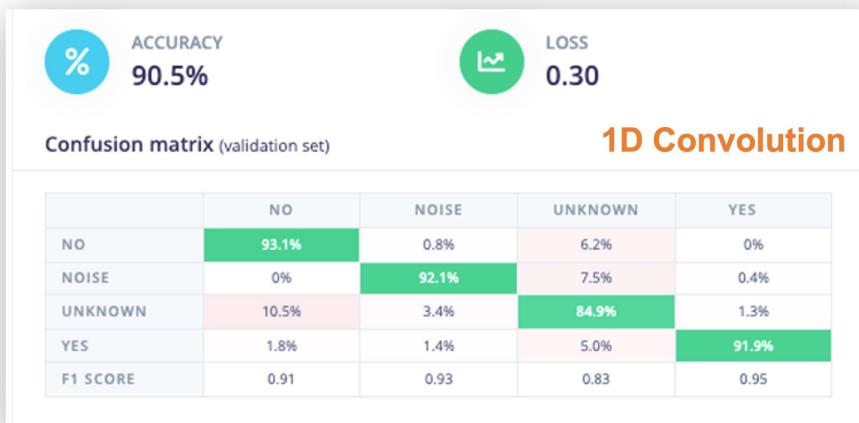
Useremo un semplice modello di rete neurale convoluzionale (CNN), testato con convoluzioni 1D e 2D. L'architettura di base ha due blocchi di Convolution + MaxPooling (filtri [8] e [16], rispettivamente) e un Dropout di [0.25] per 1D e [0.5] per 2D. Per l'ultimo layer, dopo Flattening, abbiamo [4] neuroni, uno per ogni classe:



Come iperparametri, avremo un [Learning Rate](#) di [0.005] e un modello addestrato da [100] epoche. Includeremo anche un metodo di aumento dei dati basato su [SpecAugment](#).

Abbiamo addestrato i modelli 1D e 2D con gli stessi iperparametri. L'architettura 1D ha avuto

un risultato complessivo migliore (90,5% di accuratezza rispetto all'88% del 2D, quindi useremo l'1D).



L'utilizzo di convoluzioni 1D è più efficiente perché richiede meno parametri rispetto alle convoluzioni 2D, rendendole più adatte ad ambienti con risorse limitate.

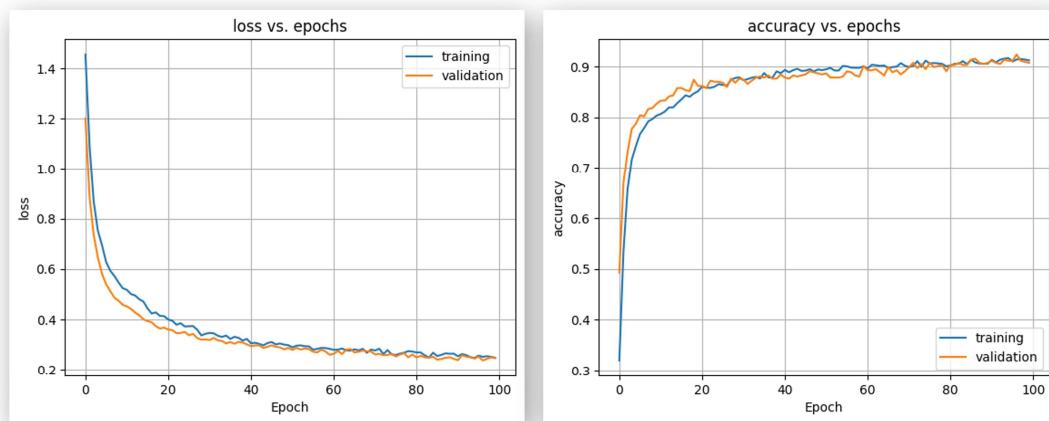
È anche interessante prestare attenzione alla Matrice di Confusione 1D. Lo Il punteggio F1 per yes è 95% e per il no, 91%. Ciò era previsto da ciò che abbiamo visto con Feature Explorer (no e unknown a distanza ravvicinata). Nel tentativo di migliorare il risultato, si possono esaminare attentamente i risultati dei campioni con un errore.



Si ascoltino i campioni che sono andati male. Ad esempio, per yes, la maggior parte degli errori erano correlati a un sì pronunciato come "yeh". Si possono acquisire campioni aggiuntivi e quindi riaddestrare il modello.

Andiamo sotto il cofano

Per capire cosa sta succedendo “sotto il cofano”, si può scaricare il set di dati pre-elaborato ((MFCC training data) dalla scheda Dashboard ed eseguire questo Jupyter Notebook, giocando con il codice o [Aprirlo in Colab]. Ad esempio, si può analizzare l’accuratezza per ogni epoca:



Test

Testando il modello con i dati riservati per il training (Test Data), abbiamo ottenuto un’accuratezza di circa il 76%.

Model testing results



ACCURACY

75.85%

	NO	NOISE	UNKNOWN	YES	UNCERTAIN
NO	57.8%	1.9%	27.8%	0.2%	12.2%
NOISE	0%	90.2%	2.3%	0.3%	7.2%
UNKNOWN	3.4%	3.7%	77.4%	0.7%	14.8%
YES	0.5%	5.0%	1.0%	82.3%	11.3%
F1 SCORE	0.72	0.89	0.70	0.90	

Ispezionando il punteggio F1, possiamo vedere che per YES abbiamo ottenuto 0.90, un risultato eccellente poiché prevediamo di utilizzare questa parola chiave come “trigger” primario per il nostro progetto KWS. Il risultato peggiore (0.70) è per UNKNOWN, il che è OK. Per NO, abbiamo ottenuto 0,72, come previsto, ma per migliorare questo risultato, possiamo spostare i campioni che non sono stati classificati correttamente nel set di dati di training e quindi ripetere il processo di training.

Classificazione Live

Possiamo procedere alla fase successiva del progetto, ma consideriamo anche che è possibile eseguire la Classificazione live utilizzando NiclaV o uno smartphone per catturare campioni dal vivo, testando il modello addestrato prima della distribuzione sul nostro dispositivo.

Distribuzione e Inferenza

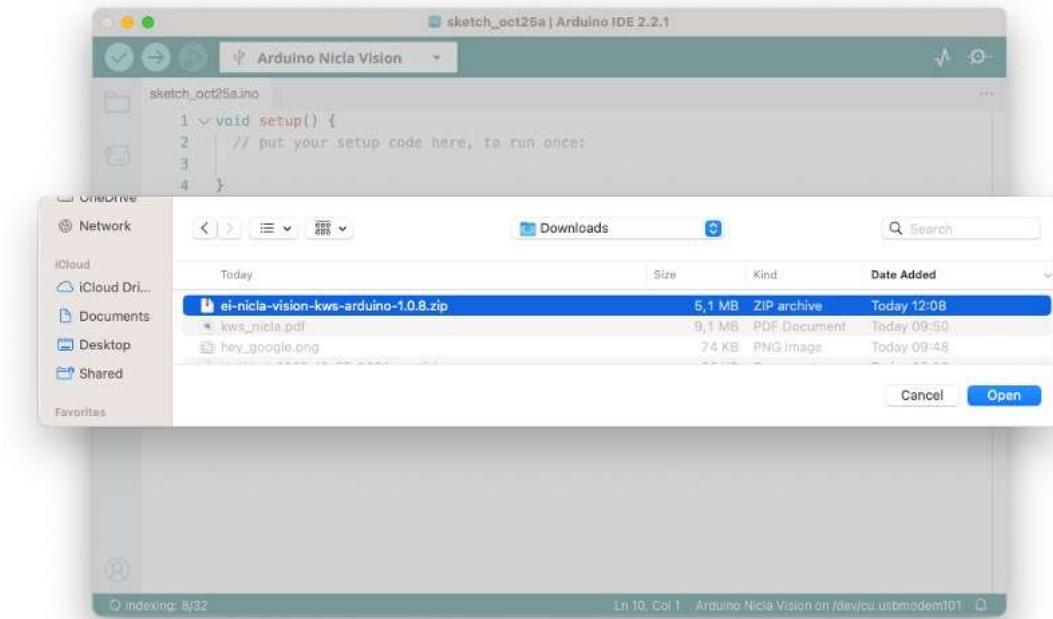
L'EIS impacchetterà tutte le librerie necessarie, le funzioni di pre-elaborazione e i modelli addestrati, scaricandoli sul computer. Si va alla sezione Deployment, si seleziona Arduino Library, e, in basso, si sceglie Quantized (Int8) e si preme Build.

The screenshot shows the 'Configure your deployment' section of the EIS interface. At the top, there's a search bar with 'Arduino library' and a clear button. Below it, under 'SELECTED DEPLOYMENT', the 'Arduino library' is selected, indicated by a blue checkmark icon. A tooltip explains it's an Arduino library for most Arm-based boards. In the 'MODEL OPTIMIZATIONS' section, the 'Quantized (int8)' option is selected, shown with a blue checkmark icon. A table provides performance metrics for this optimization level:

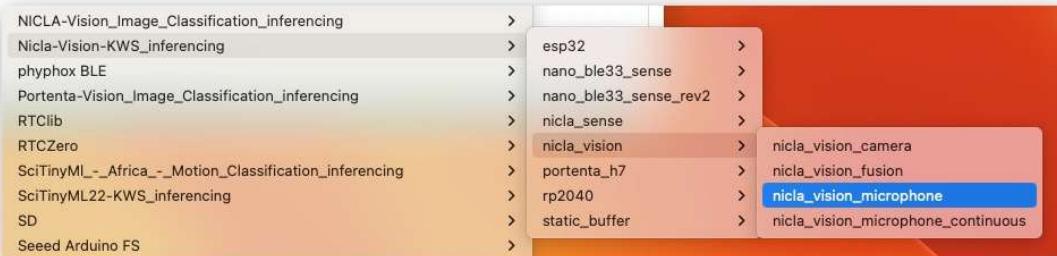
	MFCC	CLASSIFIER	TOTAL
LATENCY	24 ms	1 ms	25 ms
RAM	15.0K	3.3K	15.6K
FLASH	-	31.3K	-
ACCURACY	-	-	-

Below this, another table for 'Unoptimized (float32)' is shown, with a 'Select' button next to it. This table has similar columns and values but includes an accuracy metric: 75.85%. A note says to run model testing for all available optimizations, with a 'Run model testing' button. At the bottom, there's a 'Build' button.

Quando si seleziona il pulsante Build, verrà creato un file zip che verrà scaricato sul computer. Sull'Arduino IDE, si va alla scheda Sketch, si seleziona l'opzione Add .ZIP Library e si sceglie il file .zip scaricato da EIS:



Ora è il momento di un vero test. Faremo delle inferenze completamente disconnesse da EIS. Usiamo l'esempio di codice NiclaV creato quando abbiamo distribuito la Libreria Arduino. Nell'IDE Arduino, si va alla scheda `File/Examples`, si cerca il progetto e si seleziona `nicla-vision/nicla-vision_microphone` (o `nicla-vision_microphone_continuous`)



Premere due volte il pulsante di reset per mettere NiclaV in modalità di avvio, caricare lo sketch sulla board e provare alcune inferenze reali:



Post-elaborazione

Ora che sappiamo che il modello funziona perché rileva le nostre parole chiave, modifichiamo il codice per vedere il risultato con NiclaV completamente offline (scollegato dal PC e alimentato da una batteria, un power bank o un alimentatore indipendente da 5 V). L'idea è che ogni volta che viene rilevata la parola chiave YES, si accende il LED verde; se si sente un NO, si accende il LED rosso, se è un UNKNOWN, si accende il LED blu; e in presenza di rumore (nessuna parola chiave), i LED saranno SPENTI.

Dovremmo modificare uno degli esempi di codice. Facciamolo ora con `nicla-vision_microphone_continuous`.

Prima di tutto l'inizializzazione dei LED:

```
...
void setup()
{
    // Once you finish debugging your code, you can comment or delete the Serial part
    // of the code
    Serial.begin(115200);
    while (!Serial);
    Serial.println("Inferencing - Nicla Vision KWS with LEDs");

    // Pins for the built-in RGB LEDs on the Arduino NiclaV
    pinMode(LED_R, OUTPUT);
    pinMode(LED_G, OUTPUT);
    pinMode(LED_B, OUTPUT);

    // Ensure the LEDs are OFF by default.
    // Note: The RGB LEDs on the Arduino Nicla Vision
    // are ON when the pin is LOW, OFF when HIGH.
    digitalWrite(LED_R, HIGH);
    digitalWrite(LED_G, HIGH);
    digitalWrite(LED_B, HIGH);
}

```

Creare due funzioni, la funzione `turn_off_leds()`, per spegnere tutti i LED RGB

```
/*
 * @brief      turn_off_leds function - turn-off all RGB LEDs
 */
void turn_off_leds(){
    digitalWrite(LED_R, HIGH);
    digitalWrite(LED_G, HIGH);
    digitalWrite(LED_B, HIGH);
}
```

Un'altra funzione `turn_on_led()` viene utilizzata per accendere i LED RGB in base al risultato più probabile del classificatore.

```
/*
 * @brief      turn_on_leds function used to turn on the RGB LEDs
 * @param[in]   pred_index
 *             no:          [0] ==> Red ON
 *             noise:       [1] ==> ALL OFF
 *             unknown:     [2] ==> Blue ON
 *             Yes:         [3] ==> Green ON
 */
void turn_on_leds(int pred_index) {
    switch (pred_index)
    {
        case 0:
            turn_off_leds();
            digitalWrite(LED_R, LOW);
            break;

        case 1:
            turn_off_leds();
            break;

        case 2:
            turn_off_leds();
            digitalWrite(LED_B, LOW);
            break;

        case 3:
            turn_off_leds();
            digitalWrite(LED_G, LOW);
            break;
    }
}
```

E modificare la parte // print the predictions del codice su `loop()`:

```
...
if (++print_results >= (EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)) {
    // print the predictions
    ei_printf("Predictions ");
    ei_printf("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",

```

```
    result.timing.dsp, result.timing.classification, result.timing.anomaly);
    ei_printf(": \n");

    int pred_index = 0;      // Initialize pred_index
    float pred_value = 0;    // Initialize pred_value

    for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
        if (result.classification[ix].value > pred_value) {
            pred_index = ix;
            pred_value = result.classification[ix].value;
        }

        // ei_printf("    %s: ", result.classification[ix].label);
        // ei_printf_float(result.classification[ix].value);
        // ei_printf("\n");
    }

    ei_printf(" PREDICTION: ==> %s with probability %.2f\n",
              result.classification[pred_index].label, pred_value);
    turn_on_leds (pred_index);

#if EI_CLASSIFIER_HAS_ANOMALY == 1
    ei_printf("    anomaly score: ");
    ei_printf_float(result.anomaly);
    ei_printf("\n");
#endif

    print_results = 0;
}
}

...
```

Il codice completo si trova tra i progetti [GitHub](#).

Caricarere lo sketch sulla board e provare alcune inferenze reali. L'idea è che il LED verde sarà ACCESO ogni volta che viene rilevata la parola chiave YES, il rosso si accenderà per un NO e qualsiasi altra parola accenderà il LED blu. Tutti i LED dovrebbero essere spenti se è presente silenzio o rumore di fondo. Ricordare che la stessa procedura può "attivare" un dispositivo esterno per eseguire un'azione desiderata invece di accendere un LED, come abbiamo visto nell'introduzione.

<https://youtu.be/25Rd76OTXLY>

Conclusione

Si troveranno i notebook e i codici utilizzati in questo tutorial nel repository [GitHub](#).

Prima di concludere, considerare che la classificazione dei suoni è più di una semplice voce. Ad esempio, si possono sviluppare progetti TinyML sul suono in diverse aree, come:

- **Sicurezza** (Rilevamento di vetri rotti, spari)
- **Industria** (Rilevamento di Anomalie)
- **Medicina** (Russamento, tosse, malattie polmonari)
- **Natura** (Controllo degli alveari, suono degli insetti, riduzione dei sacchetti di raccolta)

Risorse

- Sottoinsieme del Dataset dei Comandi Vocali di Google
- KWS MFCC Analysis Colab Notebook
- KWS_CNN_training Colab Notebook
- Codice di Post-elaborazione Arduino
- Progetto Edge Impulse

Classificazione del Movimento e Rilevamento delle Anomalie



DALL-E 3 Prompt: Illustrazione in stile fumetto anni '50 raffigurante una sala di ricerca sul movimento. Al centro della stanza c'è un container simulato utilizzato per il trasporto di merci su camion, barche e carrelli elevatori. Il container è dettagliato con rivetti e marcature tipiche dei cassoni di carico industriali. Attorno al container, la stanza è piena di attrezzature d'epoca, tra cui un oscilloscopio, vari array di sensori e grandi rotoli di carta di dati registrati. Le pareti sono adornate con poster educativi sulla sicurezza e la logistica dei trasporti. L'atmosfera generale della stanza è nostalgica e scientifica, con un pizzico di stile industriale.

Panoramica

I trasporti sono la spina dorsale del commercio globale. Milioni di container vengono trasportati ogni giorno tramite vari mezzi, come navi, camion e treni, verso destinazioni in tutto il mondo. Garantire il transito sicuro ed efficiente di questi container è un compito monumentale che richiede di sfruttare la tecnologia moderna e TinyML è senza dubbio una di queste.

In questo tutorial, lavoreremo per risolvere problemi reali relativi al trasporto. Svilupperemo un sistema di Motion Classification e Anomaly Detection utilizzando la scheda Arduino Nicla Vision, l'IDE Arduino e Edge Impulse Studio. Questo progetto ci aiuterà a comprendere come i container subiscono forze e movimenti diversi durante le varie fasi del trasporto, come il transito terrestre e marittimo, il movimento verticale tramite carrelli elevatori e i periodi di stazionamento nei magazzini.

💡 Obiettivi dell'Apprendimento

- Impostazione della scheda Arduino Nicla Vision
- Raccolta e pre-elaborazione dei dati
- Creazione del modello di classificazione del movimento
- Implementazione del rilevamento delle anomalie
- Test e Analisi nel Mondo Reale

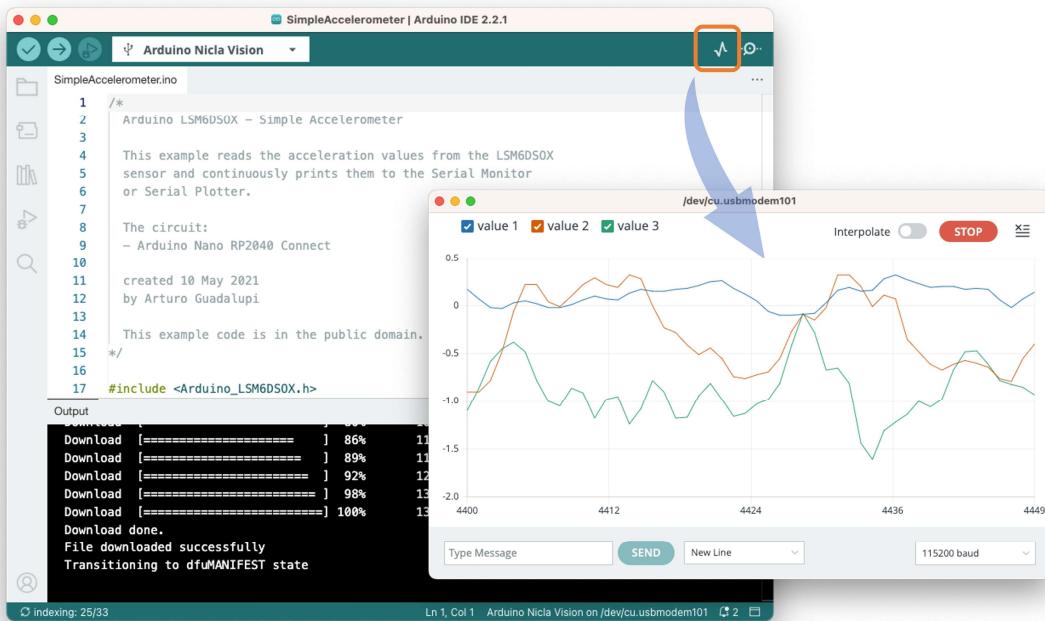
Alla fine di questo tutorial, si avrà un prototipo funzionante in grado di classificare diversi tipi di movimento e rilevare anomalie durante il trasporto di container. Questa conoscenza può essere un trampolino di lancio per progetti più avanzati nel campo emergente di TinyML che coinvolge le vibrazioni.

Installazione e test dell'IMU

Per questo progetto, useremo un accelerometro. Come spiegato nel Tutorial, *Configurazione di Nicla Vision*, la Nicla Vision Board ha un **IMU a 6 assi** integrato: giroscopio 3D e accelerometro 3D, il **LSM6DSOX**. Verifichiamo se la **libreria IMU LSM6DSOX** è installata. In caso contrario, la si installa.



Poi si va su `Examples > Arduino_LSM6DSOX > SimpleAccelerometer` e si esegue il test dell'accelerometro. Si può verificare se funziona aprendo l'IDE Serial Monitor o Plotter. I valori sono in g (gravità terrestre), con un range di default di +/- 4g:



Definizione della Frequenza di Campionamento:

La scelta di una frequenza di campionamento appropriata è fondamentale per catturare le caratteristiche del movimento che interessa studiare. Il teorema di campionamento di Nyquist-Shannon afferma che la frequenza di campionamento dovrebbe essere almeno il doppio della componente di frequenza più alta nel segnale per ricostruirlo correttamente. Nel contesto della classificazione del movimento e del rilevamento delle anomalie per il trasporto, la scelta della frequenza di campionamento dipenderebbe da diversi fattori:

- Natura del movimento:** Diversi tipi di trasporto (terrestre, marittimo, ecc.) possono comportare diversi intervalli di frequenze di movimento. Movimenti più rapidi potrebbero richiedere frequenze di campionamento più elevate.
- Limitazioni hardware:** La scheda Arduino Nicla Vision e tutti i sensori associati potrebbero avere limitazioni sulla velocità con cui possono campionare i dati.
- Risorse di Calcolo:** Frequenze di campionamento più elevate genereranno più dati, il che potrebbe essere computazionalmente intenso, particolarmente critico in un ambiente TinyML.
- Durata della Batteria:** Una frequenza di campionamento più elevata consumerà più energia. Se il sistema funziona a batteria, questa è una considerazione importante.
- Archiviazione Dati:** Un campionamento più frequente richiederà più spazio di archiviazione, un'altra considerazione cruciale per i sistemi embedded con memoria limitata.

In molte attività di riconoscimento delle attività umane, vengono comunemente utilizzate **frequenze di campionamento da 50 Hz a 100 Hz circa**. Dato che stiamo simulando scenari di trasporto, che in genere non sono eventi ad alta frequenza, una frequenza di campionamento in quell'intervallo (50-100 Hz) potrebbe essere un punto di partenza ragionevole.

Definiamo uno sketch che ci consentirà di acquisire i nostri dati con una frequenza di campionamento definita (ad esempio, 50 Hz):

```
/*
 * Based on Edge Impulse Data Forwarder Example (Arduino)
 * - https://docs.edgeimpulse.com/docs/cli-data-forwarder
 * Developed by M.Rovai @11May23
 */
```

```
/* Include ----- */
#include <Arduino_LSM6DSOX.h>

/* Constant defines ----- */
#define CONVERT_G_TO_MS2 9.80665f
#define FREQUENCY_HZ      50
#define INTERVAL_MS        (1000 / (FREQUENCY_HZ + 1))

static unsigned long last_interval_ms = 0;
float x, y, z;

void setup() {
    Serial.begin(9600);
    while (!Serial);

    if (!IMU.begin()) {
        Serial.println("Failed to initialize IMU!");
        while (1);
    }
}

void loop() {
    if (millis() > last_interval_ms + INTERVAL_MS) {
        last_interval_ms = millis();

        if (IMU.accelerationAvailable()) {
            // Read raw acceleration measurements from the device
            IMU.readAcceleration(x, y, z);

            // converting to m/s2
            float ax_m_s2 = x * CONVERT_G_TO_MS2;
            float ay_m_s2 = y * CONVERT_G_TO_MS2;
            float az_m_s2 = z * CONVERT_G_TO_MS2;

            Serial.print(ax_m_s2);
            Serial.print("\t");
            Serial.print(ay_m_s2);
            Serial.print("\t");
            Serial.println(az_m_s2);
        }
    }
}
```

Caricando lo sketch e ispezionando il Serial Monitor, possiamo vedere che stiamo catturando 50 campioni al secondo.

Output Serial Monitor X

Message (Enter to send message to 'Arduino Nicla Vision')

17:58:59.986 -> 0.62	-0.08	9.85
17:59:00.021 -> 0.63	-0.08	9.85
17:59:00.054 -> 0.62	-0.08	9.85
17:59:00.054 -> 0.62	-0.08	9.86
17:59:00.087 -> 0.62	-0.08	9.85
17:59:00.087 -> 0.63	-0.07	9.86
17:59:00.120 -> 0.63	-0.08	9.86
17:59:00.120 -> 0.62	-0.08	9.85
17:59:00.153 -> 0.62	-0.08	9.86
.	.	.
.	.	.
17:59:00.888 -> 0.63	-0.08	9.85
17:59:00.920 -> 0.64	-0.08	9.86
17:59:00.920 -> 0.62	-0.08	9.85
17:59:00.952 -> 0.62	-0.08	9.86
17:59:00.986 -> 0.63	-0.07	9.85
17:59:00.986 -> 0.63	-0.08	9.86
17:59:01.017 -> 0.62	-0.07	9.85

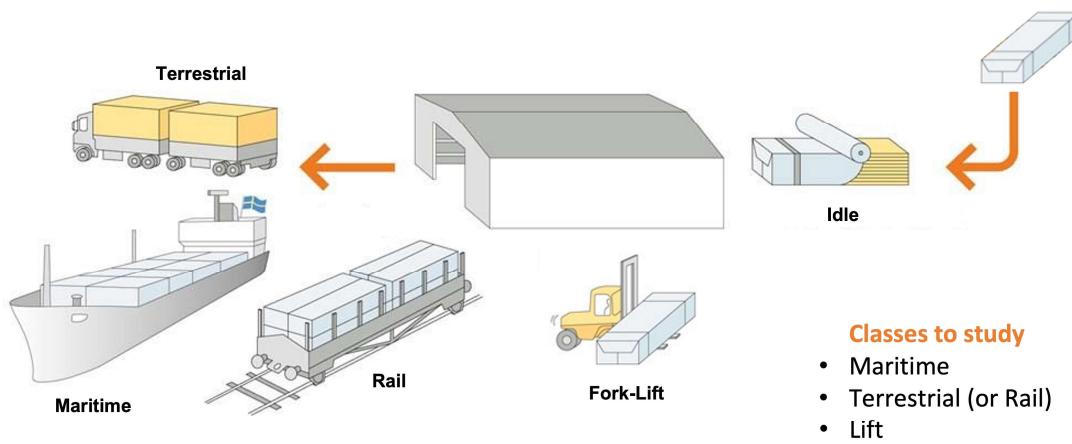
50 samples / second

Si noti che con la scheda Nicla appoggiata su un tavolo (con la telecamera rivolta verso il basso), l'asse z misura circa 9.8m/s^2 , l'accelerazione terrestre prevista.

Il Caso di Studio: Trasporto Simulato di Container

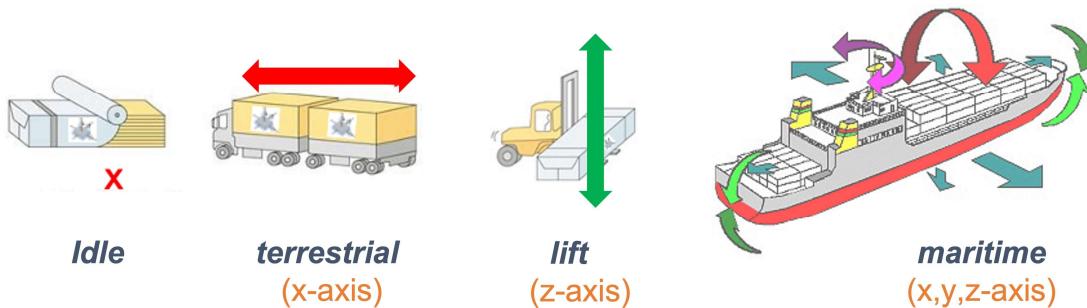
Simuleremo il trasporto di container (o meglio di pacchi) attraverso diversi scenari per rendere questo tutorial più comprensibile e pratico. Utilizzando l'accelerometro integrato della scheda Arduino Nicla Vision, cattureremo i dati di movimento simulando manualmente le condizioni di:

1. Trasporto **Terrestre** (su strada o treno)
2. Trasporto **Marittimo**
3. Movimento Verticale tramite Carrello **Elevatore**
4. Periodo di **stazionamento** (inattivo) in un Magazzino

**Classes to study**

- Maritime
- Terrestrial (or Rail)
- Lift
- Idle

Dalle immagini sopra, possiamo definire per la nostra simulazione che i movimenti principalmente orizzontali (asse x o y) dovrebbero essere associati alla “classe Terrestre”, i movimenti verticali (asse z) alla “classe di Sollevamento”, nessuna attività alla “classe di Inattività” e il movimento su tutti e tre gli assi alla **classe Marittima**.

**Raccolta Dati**

Per la raccolta dati, possiamo avere diverse opzioni. In un caso reale, possiamo avere il nostro dispositivo, ad esempio, collegato direttamente a un contenitore e i dati raccolti su un file (ad esempio .CSV) e archiviati su una scheda SD (tramite connessione SPI) o un repository offline nel computer. I dati possono anche essere inviati in remoto a un repository nelle vicinanze, come un telefono cellulare, tramite Bluetooth (come fatto in questo progetto: [Sensor DataLogger](#)). Una volta che il dataset è stato raccolto e archiviato come file .CSV, può essere caricato su Studio utilizzando lo strumento [CSV Wizard](#).

In questo [video](#), si possono imparare modi alternativi per inviare dati a Edge Impulse Studio.

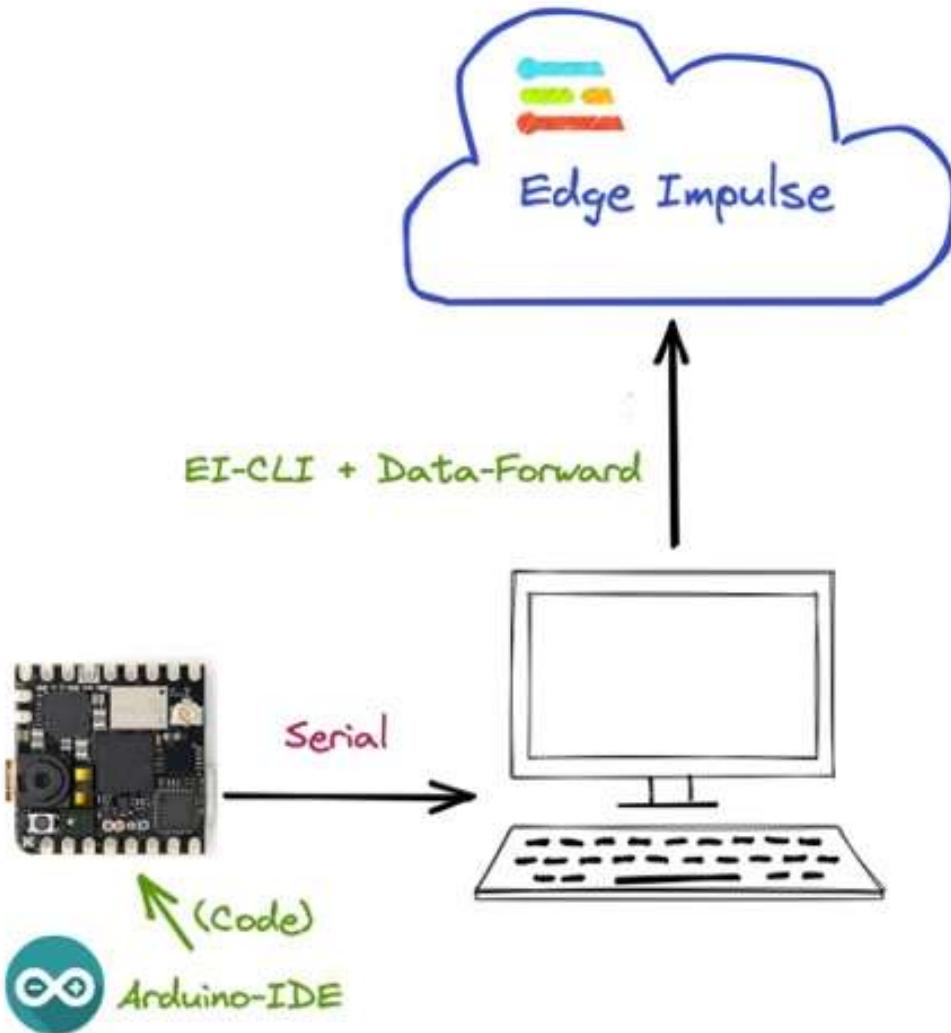
Collegamento del dispositivo a Edge Impulse

Collegheremo Nicla direttamente a Edge Impulse Studio, che verrà utilizzato anche per la pre-elaborazione dei dati, l’addestramento del modello, i test e la distribuzione. Per questo, ci sono due possibilità:

1. Scaricare il firmware più recente e collegarlo direttamente alla sezione [Raccolta Dati](#).
2. Utilizzare il tool [CLI Data Forwarder](#) per acquisire i dati dal sensore e inviarli a Studio.

L’opzione 1 è più semplice, come abbiamo visto nell’esercitazione *Configurazione di Nicla Vision*, ma l’opzione 2 darà maggiore flessibilità per quanto riguarda l’acquisizione dei dati, come la definizione della frequenza di campionamento. Facciamolo con l’ultima. Creare un nuovo progetto su Edge Impulse Studio (EIS) e collegarvi Nicla, seguendo questi passaggi:

1. Installare Edge Impulse CLI e Node.js sul computer.
2. Caricare uno sketch per l'acquisizione dati (quello discusso in precedenza in questo tutorial).
3. Utilizzare [CLI Data Forwarder](#) per acquisire dati dall'accelerometro di Nicla e inviarli a Studio, come mostrato in questo diagramma:



Avviare [CLI Data Forwarder](#) sul terminale, immettendo (se è la prima volta) il seguente comando:

```
$ edge-impulse-data-forwarder --clean
```

Quindi, immettere le proprie credenziali EI e scegliere il progetto, le variabili (ad esempio, `accX`, `accY` e `accZ`) e il nome del dispositivo (ad esempio, `NiclaV`):

```

marcelo_rovai — node ~/.npm-global/bin/edge-impulse-data-forwarder --clean — 88x16
Last login: Tue Oct 31 13:16:03 on ttys000
(base) marcelo_rovai@Marcelos-MacBook-Pro ~ % edge-impulse-data-forwarder --clean
Edge Impulse data forwarder v1.21.1
? What is your user name or e-mail address (edgeimpulse.com)? rovai@mjrobot.org
? What is your password? [hidden]

Endpoints:
  WebSocket: wss://remote-mgmt.edgeimpulse.com
  API: https://studio.edgeimpulse.com
  Ingestion: https://ingestion.edgeimpulse.com

[SER] Connecting to /dev/tty.usbmodem101
[SER] Serial is connected (00:2C:00:27:30:31:51:0C:39:31:35:32)
[WS ] Connecting to wss://remote-mgmt.edgeimpulse.com
[WS ] Connected to wss://remote-mgmt.edgeimpulse.com

? To which project do you want to connect this device? MJRoBot (Marcelo Rovai) / NICLA
Vision Mov
ement Classification
[SER] Detecting data frequency...
[SER] Detected data frequency: 50Hz
? 3 sensor axes detected (example values: [-1.26,-0.37,-9.79]). What do you want to cal
l them? Separate the names wit
h ',' : accX, accY, accZ
? What name do you want to give this device? NiclaV
[WS ] Device "NiclaV" is now connected to project "NICLA Vision Movement Classificatio
n". To connect to another project, run `edge-impulse-data-forwarder --clean`.
[WS ] Go to https://studio.edgeimpulse.com/studio/302078/acquisition/training to build
your machine learning model!

```

Andare alla sezione `Devices` sul progetto EI e verificare se il dispositivo è connesso (il punto dovrebbe essere verde):

NAME	ID	TYPE	SENSORS	REM...	LAST SEEN
NiclaV	00:2C:00:27:30:31:51:0C...	DATA_FORWARDER...	Sensor with 3 axes (...)	.	Today, 14:43:30

Si può clonare il progetto sviluppato per questa esercitazione pratica: [NICLA Vision Movement Classification](#).

Raccolta Dati

Nella sezione `Data Acquisition`, si dovrebbe vedere che la scheda `[NiclaV]` è connessa. Il sensore è disponibile: `[sensor with 3 axes (accX, accY, accZ)]` con una frequenza di campionamento di `[50Hz]`. Studio suggerisce una lunghezza di campionamento di `[10000]` ms (10s). L'ultima cosa rimasta è definire l'etichetta del campione. Cominciamo con `[terrestrial]`:

Collect data

Device ②

NiclaV

Label terrestrial

Sample length (ms.) 10000

Sensor Sensor with 3 axes (accX, accY, accZ)

Frequency 50Hz

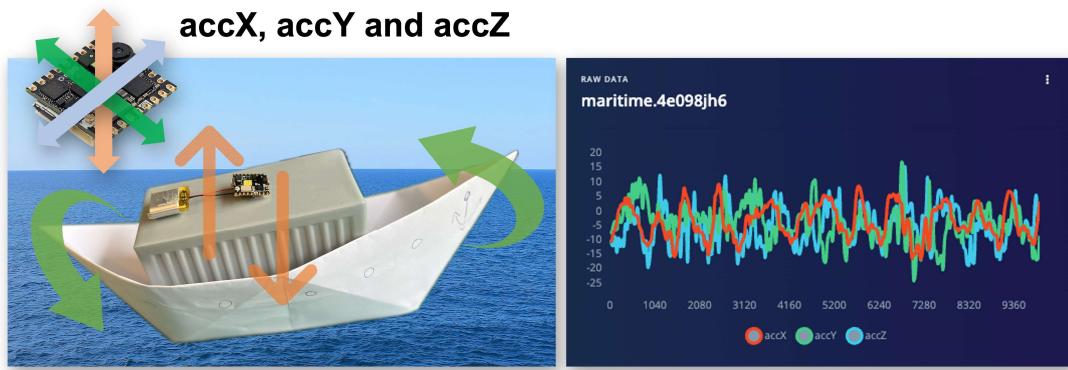
Start sampling

Terrestrial (pallet in un Camion o Treno), muovendosi orizzontalmente. Premere [Start Sample] e spostare il device orizzontalmente, mantenendo una direzione sopra il tavolo. Dopo 10s, i dati saranno caricati nello Studio. Ecco come è stato raccolto il campione:

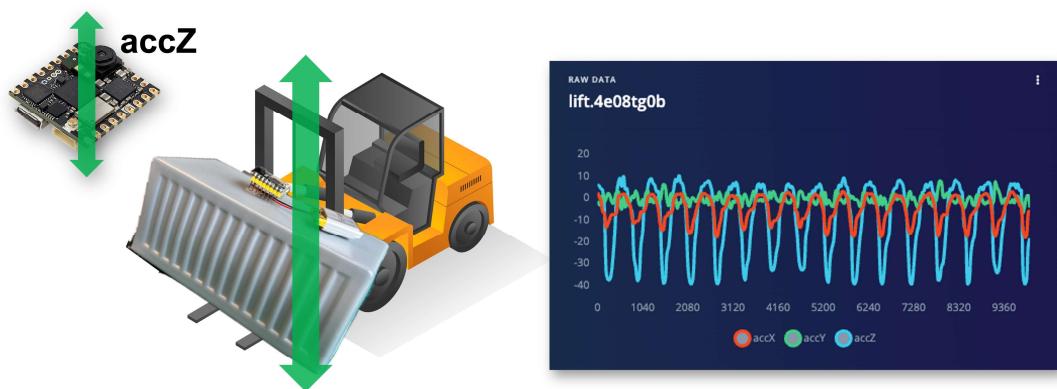


Come previsto, il movimento è stato catturato principalmente nell'asse Y (verde). Nel blu, vediamo l'asse Z, circa -10 m/s^2 (Nicla ha la telecamera rivolta verso l'alto). Come discusso in precedenza, dovremmo catturare dati da tutte e quattro le classi "Transportation". Quindi, si immagini di avere un container con un accelerometro integrato nelle seguenti situazioni:

Maritime (pallet in barche in un oceano in tempesta). Il movimento viene catturato su tutti e tre gli assi:



Lift (Pallet movimentati verticalmente da un Carrello elevatore). Movimento catturato solo sull'asse Z:

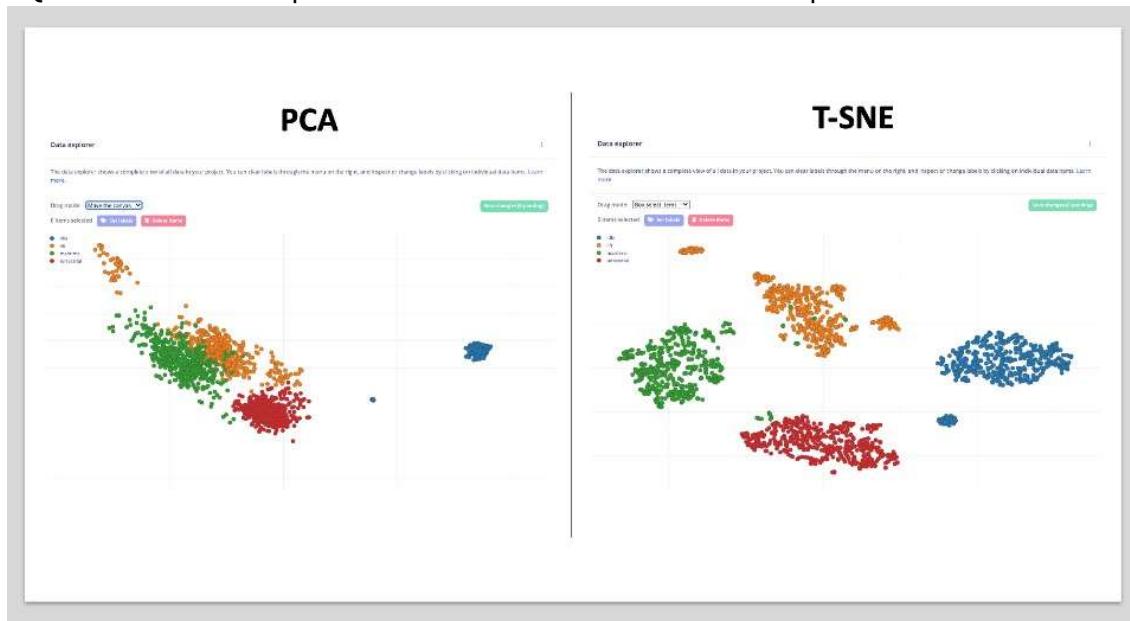


Idle (Pallet in un magazzino). Nessun movimento rilevato dall'accelerometro:



Ad esempio, si possono catturare 2 minuti (dodici campioni di 10 secondi) per ciascuna delle quattro classi (per un totale di 8 minuti di dati). Utilizzando il menu **tre puntini** dopo ciascuno dei campioni, selezionarne 2, riservandoli per il set di Test. In alternativa, si può utilizzare lo strumento automatico **Train/Test Split tool** nella scheda **Danger Zone** della scheda **Dashboard**. Di seguito il set di dati risultante:

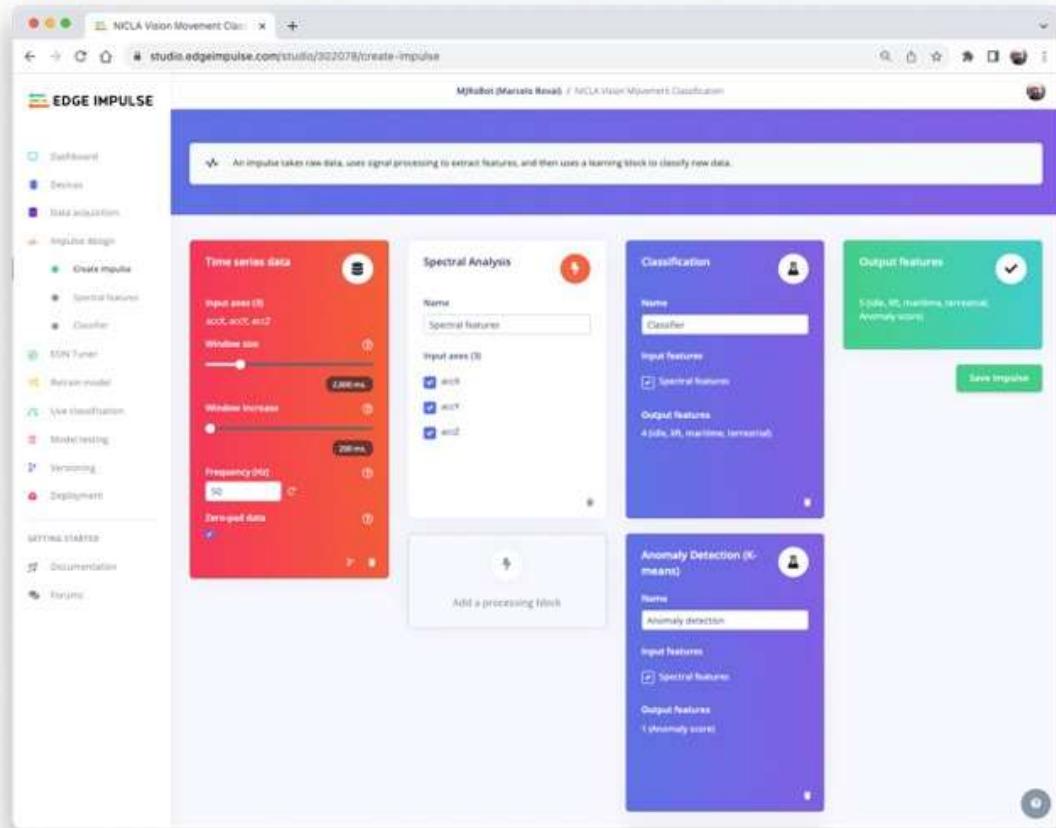
Una volta acquisito il dataset, lo si può esplorare più in dettaglio utilizzando **Data Explorer**, uno strumento visivo per trovare valori anomali o dati etichettati in modo errato (aiutando a correggerli). Data Explorer tenta prima di estrarre feature significative dai dati (applicando l'elaborazione del segnale e gli “embedding” della rete neurale) e poi utilizza un algoritmo di riduzione della dimensionalità come **PCA** o **t-SNE** per mappare queste feature in uno spazio 2D. Questo fornisce una panoramica immediata del dataset completo.



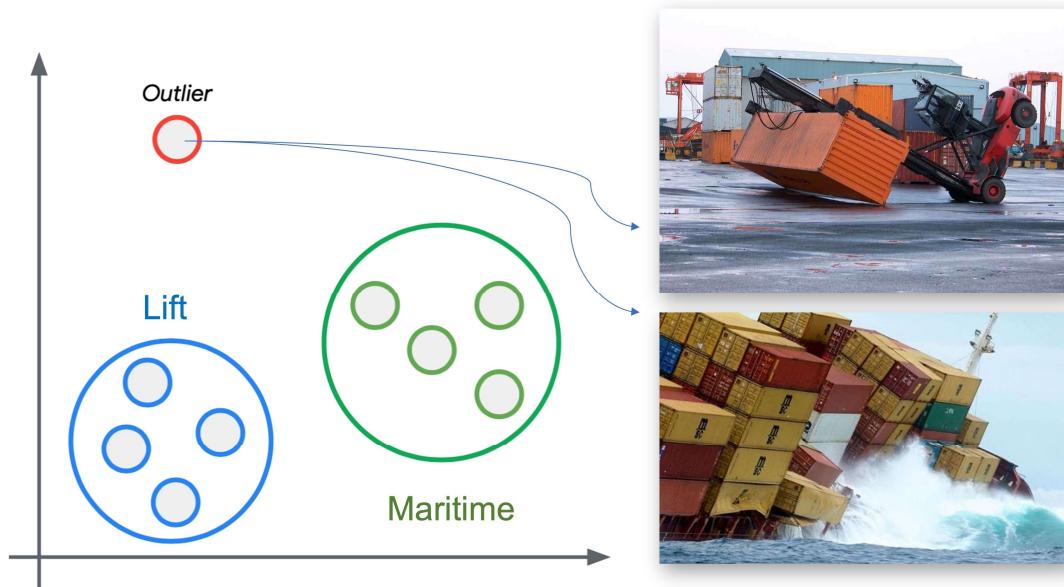
Nel nostro caso, il set di dati sembra OK (buona separazione). Ma il PCA mostra che possiamo avere problemi tra marittimo (verde) e sollevamento (arancione). Ciò è prevedibile, una volta su una barca, a volte il movimento può essere solo “verticale”.

Impulse Design

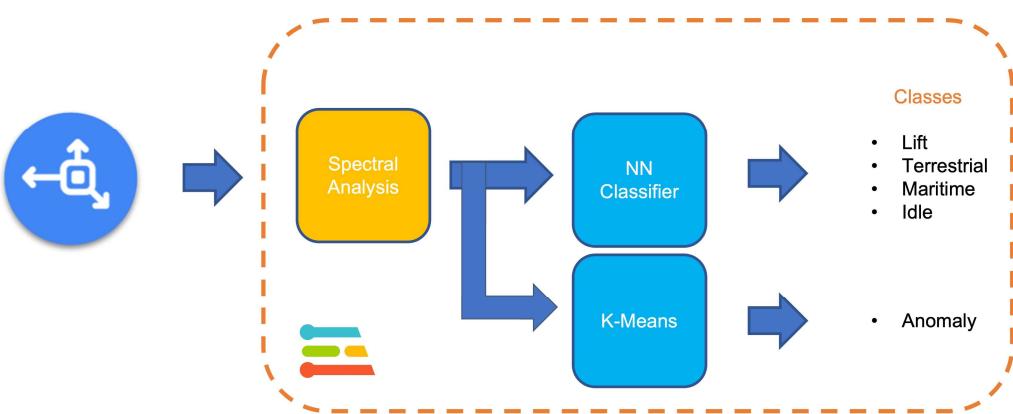
Il passo successivo è la definizione del nostro Impulse, che prende i dati grezzi e usa l'elaborazione del segnale per estrarre le feature, passandole come tensore di input di un *learning block* [blocco di apprendimento] per classificare nuovi dati. Si va in `Impulse Design` e `Create Impulse`. Studio suggerirà la progettazione di base. Aggiungiamo anche un secondo *Learning Block* per `Anomaly Detection`.



Questo secondo modello usa un modello K-means. Se immaginiamo di poter avere le nostre classi note come cluster, qualsiasi campione che si adatterebbe potrebbe essere un “outlier”, un’anomalia come un container che rotola fuori da una nave in mare o cade da un carrello elevatore.



La frequenza di campionamento dovrebbe essere catturata automaticamente, in caso contrario, inserirla: [50] Hz. Studio suggerisce una *Window Size* di 2 secondi ([2000] ms) con una *sliding window* di [20] ms. Ciò che stiamo definendo in questa fase è che pre-elaboreremo i dati catturati (dati di serie temporali), creando un dataset tabellare (caratteristiche) che saranno l'input per un classificatore di reti neurali (DNN) e un modello di rilevamento delle anomalie (K-Means), come mostrato di seguito:



Analizziamo attentamente quei passaggi e i parametri per capire meglio cosa stiamo facendo qui.

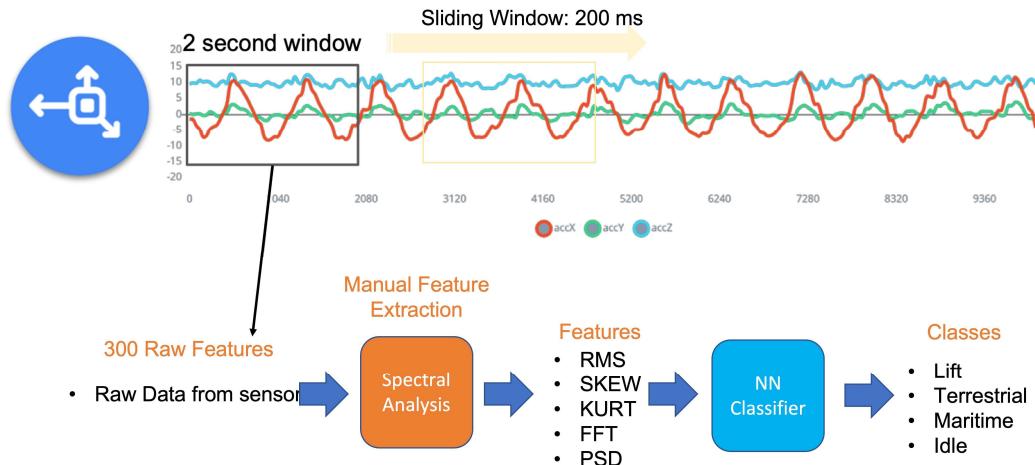
Panoramica sulla Pre-Elaborazione dei Dati

La pre-elaborazione dei dati consiste nell'estrapolare le feature dal set di dati acquisito con l'accelerometro, il che implica l'elaborazione e l'analisi dei dati grezzi. Gli accelerometri misurano l'accelerazione di un oggetto lungo uno o più assi (in genere tre, indicati come X, Y e Z). Queste misure possono essere utilizzate per comprendere vari aspetti del movimento dell'oggetto, come pattern di movimento e vibrazioni.

I dati grezzi dell'accelerometro possono essere rumorosi e contenere errori o informazioni irrilevanti. Le fasi di pre-elaborazione, come il filtraggio e la normalizzazione, possono pulire e standardizzare i dati, rendendoli più adatti all'estrazione di feature. Nel nostro caso, dovremmo dividere i dati in segmenti più piccoli o **windows** [finestre]. Ciò può aiutare a concentrarsi su eventi o attività specifici all'interno del dataset, rendendo l'estrazione di feature più gestibile e significativa. La scelta della **window size** e della sovrapposizione

(**window increase**) dipende dall'applicazione e dalla frequenza degli eventi di interesse. Come regola generale, dovremmo provare a catturare un paio di “cicli di dati”.

Con un “sampling rate” (SR) [frequenza di campionamento] di 50 Hz e una dimensione della finestra di 2 secondi, otterremo 100 campioni per asse, o 300 in totale (3 assi x 2 secondi x 50 campioni). Faremo scorrere questa finestra ogni 200 ms, creando un set di dati più grande in cui ogni istanza ha 300 feature grezze.



Una volta che i dati sono stati pre-elaborati e segmentati, si possono estrarre feature che descrivono le caratteristiche del movimento. Alcune feature tipiche estratte dai dati dell'accelerometro includono:

- Le feature del **Time-domain** descrivono le proprietà statistiche dei dati all'interno di ciascun segmento, come media, mediana, deviazione standard, asimmetria, curtosi e tasso di attraversamento dello zero.
- Le feature **Frequency-domain** si ottengono trasformando i dati nel dominio della frequenza utilizzando tecniche come la Fast Fourier Transform (FFT). Alcune feature tipiche del dominio della frequenza includono lo spettro di potenza, l'energia spettrale, le frequenze dominanti (ampiezza e frequenza) e l'entropia spettrale.
- Le feature del dominio **Time-frequency** combinano le informazioni del dominio del tempo e della frequenza, come la Short-Time Fourier Transform (STFT) o la Discrete Wavelet Transform (DWT). Possono fornire una comprensione più dettagliata di come il contenuto di frequenza del segnale cambia nel tempo.

In molti casi, il numero di feature estratte può essere elevato, il che può portare a un overfitting o a una maggiore complessità computazionale. Le tecniche di selezione delle feature, come le informazioni reciproche, i metodi basati sulla correlazione o l'analisi delle componenti principali (PCA), possono aiutare a identificare le feature più rilevanti per una determinata applicazione e ridurre la dimensionalità del dataset. Studio può aiutare con tali calcoli di importanza delle feature.

Feature Spettrali di El Studio

La pre-elaborazione dei dati è un'area impegnativa per il machine learning embedded; tuttavia, Edge Impulse aiuta a superarla con la sua fase di pre-elaborazione del segnale digitale (DSP) e, più specificamente, con lo [Spectral Features Block](#).

In Studio, il dataset grezzo raccolto sarà l'input di un blocco di Spectral Analysis, che è eccellente per analizzare il movimento ripetitivo, come i dati degli accelerometri. Questo blocco eseguirà un DSP (Digital Signal Processing), estraendo feature come la [FFT](#) o le [Wavelet](#).

Per il nostro progetto, una volta che il segnale temporale è continuo, dovremmo usare FFT con, ad esempio, una lunghezza di [32].

Le feature **Time Domain Statistical** per asse/canale sono:

- **RMS**: 1 feature
- **Skewness** [Asimmetria]: 1 feature
- **Curtosi**: 1 feature

Le feature **Frequency Domain Spectral features** per asse/canale sono:

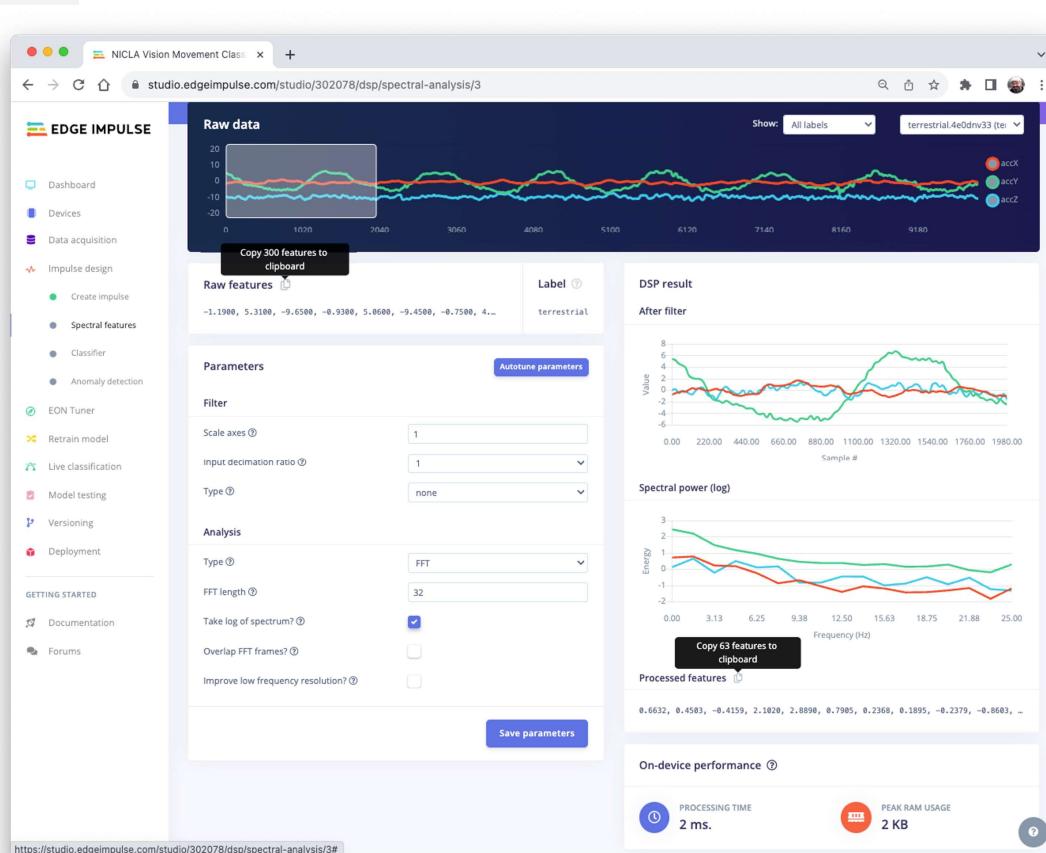
- **Potenza Spettrale**: 16 feature (lunghezza FFT/2)
- Asimmetria: 1 feature
- Curtosi: 1 feature

Quindi, per una lunghezza FFT di 32 punti, l'output risultante dello "Spectral Analysis Block" sarà di 21 feature per asse (un totale di 63 feature).

Si può scoprire di più su come viene calcolata ogni feature scaricando il notebook [Edge Impulse - Spectral Features Block Analysis TinyML under the hood: Spectral Analysis](#) o aprendolo direttamente su Google CoLab.

Generazione di feature

Una volta capito cosa fa la pre-elaborazione, è il momento di finire il lavoro. Quindi, prendiamo i dati grezzi (tipo serie temporale) e convertiamoli in dati tabellari. Per farlo, si va alla sezione **Spectral Features** nella scheda **Parameters**, si definiscono i parametri principali come discusso nella sezione precedente ([FFT] con [32] punti), e si seleziona **[Save Parameters]**:



Nel menù in alto, si seleziona l'opzione **Generate Features** e il pulsante **Generate Features**. Ogni dato della finestra di 2 secondi verrà convertito in un punto dati di 63 feature.

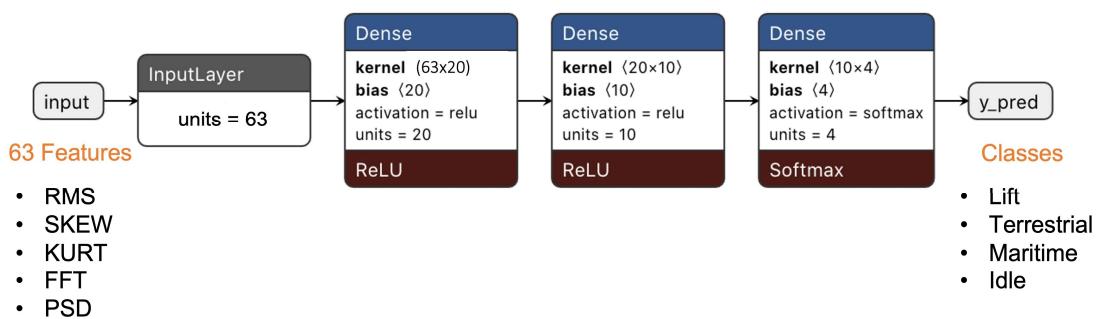
Feature Explorer mostrerà tali dati in 2D utilizzando **UMAP**. Uniform Manifold Approximation and Projection (UMAP) è una tecnica di riduzione delle dimensioni che può essere utilizzata per la visualizzazione in modo simile a t-SNE, ma è applicabile anche per la riduzione generale delle dimensioni non lineari.

La visualizzazione consente di verificare che dopo la generazione delle feature, le classi presenti mantengano la loro eccellente separazione, il che indica che il classificatore dovrebbe funzionare bene. Facoltativamente, si può analizzare quanto è importante ciascuna delle feature per una classe rispetto alle altre.

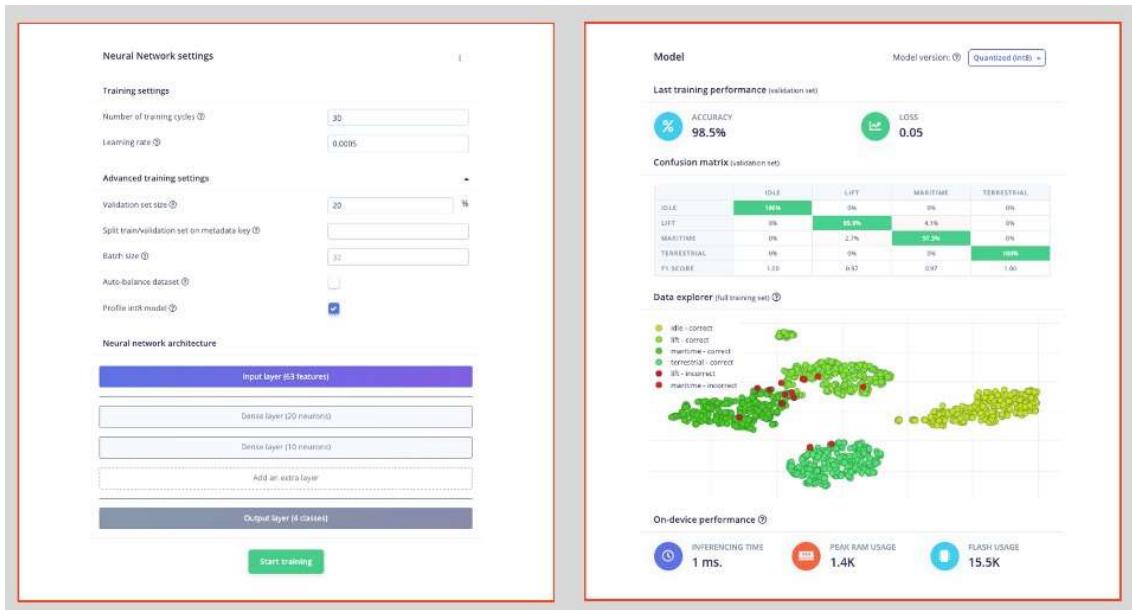


Adddestramento dei Modelli

Il nostro classificatore sarà una Dense Neural Network (DNN) che avrà 63 neuroni sul suo layer di input, due layer nascosti con 20 e 10 neuroni e un layer di output con quattro neuroni (uno per ogni classe), come mostrato qui:



Come iperparametri, useremo un Learning Rate di [0.005], una dimensione Batch di [32] e [20] % di dati per la convalida per [30] epoche. Dopo l'addestramento, possiamo vedere che l'accuratezza è del 98.5%. Il costo della memoria e della latenza è esiguo.



Per la “Anomaly Detection”, sceglieremo le feature suggerite che sono esattamente le più importanti nell'estrazione delle feature, più l'RMS accZ. Il numero di cluster sarà [32], come suggerito da Studio:



Test

Possiamo verificare come si comporterà il nostro modello con dati sconosciuti utilizzando il 20% dei dati lasciati da parte durante la fase di acquisizione dei dati. Il risultato è stato quasi del 95%, il che è positivo. Si può sempre lavorare per migliorare i risultati, ad esempio, per capire cosa è andato storto con uno dei risultati sbagliati. Se si tratta di una situazione unica, la si può aggiungere al set di dati di training e quindi ripeterla.

La soglia minima di default per un risultato considerato incerto è [0.6] per la classificazione e [0.3] per l'anomalia. Una volta che abbiamo quattro classi (la loro somma di output dovrebbe essere 1.0), si può anche impostare una soglia inferiore per una classe da considerare valida (ad esempio, 0.4). Si possono impostare le soglie di confidenza nel menu a tre puntini, oltre al pulsante Classify all.

This screenshot shows the Edge Impulse Studio interface for model validation. The left sidebar includes options like Dashboard, Devices, Data acquisition, Impulse design, EON Tuner, Retrain model, Live classification, Model testing (which is selected), Versioning, Deployment, and Getting Started. The main area has tabs for Test data, Model testing output, and Model testing results.

Test data:

SAMPLE NAME	EXPECTED	LENGTH	ANOMALY	ACCURA...	RESULT
maritime.4e0d9...	maritime	10s	-0.26	98%	40 maritime, 1...
maritime.4e0d4...	maritime	10s	-0.20	100%	41 maritime
terrestrial.4e0d1...	terrestrial	10s	-0.63	100%	41 terrestrial
terrestrial.4e0df...	terrestrial	10s	0.09	61%	25 terrestrial, ...
idle.4e0e30ec.s3	idle	10s	-0.26	100%	41 idle
idle.4e0e30ec.s9	idle	10s	-0.23	100%	41 idle
lift.4e0dsus8	lift	10s	-0.41	100%	41 lift
lift.4e0drssu	lift	10s	-0.46	100%	41 lift

Model testing output:

ACCURACY: 94.82%

	IDLE	LIFT	MARITIME	TERRESTRIAL	ANOMALY	UNCERTAIN
IDLE	100%	0%	0%	0%	0%	0%
LIFT	0%	100%	0%	0%	0%	0%
MARITIME	0%	1.2%	98.8%	0%	0%	0%
TERRESTRIAL	0%	0%	7.3%	92.5%	7.3%	4.9%
ANOMALY	-	-	-	-	-	-
F1 SCORE	1.00	0.99	0.96	0.89	0.00	

Model testing results:

Feature explorer

Legend: idle - correct, lift - correct, maritime - correct, terrestrial - correct, maritime - incorrect, terrestrial - incorrect

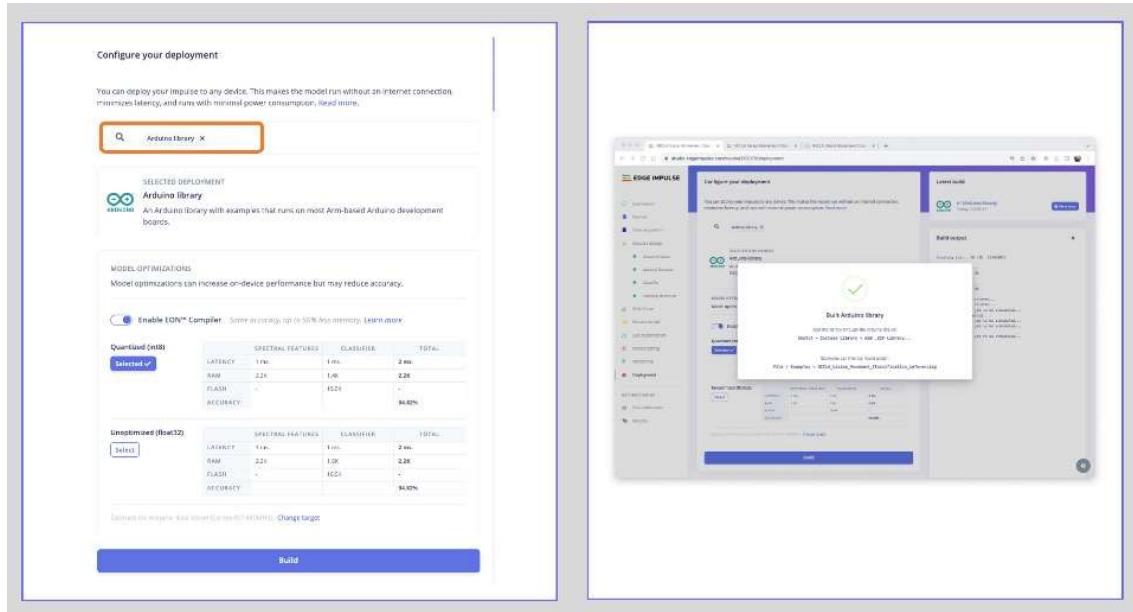
The feature explorer displays clusters of green and red points representing correctly and incorrectly classified samples for categories like IDLE, LIFT, MARITIME, TERRESTRIAL, and ANOMALY.

Puoi anche eseguire “Live Classification” col dispositivo (che dovrebbe essere ancora connesso a Studio).

Da tenere presente che qui, si cattureranno dati reali col dispositivo e si caricheranno in Studio, dove verrà presa un’inferenza utilizzando il modello addestrato (ma il **modello NON è nel device**).

Distribuzione

È il momento di distribuire il blocco di pre-elaborazione e il modello addestrato al Nicla. Studio impacchetterà tutte le librerie necessarie, le funzioni di pre-elaborazione e i modelli addestrati, scaricandoli sul computer. Si deve selezionare l’opzione `Arduino Library`, e in basso si può scegliere `Quantized (Int8)` o `Unoptimized (float32)` e `[Build]`. Verrà creato un file Zip e scaricato sul computer.



Su Arduino IDE, si va alla scheda Sketch, si seleziona Add.ZIP Library e si sceglie il file .zip scaricato da Studio. Un messaggio apparirà nel terminale IDE: Library installed.

Inferenza

Ora è il momento di un vero test. Faremo inferenze completamente scollegate da Studio. Modifichiamo uno degli esempi di codice creati quando si distribuisce la libreria Arduino. Nell'IDE Arduino, si va alla scheda File/Examples e si cerca il progetto, e negli esempi, si seleziona Nicla_vision_fusion:



Notare che il codice creato da Edge Impulse considera un approccio *sensor fusion* in cui vengono utilizzati IMU (accelerometro e giroscopio) e ToF. All'inizio del codice, ci sono le librerie relative al nostro progetto, IMU e ToF:

```
/* Includes ----- */
#include <NICLA_Vision_Movement_Classification_inferencing.h>
#include <Arduino_LSM6DSOX.h> //IMU
#include "VL53L1X.h" // ToF
```

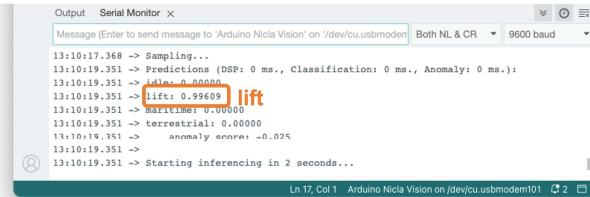
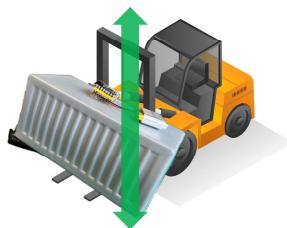
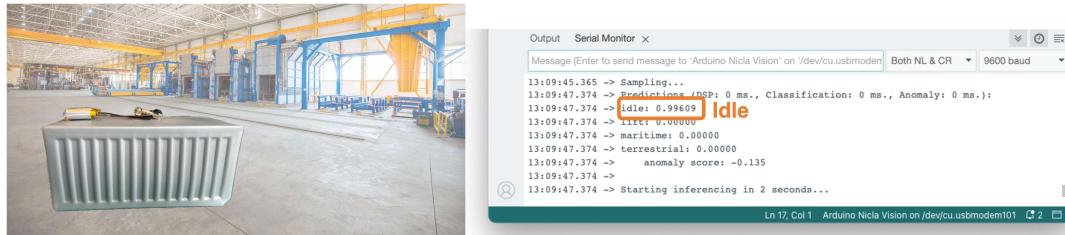
Si può mantenere il codice in questo modo per i test perché il modello addestrato utilizzerà solo le funzionalità pre-elaborate dall'accelerometro. Ma si consideri che si scriverà il codice solo con le librerie necessarie per un progetto reale.

E questo è tutto!

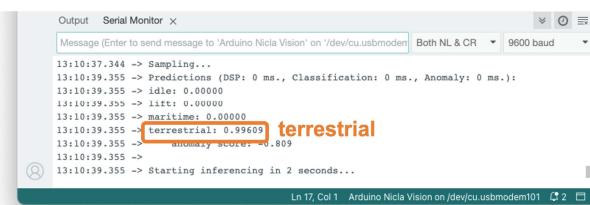
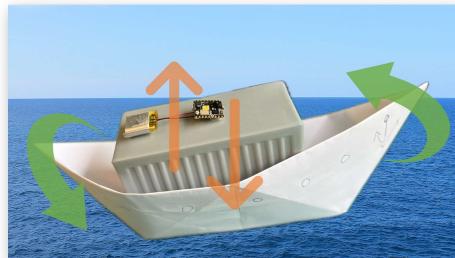
Ora si può caricare il codice sul dispositivo e procedere con le inferenze. Si preme due volte il pulsante [RESET] di Nicla per metterlo in modalità boot (disconnetterlo da Studio se è ancora connesso) e caricare lo sketch sulla board.

Ora si devono provare diversi movimenti con la scheda (simili a quelli eseguiti durante l'acquisizione dei dati), osservando il risultato dell'inferenza di ciascuna classe sul Serial Monitor:

- **Classi Idle e lift:**

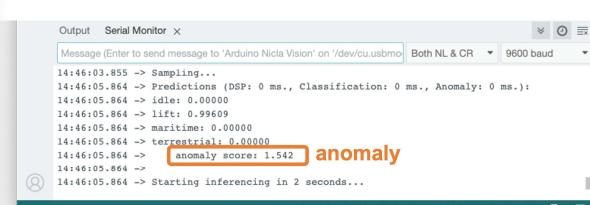


- Maritime e terrestriale:**



Notare che in tutte le situazioni sopra, il valore di `anomaly score` era inferiore a 0.0. Provare un nuovo movimento che non faceva parte del dataset originale, ad esempio, "facendo rotolare" la Nicla, rivolta verso la telecamera capovolta, come un contenitore che cade da una barca o persino un incidente in barca:

- Rilevamento delle Anomalie**



In questo caso, l'anomalia è molto più grande, oltre 1.00

Post-elaborazione

Ora che sappiamo che il modello funziona poiché rileva i movimenti, suggeriamo di modificare il codice per vedere il risultato con il NiclaV completamente offline (scollegato dal PC e alimentato da una batteria, un power bank o un alimentatore indipendente da 5V).

L'idea è di fare lo stesso del progetto KWS: se viene rilevato un movimento specifico, potrebbe accendersi un LED specifico. Ad esempio, se viene rilevato un movimento *terrestre*, si accenderà il LED verde; se è un movimento *marittimo*, si accenderà il LED rosso, se è un *ascensore*, si accenderà il LED blu; e se non viene rilevato alcun movimento (*inattivo*), i LED saranno SPENTI. Si può anche aggiungere una condizione quando viene rilevata un'anomalia, in questo caso, ad esempio, può essere utilizzato un colore bianco (tutti i LED e si accendono contemporaneamente).

Conclusione

I notebook e i codici utilizzati in questo tutorial pratico si trovano nel repository [GitHub](#).

Prima di concludere, considerare che la classificazione del movimento e il rilevamento degli oggetti possono essere utilizzati in molte applicazioni in vari domini. Ecco alcune delle potenziali applicazioni:

Applicazioni di Casi

Industriale e Manifatturiero

- **Manutenzione Predittiva:** Rilevamento di anomalie nel movimento dei macchinari per prevedere guasti prima che si verifichino.
- **Controllo Qualità:** Monitoraggio del movimento di linee di assemblaggio o bracci robotici per la valutazione della precisione e il rilevamento delle deviazioni dal pattern di movimenti standard.
- **Logistica di Magazzino:** Gestione e tracciamento del movimento delle merci con sistemi automatizzati che classificano diversi tipi di movimento e rilevano anomalie nella movimentazione.

Assistenza Sanitaria

- **Monitoraggio Pazienti:** Rilevamento di cadute o movimenti anomali negli anziani o in coloro che hanno problemi di mobilità.
- **Riabilitazione:** Monitoraggio dei progressi dei pazienti in fase di recupero da infortuni tramite classificazione dei pattern di movimento durante le sedute di fisioterapia.
- **Riconoscimento Attività:** Classificazione dei tipi di attività fisica per applicazioni di fitness o monitoraggio dei pazienti.

Elettronica di Consumo

- **Gesture Control:** Interpretazione di movimenti specifici per controllare i dispositivi, come accendere le luci con un gesto della mano.
- **Gaming:** Miglioramento delle esperienze di gioco con input controllati dal movimento.

Trasporti e Logistica

- **Telematica dei Veicoli:** Monitoraggio del movimento del veicolo per comportamenti insoliti come frenate brusche, curve strette o incidenti.
- **Monitoraggio del Carico:** Garantire l'integrità delle merci durante il trasporto rilevando movimenti insoliti che potrebbero indicare manomissioni o cattiva gestione.

Città Intelligenti e Infrastrutture

- **Monitoraggio Strutturale:** Rilevare vibrazioni o movimenti all'interno delle strutture che potrebbero indicare potenziali guasti o necessità di manutenzione.

- **Gestione del Traffico:** Analizzare il flusso di pedoni o veicoli per migliorare la mobilità e la sicurezza urbana.

Sicurezza e Sorveglianza

- **Rilevamento Intrusi:** Rilevare pattern di movimento tipici di accessi non autorizzati o altre violazioni della sicurezza.
- **Monitoraggio della Fauna Selvatica:** Rilevare bracconieri o movimenti anomali di animali in aree protette.

Agricoltura

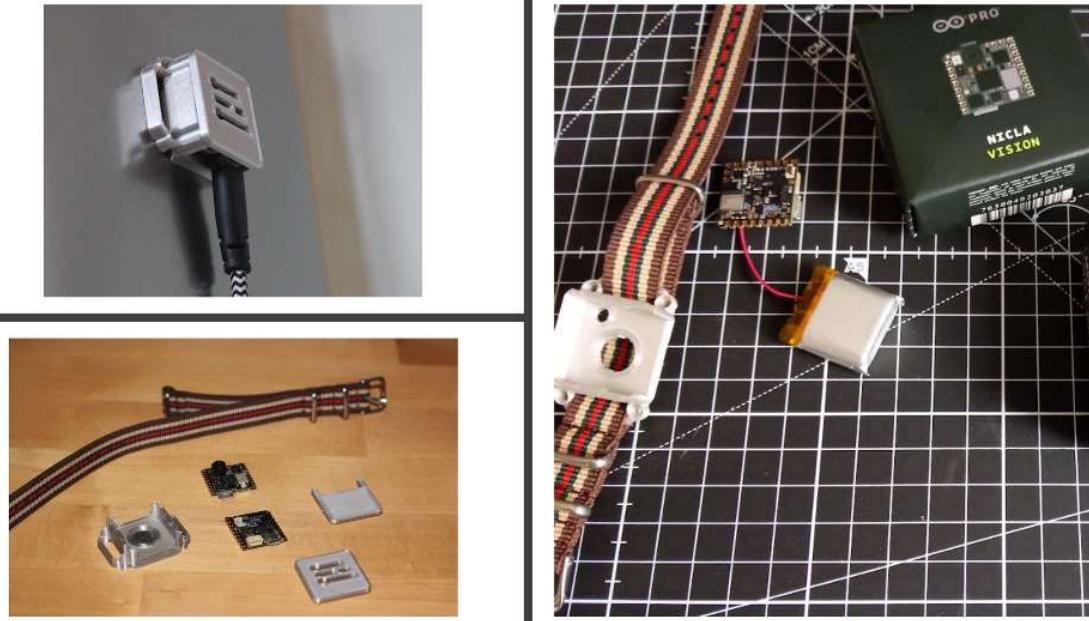
- **Monitoraggio delle Attrezzature:** Monitoraggio delle prestazioni e dell'utilizzo di macchinari agricoli.
- **Analisi del Comportamento Animale:** Monitoraggio dei movimenti del bestiame per rilevare comportamenti che indicano problemi di salute o stress.

Monitoraggio Ambientale

- **Attività Sismica:** Rilevamento di pattern di movimento irregolari che precedono terremoti o altri eventi geologicamente rilevanti.
- **Oceanografia:** Studio di pattern di onde o movimenti marini per scopi di ricerca e sicurezza.

Custodia per Nicla 3D

Per applicazioni reali, come alcuni hanno descritto in precedenza, possiamo aggiungere una custodia al nostro dispositivo ed Eoin Jordan, di Edge Impulse, ha sviluppato un'ottima custodia indossabile e per la salute delle macchine per la gamma di schede Nicla. Funziona con un magnete da 10 mm, viti da 2M e una cinghia da 16mm per scenari di utilizzo per la salute umana e delle macchine. Ecco il link: [Arduino Nicla Voice e Vision Wearable Case](#).



Le applicazioni per la classificazione del movimento e il rilevamento delle anomalie sono estese e Arduino Nicla Vision è adatto per scenari in cui il basso consumo energetico e l'edge processing sono vantaggiosi. Il suo piccolo fattore di forma e l'efficienza nell'elaborazione lo rendono una scelta ideale per l'implementazione di applicazioni portatili e remote in cui l'elaborazione in tempo reale è fondamentale e la connettività potrebbe essere limitata.

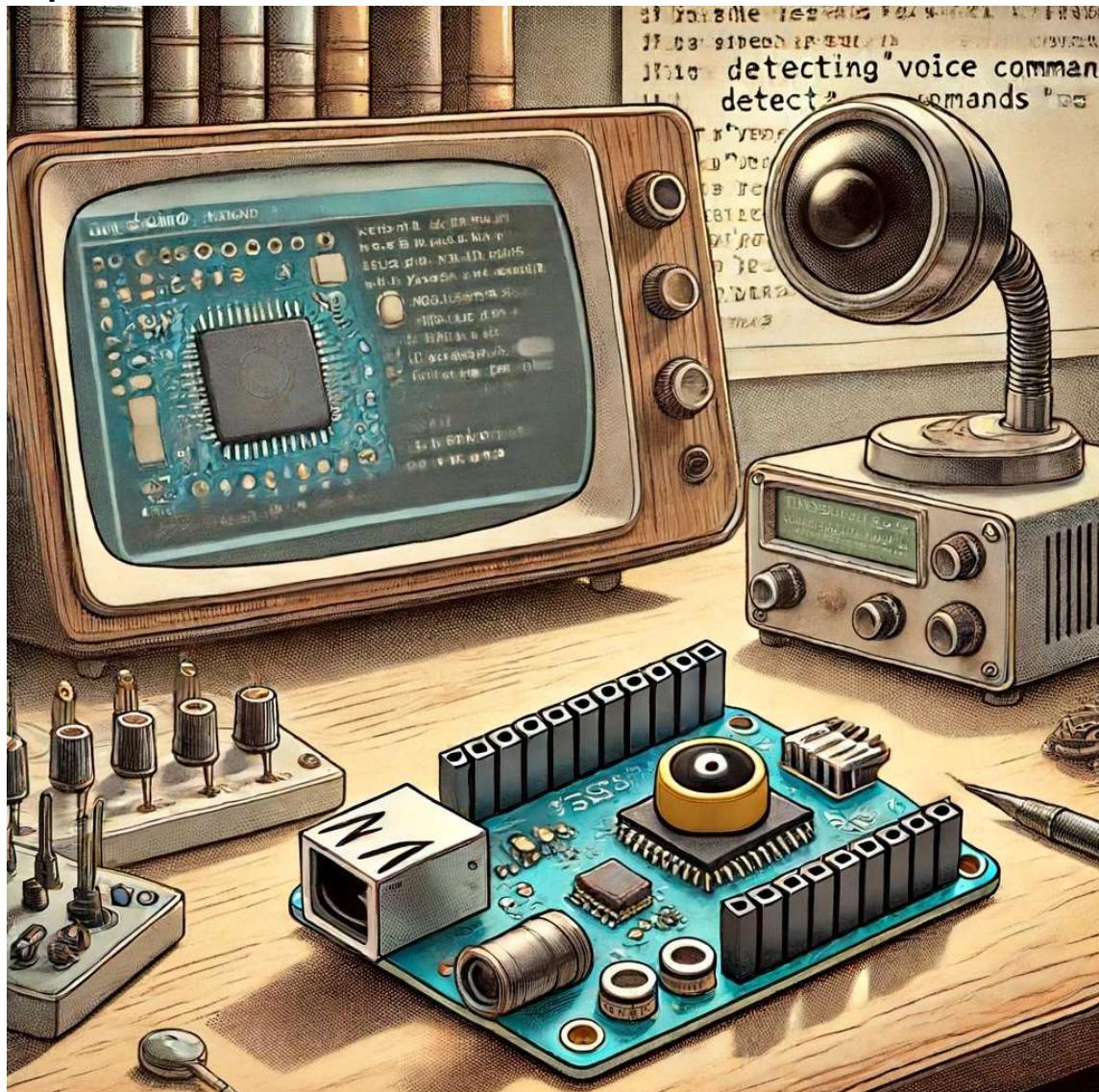
Risorse

- [Codice Arduino](#)

- Edge Impulse Spectral Features Block Colab Notebook
- Progetto Edge Impulse

XIAO ESP32S3

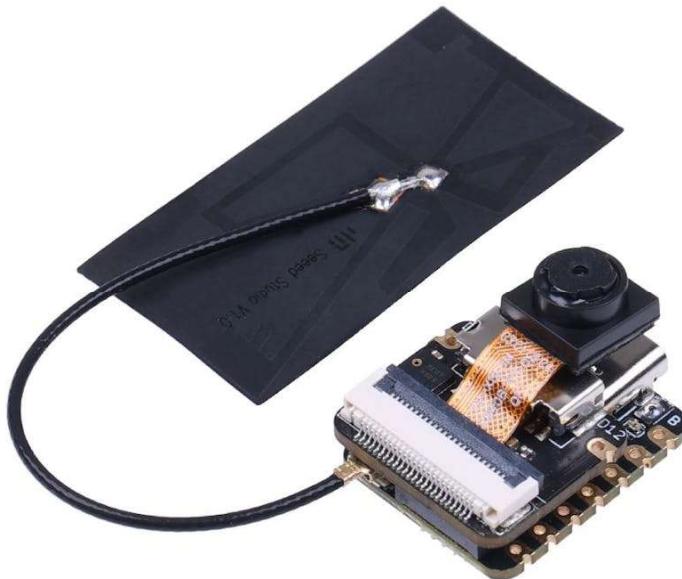
Setup



DALL-E prompt - Disegno in stile cartone animato anni '50 di una scheda XIAO ESP32S3 con un modulo telecamera distintivo, come mostrato nell'immagine fornita. La scheda è posizionata su un classico tavolo da laboratorio con vari sensori, tra cui un microfono. Dietro la scheda, uno schermo di computer vintage visualizza l'IDE Arduino in colori tenui, con codice incentrato sulle configurazioni dei pin LED e inferenza di apprendimento automatico per i comandi vocali. Il Serial Monitor sull'IDE mostra output che rilevano comandi vocali come 'yes' e 'no'. La scena fonde il fascino retrò dei laboratori di metà secolo con l'elettronica moderna.

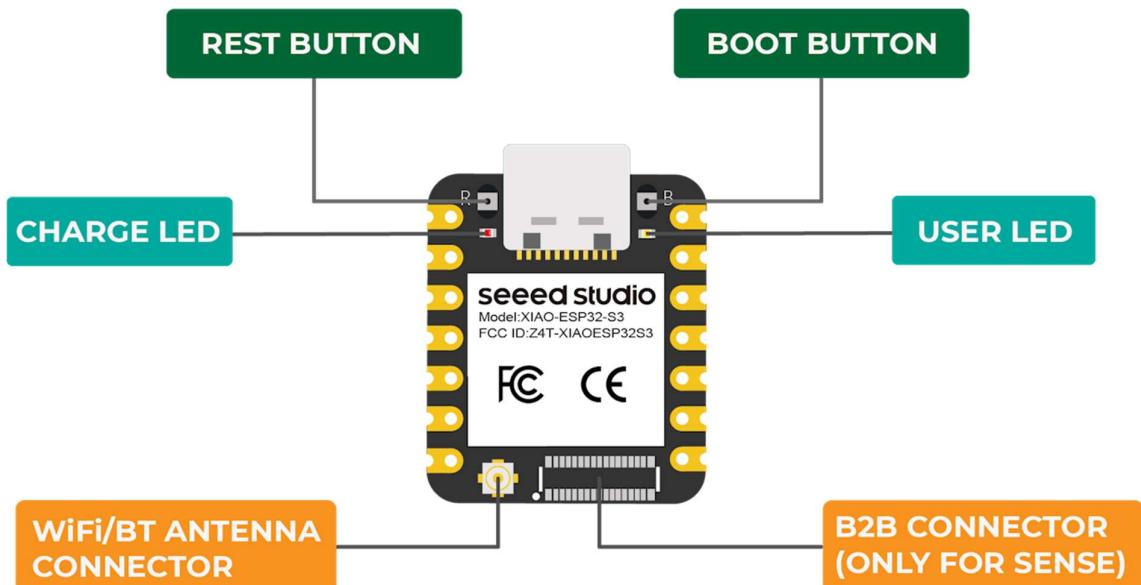
Panoramica

XIAO ESP32S3 Sense è la scheda di sviluppo conveniente di Seeed Studio, che integra un sensore della fotocamera, un microfono digitale e il supporto per schede SD. Combinando la potenza di elaborazione ML embedded e la capacità fotografica, questa scheda di sviluppo è un ottimo strumento per iniziare con TinyML (IA vocale e visiva).

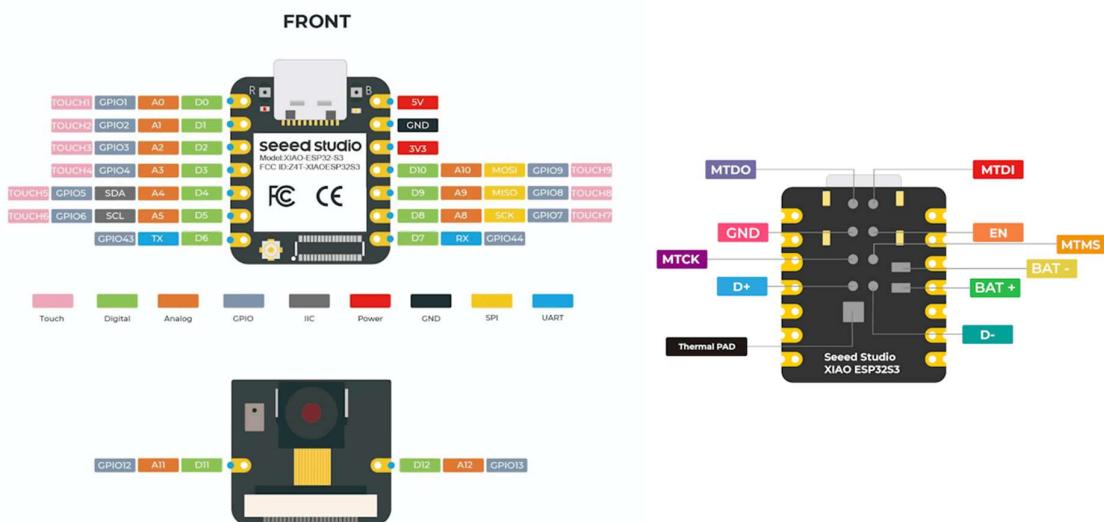


Caratteristiche Principali di XIAO ESP32S3 Sense

- **Potente Scheda MCU:** Incorpora il chip del processore Xtensa dual-core ESP32S3 a 32 bit che funziona fino a 240 MHz, più porte di sviluppo montate, supporto per Arduino/MicroPython
- **Funzionalità Avanzate:** Sensore della fotocamera OV2640 staccabile per una risoluzione di 1600 * 1200, compatibile con il sensore della fotocamera OV5640, integra un microfono digitale aggiuntivo
- **Progetto di Alimentazione Elaborato:** La capacità di gestione della carica della batteria al litio offre quattro modelli di consumo energetico, che consentono la modalità di “deep sleep” [sospensione profonda] con un consumo energetico basso fino a 14 μ A
- **Grande Memoria per più Possibilità:** Offre 8 MB di PSRAM e 8 MB di FLASH, supportando uno slot per schede SD di memoria FAT esterno da 32 GB
- **Prestazioni RF Eccezionali:** Supporta la comunicazione wireless duale Wi-Fi e BLE a 2,4 GHz, supporta la comunicazione remota a 100m+ se connesso all'antenna U.FL
- **Design Compatto delle Dimensioni di un Pollice:** 21 x 17,5 mm, adotta il fattore di forma classico di XIAO, adatto per progetti con spazio limitato come i dispositivi indossabili



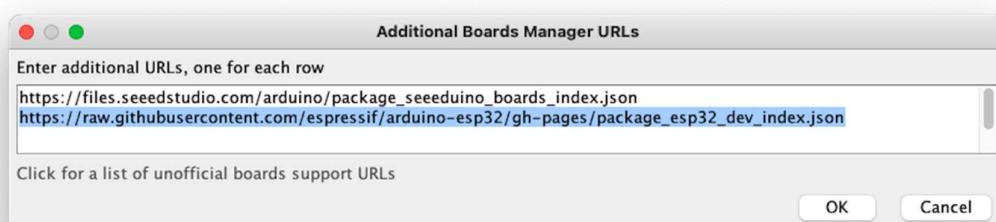
Di seguito è riportato il pinout generale della scheda:



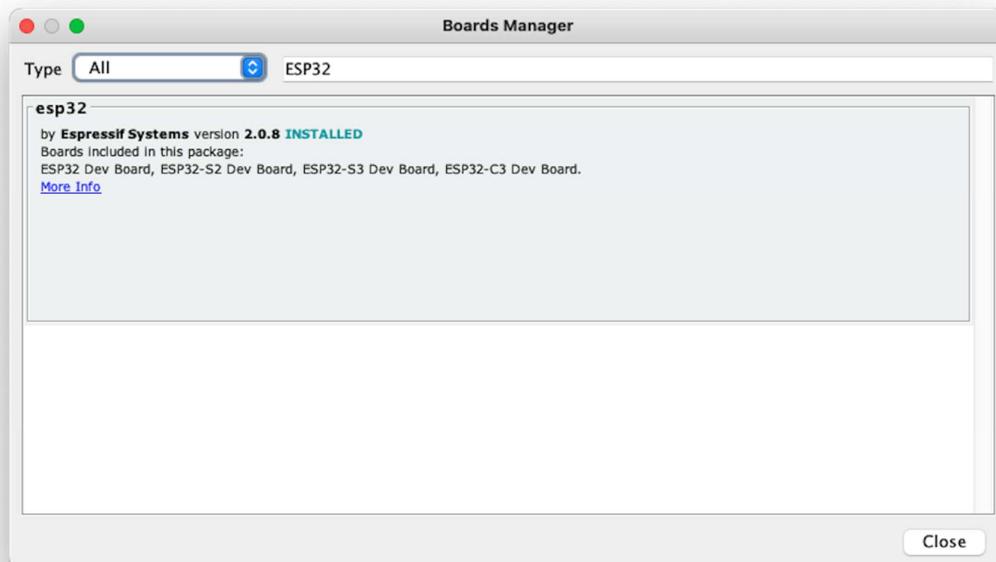
Per maggiori dettagli, fare riferimento alla pagina Seeed Studio WiKi:
https://wiki.seeedstudio.com/xiao_esp32s3_getting_started/

Installazione di XIAO ESP32S3 Sense su Arduino IDE

Su Arduino IDE, si va su **File > Preferences** e si inserisce l'URL:
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_dev_index.json
 nel campo ==> **Additional Boards Manager URLs**



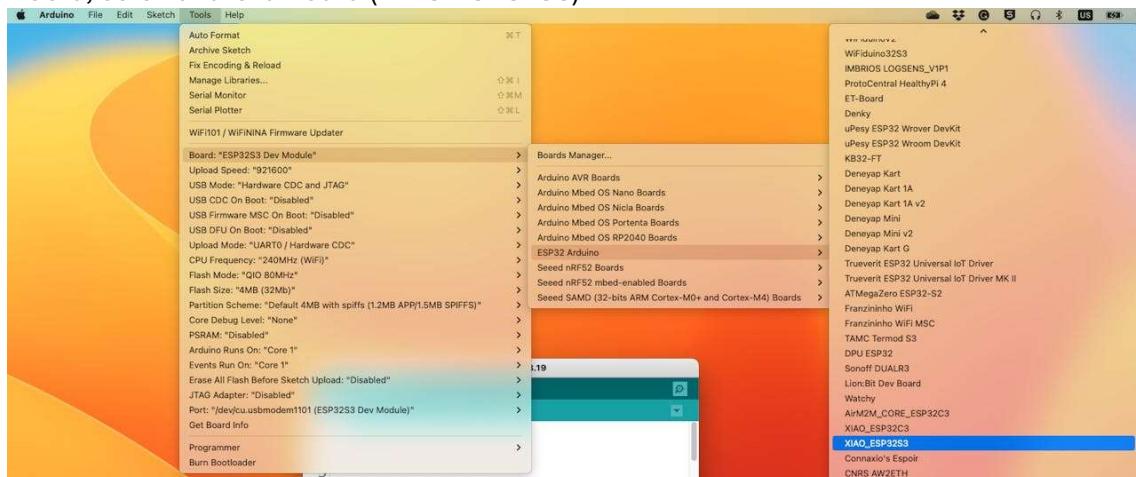
Poi si apre il gestore delle schede. Si va su **Tools > Board > Boards Manager...** e immettendo esp32. Selezionare e installare il pacchetto più aggiornato e stabile (evitare le versioni *alpha*):



⚠ Attenzione

Le versioni Alpha (ad esempio, 3.x-alpha) non funzionano correttamente con XIAO ed Edge Impulse. Utilizzare invece l'ultima versione stabile (ad esempio, 2.0.11).

Su **Tools**, selezionare la Board (**XIAO ESP32S3**):



Ultimo ma non meno importante, scegliere la **Porta** a cui è collegato l'ESP32S3.

Ecco fatto! Il dispositivo dovrebbe funzionare. Facciamo qualche test.

Test della scheda con BLINK

XIAO ESP32S3 Sense ha un LED integrato che è collegato a GPIO21. Quindi, si può eseguire lo sketch Blink così com'è (utilizzando la costante Arduino `LED_BUILTIN`) o modificando di conseguenza lo sketch Blink:

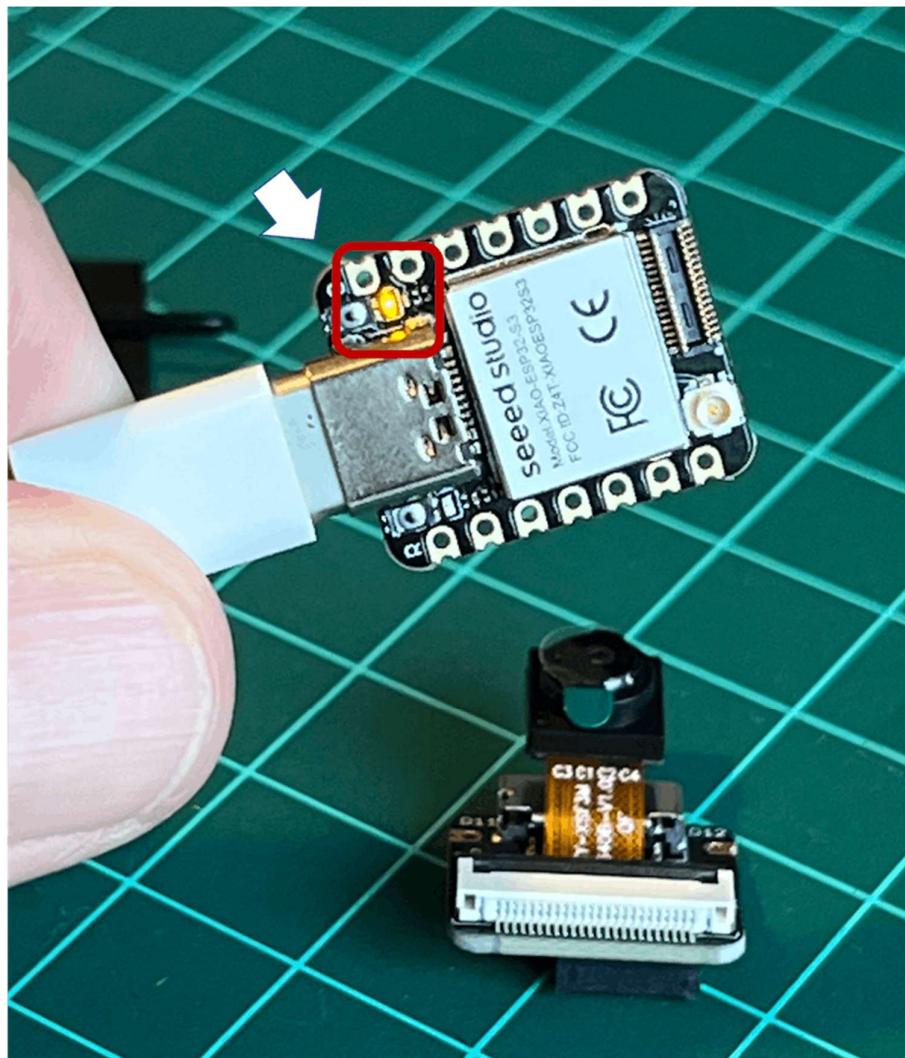
```
#define LED_BUILT_IN 21

void setup() {
  pinMode(LED_BUILT_IN, OUTPUT); // Set the pin as output
}

// Remember that the pin work with inverted logic
// LOW to Turn on and HIGH to turn off
void loop() {
  digitalWrite(LED_BUILT_IN, LOW); //Turn on
  delay (1000); //Wait 1 sec
```

```
digitalWrite(LED_BUILT_IN, HIGH); //Turn off  
delay (1000); //Wait 1 sec  
}
```

Notare che i pin funzionano con logica invertita: BASSO per accendere e ALTO per spegnere.



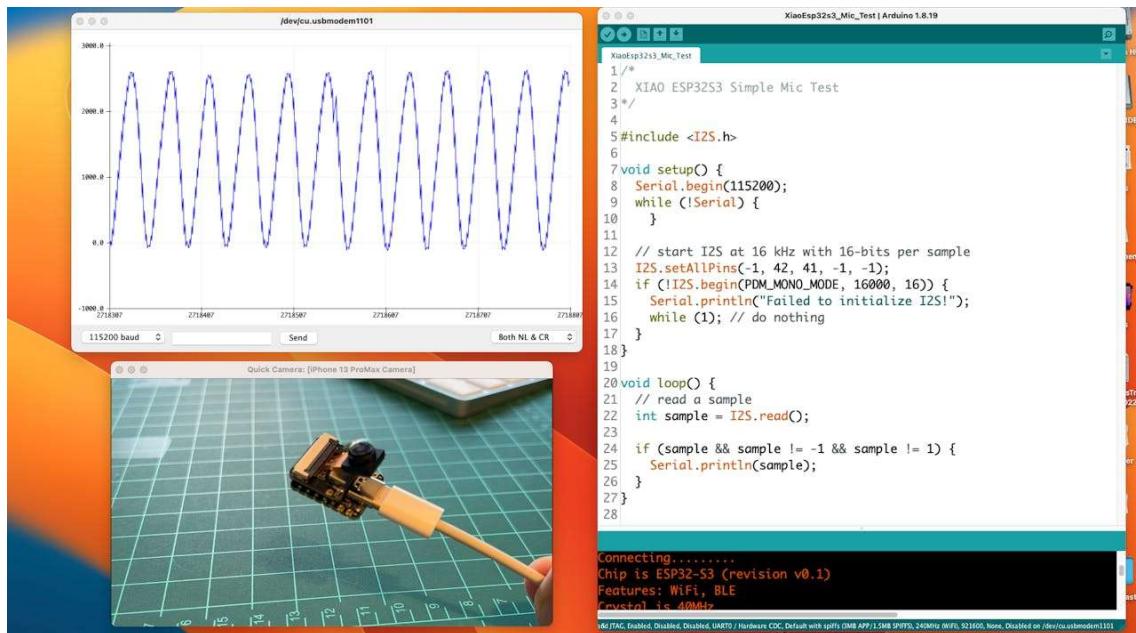
Collegamento del Modulo Sense (Scheda di Espansione)

Quando viene acquistata, la scheda di espansione è separata da quella principale, ma installarla è molto semplice. Si deve allineare il connettore sulla scheda di espansione col connettore B2B sullo XIAO ESP32S3, premerlo con forza e quando si sente un “clic”, l’installazione è completa.

Come commentato nell’introduzione, la scheda di espansione, o la parte “sense” del dispositivo, ha una fotocamera OV2640 da 1600x1200, uno slot per schede SD e un microfono digitale.

Test del Microfono

Cominciamo con il rilevamento del suono. Si va nei [progetti GitHub](#) e si scarica lo sketch: [XIAOEsp2s3_Mic_Test](#) poi lo si esegue sull’IDE Arduino:



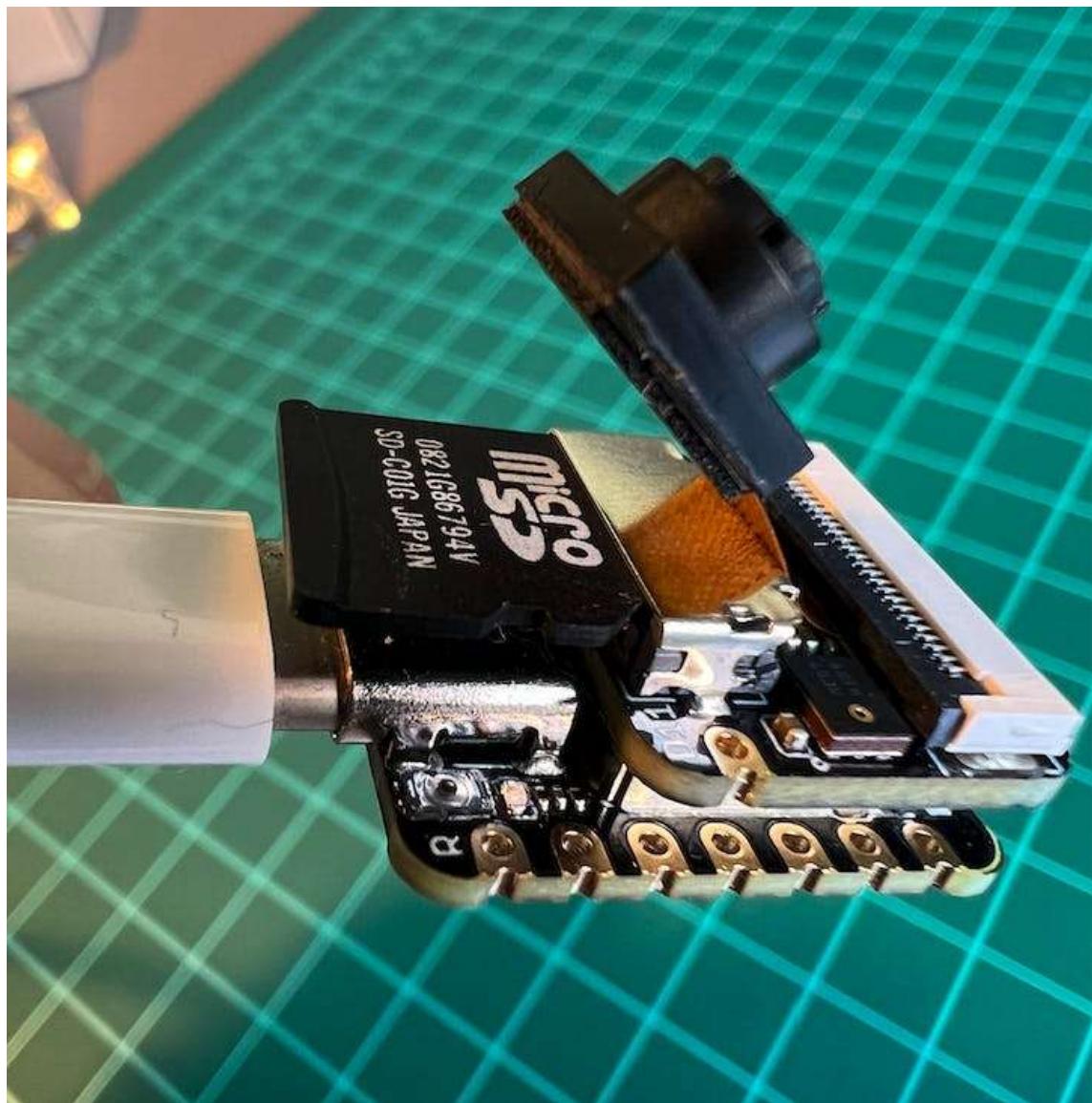
Quando si produce un suono, è possibile verificarlo sul Serial Plotter.

Salvare il suono registrato (file audio .wav) su una scheda microSD.

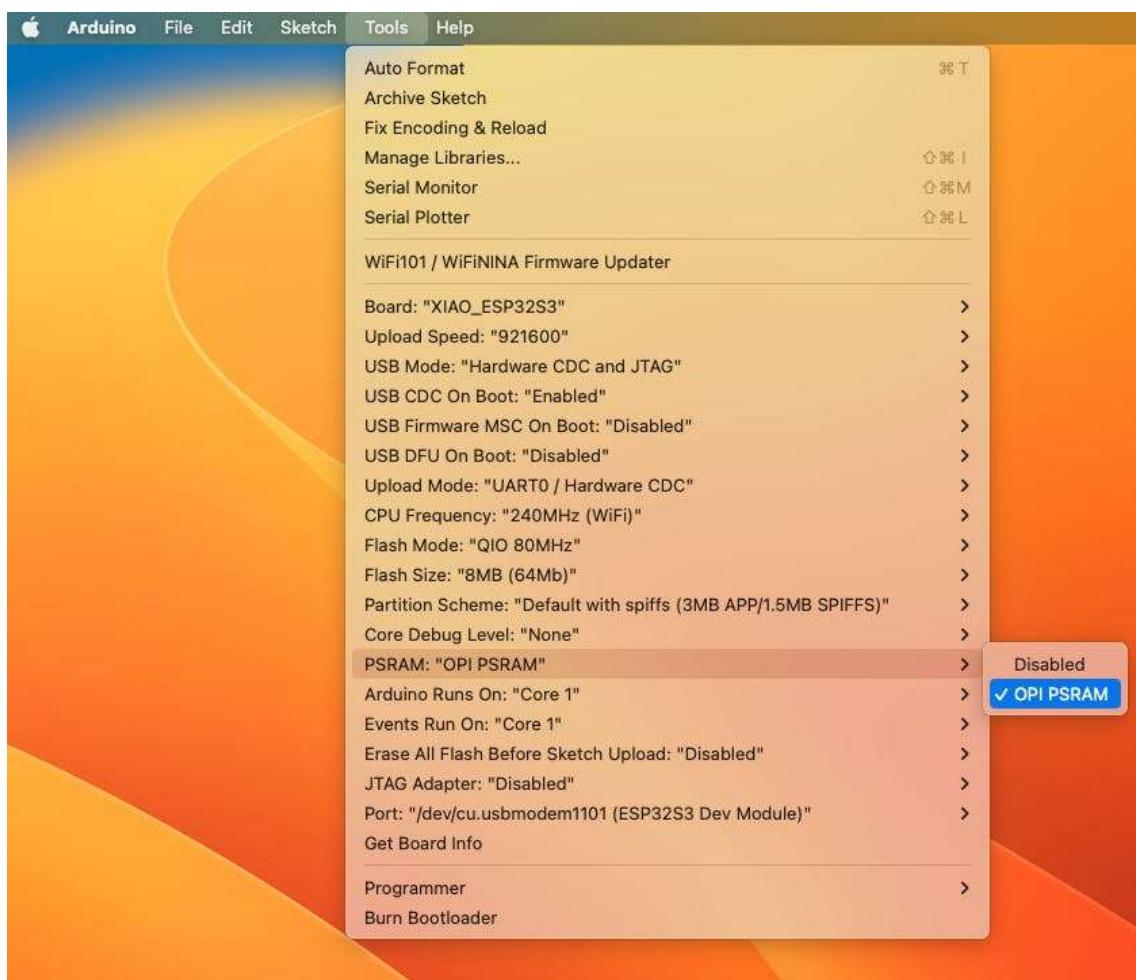
Ora, il lettore di schede SD integrato può salvare file audio .wav. Per farlo, dobbiamo abilitare la PSRAM XIAO.

ESP32-S3 ha solo poche centinaia di kilobyte di RAM interna sul chip MCU. Questo può essere insufficiente per alcuni scopi, quindi fino a 16 MB di PSRAM esterna (RAM pseudo-statica) possono essere collegati al chip flash SPI. La memoria esterna è incorporata nella mappa di memoria e, con alcune restrizioni, è utilizzabile allo stesso modo della RAM dati interna.

Per iniziare, inserire la scheda SD su XIAO come mostrato nella foto qui sotto (la scheda SD deve essere formattata in **FAT32**).



- Scaricare lo sketch [Wav_Record](#), che si può trovare su GitHub.
- Per eseguire il codice (Wav Record), è necessario utilizzare la funzione PSRAM del chip ESP-32, quindi la si deve attivare prima di caricare: Tools>PSRAM: “OPI PSRAM”>OPI PSRAM



- Eseguire il codice `Wav_Record.ino`
- Questo programma viene eseguito solo una volta dopo che l'utente accende il monitor seriale. Registra per 20 secondi e salva il file su una scheda microSD come "arduino_rec.wav."
- Quando “.” viene emesso ogni 1 secondo nel monitor seriale, l'esecuzione del programma è terminata e si può riprodurre il file audio registrato con l'aiuto di un lettore di schede.



La qualità del suono è eccellente!

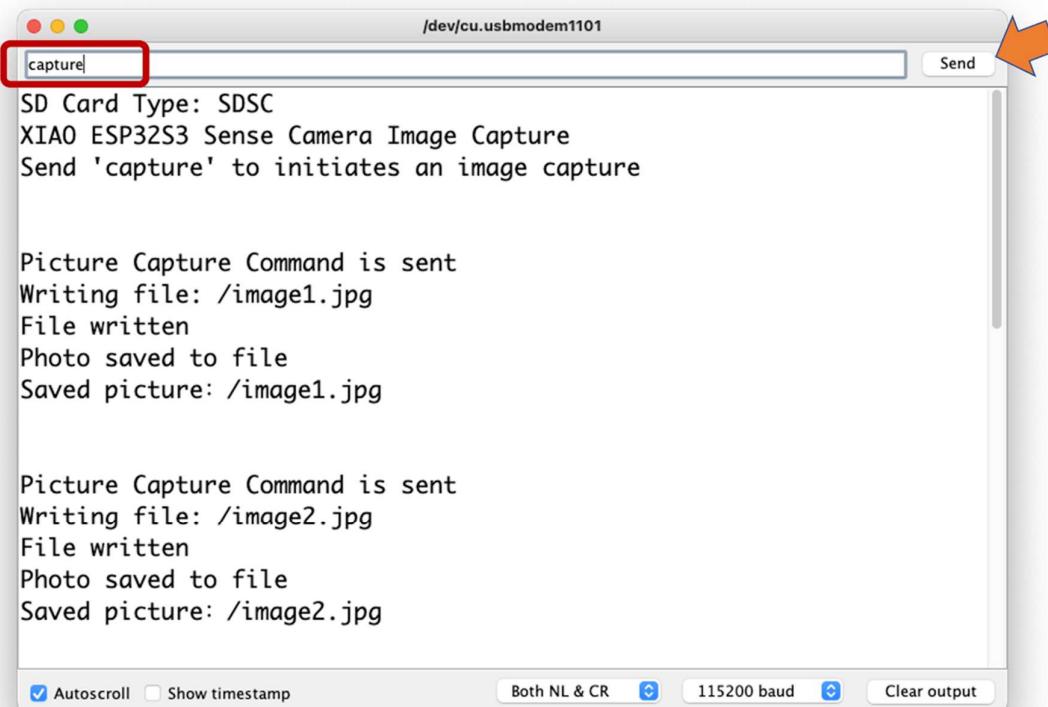
La spiegazione di come funziona il codice va oltre lo scopo di questo tutorial, ma c'è un'eccellente descrizione sulla pagina [wiki](#).

Test della Fotocamera

Per testare la fotocamera, si deve scaricare la cartella [take_photos_command](#) da GitHub. La cartella contiene lo sketch (.ino) e due file .h con i dettagli della fotocamera.

- Eseguire il codice: `take_photos_command.ino`. Aprire il Serial Monitor e inviare il comando `capture` per catturare e salvare l'immagine sulla scheda SD:

Verificare che `[Both NL & CR]` sia selezionato sul Serial Monitor.



```
capture
SD Card Type: SDSC
XIAO ESP32S3 Sense Camera Image Capture
Send 'capture' to initiates an image capture

Picture Capture Command is sent
Writing file: /image1.jpg
File written
Photo saved to file
Saved picture: /image1.jpg

Picture Capture Command is sent
Writing file: /image2.jpg
File written
Photo saved to file
Saved picture: /image2.jpg
```

Autoscroll Show timestamp Both NL & CR 115200 baud Clear output

Ecco un esempio di una foto scattata:

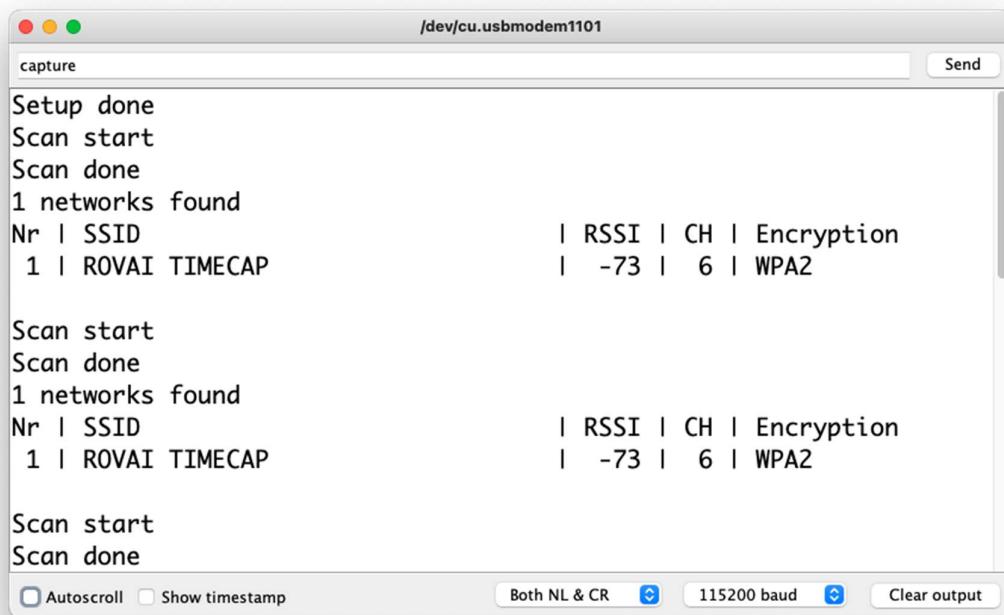


Test del WiFi

Uno dei punti di forza di XIAO ESP32S3 è la sua capacità WiFi. Quindi, testiamo la sua radio eseguendo la scansione delle reti Wi-Fi attorno ad essa. Lo si può fare eseguendo uno degli esempi di codice sulla scheda.

Si va su Arduino IDE Examples e si cerca **WiFi ==> WiFiScan**

Si dovrebbero vedere le reti Wi-Fi (SSID e RSSI) nel raggio d'azione del dispositivo sul monitor seriale. Ecco cosa abbiamo ottenuto in laboratorio:



The screenshot shows a Mac OS X terminal window titled "capture" with the path "/dev/cu.usbmodem1101". The window displays two sets of WiFi scan results. The first set shows a single network named "ROVAI TIMECAP" with RSSI -73, Channel 6, and WPA2 encryption. The second set shows another scan with the same result. The bottom of the window has checkboxes for "Autoscroll" and "Show timestamp", and buttons for "Both NL & CR", "115200 baud", and "Clear output".

```

Setup done
Scan start
Scan done
1 networks found
Nr | SSID | RSSI | CH | Encryption
1 | ROVAI TIMECAP | -73 | 6 | WPA2

Scan start
Scan done
1 networks found
Nr | SSID | RSSI | CH | Encryption
1 | ROVAI TIMECAP | -73 | 6 | WPA2

Scan start
Scan done
 Autoscroll  Show timestamp Both NL & CR 115200 baud Clear output

```

Un Semplice Server WiFi (accensione/spegnimento del LED)

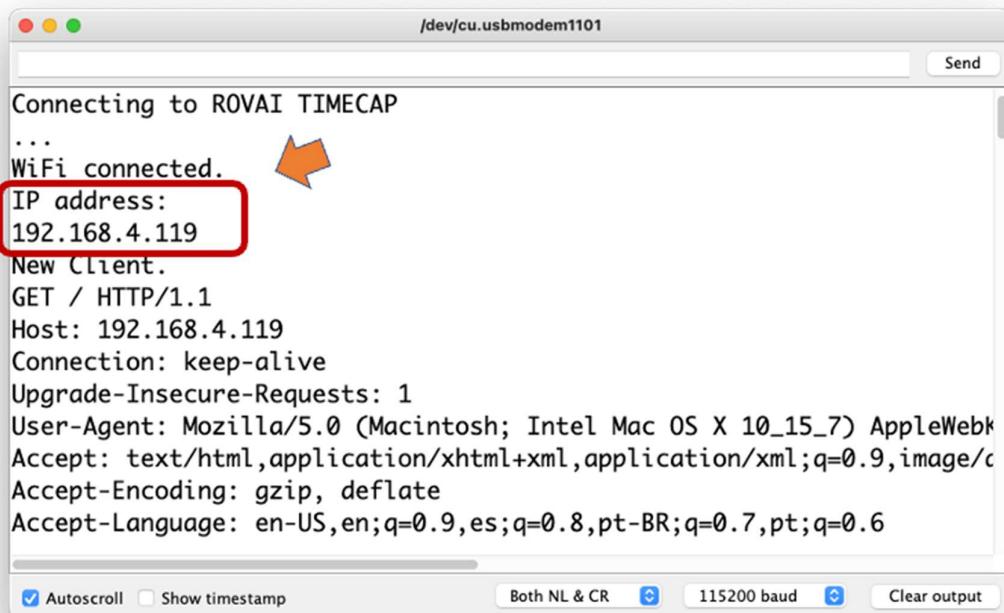
Testiamo la capacità del dispositivo di comportarsi come un server WiFi. Ospiteremo una semplice pagina sul dispositivo che invia comandi per accendere e spegnere il LED integrato di XIAO.

Come prima, si va su GitHub per scaricare la cartella usando lo sketch [SimpleWiFiServer](#).

Prima di eseguire lo sketch, si devono inserire le credenziali di rete:

```
const char* ssid      = "Your credentials here";
const char* password = "Your credentials here";
```

Si può monitorare il funzionamento del server con Serial Monitor.



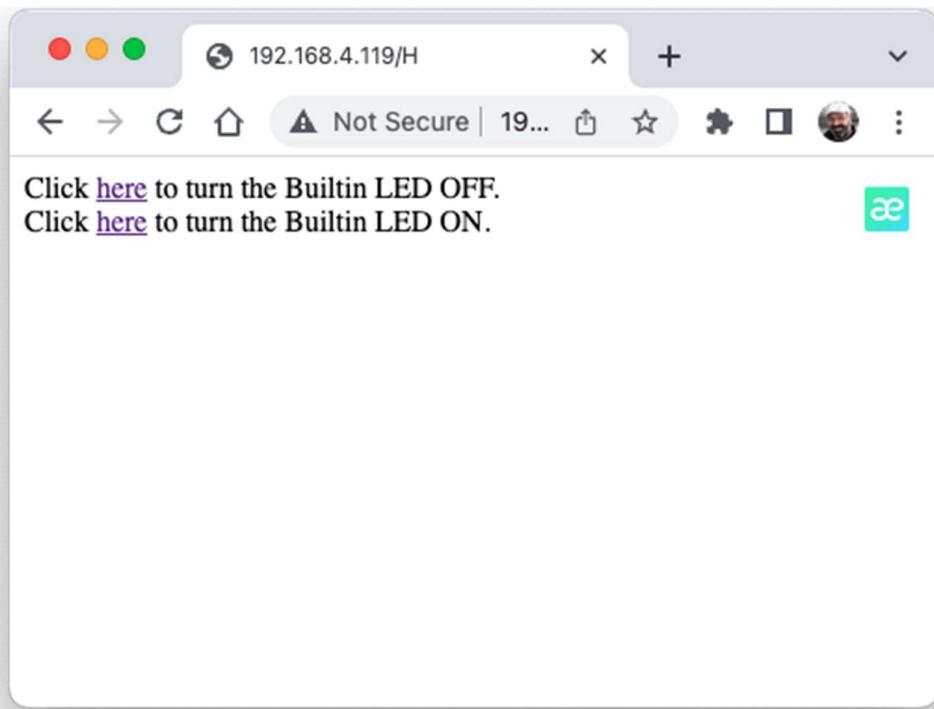
The screenshot shows a Mac OS X terminal window titled "capture" with the path "/dev/cu.usbmodem1101". It displays a WiFi connection log to "ROVAI TIMECAP" and an incoming HTTP request from a client. The IP address "192.168.4.119" is highlighted with a red box and an orange arrow points to it. The bottom of the window has checkboxes for "Autoscroll" and "Show timestamp", and buttons for "Both NL & CR", "115200 baud", and "Clear output".

```

Connecting to ROVAI TIMECAP
...
WiFi connected.
IP address:
192.168.4.119
New Client.
GET / HTTP/1.1
Host: 192.168.4.119
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.0 Safari/605.1.15
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,es;q=0.8,pt-BR;q=0.7,pt;q=0.6
 Autoscroll  Show timestamp Both NL & CR 115200 baud Clear output

```

Prendere l'indirizzo IP e inserirlo nel browser:



Si vedrà una pagina con link che possono accendere e spegnere il LED integrato del XIAO.

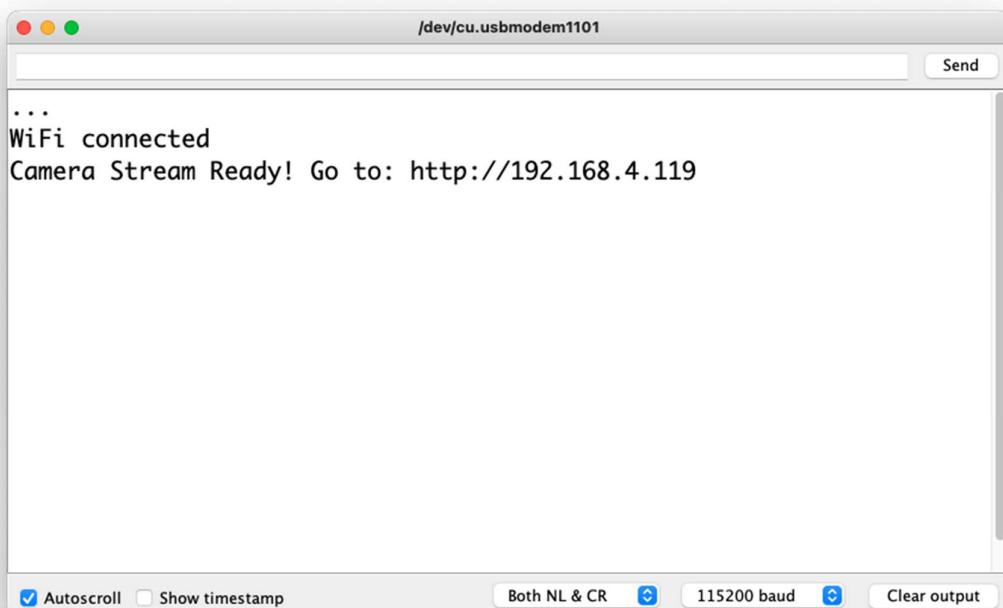
Streaming video sul Web

Ora che sappiamo che si possono inviare comandi dalla pagina Web al dispositivo, facciamo il contrario. Prendiamo l'immagine catturata dalla telecamera e la trasmettiamo in streaming su una pagina Web:

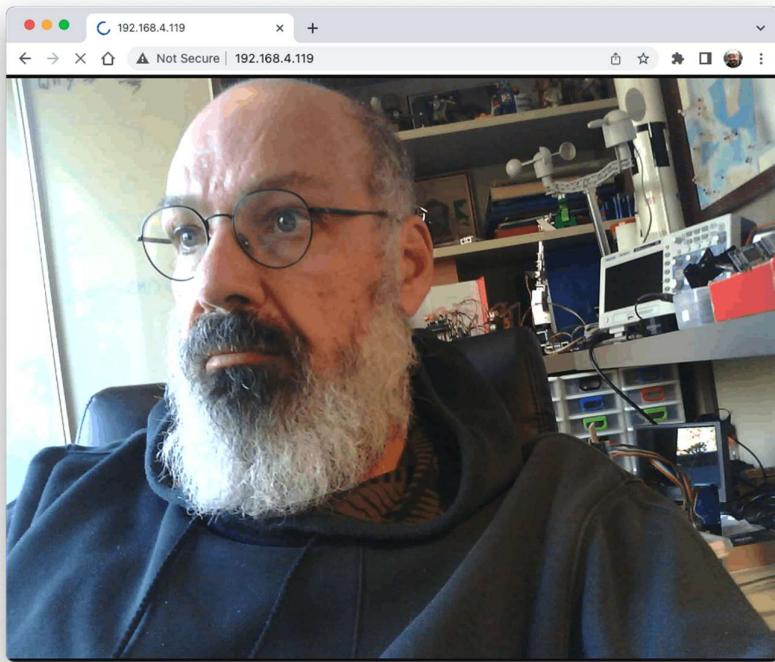
Scaricare da GitHub la [cartella](#) che contiene il codice: XIAO-ESP32S3-Streeming_Video.ino.

Ricordarsi che la cartella contiene il file .ino e un paio di file .h necessari per gestire la telecamera.

Inserire le credenziali ed eseguire lo sketch. Sul monitor seriale, si può trovare l'indirizzo della pagina da inserire nel browser:

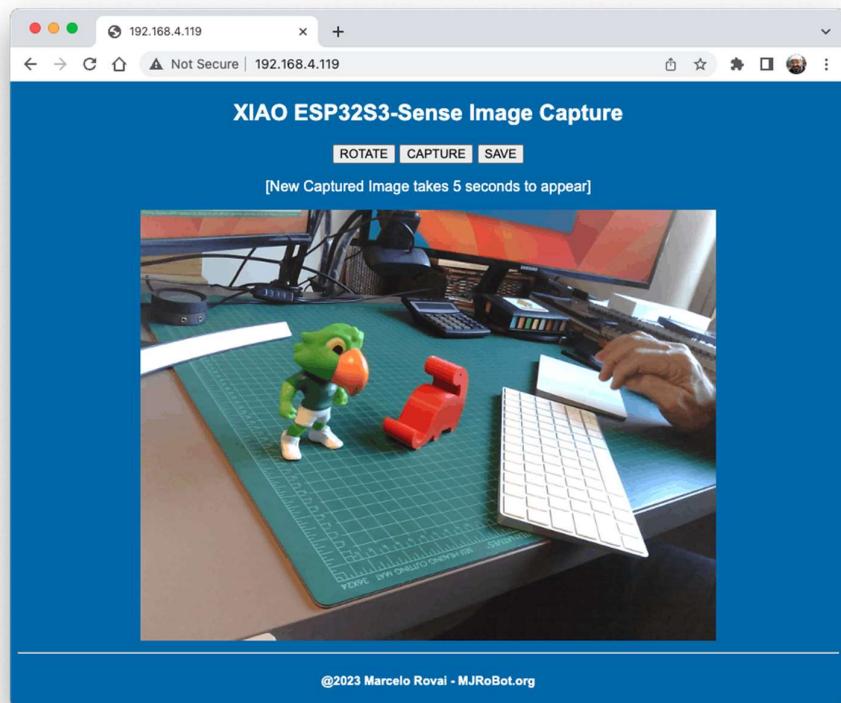


Apri la pagina sul browser (attendere qualche secondo per avviare lo streaming). Tutto qui.



Mettere in streaming ciò che la telecamera “vede” può essere importante quando la si posiziona per catturare un set di dati per un progetto ML (ad esempio, usando il codice “take_photos_commands.ino”).

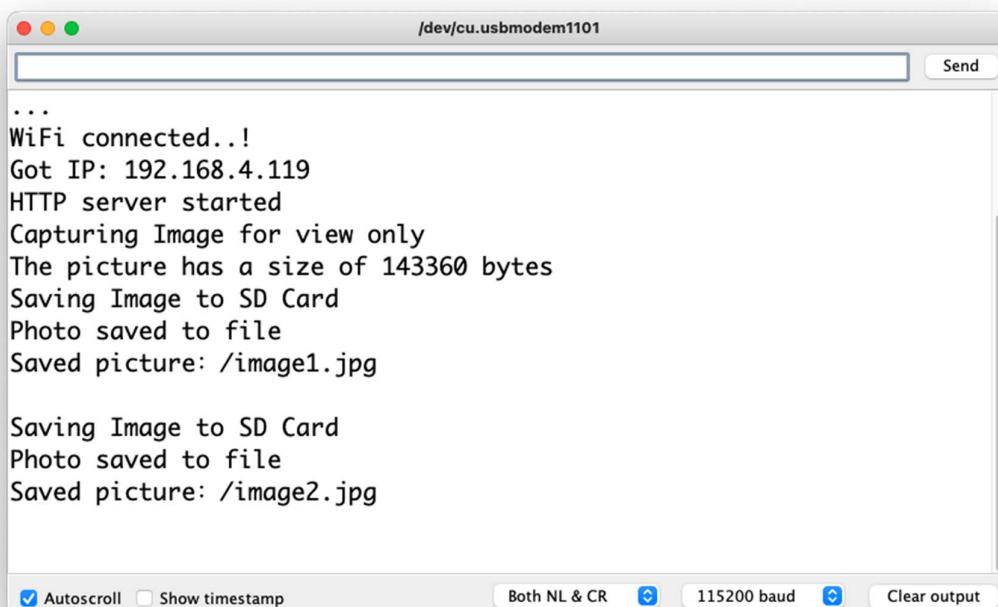
Naturalmente, possiamo fare entrambe le cose contemporaneamente: mostrare ciò che la telecamera vede sulla pagina e inviare un comando per catturare e salvare l’immagine sulla scheda SD. Per questo, si può usare il codice Camera_HTTP_Server_STA, scaricabile da GitHub.



Il programma eseguirà le seguenti attività:

- Imposta la fotocamera in modalità di output JPEG.

- Crea una pagina Web (ad esempio ==> <http://192.168.4.119/>). L'indirizzo corretto verrà visualizzato sul Serial Monitor.
- Se server.on (“/capture”, HTTP_GET, serverCapture), il programma scatta una foto e la invia al Web.
- È possibile ruotare l'immagine sulla pagina Web utilizzando il pulsante [ROTATE]
- Il comando [CAPTURE] visualizzerà solo l'anteprima dell'immagine sulla pagina Web, mostrandone le dimensioni sul Serial Monitor
- Il comando [SAVE] salverà un'immagine sulla scheda SD e la mostrerà sul browser.
- Le immagini salvate seguiranno una denominazione sequenziale (image1.jpg, image2.jpg).



Questo programma può catturare un set di dati di immagini con un progetto di classificazione delle immagini.

Esaminate il codice; sarà più facile capire come funziona la fotocamera. Questo codice è stato sviluppato sulla base del fantastico tutorial di Rui Santos [ESP32-CAM Take Photo and Display in Web Server](#), che invito tutti a visitare.

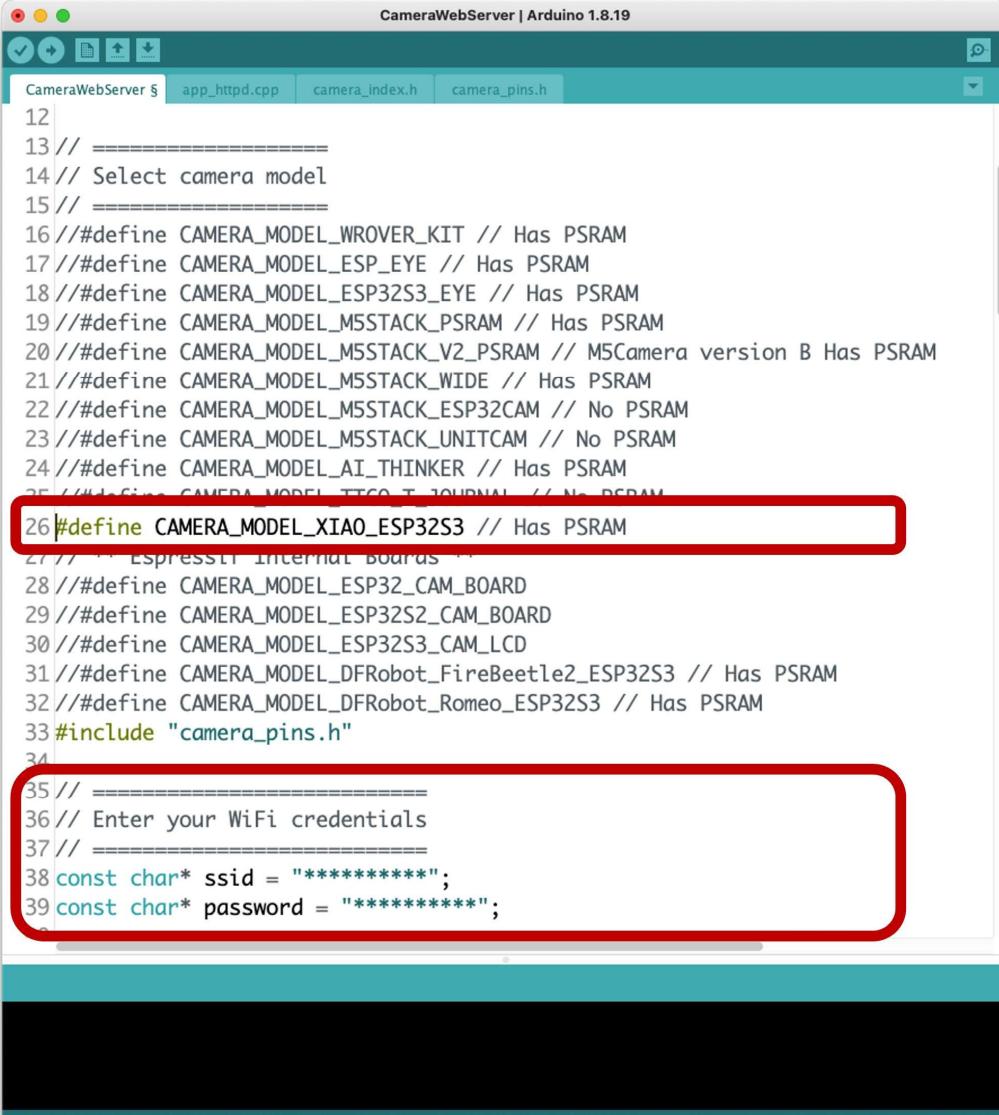
Uso di CameraWebServer

Nell'IDE Arduino, si va su `File > Examples > ESP32 > Camera` e si seleziona `CameraWebServer`. Si devono anche commentare tutti i modelli di fotocamere, eccetto i pin del modello XIAO:

```
#define CAMERA_MODEL_XIAO_ESP32S3 // Has PSRAM
```

Non dimenticare `Tools` per abilitare la PSRAM.

Inserire le credenziali wifi e caricare il codice sul dispositivo:

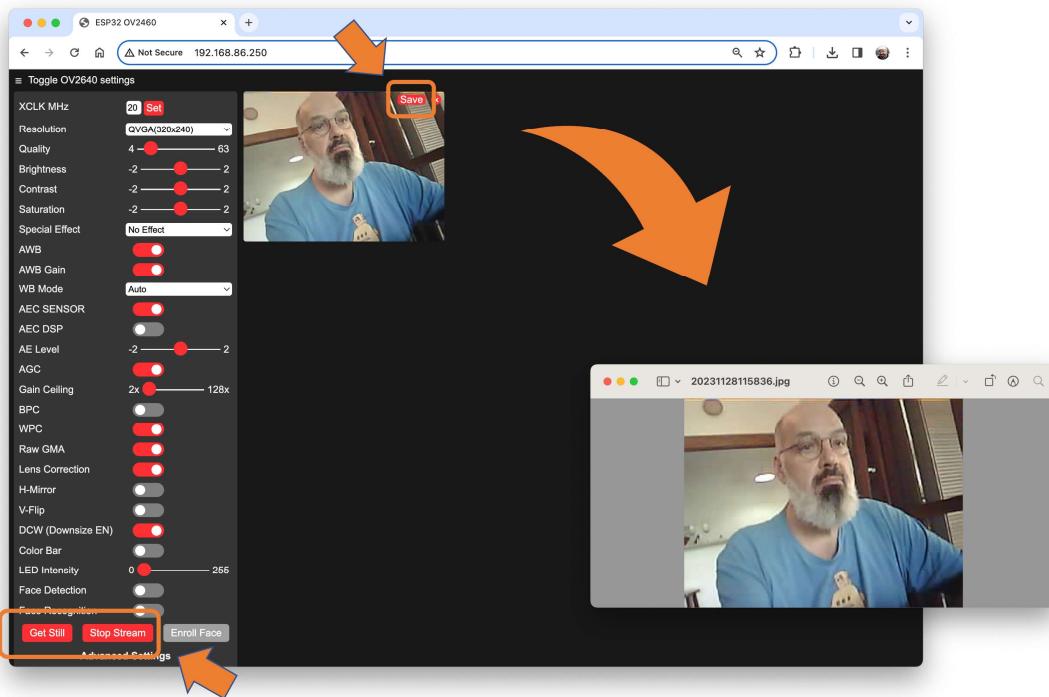


```
CameraWebServer | Arduino 1.8.19
CameraWebServer S app_httpd.cpp camera_index.h camera_pins.h
12
13 // =====
14 // Select camera model
15 // =====
16 #define CAMERA_MODEL_WROVER_KIT // Has PSRAM
17 #define CAMERA_MODEL_ESP_EYE // Has PSRAM
18 #define CAMERA_MODEL_ESP32S3_EYE // Has PSRAM
19 #define CAMERA_MODEL_M5STACK_PSRAM // Has PSRAM
20 #define CAMERA_MODEL_M5STACK_V2_PSRAM // M5Camera version B Has PSRAM
21 #define CAMERA_MODEL_M5STACK_WIDE // Has PSRAM
22 #define CAMERA_MODEL_M5STACK_ESP32CAM // No PSRAM
23 #define CAMERA_MODEL_M5STACK_UNITCAM // No PSRAM
24 #define CAMERA_MODEL_AI_THINKER // Has PSRAM
25 // Define CAMERA_MODEL_TTC2_T_JOURNAL // No PSRAM
26 #define CAMERA_MODEL_XIAO_ESP32S3 // Has PSRAM
27 // `` Espressif internal boards ''
28 #define CAMERA_MODEL_ESP32_CAM_BOARD
29 #define CAMERA_MODEL_ESP32S2_CAM_BOARD
30 #define CAMERA_MODEL_ESP32S3_CAM_LCD
31 #define CAMERA_MODEL_DFRobot_FireBeetle2_ESP32S3 // Has PSRAM
32 #define CAMERA_MODEL_DFRobot_Romeo_ESP32S3 // Has PSRAM
33 #include "camera_pins.h"
34
35 // =====
36 // Enter your WiFi credentials
37 // =====
38 const char* ssid = "*****";
39 const char* password = "*****";
```

Se il codice viene eseguito correttamente, si vedrà l'indirizzo sul monitor seriale:

```
WiFi connected
[ 1946][I][app_httpd.cpp:1361] startCameraServer(): Starting web server on port: '80'
[ 1948][I][app_httpd.cpp:13791 startCameraServer(): Starting stream server on port: '81'
Camera Ready! Use 'http://192.168.86.250' to connect
```

Copiare l'indirizzo sul browser e attendere che la pagina venga caricata. Selezionare la risoluzione della telecamera (ad esempio, QVGA) e selezionare [START STREAM]. Attendi qualche secondo/minuto, a seconda della connessione. Utilizzando il pulsante [Save], si può salvare un'immagine nell'area download del computer.



Ecco fatto! Si possono salvare le immagini direttamente sul computer per usarle nei progetti.

Conclusione

The XIAO ESP32S3 Sense è flessibile, economico e facile da programmare. Con 8 MB di RAM, la memoria non è un problema e il dispositivo può gestire molte attività di post-elaborazione, inclusa la comunicazione.

L'ultima versione del codice si trova nel repository GitHub: [XIAO-ESP32S3-Sense](#).

Risorse

- [XIAO ESP32S3 Code](#)

Classificazione delle Immagini

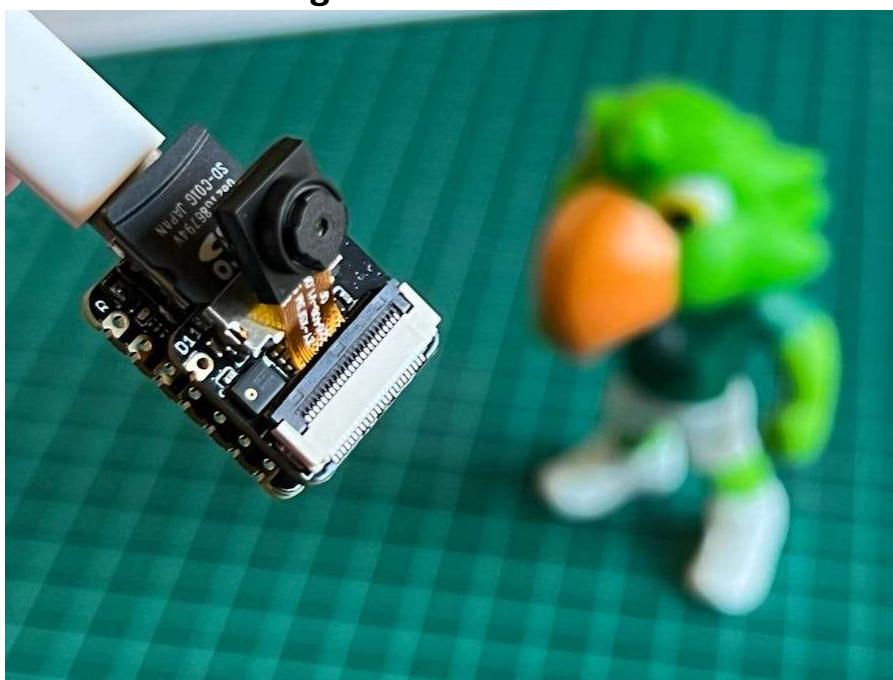


Immagine di Marcelo Rovai

Panoramica

Sempre di più, ci troviamo di fronte a una rivoluzione dell'intelligenza artificiale (IA) in cui, come affermato da Gartner, **Edge AI** ha un potenziale di impatto molto elevato, ed è ora!



In prima linea nel Radar delle Tecnologie Emergenti c'è il linguaggio universale di Edge Computer Vision. Quando esaminiamo il Machine Learning (ML) applicato alla visione, il primo concetto che ci accoglie è la classificazione delle immagini, una specie "Hello World" del ML che è sia semplice che profondo!

Seeed Studio XIAO ESP32S3 Sense è un potente strumento che combina il supporto per fotocamera e scheda SD. Con la sua potenza di elaborazione ML embedded e la capacità di fotografia, è un ottimo punto di partenza per esplorare l'intelligenza artificiale per la visione TinyML.

Un progetto TinyML di Motion Classification: Frutta contro Verdura



L'idea di base del nostro progetto sarà quella di addestrare un modello e procedere con l'inferenza su XIAO ESP32S3 Sense. Per l'addestramento, dovremmo trovare alcuni dati (**in effetti, tonnellate di dati!**).

Ma prima di tutto, abbiamo bisogno di un obiettivo! Cosa vogliamo classificare?

Con TinyML, un set di tecniche associate all'inferenza di apprendimento automatico su dispositivi embedded, dovremmo limitare la classificazione a tre o quattro categorie a causa di limitazioni (principalmente della memoria). Differenziamo **mele** da **banane** e **patate** (si possono provare altre categorie).

Quindi, cerchiamo un set di dati specifico che includa immagini da quelle categorie. Kaggle è un buon inizio:

<https://www.kaggle.com/kritikseth/fruit-and-vegetable-image-recognition>

Questo set di dati contiene immagini dei seguenti alimenti:

- **Frutta** - banana, mela, pera, uva, arancia, kiwi, anguria, melograno, ananas, mango.
- **Verdura** - cetriolo, carota, peperone, cipolla, patata, limone, pomodoro, ravanello, barbabietola, cavolo, lattuga, spinaci, soia, cavolfiore, peperone, peperoncino, rapa, mais, mais dolce, patata dolce, paprika, jalepeño, zenzero, aglio, piselli, melanzane.

Ogni categoria è suddivisa in **train** [addestramento] (100 immagini), **test** (10 immagini) e **validation** (10 immagini).

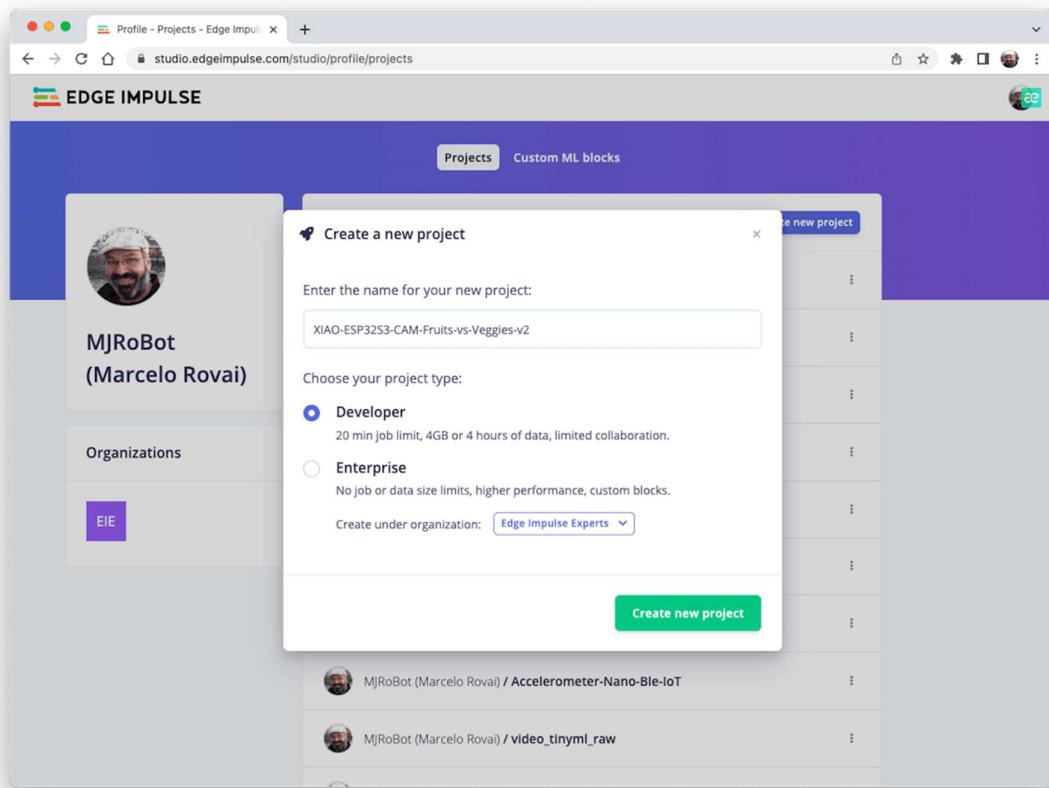
- Scaricare il dataset dal sito Web di Kaggle e metterlo sul computer.

Facoltativamente, si possono aggiungere alcune foto fresche di banane, mele e patate direttamente dalla cucina di casa, utilizzando, ad esempio, il codice discusso nel lab della configurazione.

Addestramento del modello con Edge Impulse Studio

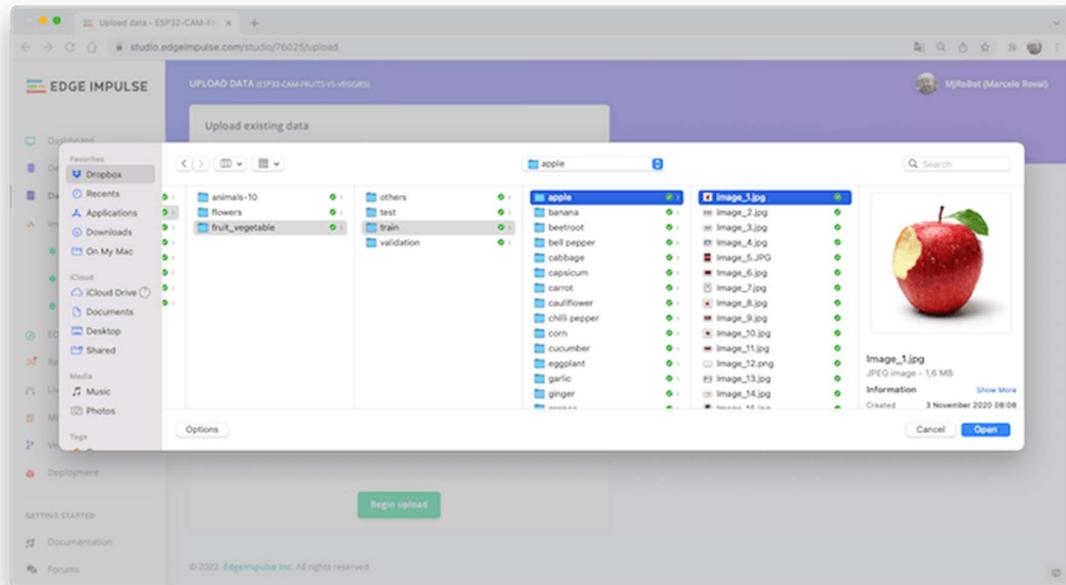
Utilizzeremo Edge Impulse Studio per addestrare il modello. Come si saprà, **Edge Impulse** è una piattaforma di sviluppo leader per l'apprendimento automatico su dispositivi edge.

Inserire le credenziali del proprio account (o crearne uno gratuito) su Edge Impulse. Quindi, creare un nuovo progetto:



Raccolta Dati

Successivamente, nella sezione **UPLOAD DATA**, caricare dal computer i file delle categorie scelte:



Sarebbe meglio se ora si avesse il proprio set di dati di training suddiviso in tre classi di dati:

The screenshot shows the Edge Impulse Data acquisition interface. On the left, there's a sidebar with various options like Dashboard, Devices, Data acquisition (which is selected), and Transfer Learning. The main area has tabs for Training data, Test data, and Export data. A message at the top says "Did you know? You can capture data from any device or development board, or upload your existing datasets - Show options". Below that, it shows "DATA COLLECTED 279 items" and a "TRAIN / TEST SPLIT 100% / 0%" button. A "Record new data" section indicates "No devices connected to the remote management API". A "RAW DATA" section shows a preview of an image named "Image_89.jpg.2p4ed0vt" which is a bunch of bananas.

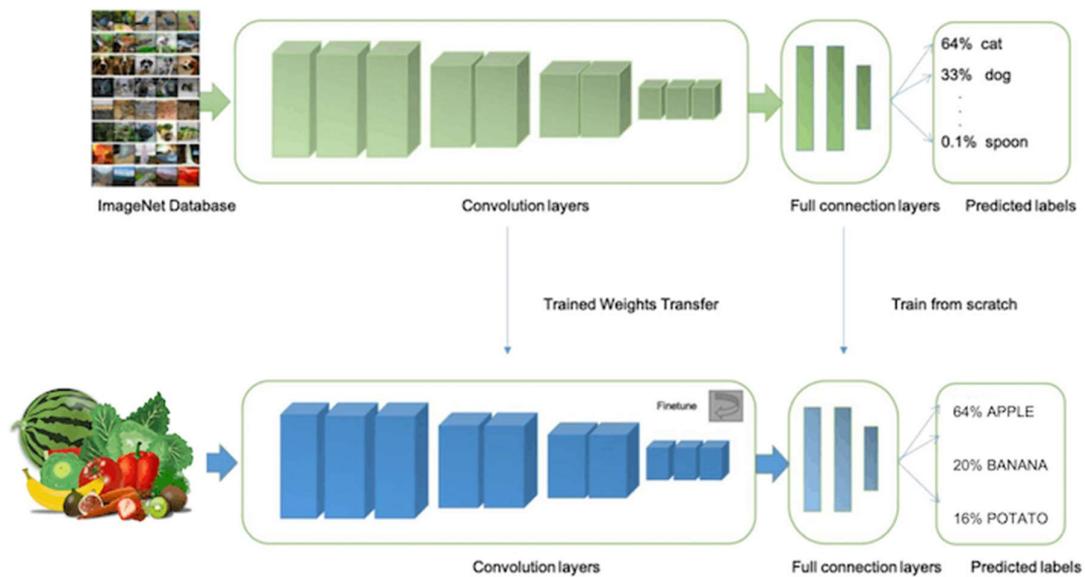
SAMPLE NAME	LABEL	ADDED	LENGTH
Image_89.jpg.2p4ed0vt	banana	Jan 12 2022, 15:45...	1
Image_91.jpg.2p4ed1po	banana	Jan 12 2022, 15:45...	1
Image_88.png.2p4ecv5e	banana	Jan 12 2022, 15:45...	1
Image_92.jpg.2p4ecus5	banana	Jan 12 2022, 15:45...	1
Image_100.jpg.2p4ectpi	banana	Jan 12 2022, 15:45...	1
Image_90.jpg.2p4ecsncc	banana	Jan 12 2022, 15:45...	1
Image_87.jpg.2p4ecr6	banana	Jan 12 2022, 15:45...	1
Image_83.png.2p4ecr56	banana	Jan 12 2022, 15:45...	1
Image_84.jpg.2p4ecr6e	banana	Jan 12 2022, 15:45...	1
Image_79.jpg.2p4ecrq	banana	Jan 12 2022, 15:45...	1
Image_86.jpg.2p4ecqbh	banana	Jan 12 2022, 15:45...	1
Image_85.jpg.2p4ecqsc	banana	Jan 12 2022, 15:45...	1

Si possono caricare dati extra per ulteriori test del modello o dividere i dati di training. Lo lasceremo così com'è per utilizzare più dati possibili.

Impulse Design

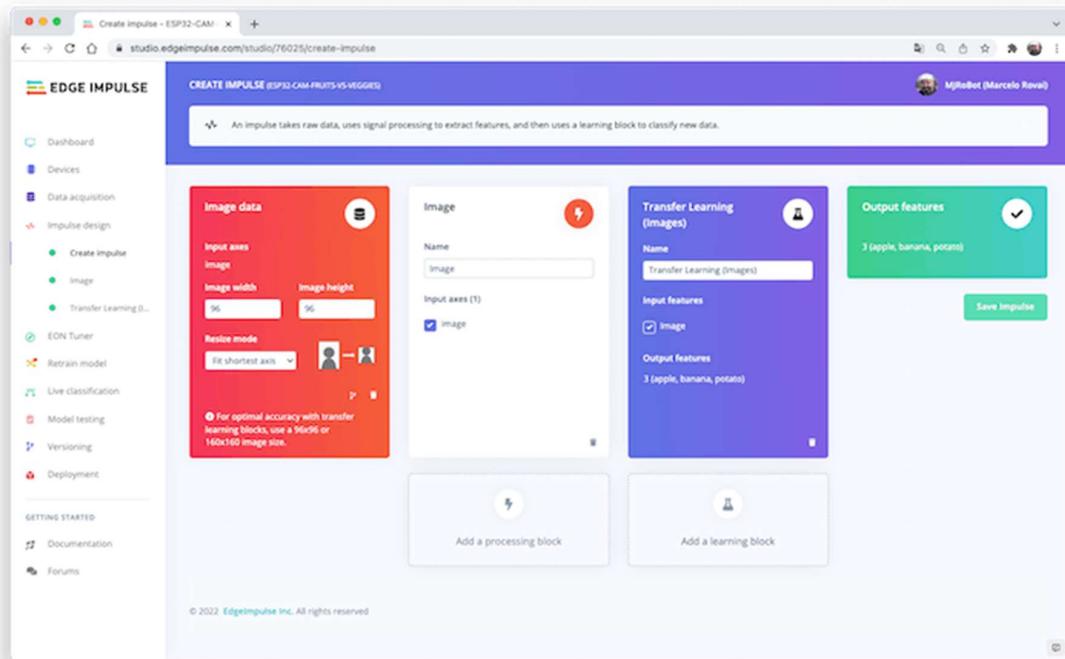
Un impulso prende dati grezzi (in questo caso, immagini), estrae feature (ridimensiona le immagini) e poi utilizza un blocco di apprendimento per classificare nuovi dati.

La classificazione delle immagini è l'uso più comune del deep learning, ma per portare a termine questo compito dovrebbero essere utilizzati molti dati. Abbiamo circa 90 immagini per ogni categoria. Questo numero è sufficiente? No, per niente! Avremo bisogno di migliaia di immagini per “insegnare o modellare” per distinguere una mela da una banana. Ma possiamo risolvere questo problema riaddestrando un modello precedentemente addestrato con migliaia di immagini. Chiamiamo questa tecnica “Transfer Learning” (TL).



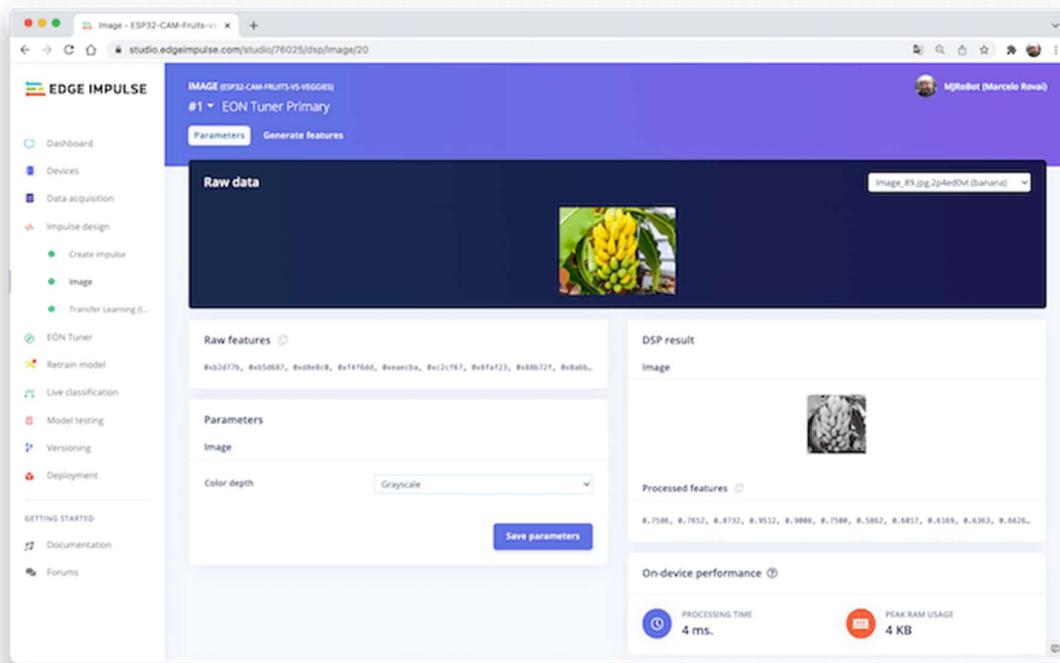
Col TL, possiamo mettere a punto un modello di classificazione delle immagini pre-addestrato sui nostri dati, che funziona bene anche con set di dati di immagini relativamente piccoli (il nostro caso).

Quindi, partendo dalle immagini grezze, le ridimensioneremo (96x96) pixel e le invieremo al nostro blocco Transfer Learning:



Pre-elaborazione (Generazione di Feature)

Oltre a ridimensionare le immagini, possiamo cambiarle in scala di grigi o mantenere la profondità di colore RGB effettiva. Iniziamo selezionando `Grayscale`. In questo modo, ognuno dei nostri campioni di dati avrà dimensione 9.216 feature (96x96x1). Con RGB, questa dimensione sarebbe tre volte più grande. Lavorare con la scala di grigi aiuta a ridurre la quantità di memoria finale necessaria per l'inferenza.



Ricordarsi di [Save parameters]. Questo genererà le feature da utilizzare nel training.

Progettazione del Modello

Transfer Learning

Nel 2007, Google ha introdotto **MobileNetV1**, una famiglia di reti neurali di visione artificiale di uso generale progettate pensando ai dispositivi mobili per supportare classificazione, rilevamento e altro. Le MobileNet sono modelli piccoli, a bassa latenza e a basso consumo, parametrizzati per soddisfare i vincoli di risorse di vari casi d'uso.

Sebbene l'architettura di base di MobileNet sia già minuscola e abbia una bassa latenza, spesso un caso d'uso o un'applicazione specifica potrebbe richiedere che il modello sia più piccolo e veloce. MobileNet introduce un parametro semplice α (alfa) chiamato moltiplicatore di larghezza per costruire questi modelli più piccoli e meno costosi dal punto di vista computazionale. Il ruolo del moltiplicatore di larghezza α è di assottigliare una rete in modo uniforme a ogni livello.

Edge Impulse Studio ha **MobileNet V1 (immagini 96x96)** e **V2 (immagini 96x96 e 160x160)** disponibili, con diversi valori α (da 0,05 a 1,0). Ad esempio, si otterrà la massima accuratezza con V2, immagini 160x160 e $\alpha=1,0$. Naturalmente, c'è un compromesso. Maggiore è la precisione, maggiore sarà la memoria (circa 1,3 M di RAM e 2,6 M di ROM) necessaria per eseguire il modello, il che implica una maggiore latenza.

L'ingombro più piccolo sarà ottenuto all'altro estremo con **MobileNet V1** e $\alpha=0,10$ (circa 53,2 K di RAM e 101 K di ROM).

Per questo primo passaggio, utilizzeremo **MobileNet V1** e $\alpha=0,10$.

Training

Data Augmentation

Un'altra tecnica necessaria da usare con il deep learning è il **data augmentation**. Il data augmentation è un metodo che può aiutare a migliorare l'accuratezza dei modelli di machine learning, creando dati artificiali aggiuntivi. Un sistema di Data Augmentation apporta piccole modifiche casuali ai dati di training (ad esempio capovolgendo, ritagliando o ruotando le immagini).

Sotto, qui si può vedere come Edge Impulse implementa una policy di data Augmentation sui dati:

```
# Implements the data augmentation policy
def augment_image(image, label):
    # Flips the image randomly
    image = tf.image.random_flip_left_right(image)

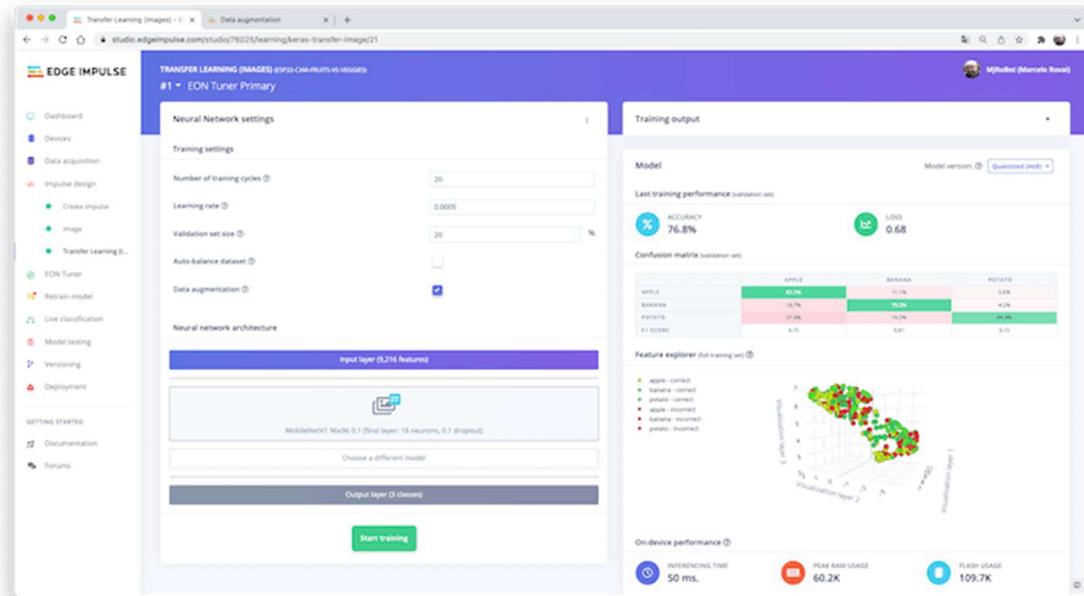
    # Increase the image size, then randomly crop it down to
    # the original dimensions
    resize_factor = random.uniform(1, 1.2)
    new_height = math.floor(resize_factor * INPUT_SHAPE[0])
    new_width = math.floor(resize_factor * INPUT_SHAPE[1])
    image = tf.image.resize_with_crop_or_pad(image, new_height, new_width)
    image = tf.image.random_crop(image, size=INPUT_SHAPE)

    # Vary the brightness of the image
    image = tf.image.random_brightness(image, max_delta=0.2)

    return image, label
```

L'esposizione a queste variazioni durante l'addestramento può aiutare a impedire al modello di prendere scorciatoie "memorizzando" indizi superficiali nei dati di addestramento, il che significa che potrebbe riflettere meglio i pattern profondi in esame nel set di dati.

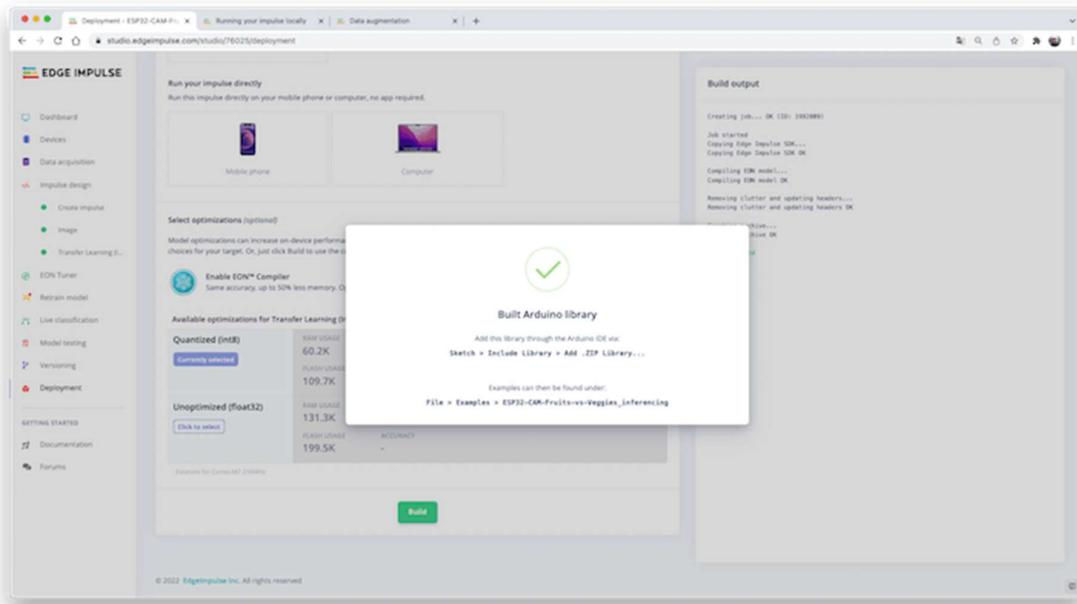
L'ultimo layer del nostro modello avrà 16 neuroni con un dropout del 10% per prevenire l'overfitting. Ecco l'output del Training:



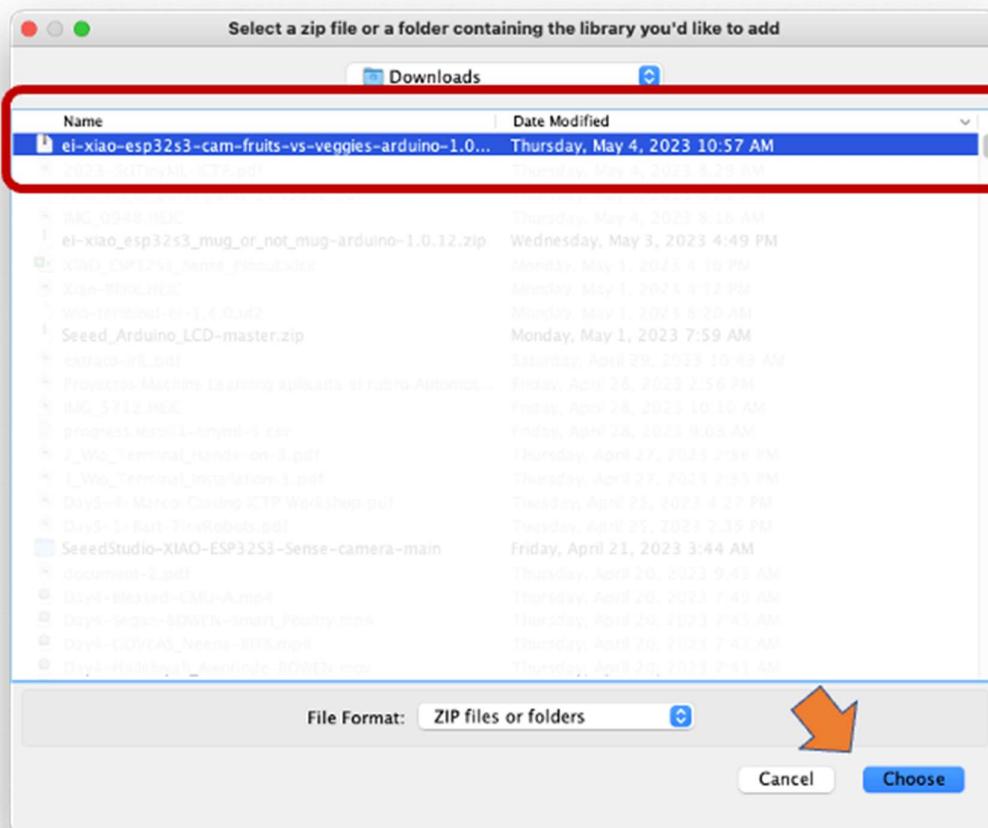
Il risultato potrebbe essere migliore. Il modello ha raggiunto circa il 77% di accuratezza, ma la quantità di RAM prevista per essere utilizzata durante l'inferenza è relativamente piccola (circa 60 KByte), il che è molto buono.

Deployment

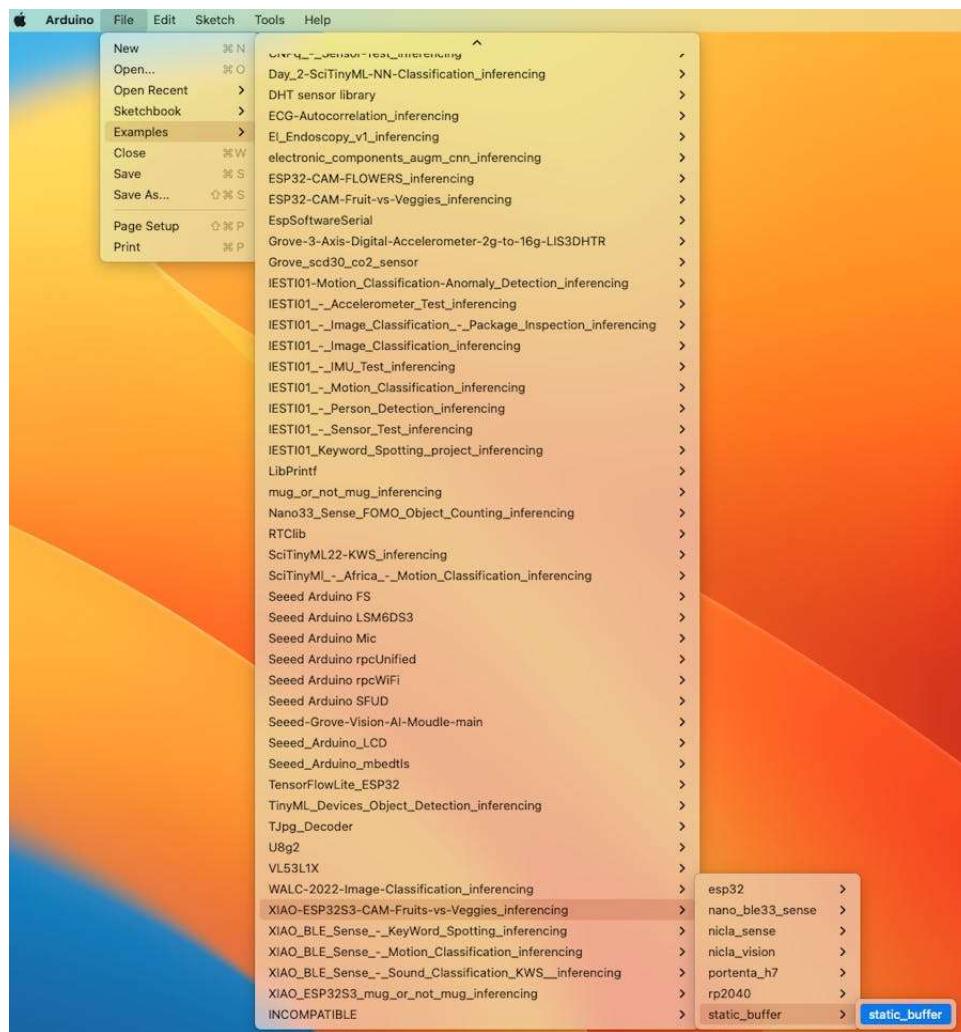
Il modello addestrato verrà distribuito come libreria Arduino .zip:



Apri l'IDE Arduino e in **Sketch**, si va su **Include Library** e **add.ZIP Library**. Selezionare il file che scaricato da Edge Impulse Studio e il gioco è fatto!



Nella scheda **Examples** su Arduino IDE, si trova un codice sketch sotto il nome del proprio progetto.



Aprire l'esempio Static Buffer:

The screenshot shows the Arduino IDE code editor with the 'static_buffer' sketch loaded. The code is as follows:

```

static const float features[] = {
    // copy raw features here (for example from the 'Live classification' page)
    // see https://docs.edgeimpulse.com/docs/running-your-impulse-arduino
};

/* @brief      Copy raw feature data in out_ptr
 * Function called by inference library
 */
/* @param[in]  offset   The offset
 * @param[in]  length   The length
 * @param     out_ptr  The out pointer
 */
/* @return      0
 */

```

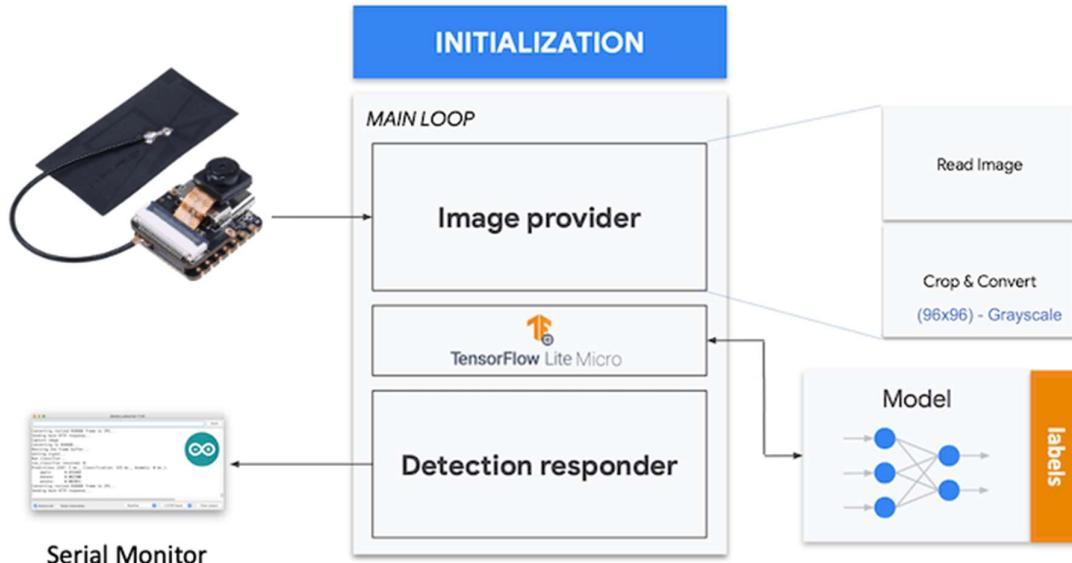
The status bar at the bottom of the IDE shows: Mb, Core 1, Core 1, Hardware CDC and JTAC, Enabled, Disabled, UART0 / Hardware CDC, Default with spiffs (3MB APP/1.5MB SPIFFS), 240MHz (WiFi), 921600, None, Disabled on /dev/cu.usbmodem1101

La prima riga di codice è esattamente la chiamata di una libreria con tutto il necessario per eseguire l'inferenza sul dispositivo.

```
#include <XIAO-ESP32S3-CAM-Fruits-vs-Veggies_inferencing.h>
```

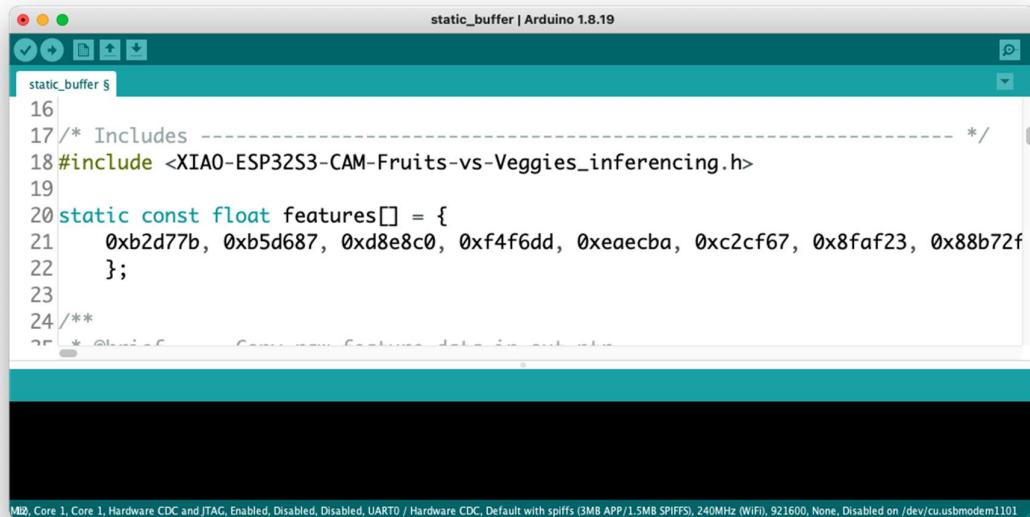
Ovviamente, questo è un codice generico (un “template”) che ottiene solo un campione di dati grezzi (memorizzati nella variabile: `features = {}`) ed esegue il classificatore, eseguendo l’inferenza. Il risultato viene mostrato sul monitor seriale.

Dovremmo ottenere il campione (immagine) dalla fotocamera e pre-elaborarlo (ridimensionandolo a 96x96, convertendolo in scala di grigi e appiattendolo). Questo sarà il tensore di input del nostro modello. Il tensore di output sarà un vettore con tre valori (etichette), che mostrano le probabilità di ciascuna delle classi.



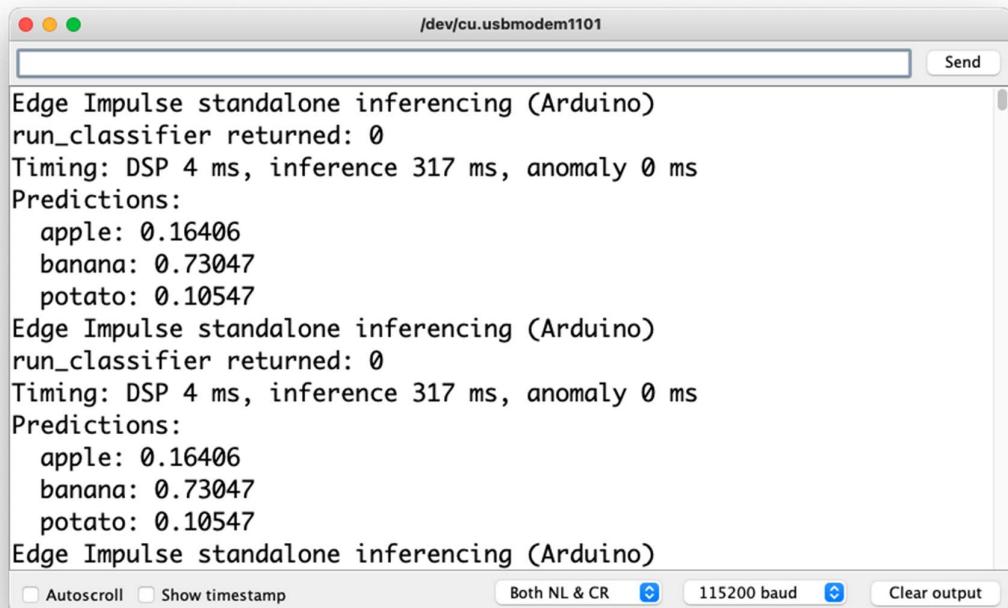
Tornando al progetto (Tab Image), copiare uno dei Raw Data Sample:

9.216 feature verranno copiate negli appunti. Questo è il tensore di input (un’immagine appiattita di 96x96x1), in questo caso, banane. “Incollare” questo tensore di input su `features[] = {0xb2d77b, 0xb5d687, 0xd8e8c0, 0xeaecba, 0xc2cf67, ...}`



The screenshot shows the Arduino IDE interface with the sketch named "static_buffer". The code includes an include statement for "XIAO-ESP32S3-CAM-Fruits-vs-Veggies_inferencing.h" and defines a static const float array "features" containing eight values. The serial monitor at the bottom shows the board configuration: M0, Core 1, Core 1, Hardware CDC and JTAG, Enabled, Disabled, UART0 / Hardware CDC, Default with spiffs (3MB APP/1.5MB SPIFFS), 240MHz (WiFi), 921600, None, Disabled on /dev/cu.usbmodem1101.

Edge Impulse ha incluso la [libreria ESP NN](#) nel suo SDK, che contiene funzioni NN (Neural Network) ottimizzate per vari chip Espressif, tra cui ESP32S3 (in esecuzione su Arduino IDE). Quando si esegue l'inferenza, si deve ottenere il punteggio più alto per “banana”.



The screenshot shows the serial monitor output from the Edge Impulse standalone inferencing (Arduino) sketch. It displays three separate inference runs. Each run shows the classifier returned value (0), timing (DSP 4 ms, inference 317 ms, anomaly 0 ms), and predictions for three classes: apple, banana, and potato. The banana prediction is consistently the highest across all three runs.

```
Edge Impulse standalone inferencing (Arduino)
run_classifier returned: 0
Timing: DSP 4 ms, inference 317 ms, anomaly 0 ms
Predictions:
apple: 0.16406
banana: 0.73047
potato: 0.10547
Edge Impulse standalone inferencing (Arduino)
run_classifier returned: 0
Timing: DSP 4 ms, inference 317 ms, anomaly 0 ms
Predictions:
apple: 0.16406
banana: 0.73047
potato: 0.10547
Edge Impulse standalone inferencing (Arduino)
```

Ottime notizie! Il nostro dispositivo gestisce un'inferenza, scoprendo che l'immagine in ingresso è una banana. Notare inoltre che il tempo di inferenza è stato di circa 317 ms, con un massimo di 3 fps se si è provato a classificare le immagini da un video. Ora, dovremmo incorporare la telecamera e classificare le immagini in tempo reale. Si va su Arduino IDE Examples e si scarica dal progetto lo sketch `esp32_camera`:



Si devono cambiare le righe dalla 32 alla 75, che definiscono il modello e i pin della telecamera, utilizzando i dati relativi al nostro modello. Copiare e incollare le righe seguenti, sostituendo le righe 32-75:

```
#define PWDN_GPIO_NUM      -1
#define RESET_GPIO_NUM     -1
#define XCLK_GPIO_NUM       10
#define SIOD_GPIO_NUM        40
#define SIOC_GPIO_NUM        39
#define Y9_GPIO_NUM          48
#define Y8_GPIO_NUM          11
#define Y7_GPIO_NUM          12
#define Y6_GPIO_NUM          14
#define Y5_GPIO_NUM          16
#define Y4_GPIO_NUM          18
#define Y3_GPIO_NUM          17
#define Y2_GPIO_NUM          15
#define VSYNC_GPIO_NUM        38
#define HREF_GPIO_NUM         47
#define PCLK_GPIO_NUM         13
```

Ecco il codice risultante:

The screenshot shows the Arduino IDE with the sketch titled 'esp32_camera'. The code has been modified to include the new camera model and pin definitions. A red box highlights the block of code from line 32 to line 75, which corresponds to the copied text above. The rest of the code remains the same, including the camera selection and pin definitions.

```
esp32_camera | Arduino 1.8.19

23 /* Includes */
24 #include <XIAO-ESP32S3-CAM-Fruits-vs-Veggies_inferencing.h>
25 #include "edge-impulse-sdk/dsp/image/image.hpp"
26
27 #include "esp_camera.h"
28
29 // Select camera model - find more camera models in camera_pins.h file here
30 // https://github.com/espressif/arduino-esp32/blob/master/libraries/ESP32/examples/Camera
31
32 #define CAMERA_MODEL_XIAO_ESP32S3 // Has PSRAM
33
34 #define PWDN_GPIO_NUM      -1
35 #define RESET_GPIO_NUM     -1
36 #define XCLK_GPIO_NUM       10
37 #define SIOD_GPIO_NUM        40
38 #define SIOC_GPIO_NUM        39
39
40 #define Y9_GPIO_NUM          48
41 #define Y8_GPIO_NUM          11
42 #define Y7_GPIO_NUM          12
43 #define Y6_GPIO_NUM          14
44 #define Y5_GPIO_NUM          16
45 #define Y4_GPIO_NUM          18
46 #define Y3_GPIO_NUM          17
47 #define Y2_GPIO_NUM          15
48 #define VSYNC_GPIO_NUM        38
49 #define HREF_GPIO_NUM         47
50 #define PCLK_GPIO_NUM         13
51
52 #define LED_GPIO_NUM        21
```

MB, Core 1, Core 1, Hardware CDC and I2C, Enabled, Disabled, Disabled, UART0 / Hardware CDC, Default with spiFFS (3MB APP/1.5MB SPIFFS), 240MHz WiFi, 921600, None, Disabled on /dev/cu.usbmodem1101

Lo sketch modificato è scaricabile da GitHub: [xiao_esp32s3_camera](#).

Notare che si possono facoltativamente mantenere i pin come file .h come abbiamo fatto nel Setup Lab.

Caricare il codice sullo XIAO ESP32S3 Sense e si sarà pronti per iniziare a classificare la frutta e la verdura! Si può controllare il risultato su Serial Monitor.

Test del modello (inferenza)



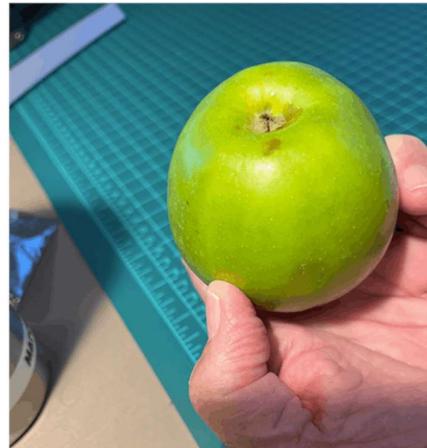
Scattando una foto con la fotocamera, il risultato della classificazione apparirà su Serial Monitor:

```
banana: 0.90234
potato: 0.03906
Predictions (DSP: 4 ms., Classification: 318 ms., Anomaly: 0 ms.):
apple: 0.03906
banana: 0.93359
potato: 0.02734
Predictions (DSP: 4 ms., Classification: 317 ms., Anomaly: 0 ms.):
apple: 0.05469
banana: 0.90625
potato: 0.03906
Predictions (DSP: 4 ms., Classification: 318 ms., Anomaly: 0 ms.):
apple: 0.04297
banana: 0.92578
potato: 0.03125
```

The screenshot shows a terminal window titled "/dev/cu.usbmodem1101". The window displays three sets of classification predictions for different objects (apple, banana, potato) across three separate DSP operations. The results are presented as key-value pairs where the key is the object name and the value is a confidence score. The terminal also includes standard control buttons (red, yellow, green circles) and a "Send" button at the top right. At the bottom, there are checkboxes for "Autoscroll" and "Show timestamp", and a dropdown menu for baud rate set to "115200 baud". A "Clear output" button is also visible.

Altri test:

```
/dev/cu.usbmodem1101
Send
banana: 0.14844
potato: 0.12891
Predictions (DSP: 4 ms., Classification: 318 ms., Anomaly: 0 ms.):
apple: 0.78906
banana: 0.06641
potato: 0.14453
Predictions (DSP: 4 ms., Classification: 317 ms., Anomaly: 0 ms.):
apple: 0.71484
banana: 0.06641
potato: 0.21875
Predictions (DSP: 4 ms., Classification: 318 ms., Anomaly: 0 ms.):
apple: 0.79297
banana: 0.05469
potato: 0.14844
Autoscroll Show timestamp Both NL & CR 115200 baud Clear output
```

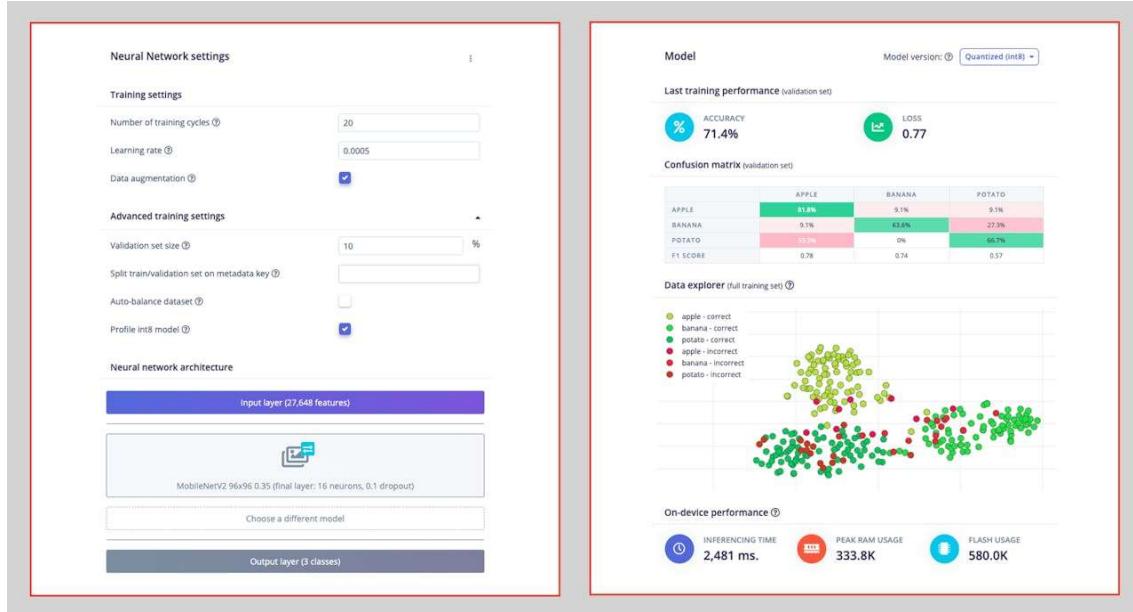


```
/dev/cu.usbmodem1101
Send
banana: 0.03125
potato: 0.79688
Predictions (DSP: 4 ms., Classification: 318 ms., Anomaly: 0 ms.):
apple: 0.32812
banana: 0.03906
potato: 0.63281
Predictions (DSP: 4 ms., Classification: 318 ms., Anomaly: 0 ms.):
apple: 0.40625
banana: 0.05469
potato: 0.53906
Predictions (DSP: 4 ms., Classification: 318 ms., Anomaly: 0 ms.):
apple: 0.16406
banana: 0.02344
potato: 0.81250
Autoscroll Show timestamp Both NL & CR 115200 baud Clear output
```



Test con un Modello Più Grande

Ora, passiamo all'altro lato delle dimensioni del modello. Selezioniamo un MobilinetV2 96x96 0.35, con immagini RGB in input.



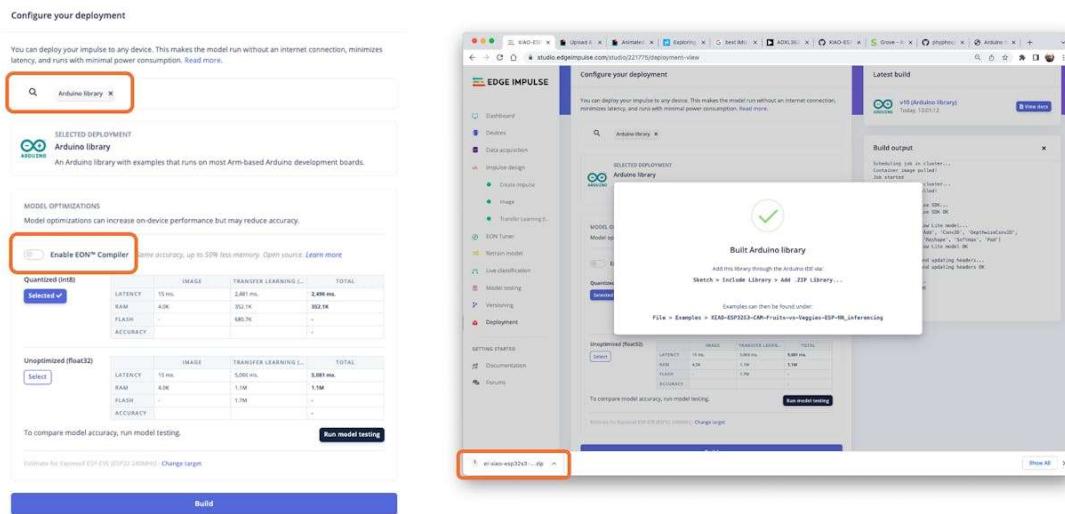
Anche con un modello più grande, la precisione potrebbe essere migliore e la quantità di memoria necessaria per eseguire il modello aumenta di cinque volte, con una latenza che aumenta di sette volte.

Notare che le prestazioni qui sono stimate con un dispositivo più piccolo, l'ESP-EYE. L'inferenza effettiva con l'ESP32S3 dovrebbe essere migliore.

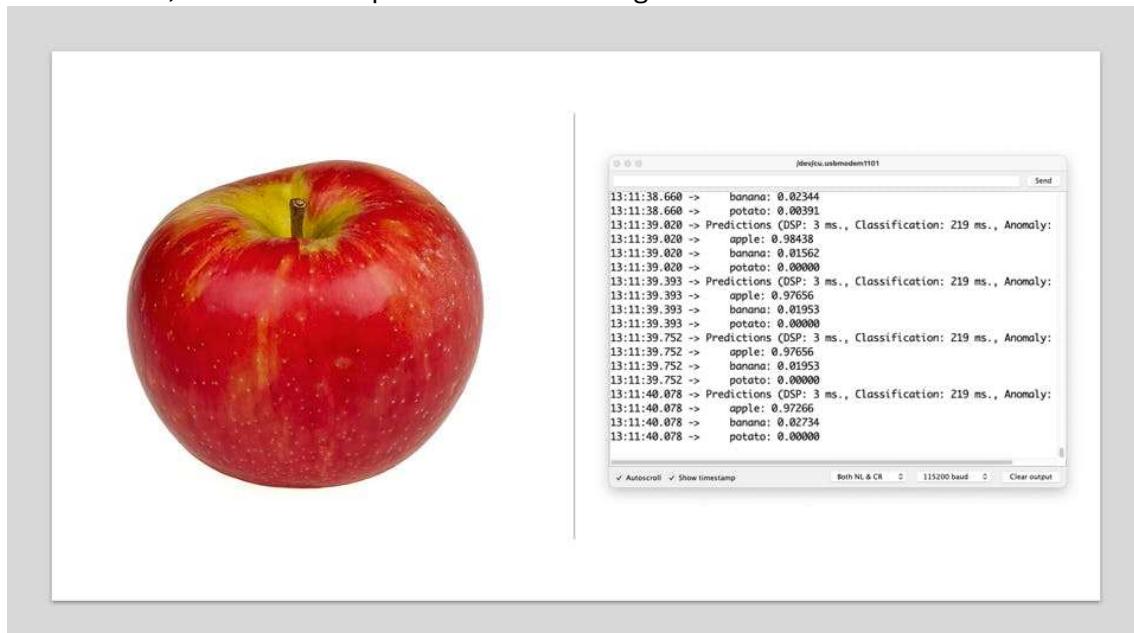
Per migliorare il nostro modello, dovremo addestrare più immagini.

Anche se il nostro modello non ha migliorato la precisione, testiamo se l'XIAO può gestire un modello così grande. Faremo un semplice test di inferenza con lo sketch Static Buffer. Ridistribuiamo il modello. Se il compilatore EON è abilitato quando si genera la libreria, la memoria totale necessaria per l'inferenza dovrebbe essere ridotta, ma ciò non influisce sulla precisione.

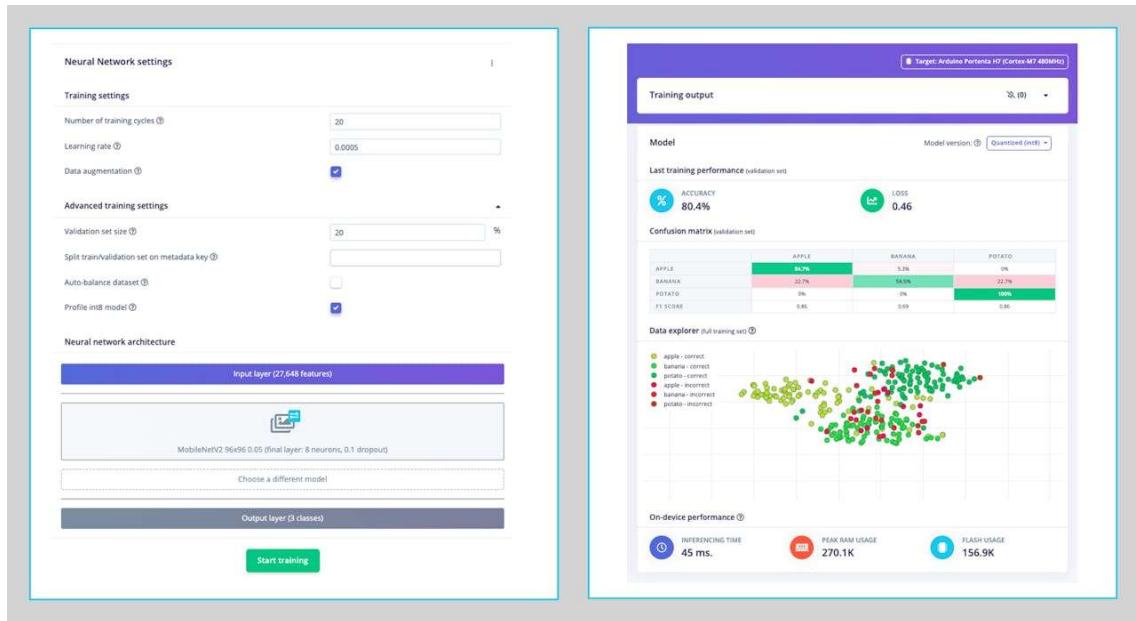
⚠️ Attenzione - Xiao ESP32S3 con PSRAM abilitata ha memoria sufficiente per eseguire l'inferenza, anche in un modello così grande. Mantenere il Compilatore EON NOT ENABLED.



Facendo un'inferenza con MobilinetV2 96x96 0.35, con immagini RGB in input, la latenza è stata di 219 ms, il che è ottimo per un modello così grande.



Per il test, possiamo addestrare di nuovo il modello, usando la versione più piccola di MobileNet V2, con un alpha di 0,05. È interessante che il risultato in accuratezza sia stato più alto.



Notare che la latenza stimata per un Arduino Portenta (o Nicla), in esecuzione con un clock di 480 MHz è di 45 ms.

Distribuendo il modello, abbiamo ottenuto un'inferenza di soli 135 ms, ricordando che XIAO funziona con metà del clock utilizzato da Portenta/Nicla (240 MHz):

```
/dev/cu.usbmodem1101
Send
10:44:47.849 ->    banana: 0.01953
10:44:47.849 ->    potato: 0.12891
10:44:48.103 -> Predictions (DSP: 3 ms., Classification: 135 ms., Anomaly: 0 ms.):
10:44:48.103 ->    apple: 0.86328
10:44:48.103 ->    banana: 0.03906
10:44:48.103 ->    potato: 0.10156
10:44:48.356 -> Predictions (DSP: 3 ms., Classification: 135 ms., Anomaly: 0 ms.):
10:44:48.356 ->    apple: 0.90234
10:44:48.356 ->    banana: 0.02344
10:44:48.356 ->    potato: 0.07422
10:44:48.612 -> Predictions (DSP: 3 ms., Classification: 135 ms., Anomaly: 0 ms.):
10:44:48.612 ->    apple: 0.91797
10:44:48.612 ->    banana: 0.02344
10:44:48.612 ->    potato: 0.05859
10:44:48.861 -> Predictions (DSP: 3 ms., Classification: 135 ms., Anomaly: 0 ms.):
10:44:48.861 ->    apple: 0.88281
10:44:48.861 ->    banana: 0.03516
10:44:48.861 ->    potato: 0.08203
10:44:49.114 -> Predictions (DSP: 3 ms., Classification: 135 ms., Anomaly: 0 ms.):

```

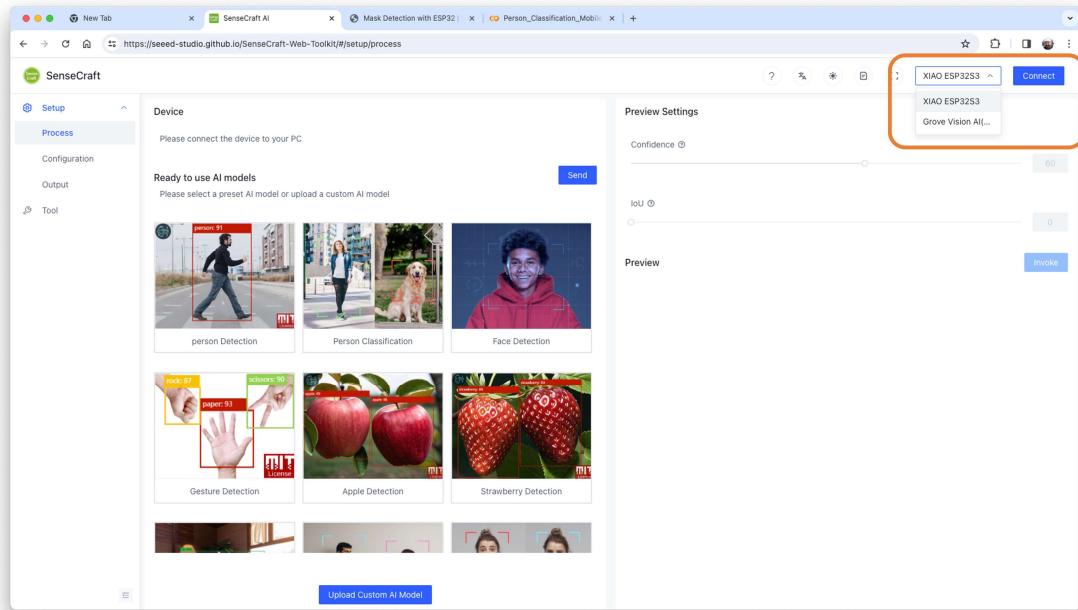
✓ Autoscroll ✓ Show timestamp Both NL & CR 115200 baud Clear output

Esecuzione dell'inferenza su SenseCraft-Web-Toolkit

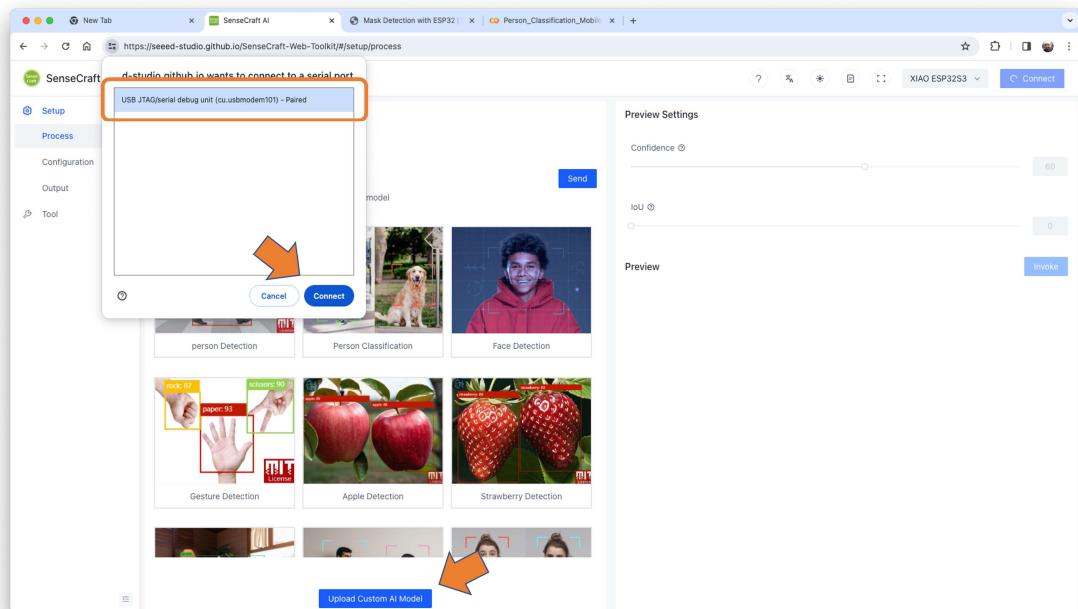
Una limitazione significativa della visualizzazione dell'inferenza su Arduino IDE è che non possiamo vedere su cosa punta la telecamera. Una buona alternativa è **SenseCraft-Web-Toolkit**, uno strumento di distribuzione del modello visivo fornito da **SSCMA**(Seeed SenseCraft Model Assistant). Questo strumento consente di distribuire facilmente modelli su varie piattaforme tramite semplici operazioni. Lo strumento offre un'interfaccia intuitiva e non richiede alcuna codifica.

Seguire i seguenti passaggi per avviare SenseCraft-Web-Toolkit:

1. Aprire il sito web di SenseCraft-Web-Toolkit.
2. Collega XIAO al computer:
- Dopo aver collegato XIAO, selezionarlo come di seguito:



- Selezionare il dispositivo/Porta e premere [Connect]:



Si possono provare diversi modelli di Computer Vision caricati in precedenza da Seeed Studio. Da provare e verificarli!

Nel nostro caso, useremo il pulsante blu in fondo alla pagina: [Upload Custom AI Model]. Ma prima, dobbiamo scaricare da Edge Impulse Studio il modello **quantized.tflite**.

3. Si va sul proprio progetto su Edge Impulse Studio, oppure si clona questo:
- [XIAO-ESP32S3-CAM-Fruits-vs-Veggies-v1-ESP-NN](#)
4. Nella Dashboard, scaricare il modello (“block output”): Transfer learning model - TensorFlow Lite (int8 quantized).

Download block output

TITLE	TYPE	SIZE
Image training data	NPY file	279 windows
Image training labels	NPY file	279 windows
Image testing data	NPY file	15 windows
Image testing labels	NPY file	15 windows
Transfer learning model	TensorFlow Lite (float32)	245 KB
Transfer learning model	TensorFlow Lite (int8 quantized)	171 KB
Transfer learning model	TensorFlow SavedModel	314 KB
Transfer learning model	Keras h5 model	248 KB

Collaborators (1/4)

MJRobot (Marcelo Rovai) OWNER

Summary

DEVICES CONNECTED 0

DATA COLLECTED 294 items

Project info

5. Su SenseCraft-Web-Toolkit, usare il pulsante blu in fondo alla pagina: [Upload Custom AI Model]. Si aprirà una finestra. Inserire il file del Modello scaricato sul computer da Edge Impulse Studio, scegliere un nome del modello e inserirlo con le etichette (ID: Object):

Custom AI Model

Model Name: Fruits_vs_Veggies_Img Classification

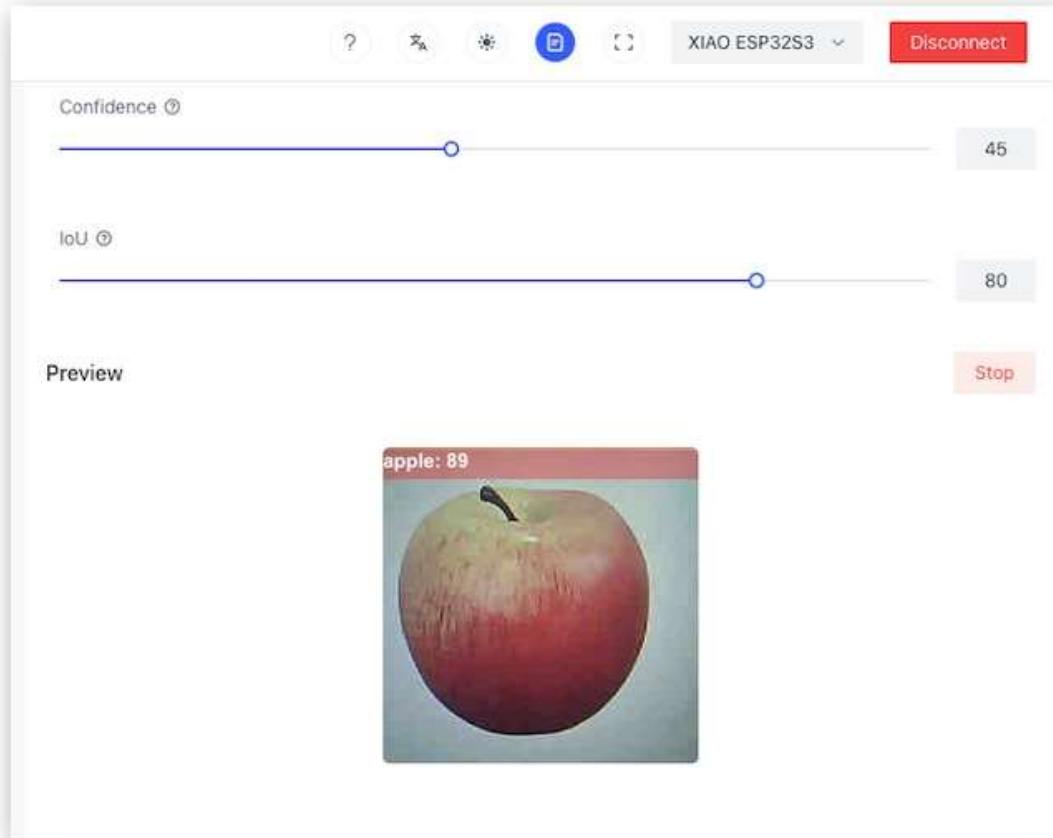
Model File: fruits-vs-veggies-v1-esp-nn-transfer...

ID Object: apple X banana X potato X

Cancel Send Model

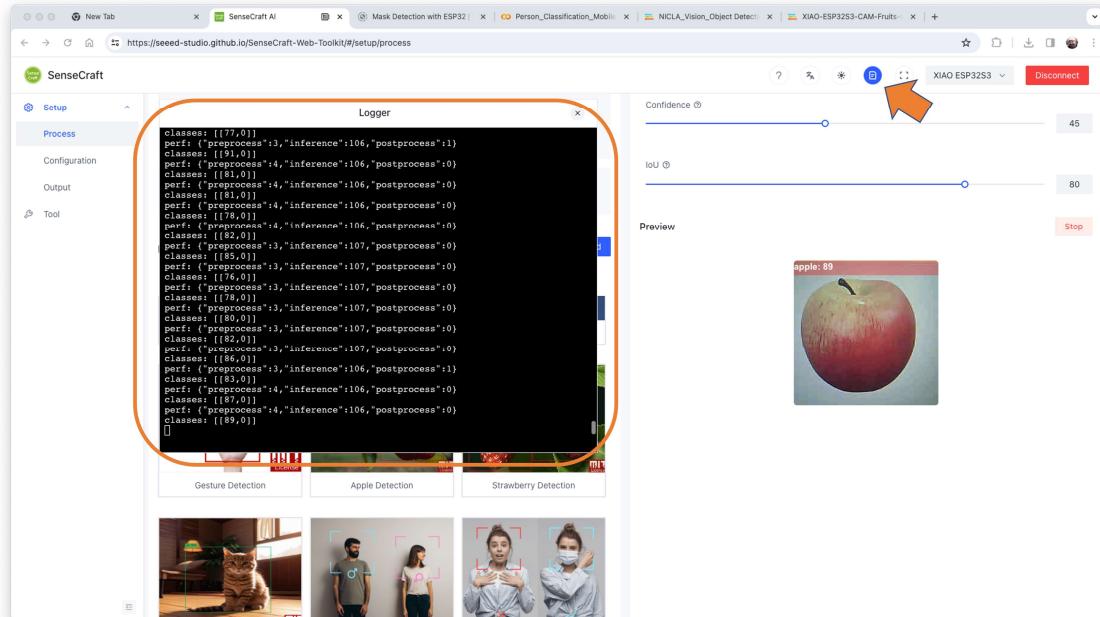
Notare che si devono usare le etichette addestrate su EI Studio, inserendole in ordine alfabetico (nel nostro caso: apple, banana, potato).

Dopo alcuni secondi (o minuti), il modello verrà caricato sul dispositivo e l'immagine della telecamera apparirà in tempo reale nel Preview Sector:



Il risultato della classificazione sarà in cima all'immagine. Si può anche selezionare la "Confidence" del cursore di inferenza Confidence.

Cliccando sul pulsante in alto (Device Log), si può aprire un Serial Monitor per seguire l'inferenza, come fatto con l'IDE Arduino:



Su Device Log, si otterranno informazioni come:

```
perf: {"preprocess":4,"inference":106,"postprocess":0}
classes: [[89,0]]
[]
```

- Tempo di pre-elaborazione (cattura immagine e Crop): 4ms,
- Tempo di inferenza (latenza modello): 106ms,
- Tempo di Post-elaborazione (visualizzazione dell'immagine e inclusione dei dati): 0ms,
- Tensore di output (classi), ad esempio: [[89,0]]; dove 0 è Apple (1 è banana e 2 è patata).

Ecco altri screenshot:



Conclusione

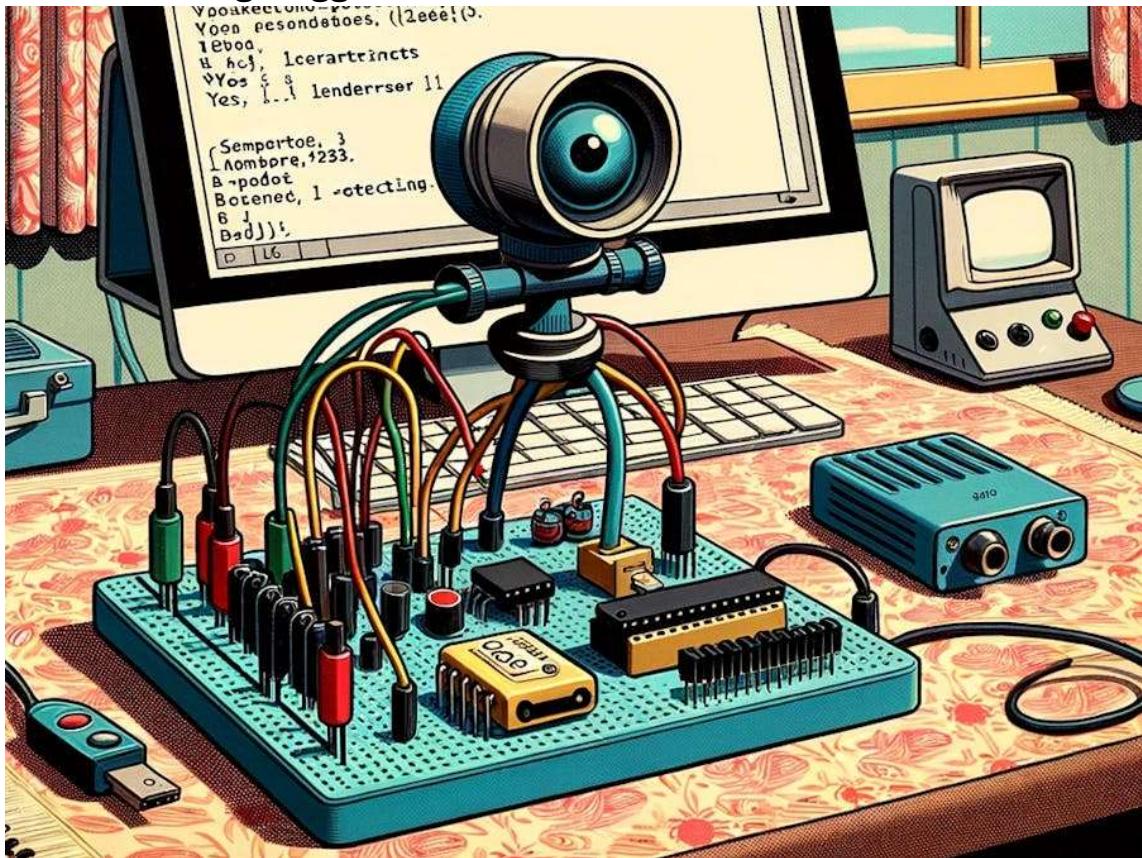
XIAO ESP32S3 Sense è molto flessibile, poco costoso e facile da programmare. Il progetto dimostra il potenziale di TinyML. La memoria non è un problema; il dispositivo può gestire molte attività di post-elaborazione, tra cui la comunicazione.

L'ultima versione del codice si trova nel repository GitHub: [XIAO-ESP32S3-Sense](#).

Risorse

- [Codici XIAO ESP32S3](#)
- [Dataset](#)
- [Progetto Edge Impulse](#)

Rilevamento degli Oggetti



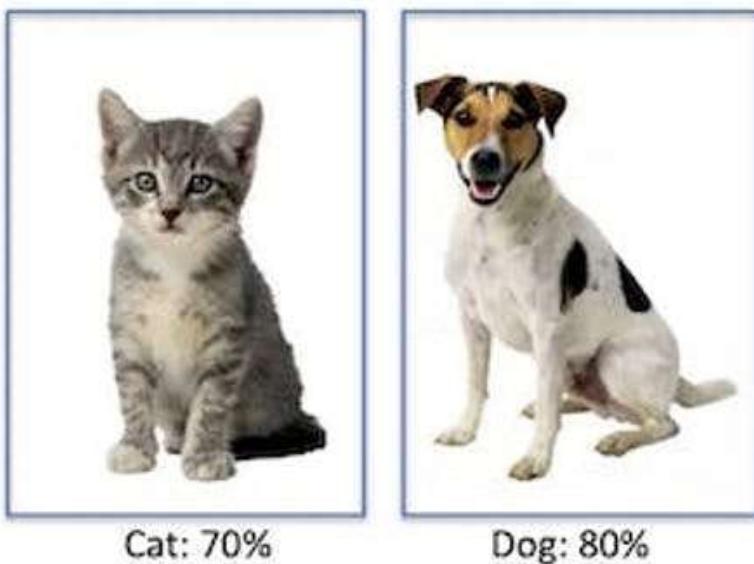
DALL-E prompt - Fumetto in stile anni '50, che mostra una scheda dettagliata con sensori, in particolare una telecamera, su un tavolo con un panno fantasia. Dietro la scheda, un computer con un ampio retro mostra l'IDE Arduino. Il contenuto dell'IDE accenna alle assegnazioni dei pin LED e all'inferenza di apprendimento automatico per rilevare i comandi vocali. Il monitor seriale, in una finestra distinta, rivela gli output per i comandi 'yes' e 'no'.

Panoramica

Nell'ultima sezione riguardante Computer Vision (CV) e XIAO ESP32S3, *Classificazione delle immagini*, abbiamo imparato come impostare e classificare le immagini con questa straordinaria scheda di sviluppo. Continuando il nostro viaggio con CV, esploreremo il **Rilevamento degli oggetti** sui microcontrollori.

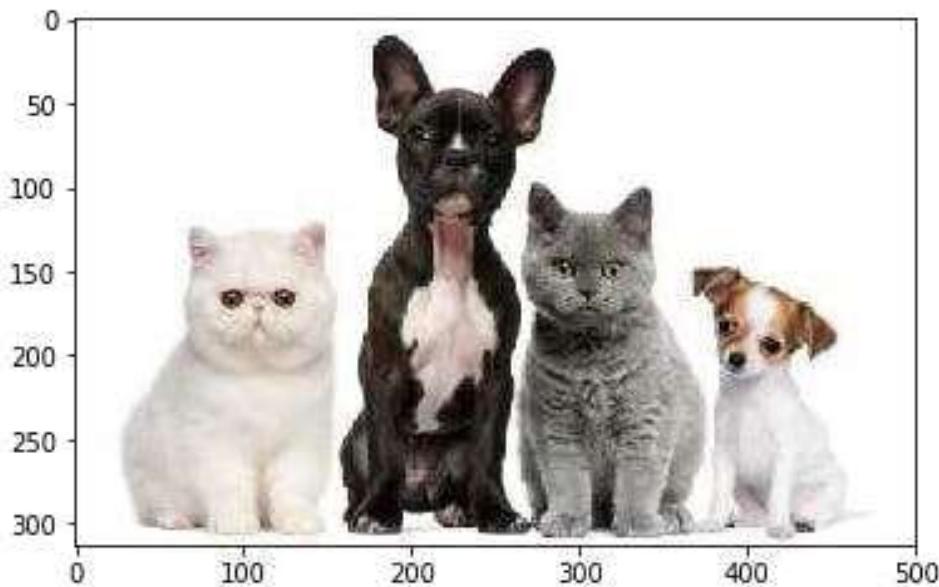
Object Detection e Image Classification

Il compito principale con i modelli di Classificazione delle immagini è identificare la categoria di oggetti più probabile presente su un'immagine, ad esempio, per classificare tra un gatto o un cane, gli "oggetti" dominanti in un'immagine:



Ma cosa succede se non c'è una categoria dominante nell'immagine?

[PREDICTION]	[Prob]
ashcan	: 27%
Egyptian cat	: 19%
hamper	: 13%

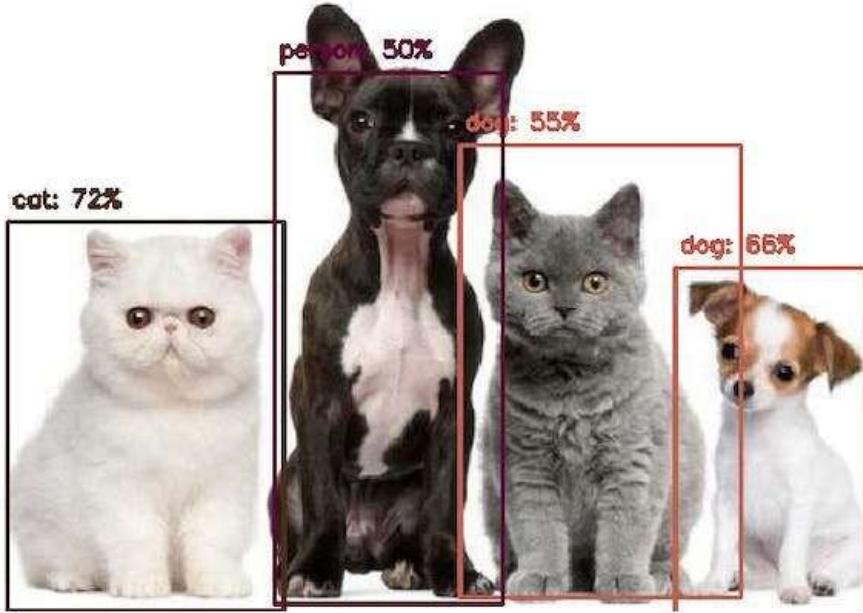


Un modello di classificazione delle immagini identifica l'immagine soprastante in modo completamente sbagliato come un “ashcan”, probabilmente a causa delle tonalità di colore.

Il modello utilizzato nelle immagini precedenti è MobileNet, che è addestrato con un ampio set di dati, *ImageNet*, in esecuzione su un Raspberry Pi.

Per risolvere questo problema, abbiamo bisogno di un altro tipo di modello, in cui non solo possono essere trovate **più categorie** (o etichette), ma anche **dove** si trovano gli oggetti in una determinata immagine.

Come possiamo immaginare, tali modelli sono molto più complicati e più grandi, ad esempio, **MobileNetV2 SSD FPN-Lite 320x320, addestrato con il set di dati COCO**. Questo modello di rilevamento degli oggetti pre-addestrato è progettato per individuare fino a 10 oggetti all'interno di un'immagine, generando un riquadro di delimitazione per ogni oggetto rilevato. L'immagine sottostante è il risultato di un tale modello in esecuzione su un Raspberry Pi:



I modelli utilizzati per il rilevamento di oggetti (come MobileNet SSD o YOLO) hanno solitamente dimensioni di diversi MB, il che è OK per l'uso con Raspberry Pi ma non è adatto per l'uso con dispositivi embedded, dove la RAM di solito ha, al massimo, pochi MB come nel caso di XIAO ESP32S3.

Una soluzione innovativa per il Rilevamento di Oggetti: FOMO

Edge Impulse ha lanciato nel 2022, **FOMO** (Faster Objects, More Objects), una nuova soluzione per eseguire il rilevamento di oggetti su dispositivi embedded, come Nicla Vision e Portenta (Cortex M7), su CPU Cortex M4F (serie Arduino Nano33 e OpenMV M4) e sui dispositivi Espressif ESP32 (ESP-CAM, ESP-EYE e XIAO ESP32S3 Sense).

In questo progetto pratico, esploreremo l'Object Detection utilizzando FOMO.

Per saperne di più sulla FOMO, si può leggere l'[annuncio ufficiale su FOMO](#) di Edge Impulse, dove Louis Moreau e Mat Kelcey spiegano in dettaglio come funziona.

Obiettivo del Progetto di Object Detection

Tutti i progetti di apprendimento automatico devono iniziare con un obiettivo dettagliato. Supponiamo di trovarci in una struttura industriale o rurale e di dover smistare e contare arance (frutti) e in particolare rane (Insetti).



In altre parole, dovremmo eseguire una classificazione multi-etichetta, in cui ogni immagine può avere tre classi:

- Background [Sfondo] (nessun oggetto)
- Fruit
- Bug

Ecco alcuni esempi di immagini non etichettate che dovremmo utilizzare per rilevare gli oggetti (frutti e insetti):



Siamo interessati a quale oggetto è presente nell'immagine, alla sua posizione (centroide) e a quanti ne possiamo trovare su di essa. La dimensione dell'oggetto non viene rilevata con FOMO, come con MobileNet SSD o YOLO, in cui il Bounding Box è uno degli output del modello.

Svilupperemo il progetto utilizzando XIAO ESP32S3 per l'acquisizione di immagini e l'inferenza del modello. Il progetto ML verrà sviluppato utilizzando Edge Impulse Studio. Ma prima di iniziare il progetto di "object detection" in Studio, creiamo un *dataset grezzo* (non etichettato) con immagini che contengono gli oggetti da rilevare.

Raccolta Dati

Si possono catturare immagini usando XIAO, il telefono o altri dispositivi. Qui, useremo XIAO con codice dalla libreria Arduino IDE ESP32.

Raccolta di Dataset con XIAO ESP32S3

Aprire Arduino IDE e selezionare la scheda XIAO_ESP32S3 (e la porta a cui è collegata). Su File > Examples > ESP32 > Camera, selezionare CameraWebServer.

Nel pannello BOARDS MANAGER, confermare di aver installato l'ultimo pacchetto "stable".

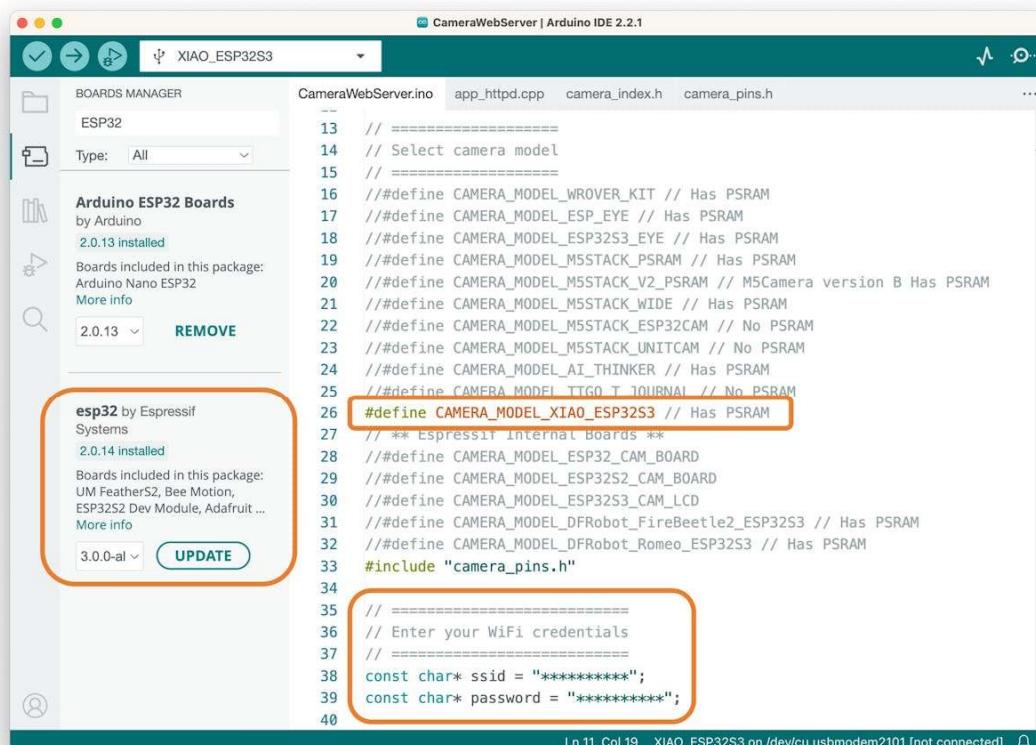
Attenzione

Le versioni Alpha (ad esempio, 3.x-alpha) non funzionano correttamente con XIAO ed Edge Impulse. Utilizzare invece l'ultima versione stabile (ad esempio, 2.0.11).

Si devono anche commentare tutti i modelli di fotocamere, eccetto i pin del modello XIAO:

```
#define CAMERA_MODEL_XIAO_ESP32S3 // Has PSRAM
```

E su Tools, abilitare la PSRAM. Inserisci le credenziali wifi e caricare il codice sul dispositivo:

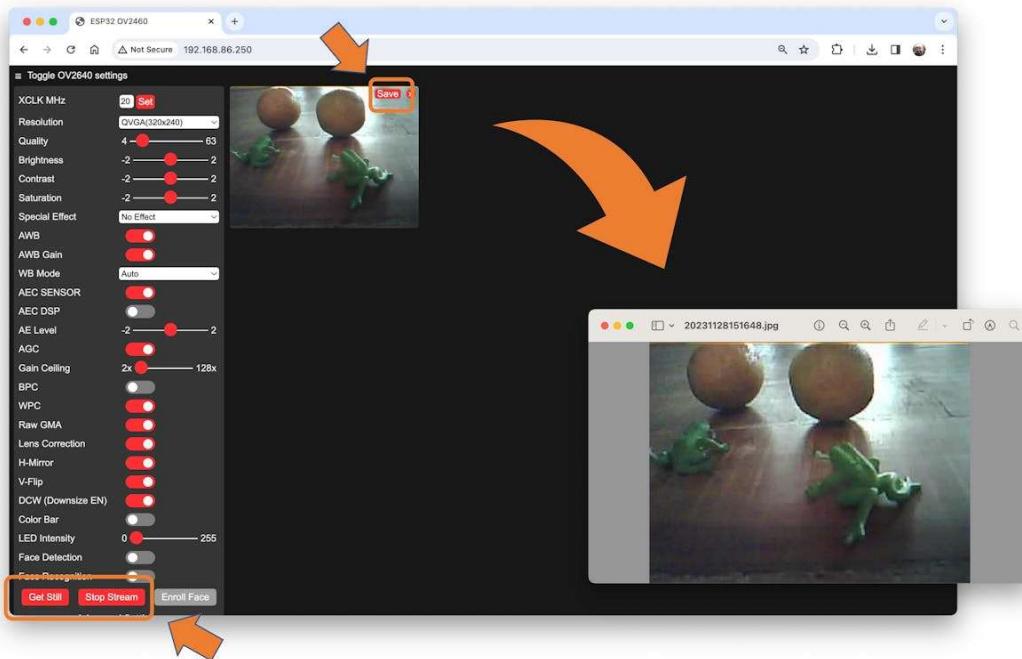


Se il codice viene eseguito correttamente, si vedrà l'indirizzo sul monitor seriale:



Copiare l'indirizzo sul browser e attendere che la pagina venga caricata. Selezionare la risoluzione della telecamera (ad esempio, QVGA) e selezionare [START STREAM]. Attendi

qualche secondo/minuto, a seconda della connessione. Si può salvare un'immagine nell'area download del computer usando il pulsante [Save].



Edge impulse suggerisce che gli oggetti dovrebbero essere simili per dimensione e non sovrapposti per prestazioni migliori. Questo va bene in una struttura industriale, dove la telecamera dovrebbe essere fissa, mantenendo la stessa distanza dagli oggetti da rilevare. Nonostante ciò, proveremo anche a usare dimensioni e posizioni miste per vedere il risultato.

Non abbiamo bisogno di creare cartelle separate per le nostre immagini perché ognuna contiene più etichette.

Suggeriamo di usare circa 50 immagini per mescolare gli oggetti e variare il numero di ciascuno che appare sulla scena. Provare ad acquisire con diverse angolazioni, sfondi e condizioni di luce.

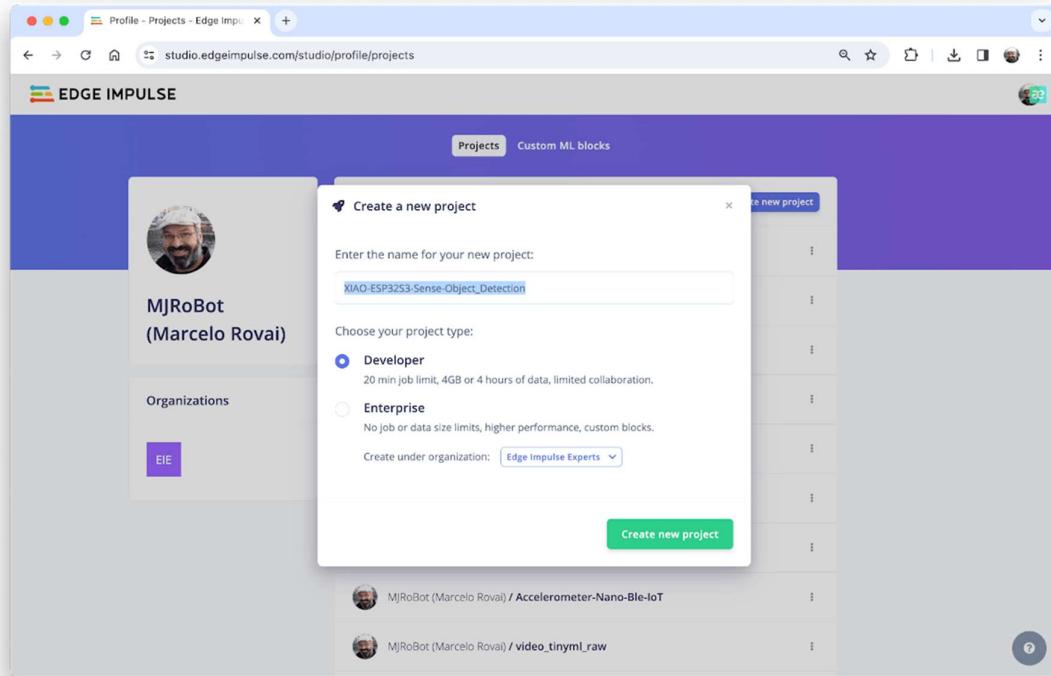
Le immagini archiviate usano una dimensione del frame QVGA di 320x240 e RGB565 (formato pixel colore).

Dopo aver acquisito il dataset, `[Stop Stream]` e spostare le immagini in una cartella.

Edge Impulse Studio

Setup del progetto

Si va su [Edge Impulse Studio](#), si inseriscono le proprie credenziali in **Login** (o si crea un account) e si avvia un nuovo progetto.

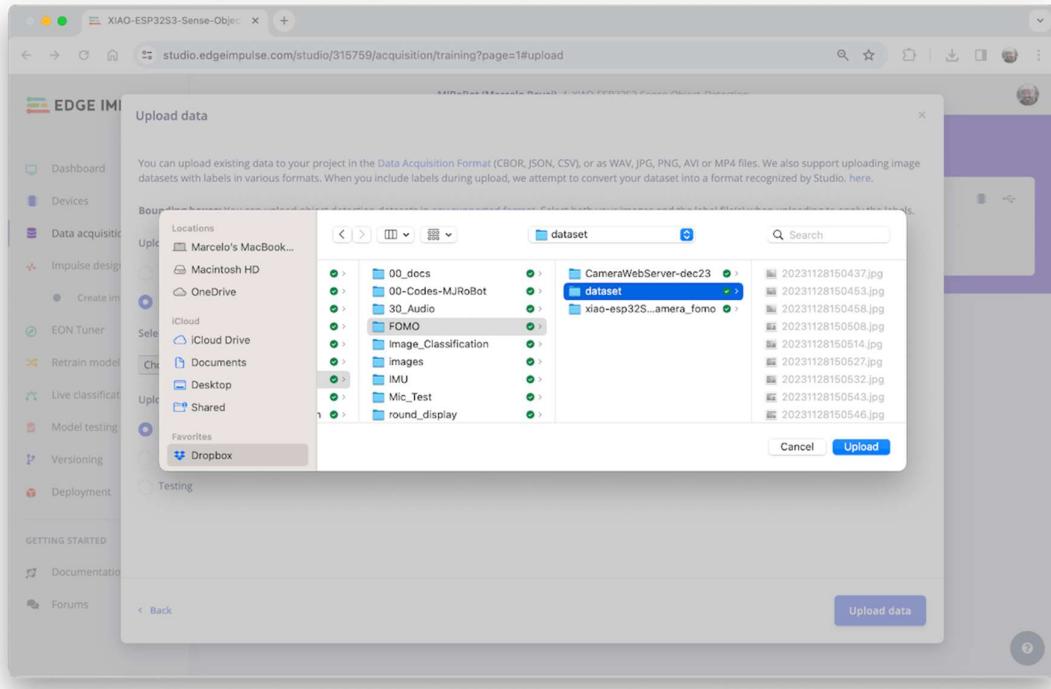


Qui, è possibile clonare il progetto sviluppato per questa esercitazione pratica:
[XIAO-ESP32S3-Sense-Object_Detection](#)

Nella dashboard del progetto, andare in basso e su **Project info** e selezionare **Bounding boxes (object detection)** e **Espressif ESP-EYE** (il più simile alla nostra scheda) come Target Device:

Caricamento dei dati non etichettati

In Studio, si va alla scheda **Data acquisition** e nella sezione **UPLOAD DATA** caricare i file acquisiti come cartella dal computer.



Si può lasciare che Studio divida automaticamente i dati tra “Train” e “Test” o farlo manualmente. Caricheremo tutti come training.

SAMPLE NAME	LABELS	ADDED	LENGTH
20231128151645	-	Today, 15:27:09	-
20231128150613	-	Today, 15:27:09	-
20231128150604	-	Today, 15:27:09	-
20231128150833	-	Today, 15:27:09	-
20231128150600	-	Today, 15:27:09	-
20231128150855	-	Today, 15:27:09	-
20231128150458	-	Today, 15:27:09	-
20231128150713	-	Today, 15:27:09	-
20231128150908	-	Today, 15:27:09	-

Tutte le immagini non etichettate (47) sono state caricate, ma devono essere etichettate in modo appropriato prima di essere utilizzate come dataset del progetto. Studio ha uno strumento per questo scopo, che si trova nel link **Labeling queue (47)**. Ci sono due modi per eseguire l’etichettatura assistita dall’IA su Edge Impulse Studio (versione gratuita):

- Utilizzando yolov5
- Tracciando di oggetti tra i frame

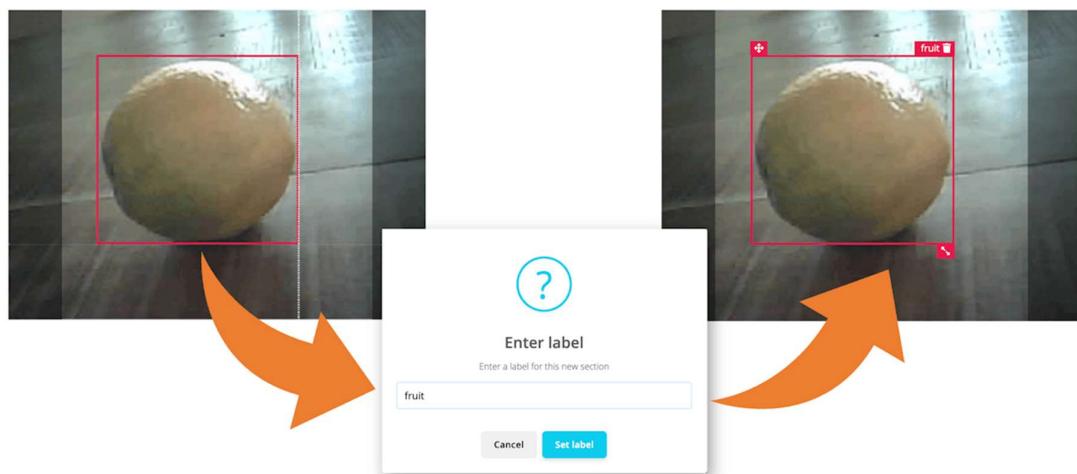
Edge Impulse ha lanciato una [funzione di auto-labeling](#) per i clienti Enterprise, semplificando le attività di etichettatura nei progetti di rilevamento degli oggetti.

Gli oggetti ordinari possono essere rapidamente identificati ed etichettati utilizzando una libreria esistente di modelli di rilevamento degli oggetti pre-addestrati da YOLOv5 (addestrati con il set di dati COCO). Ma poiché, nel nostro caso, gli oggetti non fanno parte dei dataset COCO, dovremmo selezionare l'opzione di “tracking” [tracciamento] degli oggetti. Con questa opzione, una volta disegnati i bounding box ed etichettate le immagini in un frame, gli oggetti verranno tracciati automaticamente da un frame all'altro, etichettando *partially* quelli nuovi (non tutti sono etichettati correttamente).

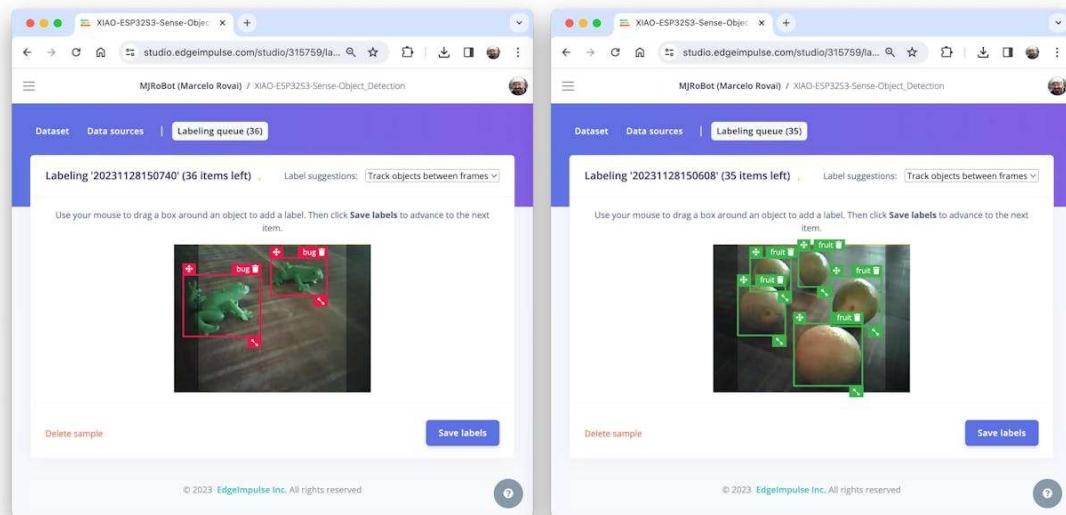
Si può usare [El uploader](#) per importare i dati se si ha già un dataset etichettato contenente dei “bounding box”.

Etichettatura del Dataset

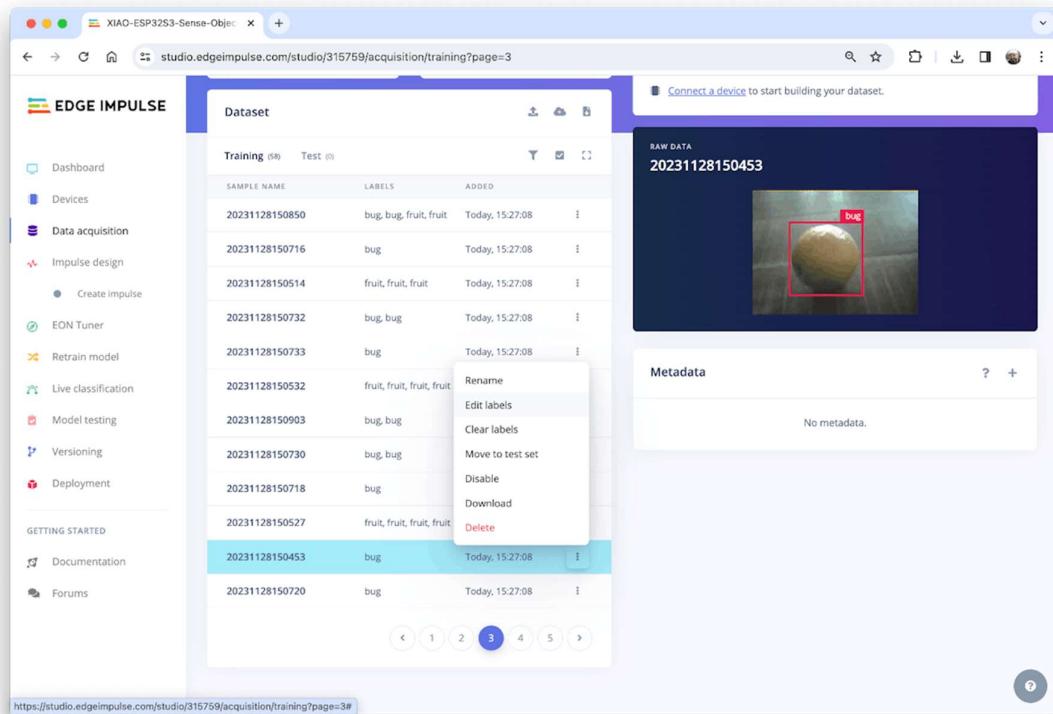
Iniziando dalla prima immagine dei dati non etichettati, si usa il mouse per trascinare una casella attorno a un oggetto per aggiungere un'etichetta. Poi si clicca su **Save labels** per passare all'elemento successivo.



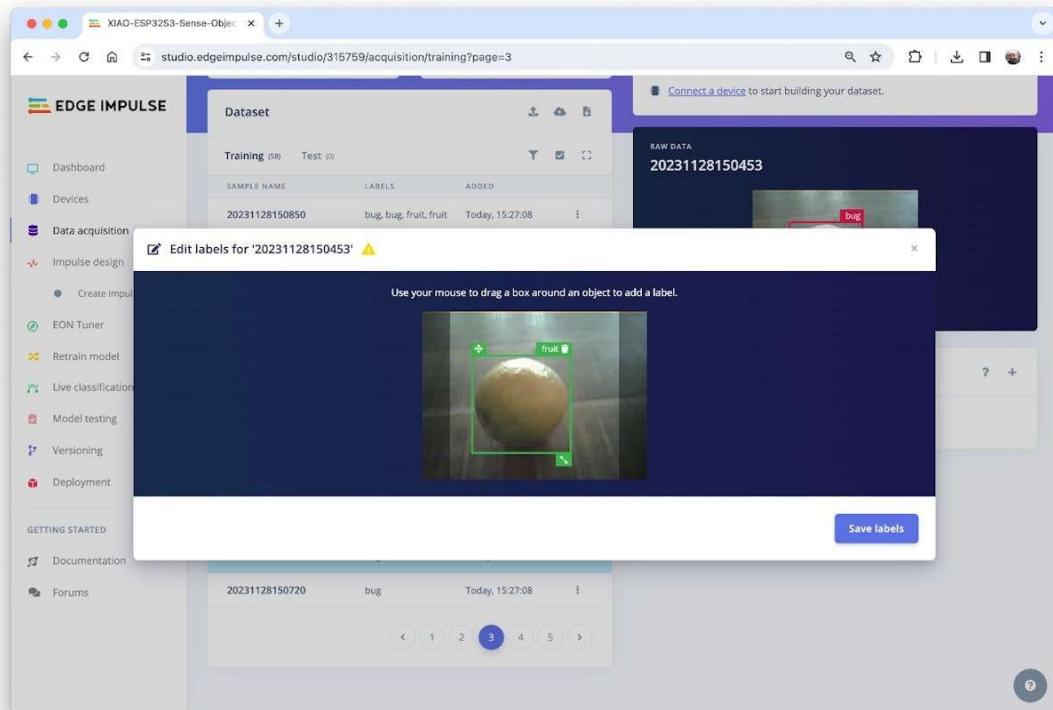
Si continua con questo processo finché la coda non è vuota. Alla fine, tutte le immagini dovrebbero avere gli oggetti etichettati come i campioni sottostanti:



Poi, si esaminano i campioni etichettati nella scheda [Data acquisition](#). Se una delle etichette è errata, la si può modificarla utilizzando il menù *tre puntini* dopo il nome del campione:



Si verrà guidati a sostituire l'etichetta errata e a correggere il dataset.



Bilanciamento del dataset e suddivisione Train/Test

Dopo aver etichettato tutti i dati, ci siamo resi conto che la classe fruit aveva molti più campioni di bug. Quindi, sono state raccolte 11 immagini di bug nuove e aggiuntive (per un totale di 58 immagini). Dopo averle etichettate, è il momento di selezionare alcune immagini e spostarle nel dataset di test. Per farlo si usa il menù a tre punti dopo il nome dell'immagine. Sono state selezionate sei immagini, che rappresentano il 13% del set di dati totale.

Impulse Design

In questa fase, si deve definire come:

- Il **Pre-processing** consiste nel ridimensionare le singole immagini da 320 x 240 a 96 x 96 e nel ridurle (forma quadrata, senza ritaglio). In seguito, le immagini vengono convertite da RGB a scala di grigi.
- **Design a Model**, in questo caso, “Object Detection”.

Pre-elaborazione di tutti i dataset

In questa sezione, selezionare **Color depth** come Grayscale, adatta per l'uso con modelli FOMO ed eseguire il “Save” dei parametri.

The screenshot shows the Edge Impulse Studio interface. On the left, there's a sidebar with various options like Dashboard, Devices, Data acquisition, and Image. The main area has tabs for 'Raw data' and 'Generate features'. Under 'Raw data', there's a preview of an image with a red bounding box labeled 'bug'. Below it, the 'Raw features' section lists some hex values. The 'Parameters' tab is active, showing 'Color depth' set to 'Grayscale'. In the 'DSP result' section, there's a preview image and a button labeled 'Copy 9216 features to clipboard'. The 'On-device performance' section shows processing time at 15 ms and peak RAM usage at 4 KB.

Studio passa automaticamente alla sezione successiva, “Generate features”, in cui tutti i campioni verranno pre-elaborati, generando un set di dati con singole immagini 96x96x1 o 9.216 feature.

This screenshot shows the 'Generate features' section of the Edge Impulse Studio. The left panel shows a 'Training set' with 52 items and 2 classes ('bug' and 'fruit'). The right panel contains a 'Feature explorer' scatter plot for the 'bug' and 'fruit' classes. Below it is a 'Feature generation output' log that is mostly filled with 'Still running...' entries, followed by 'Tue Nov 28 19:09:20 2023 Finished embedding Reducing dimensions for visualizations OK' and 'Job completed'. The 'On-device performance' section at the bottom shows a processing time of 15 ms and peak RAM usage of 4 KB.

L'esploratore di feature mostra che tutti i campioni evidenziano una buona separazione dopo la generazione delle feature.

Alcuni campioni sembrano stare nello spazio sbagliato, ma cliccandoci sopra si conferma la corretta etichettatura.

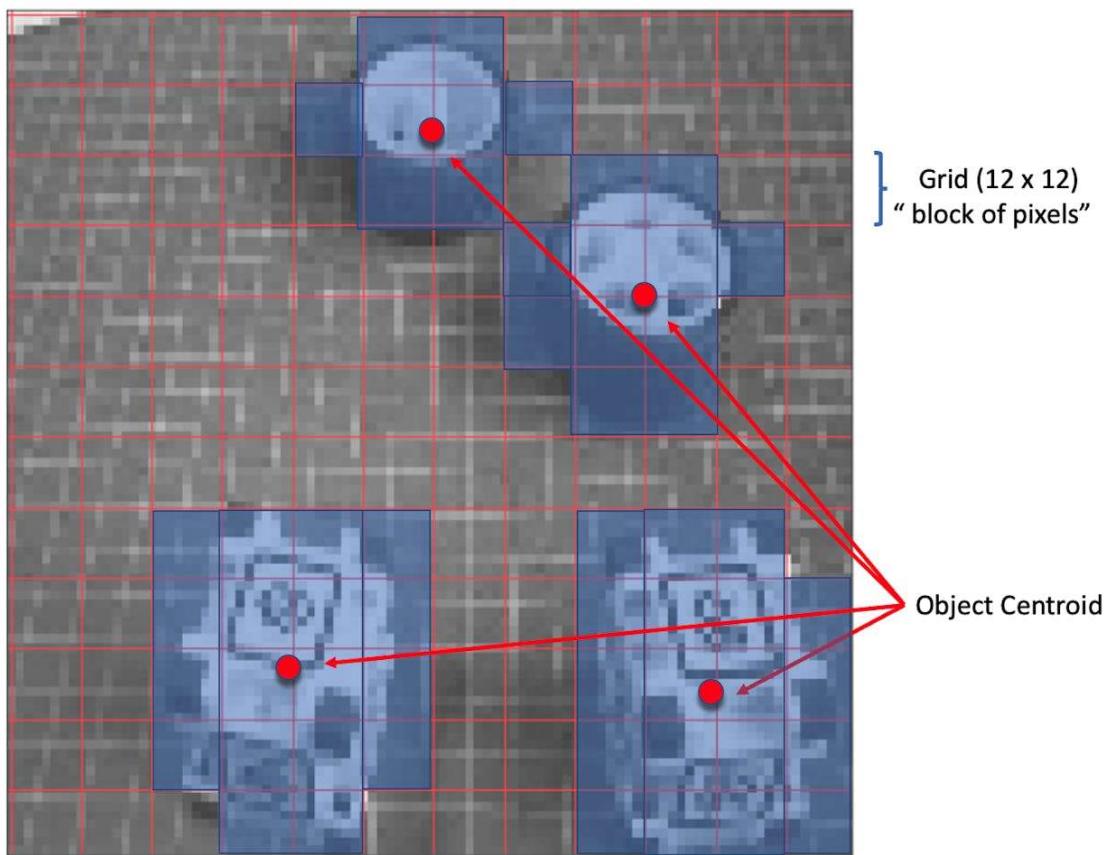
Progettazione, Addestramento e Test del Modello

Useremo FOMO, un modello di rilevamento degli oggetti basato su MobileNetV2 (alpha 0.35) progettato per segmentare grossolanamente un'immagine in una griglia di **background** rispetto a **oggetti di interesse** (in questo caso, scatole e ruote).

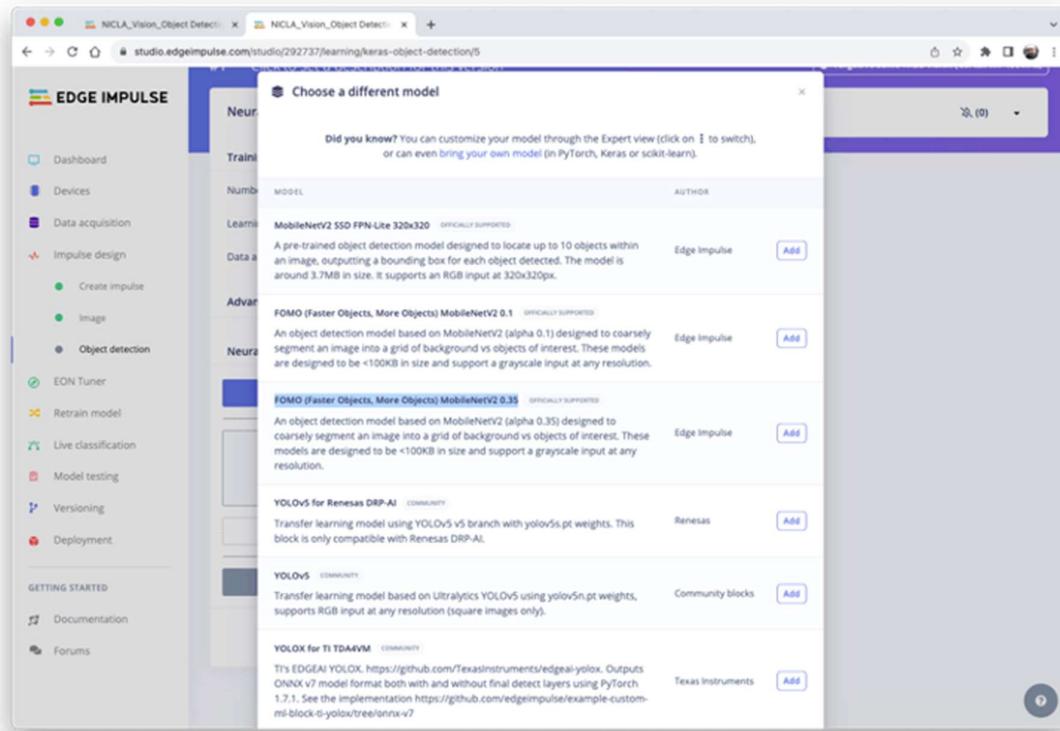
FOMO è un modello di apprendimento automatico innovativo per il rilevamento degli oggetti, che può utilizzare fino a 30 volte meno energia e memoria rispetto ai modelli tradizionali come Mobilenet SSD e YOLOv5. FOMO può funzionare su microcontrollori con meno di 200 KB di RAM. Il motivo principale per cui ciò è possibile è che mentre altri modelli calcolano le dimensioni dell'oggetto disegnando un quadrato attorno ad esso (bounding box), FOMO ignora le dimensioni dell'immagine, fornendo solo le informazioni su dove si trova l'oggetto nell'immagine tramite le sue coordinate del centroide.

Come funziona FOMO?

FOMO prende l'immagine in scala di grigi e la divide in blocchi di pixel usando un fattore di 8. Per l'input di 96x96, la griglia è 12x12 ($96/8=12$). Successivamente, FOMO eseguirà un classificatore attraverso ogni blocco di pixel per calcolare la probabilità che ci sia un box o una ruota in ognuno di essi e, successivamente, determinerà le regioni che hanno la più alta probabilità di contenere l'oggetto (se un blocco di pixel non ha oggetti, verrà classificato come *background*). Dalla sovrapposizione della regione finale, FOMO fornisce le coordinate (relative alle dimensioni dell'immagine) del centroide di questa regione.



Per l'addestramento, dovremmo selezionare un modello pre-addestrato. Usiamo **FOMO (Faster Objects, More Objects) MobileNetV2 0.35**. Questo modello utilizza circa 250 KB di RAM e 80 KB di ROM (Flash), che si adatta bene alla nostra scheda.



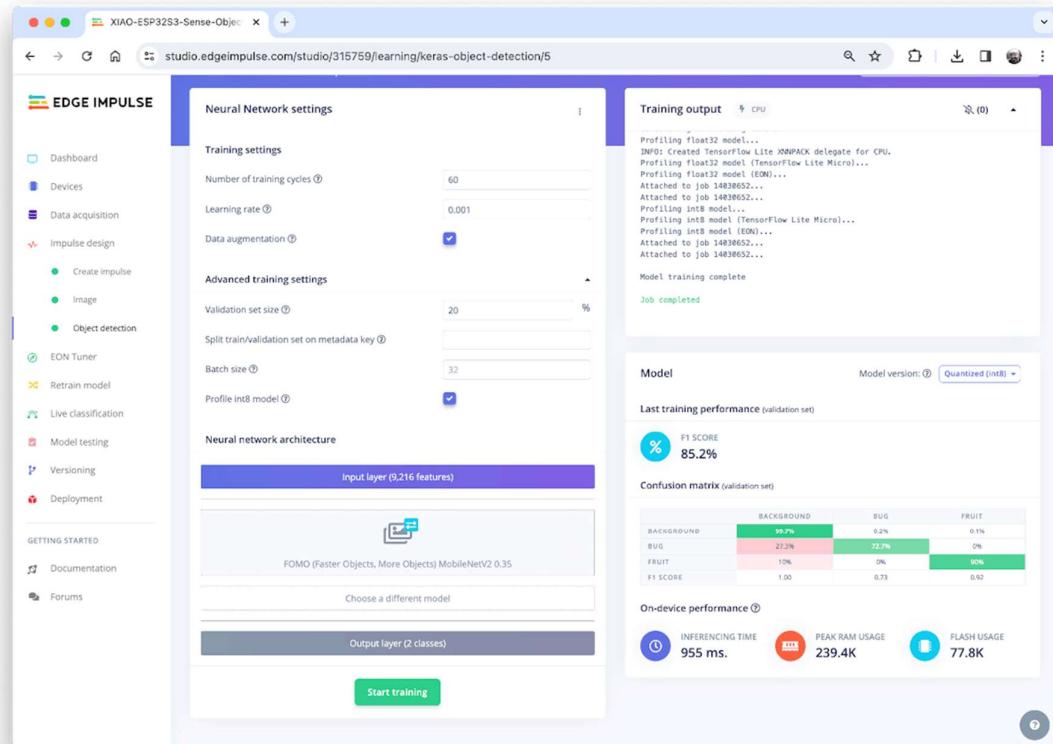
Per quanto riguarda gli iperparametri di training, il modello verrà addestrato con:

- Epochs: 60
- Batch size: 32
- Learning Rate: 0.001.

Per la convalida durante l'addestramento, il 20% del set di dati (*validation_dataset*) verrà risparmiato. Per il restante 80% (*train_dataset*), applicheremo il “Data Augmentation”, che capovolgerà casualmente, cambierà le dimensioni e la luminosità dell'immagine e le ritaglierà, aumentando artificialmente il numero di campioni sul set di dati per l'addestramento.

Di conseguenza, il modello termina con un punteggio F1 complessivo dell'85%, simile al risultato ottenuto utilizzando i dati di prova (83%).

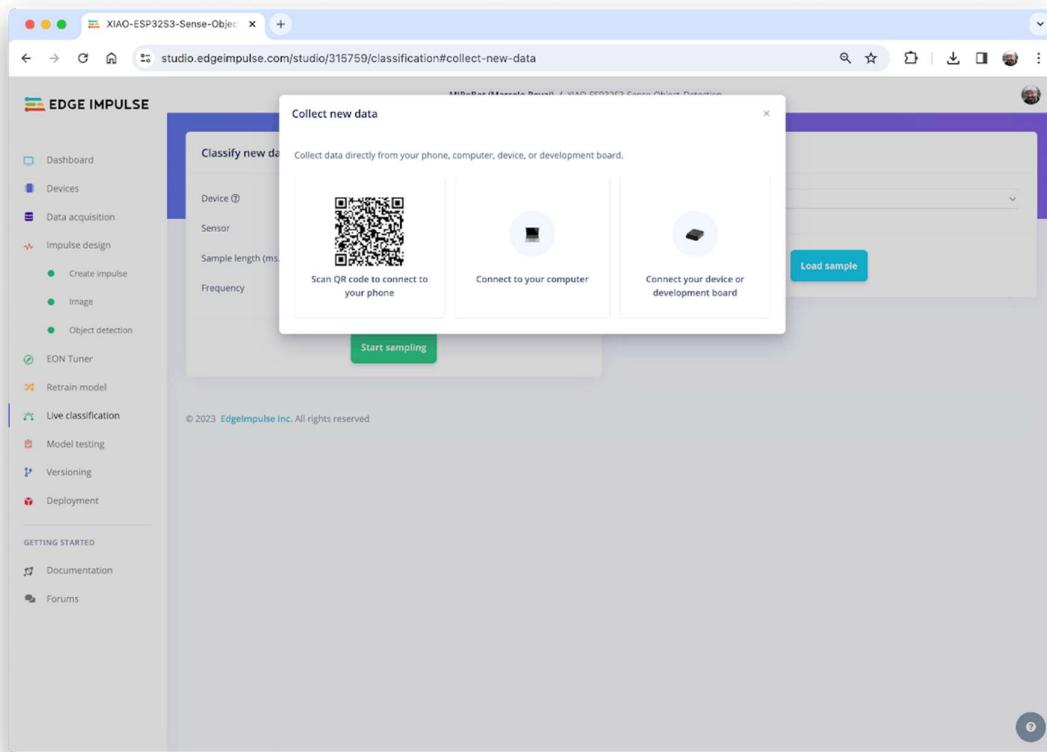
Notare che FOMO ha aggiunto automaticamente una terza etichetta di background [sfondo] ai due precedentemente definiti (*box* e *wheel*).



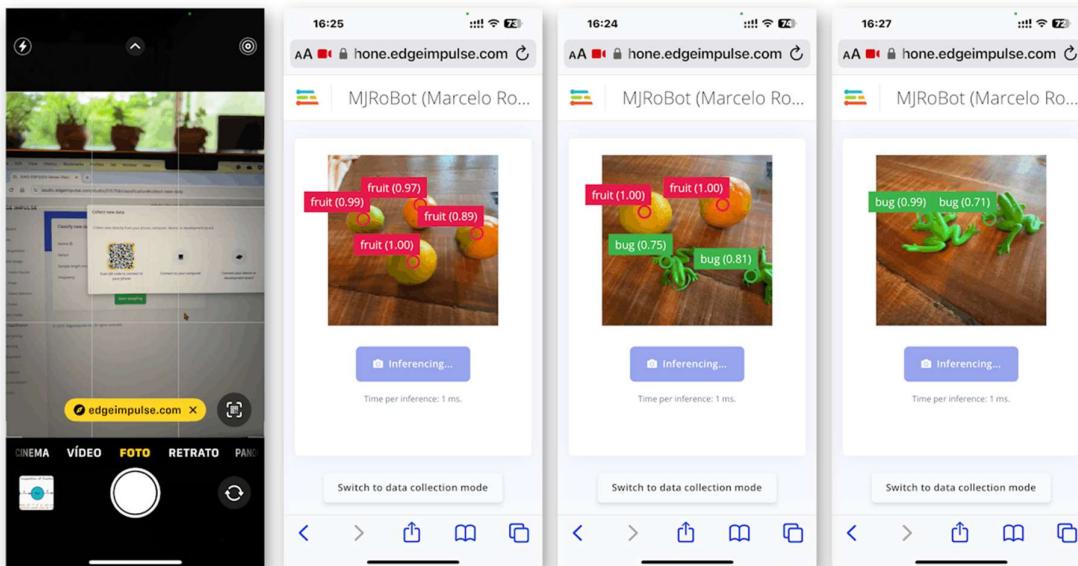
Nelle attività di rilevamento di oggetti, l’accuratezza non è in genere la **metrica di valutazione primaria**. Il rilevamento di oggetti comporta la classificazione degli oggetti e la definizione di riquadri di delimitazione attorno a essi, il che lo rende un problema più complesso della semplice classificazione. Il problema è che non abbiamo il riquadro di delimitazione, solo i centroidi. In breve, usare l’accuratezza come metrica potrebbe essere fuorviante e potrebbe non fornire una comprensione completa delle prestazioni del modello. Per questo motivo, useremo il punteggio F1.

Modello di test con “Live Classification”

Una volta addestrato il nostro modello, possiamo testarlo utilizzando lo strumento Live Classification. Nella sezione corrispondente, cliccare sull’icona “Connect a development board” (una piccola MCU) e scansionare il codice QR col telefono.



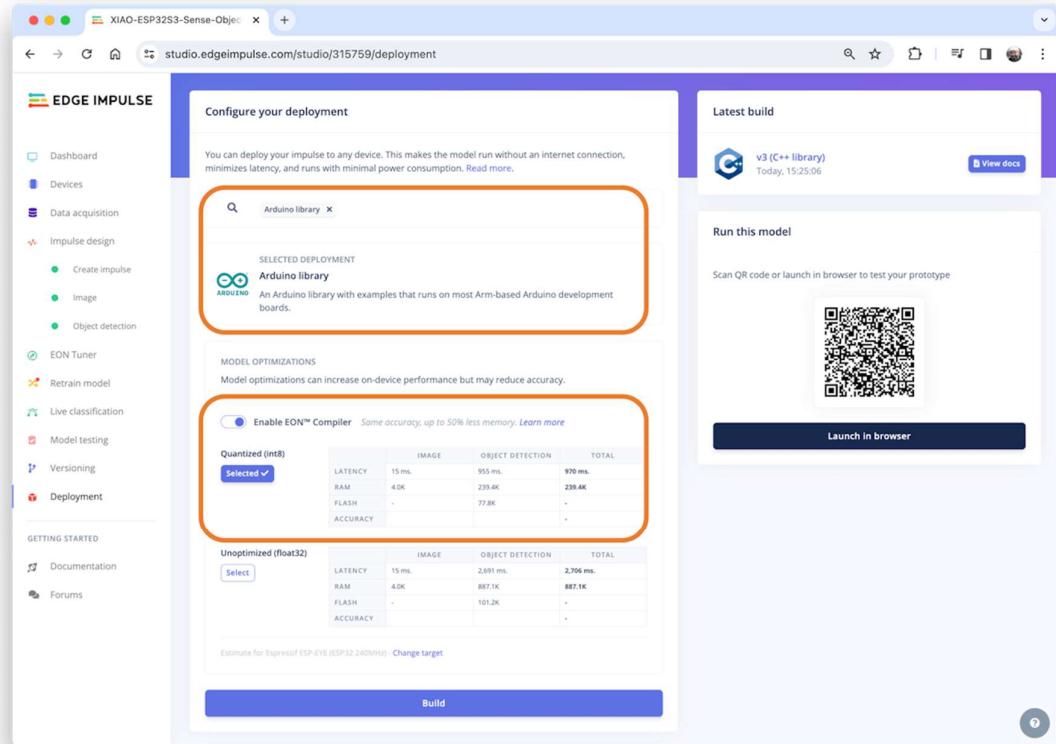
Una volta connesso, si può usare lo smartphone per catturare immagini reali da testare col modello addestrato su Edge Impulse Studio.



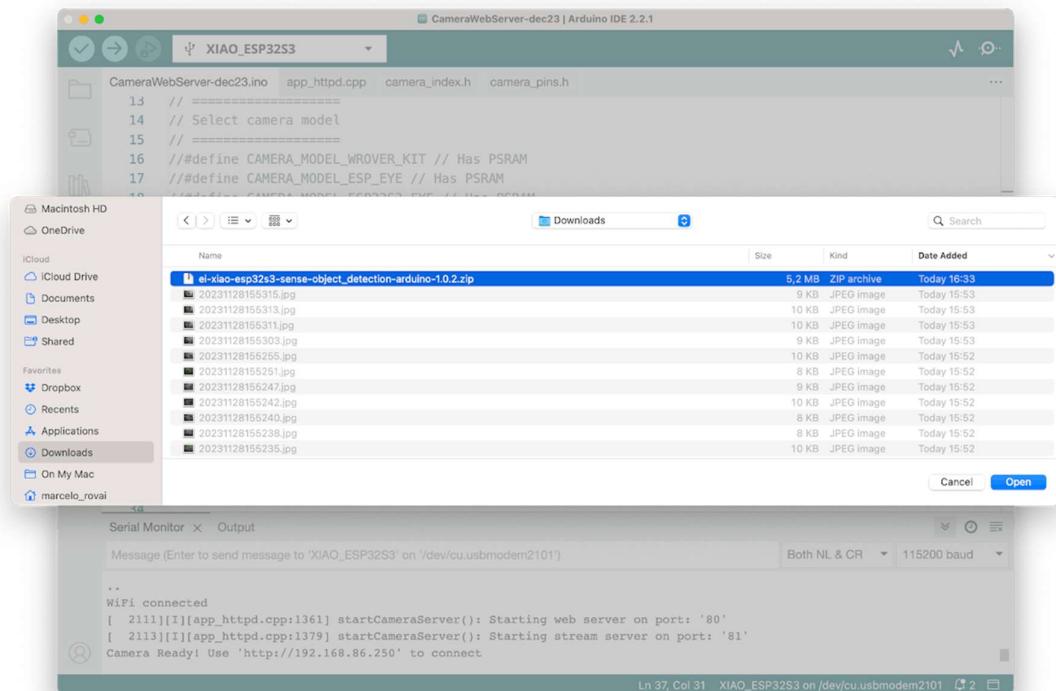
Una cosa da notare è che il modello può produrre falsi positivi e falsi negativi. Questo può essere ridotto al minimo definendo una “Confidence Threshold” appropriata (usare il menu “Tre puntini” per la configurazione). Provare con 0,8 o più.

Deploying del Modello (Arduino IDE)

Selezionare la Libreria Arduino e il modello Quantized (int8), abilitare il compilatore EON nella scheda Deploy e premere [Build].



Apri l’Arduino IDE e, in Sketch, andare su Include Library e aggiungere .ZIP Library.
Selezionare il file che scaricato da Edge Impulse Studio e il gioco è fatto!



Nella scheda Examples su Arduino IDE, si trova il codice di uno sketch (`esp32 > esp32_camera`) sotto il nome del progetto.



Si devono cambiare le righe dalla 32 alla 75, che definiscono il modello e i pin della telecamera, utilizzando i dati relativi al nostro modello. Copiare e incollare le righe seguenti, sostituendo le righe 32-75:

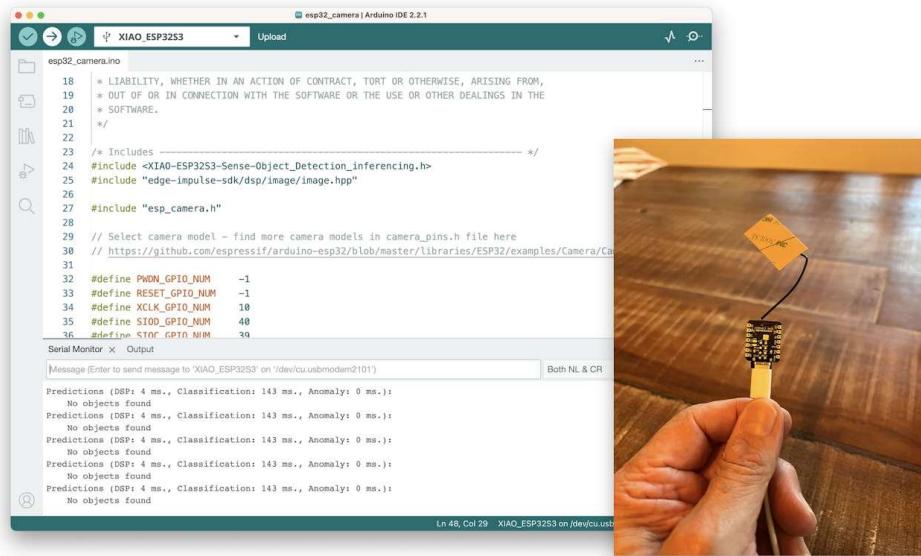
```
#define PWDN_GPIO_NUM      -1
#define RESET_GPIO_NUM     -1
#define XCLK_GPIO_NUM       10
#define SIOD_GPIO_NUM       40
#define SIOC_GPIO_NUM       39
#define Y9_GPIO_NUM          48
#define Y8_GPIO_NUM          11
#define Y7_GPIO_NUM          12
#define Y6_GPIO_NUM          14
#define Y5_GPIO_NUM          16
#define Y4_GPIO_NUM          18
#define Y3_GPIO_NUM          17
#define Y2_GPIO_NUM          15
#define VSYNC_GPIO_NUM       38
#define HREF_GPIO_NUM        47
#define PCLK_GPIO_NUM        13
```

Ecco il codice risultante:

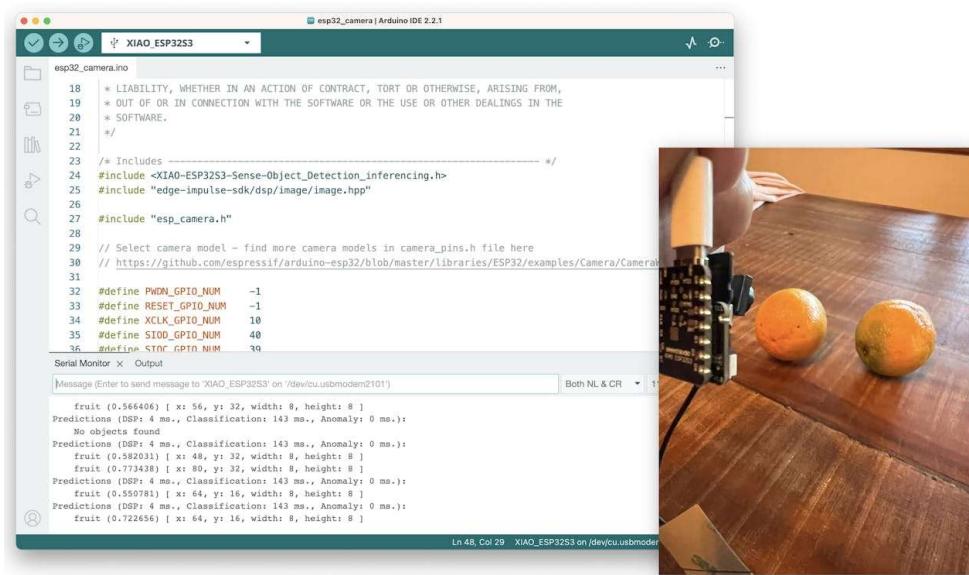
```
esp32_camera.ino
31 // LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
32 // OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
33 // SOFTWARE.
34 /*
35  * Includes -----
36  */
37 #include "XIAO-ESP32S3-Sense-Object_Detection_inferencing.h"
38 #include "edge-impulse-sd/dsp/image/image.hpx"
39 #include "esp_camera.h"
40 /*
41  * Select camera model - find more camera models in camera_pins.h file here
42  * https://github.com/arduino-esp32/blob/master/libraries/ESP32/examples/
43  */
44 #define PWDN_GPIO_NUM      -1
45 #define RESET_GPIO_NUM     -1
46 #define XCLK_GPIO_NUM       4
47 #define SIOD_GPIO_NUM       18
48 #define SIOC_GPIO_NUM       23
49 #define Y9_GPIO_NUM          36
50 #define Y8_GPIO_NUM          37
51 #define Y7_GPIO_NUM          38
52 #define Y6_GPIO_NUM          39
53 #define Y5_GPIO_NUM          35
54 #define Y4_GPIO_NUM          14
55 #define Y3_GPIO_NUM          13
56 #define Y2_GPIO_NUM          34
57 #define VSYNC_GPIO_NUM       5
58 #define HREF_GPIO_NUM        27
59 #define PCLK_GPIO_NUM        25
60 /*
61  * If defined(CAMERA_MODEL_AI_THINKER)
62  */
63 #define PWDN_GPIO_NUM      32
64 #define RESET_GPIO_NUM     -1
65 #define XCLK_GPIO_NUM       0
66 #define SIOD_GPIO_NUM       26
67 #define SIOC_GPIO_NUM       27
68 #define Y9_GPIO_NUM          35
69 #define Y8_GPIO_NUM          34
70 #define Y7_GPIO_NUM          39
71 #define Y6_GPIO_NUM          36
72 #define Y5_GPIO_NUM          21
73 #define Y4_GPIO_NUM          19
74 #define Y3_GPIO_NUM          18
75 #define Y2_GPIO_NUM          5
76 #define VSYNC_GPIO_NUM       25
77 #define HREF_GPIO_NUM        23
78 #define PCLK_GPIO_NUM        22
79 /*
80  * else
81  * #error "Camera model not selected"
82  */
83 /*
84  * Constant defines -----
85  */
86 #define EI_CAMERA_RAW_FRAME_BUFFER_COLS      320
87 #define EI_CAMERA_RAW_FRAME_BUFFER_ROWS      240
88 #define EI_CAMERA_FRAME_BYTE_SIZE            3
89 /*
90  * Private variables -----
91  */
92 static bool debug_on = false; // Set this to true to see e.g. features generated for
93 static bool is initialised = false;
94 uint8_t *snapshot_buf; // points to the output of the capture
95 /*
96  * static camera_config_t camera_config = {
97  */
98 .pin_pwdn = PWDN_GPIO_NUM,
99 .pin_reset = RESET_GPIO_NUM,
100 .pin_xclk = XCLK_GPIO_NUM,
101 .pin_ssck = SIOD_GPIO_NUM,
```

Caricare il codice sul XIAO ESP32S3 Sense e si è pronti a iniziare a rilevare frutta e insetti. Si può controllare il risultato su Serial Monitor.

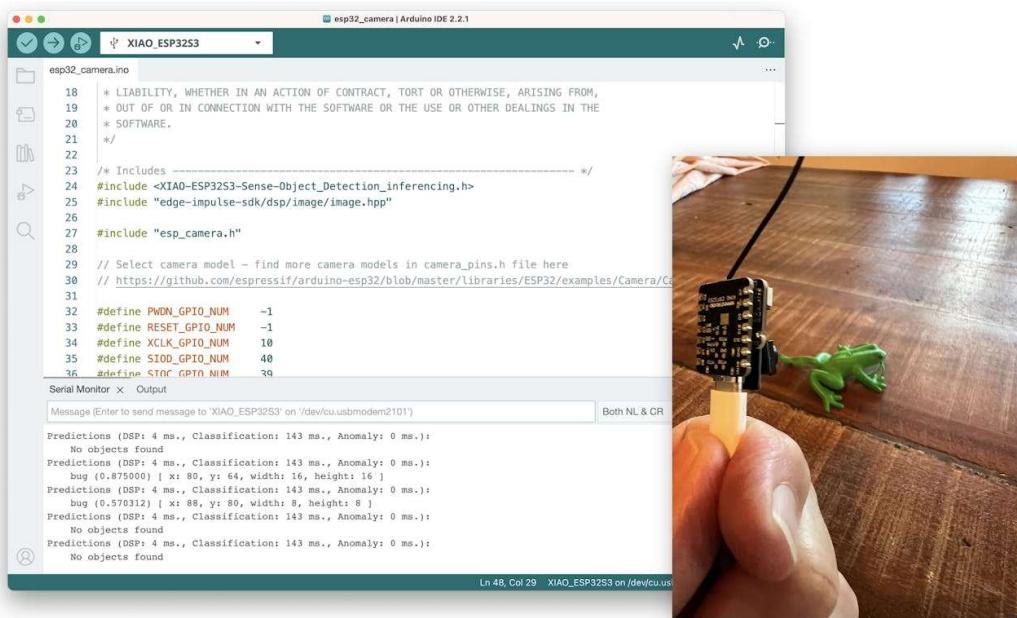
Background



Fruits



Bugs



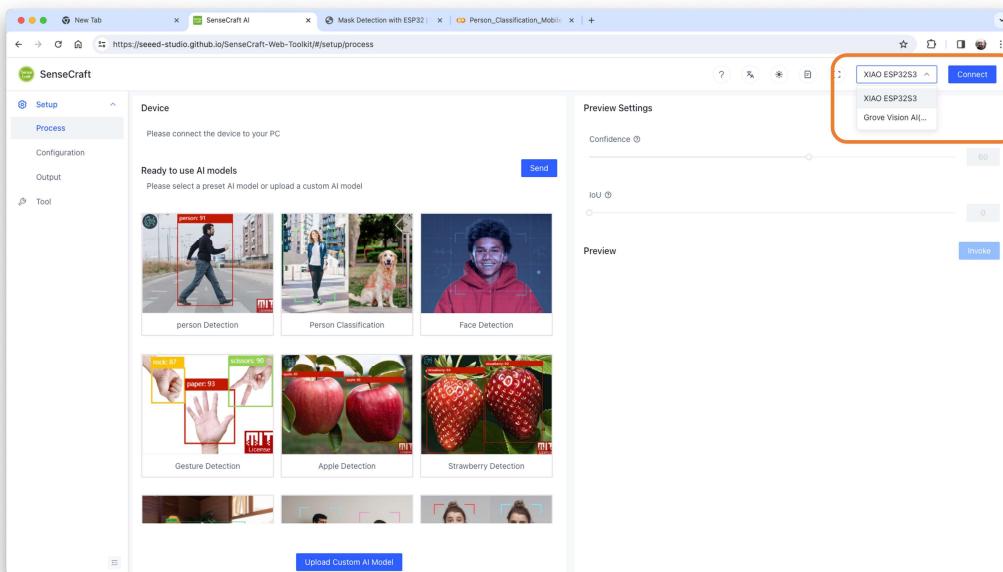
Si noti che la latenza del modello è di 143ms e il frame rate al secondo è di circa 7 fps (simile a quanto ottenuto con il progetto Image Classification). Ciò accade perché FOMO è intelligentemente costruito su un modello CNN, non con un modello di rilevamento degli oggetti come SSD MobileNet. Ad esempio, quando si esegue un modello MobileNetV2 SSD FPN-Lite 320x320 su un Raspberry Pi 4, la latenza è circa cinque volte superiore (circa 1,5 fps).

Distribuzione del Modello (SenseCraft-Web-Toolkit)

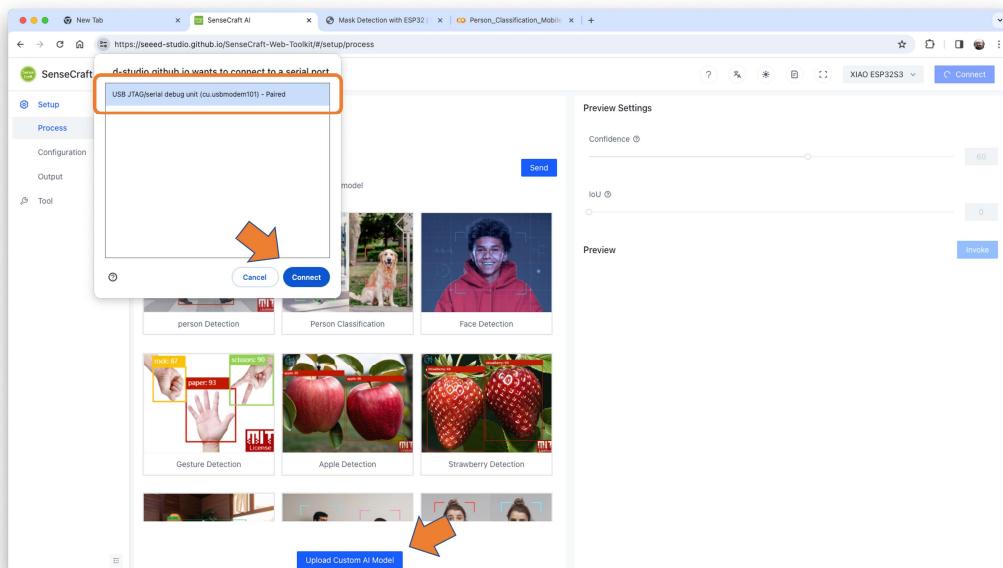
Come discusso nel capitolo Image Classification, verificare l'inferenza con i modelli di immagine su Arduino IDE è molto impegnativo perché non possiamo vedere su cosa punta la telecamera. Di nuovo, utilizziamo **SenseCraft-Web Toolkit**.

Seguire i seguenti passaggi per avviare SenseCraft-Web-Toolkit:

1. Aprire il [sito web di SenseCraft-Web-Toolkit](#).
2. Collega XIAO al computer:
 - Dopo aver collegato XIAO, selezionarlo come di seguito:



- Selezionare il dispositivo/Porta e premere [Connect]:



Si possono provare diversi modelli di Computer Vision caricati in precedenza da Seeed Studio. Da provare e verificarli!

Nel nostro caso, useremo il pulsante blu in fondo alla pagina: [Upload Custom AI Model]. Ma prima, dobbiamo scaricare da Edge Impulse Studio il modello **quantized .tflite**.

3. Si va sul proprio progetto su Edge Impulse Studio, oppure si clona questo:

- [XIAO-ESP32S3-CAM-Fruits-vs-Veggies-v1-ESP-NN](#)

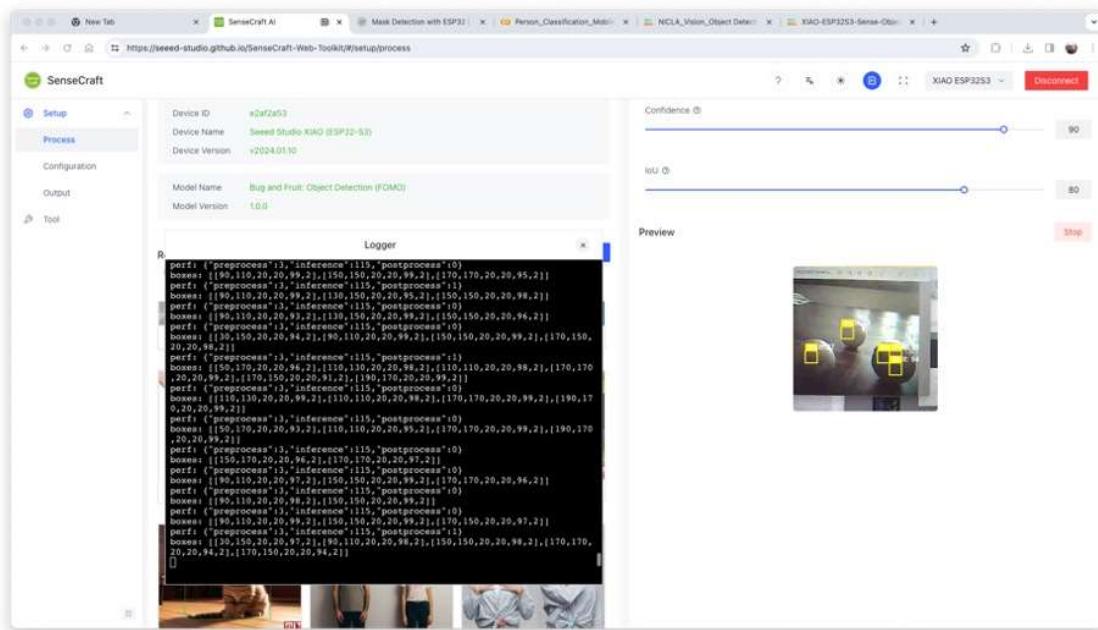
4. Su Dashboard, scaricare il modello (“block output”): Object Detection model - TensorFlow Lite (int8 quantized)

TITLE	TYPE	SIZE
Image training data	NPY file	52 windows
Image training labels	JSON file	52 windows
Image testing data	NPY file	6 windows
Image testing labels	JSON file	6 windows
Object detection model	TensorFlow Lite (float32)	82 KB
Object detection model	TensorFlow Lite (int8 quantized)	55 KB
Object detection model	TensorFlow SavedModel	186 KB
Object detection model	Keras h5 model	88 KB

5. Su SenseCraft-Web-Toolkit, usare il pulsante blu in fondo alla pagina: [Upload Custom AI Model]. Si aprirà una finestra. Inserire il file del Modello scaricato sul computer da Edge Impulse Studio, scegliere un nome del modello e inserirlo con le etichette (ID: Object):

Notare che si devono utilizzare le etichette apprese su El Studio e inserirle in ordine alfabetico (nel nostro caso, background, bug, fruit).

Dopo alcuni secondi (o minuti), il modello verrà caricato sul dispositivo e l'immagine della telecamera apparirà in tempo reale nel Preview Sector:



Gli oggetti rilevati saranno contrassegnati (il centroide). È possibile selezionare l'affidabilità del cursore di inferenza **Confidence** e **IoU**, che viene utilizzata per valutare l'accuratezza delle “bounding box” previste rispetto a quelle vere.

Cliccando sul pulsante in alto (Device Log), si può aprire un Serial Monitor per seguire l'inferenza, come abbiamo fatto con l'IDE Arduino.

```

perf: {"preprocess":3,"inference":115,"postprocess":1}
boxes: [[30,150,20,20,97,2],[90,110,20,20,98,2],[150,150,20,20,98,2],[170,170,20,20,97,2],[170,150,20,20,94,2],[170,150,20,20,94,2]]

```

Su Device Log, si otterranno informazioni come:

- Tempo di pre-elaborazione (acquisizione dell'immagine e Crop): 3 ms,
- Tempo di inferenza (latenza del modello): 115 ms,
- Tempo di post-elaborazione (visualizzazione dell'immagine e marcatura degli oggetti): 1 ms.
- Tensore di output (box), ad esempio, uno dei box: [[30,150,20,20,97,2]]; dove 30,150, 20, 20 sono le coordinate della casella (intorno al centroide); 97 è il risultato dell'inferenza e 2 è la classe (in questo caso 2: frutto).

Notare che nell'esempio precedente, abbiamo ottenuto 5 caselle perché nessuno dei frutti ha ottenuto 3 centroidi. Una soluzione sarà la post-elaborazione, dove possiamo aggregare centroidi vicini in uno.

Ecco altri screenshot:



Conclusione

FOMO è un salto significativo nello spazio di elaborazione delle immagini, come hanno affermato Louis Moreau e Mat Kelcey durante il suo lancio nel 2022:

FOMO è un algoritmo rivoluzionario che porta per la prima volta il rilevamento, il tracciamento e il conteggio degli oggetti in tempo reale sui microcontrollori.

Esistono molteplici possibilità per esplorare il rilevamento di oggetti (e, più precisamente, il loro conteggio) su dispositivi embedded.

Risorse

- [Progetto Edge Impulse](#)

Keyword Spotting (KWS)



Immagine di Marcelo Rovai

Panoramica

Keyword Spotting (KWS) è parte integrante di molti sistemi di riconoscimento vocale, consentendo ai dispositivi di rispondere a parole o frasi specifiche. Sebbene questa tecnologia sia alla base di dispositivi popolari come Google Assistant o Amazon Alexa, è ugualmente applicabile e realizzabile su dispositivi più piccoli e a basso consumo. Questo lab guiderà nell'implementazione di un sistema KWS utilizzando TinyML sulla scheda microcontrollore XIAO ESP32S3.

The XIAO ESP32S3, dotato del chip ESP32-S3 di Espressif, è un microcontrollore compatto e potente che offre un processore Xtensa LX7 dual-core, Wi-Fi integrato e Bluetooth. Il suo

equilibrio di potenza di calcolo, efficienza energetica e connettività versatile lo rendono una piattaforma fantastica per le applicazioni TinyML. Inoltre, con la sua scheda di espansione, avremo accesso alla parte “sense” del dispositivo, che ha una fotocamera OV2640 da 1600x1200, uno slot per schede SD e un **microfono digitale**. Il microfono integrato e la scheda SD saranno essenziali in questo progetto.

Utilizzeremo [Edge Impulse Studio](#), una piattaforma potente e intuitiva che semplifica la creazione e l’implementazione di modelli di apprendimento automatico su dispositivi edge. Addestreremo un modello KWS passo dopo passo, ottimizzandolo e distribuendolo su XIAO ESP32S3 Sense.

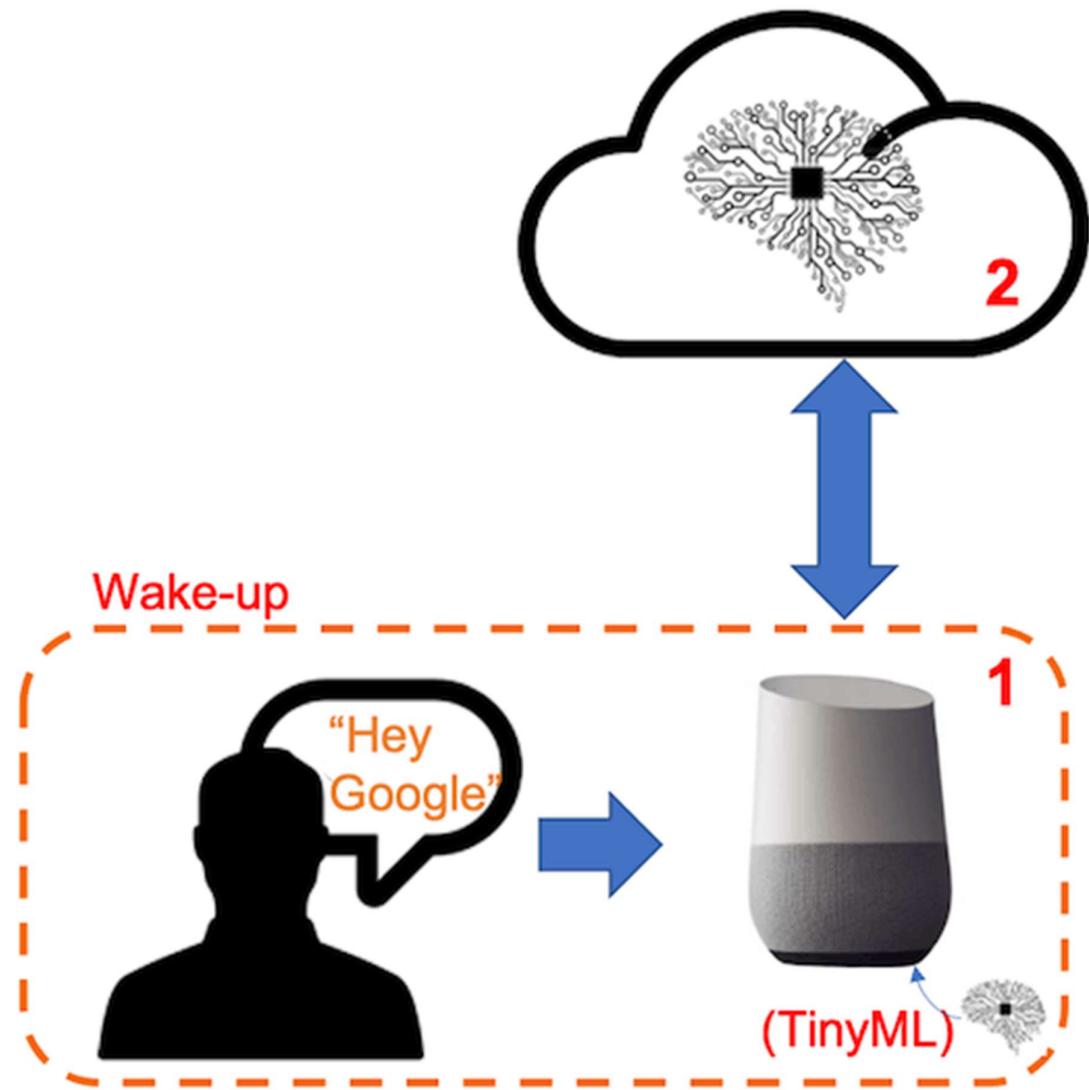
Il nostro modello sarà progettato per riconoscere parole chiave che possono attivare il “wake-up” [risveglio] del dispositivo o azioni specifiche (nel caso di “YES”), dando vita a progetti con comandi vocali.

Sfruttando la nostra esperienza con TensorFlow Lite per microcontrollori (il motore “sotto il cofano” di EI Studio), creeremo un sistema KWS in grado di apprendere in tempo reale sul dispositivo.

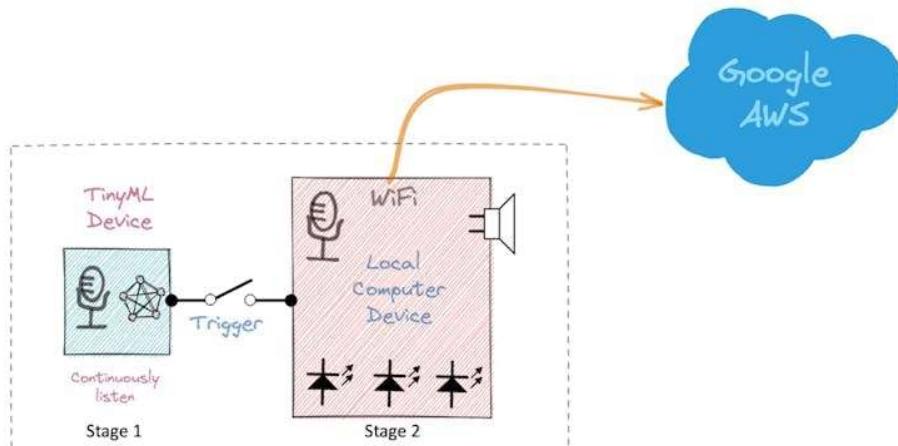
Procedendo nel lab, analizzeremo ogni fase del processo, dalla raccolta e preparazione dei dati all’addestramento e distribuzione del modello, per fornire una comprensione completa dell’implementazione di un sistema KWS su un microcontrollore.

Come funziona un assistente vocale?

Keyword Spotting (KWS) è fondamentale per molti assistenti vocali, consentendo ai dispositivi di rispondere a parole o frasi specifiche. Per iniziare, è essenziale rendersi conto che gli assistenti vocali sul mercato, come Google Home o Amazon Echo-Dot, reagiscono agli umani solo quando vengono “svegliati” da parole chiave specifiche come “Hey Google” sul primo e “Alexa” sul secondo.



In altre parole, il riconoscimento dei comandi vocali si basa su un modello multi-fase o Cascade Detection.



Fase 1: Un microprocessore più piccolo all'interno dell'Echo Dot o Google Home **ascolta continuamente** il suono, in attesa che venga individuata la parola chiave. Per tale rilevamento, viene utilizzato un modello TinyML all'edge (applicazione KWS).

Fase 2: Solo quando vengono attivati dall'applicazione KWS nella Fase 1, i dati vengono inviati al cloud ed elaborati su un modello più grande.

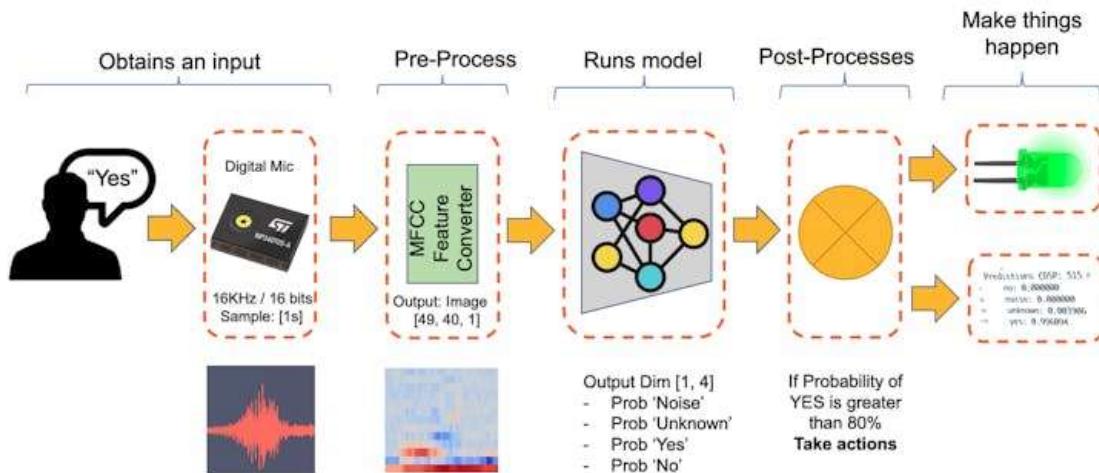
Il video qui sotto mostra un esempio in cui si emula un Google Assistant su un Raspberry Pi (Fase 2), con un Arduino Nano 33 BLE come dispositivo tinyML (Fase 1).

Per approfondire il progetto completo, guardare il tutorial: [Building an Intelligent Voice Assistant From Scratch](#).

In questo lab, ci concentreremo sulla Fase 1 (KWS o Keyword Spotting), dove utilizzeremo XIAO ESP2S3 Sense, che ha un microfono digitale per individuare la parola chiave.

Il progetto KWS

Il diagramma seguente darà un'idea di come dovrebbe funzionare l'applicazione KWS finale (durante l'inferenza):



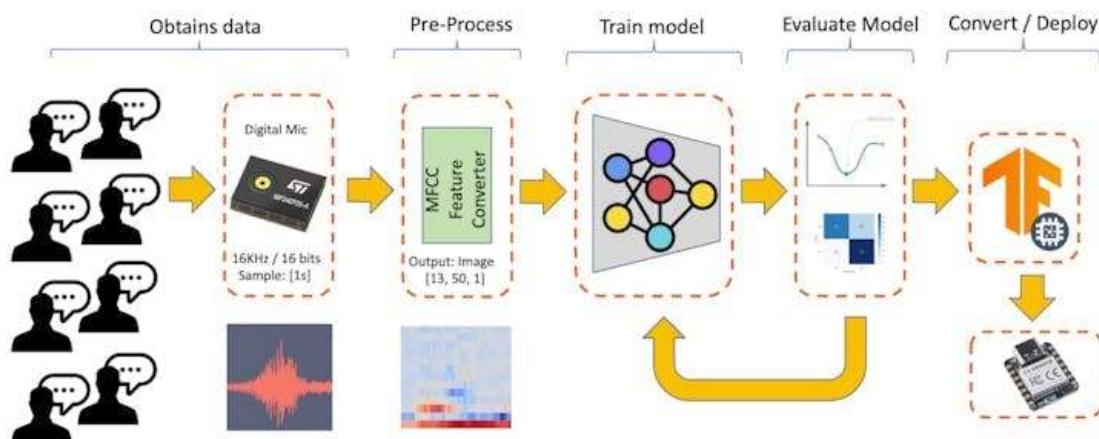
La nostra applicazione KWS riconoscerà quattro classi di suono:

- **YES** (Keyword 1)
- **NO** (Keyword 2)
- **NOISE** [rumore] (nessuna parola chiave pronunciata, è presente solo rumore di fondo)
- **UNKNOWN** (un mix di parole diverse da YES e NO)

Facoltativamente, per progetti reali, si consiglia sempre di includere parole diverse dalle parole chiave, come "Rumore" (o Sfondo) e "Sconosciuto".

Il Flusso di Lavoro del Machine Learning

Il componente principale dell'applicazione KWS è il suo modello. Quindi, dobbiamo addestrare un modello del genere con le nostre parole chiave specifiche, rumore e altre parole (lo "unknown"):



Il Dataset

Il componente critico del flusso di lavoro di apprendimento automatico è il **dataset**. Una volta decise le parole chiave specifiche (YES e NO), possiamo sfruttare il dataset sviluppato da Pete Warden, “[Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition](#)“. Questo set di dati ha 35 parole chiave (con +1.000 campioni ciascuna), come yes, no, stop e go. In altre parole, possiamo ottenere 1.500 campioni di yes e no. Si può scaricare una piccola parte del dataset da Edge Studio ([Keyword spotting pre-built dataset](#)), che include campioni dalle quattro classi che utilizzeremo in questo progetto: yes, no, noise e background. Per farlo, si seguono i passaggi seguenti:

- Download del [dataset delle parole chiave](#).
- Decomprimere il file in una posizione a scelta.

Sebbene disponiamo di molti dati dal dataset di Pete, è consigliabile raccogliere alcune parole pronunciate da noi. Lavorando con gli accelerometri, creare un dataset con dati acquisiti dallo stesso tipo di sensore era essenziale. Nel caso del suono, è diverso perché ciò che classificheremo sono, in realtà, dati *audio*.

La differenza fondamentale tra suono e audio è la loro forma di energia. Il suono è energia delle onde meccaniche (onde sonore longitudinali) che si propagano attraverso un mezzo causando variazioni di pressione all'interno del mezzo. L'audio è costituito da energia elettrica (segnali analogici o digitali) che rappresentano il suono elettricamente.

Le onde sonore dovrebbero essere convertite in dati audio quando pronunciamo una parola chiave. La conversione dovrebbe essere eseguita campionando il segnale generato dal microfono a 16 KHz con una profondità di 16 bit.

Quindi, qualsiasi dispositivo in grado di generare dati audio con questa specifica di base (16Khz/16bit) funzionerà bene. Come dispositivo, possiamo usare il XIAO ESP32S3 Sense appropriato, un computer o persino il telefono cellulare.



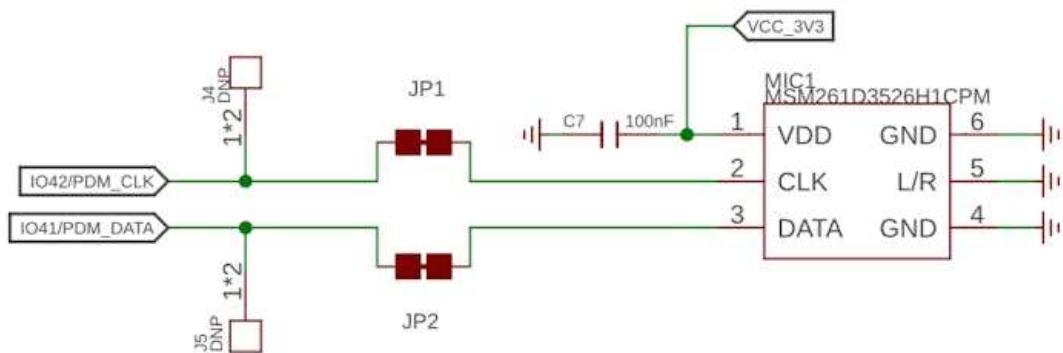
Acquisizione di Dati Audio online con Edge Impulse e uno smartphone

Nel lab “Motion Classification” e “Anomaly Detection”, collegiamo il nostro dispositivo direttamente a Edge Impulse Studio per l’acquisizione dei dati (con una frequenza di campionamento da 50 Hz a 100 Hz). Per una frequenza così bassa, potremmo usare la funzione EI CLI *Data Forwarder*, ma secondo Jan Jongboom, CTO di Edge Impulse, *l’audio (16 KHz)* è troppo veloce perché il data forwarder possa essere acquisito. Quindi, una volta che i dati digitali sono stati acquisiti dal microfono, possiamo trasformarli in un file WAV** da inviare a Studio tramite Data Uploader (lo stesso che faremo con il set di dati di Pete).

Se vogliamo raccogliere dati audio direttamente sullo Studio, possiamo usare qualsiasi smartphone connesso online. Non esploreremo questa opzione qui, ma si può facilmente seguire la [documentazione EI](#).

Acquisizione (offline) di dati audio con XIAO ESP32S3 Sense

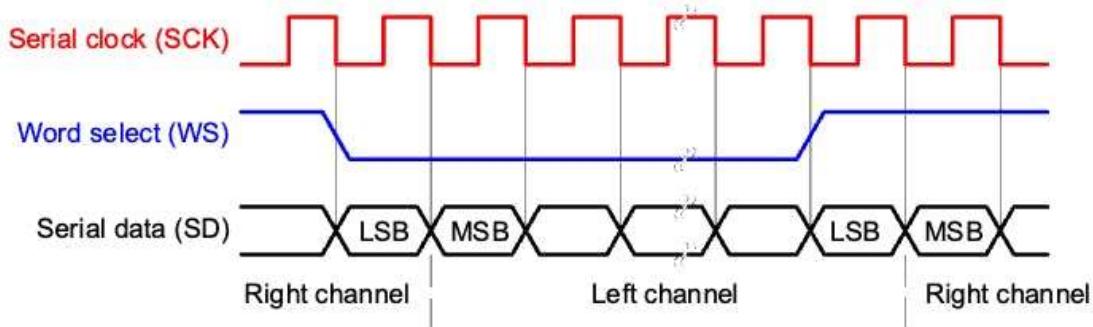
Il microfono integrato è il [MSM261D3526H1CPM](#), un microfono MEMS con uscita digitale PDM con Multi-mode. Internamente, è collegato all’ESP32S3 tramite un bus I2S utilizzando i pin IO41 (Clock) e IO41 (Data).



Cos'è I2S?

I2S, o Inter-IC Sound, è un protocollo standard per la trasmissione di audio digitale da un dispositivo a un altro. Inizialmente è stato sviluppato da Philips Semiconductor (ora NXP Semiconductors). È comunemente utilizzato in dispositivi audio come processori di segnale digitale, processori audio digitali e, più di recente, microcontrollori con capacità audio digitali (il nostro caso qui).

Il protocollo I2S è composto da almeno tre linee:



1. Linea di clock di bit (o seriale) (BCLK o CLK): Questa linea si attiva/disattiva per indicare l'inizio di un nuovo bit di dati (pin IO42).

2. Linea di "Word select (WS)": Questa linea si attiva/disattiva per indicare l'inizio di una nuova parola (canale sinistro o canale destro). La frequenza del clock di Word select (WS) definisce la frequenza di campionamento. Nel nostro caso, L/R sul microfono è impostato su massa, il che significa che utilizzeremo solo il canale sinistro (mono).

3. Data line (SD): Questa linea trasporta i dati audio (pin IO41)

In un flusso di dati I2S, i dati vengono inviati come una sequenza di frame, ciascuno contenente una parola del canale sinistro e una parola del canale destro. Ciò rende I2S particolarmente adatto per la trasmissione di dati audio stereo. Tuttavia, può anche essere utilizzato per audio mono o multicanale con linee dati aggiuntive.

Cominciamo a capire come catturare dati grezzi usando il microfono. Su va sul [progetto GitHub](#) si scarica lo sketch: [XIAOEsp2s3_Mic_Test](#):

```
/*
  XIAO ESP32S3 Simple Mic Test
*/

#include <I2S.h>

void setup() {
  Serial.begin(115200);
  while (!Serial) {
  }

  // start I2S at 16 kHz with 16-bits per sample
  I2S.setAllPins(-1, 42, 41, -1, -1);
}
```

```

if (!I2S.begin(PDM_MONO_MODE, 16000, 16)) {
    Serial.println("Failed to initialize I2S!");
    while (1); // do nothing
}

void loop() {
    // read a sample
    int sample = I2S.read();

    if (sample && sample != -1 && sample != 1) {
        Serial.println(sample);
    }
}

```

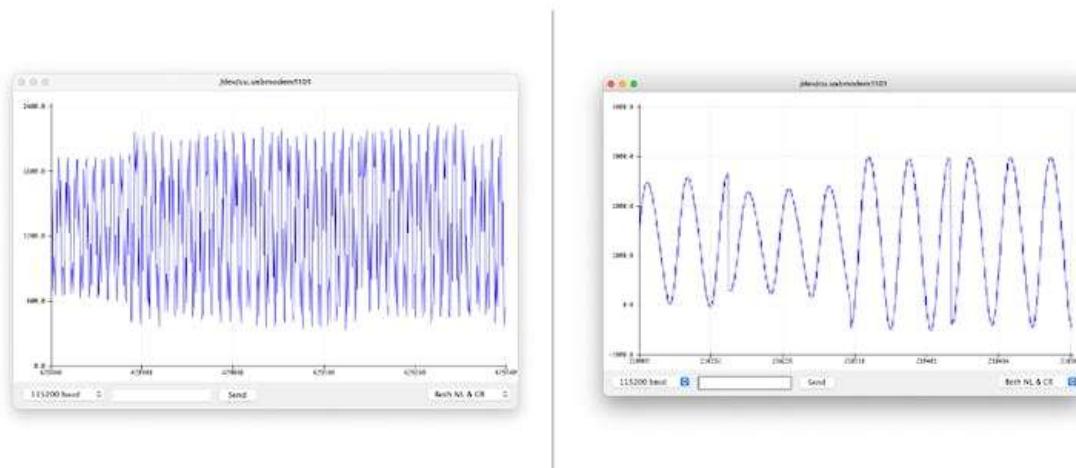
Questo codice è un semplice test del microfono per XIAO ESP32S3 che utilizza l'interfaccia I2S (Inter-IC Sound). Imposta l'interfaccia I2S per catturare dati audio a una frequenza di campionamento di 16 kHz con 16 bit per campione e quindi legge continuamente i campioni dal microfono e li stampa sul monitor seriale.

Analizziamo le parti principali del codice:

- Include la libreria I2S: Questa libreria fornisce funzioni per configurare e utilizzare l'[interfaccia I2S](#), che è uno standard per la connessione di dispositivi audio digitali.
- I2S.setAllPins(-1, 42, 41, -1, -1): Imposta i pin I2S. I parametri sono (-1, 42, 41, -1, -1), dove il secondo parametro (42) è il PIN per il clock I2S (CLK) e il terzo parametro (41) è il PIN per la linea dati I2S (DATA). Gli altri parametri sono impostati su -1, il che significa che quei pin non vengono utilizzati.
- I2S.begin(PDM_MONO_MODE, 16000, 16): Inizializza l'interfaccia I2S in modalità mono Pulse Density Modulation (PDM), con una frequenza di campionamento di 16 kHz e 16 bit per campione. Se l'inizializzazione fallisce, viene stampato un messaggio di errore e il programma si arresta.
- int sample = I2S.read(): Legge un campione audio dall'interfaccia I2S.

Se il campione è valido, viene stampato sul monitor seriale e sul plotter.

Di seguito è riportato un test “sussurrato” in due toni diversi.



Salvare campioni audio registrati (dataset) come file audio .wav su una scheda microSD

Utilizziamo il lettore di schede SD integrato per salvare i file audio .wav; dobbiamo prima abilitare la PSRAM XIAO.

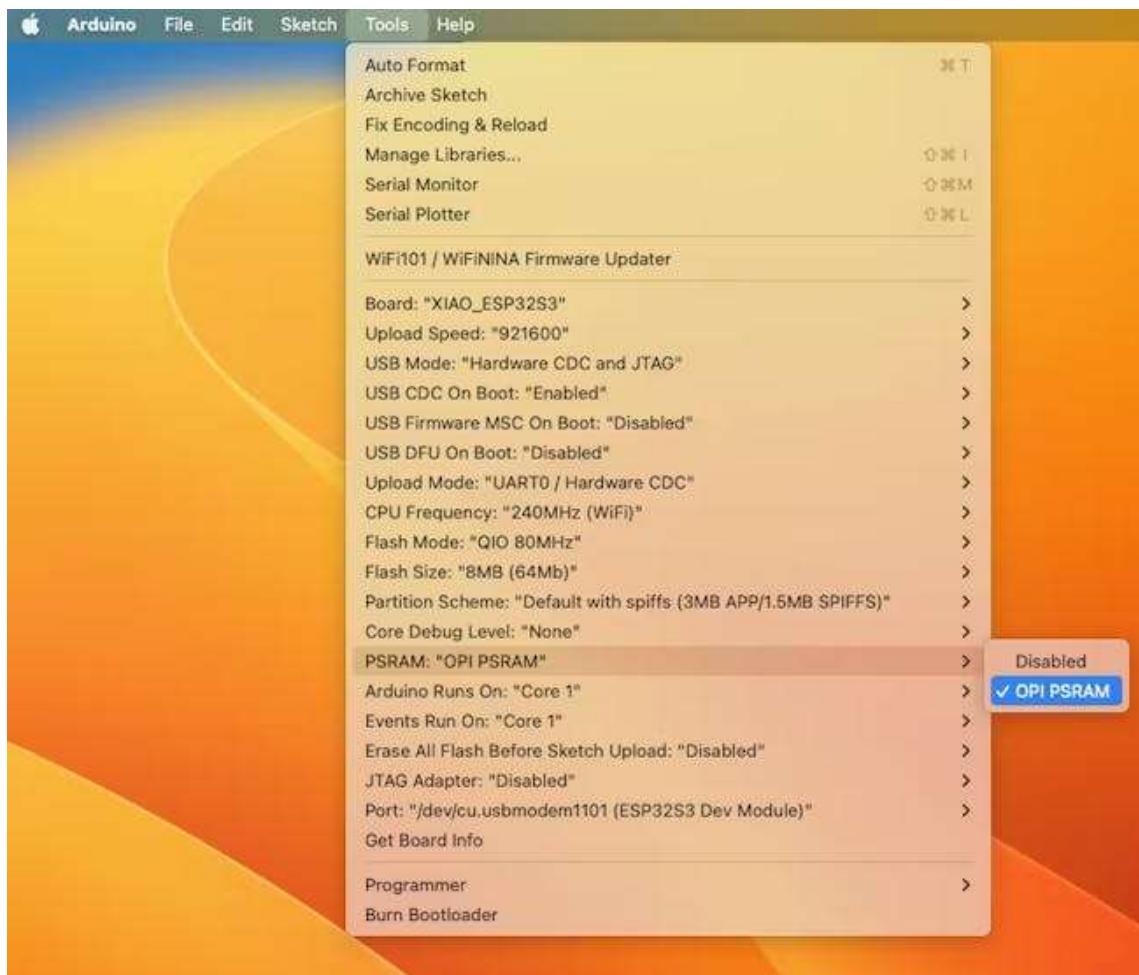
ESP32-S3 ha solo poche centinaia di kilobyte di RAM interna sul chip MCU. Potrebbe essere insufficiente per alcuni scopi, quindi ESP32-S3 può utilizzare fino a 16 MB di PSRAM esterna (Psuedostatic RAM) collegata in parallelo con il chip flash SPI. La

memoria esterna è incorporata nella mappa di memoria e, con alcune restrizioni, è utilizzabile allo stesso modo della RAM dati interna.

Per iniziare, si inserisce la scheda SD sullo XIAO come mostrato nella foto qui sotto (la scheda SD deve essere formattata in FAT32).



Attivare la funzione PSRAM del chip ESP-32 (Arduino IDE): Tools>PSRAM: “OPI PSRAM”>OPI PSRAM



- Scaricare lo sketch [Wav_Record_dataset](https://github.com/Mjrovai/XIAO-ESP32S3-Sense/tree/main/Wav_Record_dataset), https://github.com/Mjrovai/XIAO-ESP32S3-Sense/tree/main/Wav_Record_dataset che si trova sul GitHub del progetto.

Questo codice registra l'audio usando l'interfaccia I2S della scheda Seeed XIAO ESP32S3 Sense, salva la registrazione come file .wav su una scheda SD e consente il controllo del processo di registrazione tramite comandi inviati dal monitor seriale. Il nome del file audio è personalizzabile (dovrebbe essere le etichette della classe da usare con la formazione) e possono essere effettuate più registrazioni, ciascuna salvata in un nuovo file. Il codice include anche funzionalità per aumentare il volume delle registrazioni.

Analizziamo le parti più essenziali:

```
#include <I2S.h>
#include "FS.h"
#include "SD.h"
#include "SPI.h"
```

Queste sono le librerie necessarie per il programma. I2S.h consente l'input audio, FS.h fornisce capacità di gestione del file system, SD.h consente al programma di interagire con una scheda SD e SPI.h gestisce la comunicazione SPI con la scheda SD.

```
#define RECORD_TIME 10
#define SAMPLE_RATE 16000U
#define SAMPLE_BITS 16
#define WAV_HEADER_SIZE 44
#define VOLUME_GAIN 2
```

Qui vengono definite varie costanti per il programma.

- RECORD_TIME** specifica la lunghezza della registrazione audio in secondi.
- SAMPLE_RATE** e **SAMPLE_BITS** definiscono la qualità audio della registrazione.
- WAV_HEADER_SIZE** specifica la dimensione dell'intestazione del file .wav.
- VOLUME_GAIN** viene utilizzato per aumentare il volume della registrazione.

```

int fileNumber = 1;
String baseFileName;
bool isRecording = false;

void setup() {
    Serial.begin(115200);
    while (!Serial);

    I2S.setAllPins(-1, 42, 41, -1, -1);
    if (!I2S.begin(PDM_MONO_MODE, SAMPLE_RATE, SAMPLE_BITS)) {
        Serial.println("Failed to initialize I2S!");
        while (1);
    }

    if(!SD.begin(21)){
        Serial.println("Failed to mount SD Card!");
        while (1);
    }

    Serial.printf("Enter with the label name\n");
}

```

La funzione di configurazione inizializza la comunicazione seriale, l'interfaccia I2S per l'ingresso audio e l'interfaccia della scheda SD. Se l'I2S non si inizializza o la scheda SD non riesce a essere montata, verrà visualizzato un messaggio di errore e l'esecuzione verrà interrotta.

```

void loop() {
    if (Serial.available() > 0) {
        String command = Serial.readStringUntil('\n');
        command.trim();
        if (command == "rec") {
            isRecording = true;
        } else {
            baseFileName = command;
            fileNumber = 1; //reset file number each time a new basefile name is set
            Serial.printf("Send rec for starting recording label \n");
        }
    }

    if (isRecording && baseFileName != "") {
        String fileName = "/" + baseFileName + "." + String(fileNumber) + ".wav";
        fileNumber++;
        record_wav(fileName);
        delay(1000); // delay to avoid recording multiple files at once
        isRecording = false;
    }
}

```

Nel ciclo principale, il programma attende un comando dal monitor seriale. Se il comando è rec, il programma inizia a registrare. Altrimenti, si presume che il comando sia il nome base per i file .wav. Se sta attualmente registrando e un nome file base è impostato, registra l'audio e lo salva come file .wav. I nomi dei file vengono generati aggiungendo il numero al nome file base.

```

void record_wav(String fileName)
{
    ...

    File file = SD.open(fileName.c_str(), FILE_WRITE);
    ...
    rec_buffer = (uint8_t *)ps_malloc(record_size);
    ...

    esp_i2s::i2s_read(esp_i2s::I2S_NUM_0,
                      rec_buffer,
                      record_size,
                      &sample_size,
                      portMAX_DELAY);
}

```

```
    ...
}
```

Questa funzione registra l'audio e lo salva come file .wav con il nome specificato. Inizia inizializzando le variabili sample_size e record_size. record_size viene calcolato in base alla frequenza di campionamento, alla dimensione e al tempo di registrazione desiderato.

Analizziamo le sezioni essenziali;

```
File file = SD.open(fileName.c_str(), FILE_WRITE);
// Write the header to the WAV file
uint8_t wav_header[WAV_HEADER_SIZE];
generate_wav_header(wav_header, record_size, SAMPLE_RATE);
file.write(wav_header, WAV_HEADER_SIZE);
```

Questa sezione del codice apre il file sulla scheda SD per la scrittura e poi genera l'intestazione del file .wav utilizzando la funzione generate_wav_header. Quindi scrive l'intestazione nel file.

```
// PSRAM malloc for recording
rec_buffer = (uint8_t *)ps_malloc(record_size);
if (rec_buffer == NULL) {
    Serial.printf("malloc failed!\n");
    while(1) ;
}
Serial.printf("Buffer: %d bytes\n", ESP.getPsramSize() - ESP.getFreePsram());
```

La funzione ps_malloc alloca memoria nella PSRAM per la registrazione. Se l'allocazione fallisce (ad esempio, rec_buffer è NULL), stampa un messaggio di errore e interrompe l'esecuzione.

```
// Start recording
esp_i2s::i2s_read(esp_i2s::I2S_NUM_0,
    rec_buffer,
    record_size,
    &sample_size,
    portMAX_DELAY);
if (sample_size == 0) {
    Serial.printf("Record Failed!\n");
} else {
    Serial.printf("Record %d bytes\n", sample_size);
}
```

La funzione i2s_read legge i dati audio dal microfono in rec_buffer. Stampa un messaggio di errore se non vengono letti dati (sample_size è 0).

```
// Increase volume
for (uint32_t i = 0; i < sample_size; i += SAMPLE_BITS/8) {
    (*(uint16_t *) (rec_buffer+i)) <<= VOLUME_GAIN;
}
```

Questa sezione del codice aumenta il volume di registrazione spostando i valori del campione di VOLUME_GAIN.

```
// Write data to the WAV file
Serial.printf("Writing to the file ...\\n");
if (file.write(rec_buffer, record_size) != record_size)
    Serial.printf("Write file Failed!\\n");

free(rec_buffer);
file.close();
Serial.printf("Recording complete: \\n");
Serial.printf("Send rec for a new sample or enter a new label\\n\\n");
```

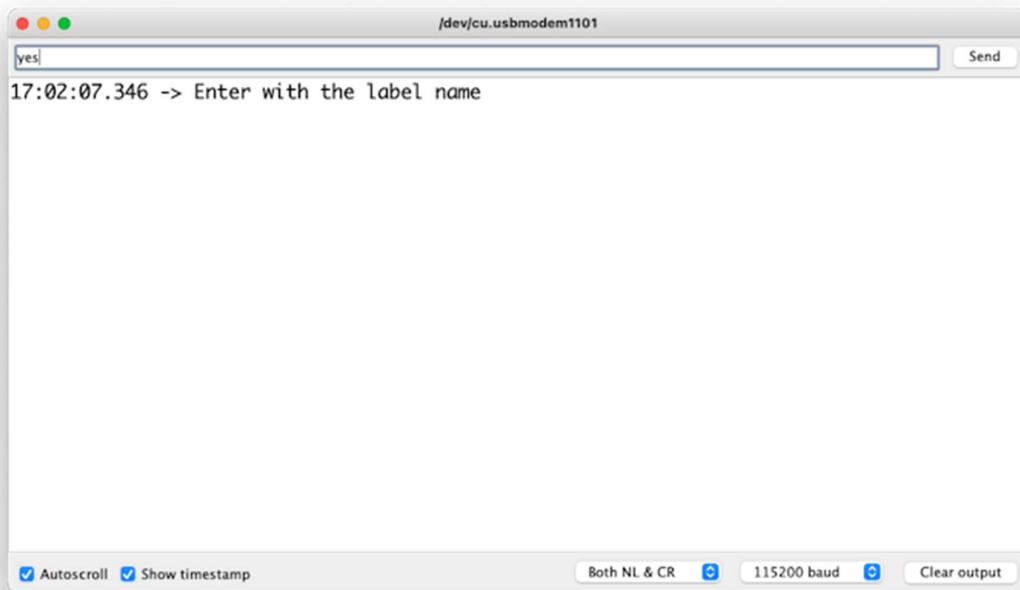
Infine, i dati audio vengono scritti nel file .wav. Se l'operazione di scrittura fallisce, viene stampato un messaggio di errore. Dopo la scrittura, la memoria allocata per rec_buffer viene liberata e il file viene chiuso. La funzione termina stampando un messaggio di completamento e chiedendo all'utente di inviare un nuovo comando.

```
void generate_wav_header(uint8_t *wav_header,
    uint32_t wav_size,
    uint32_t sample_rate)
{
    ...
    memcpy(wav_header, set_wav_header, sizeof(set_wav_header));
}
```

La funzione generate_wav_header crea un'intestazione di file .wav in base ai parametri (wav_size e sample_rate). Genera un array di byte in base al formato di file .wav, che include campi per la dimensione del file, il formato audio, il numero di canali, la frequenza di campionamento, la frequenza di byte, l'allineamento dei blocchi, i bit per campione e la dimensione dei dati. L'intestazione generata viene poi copiata nell'array wav_header passato alla funzione.

Ora, caricare il codice su XIAO e ottenere campioni dalle parole chiave (yes e no). Si possono anche catturare rumore e altre parole.

Il monitor seriale chiederà di ricevere l'etichetta da registrare.

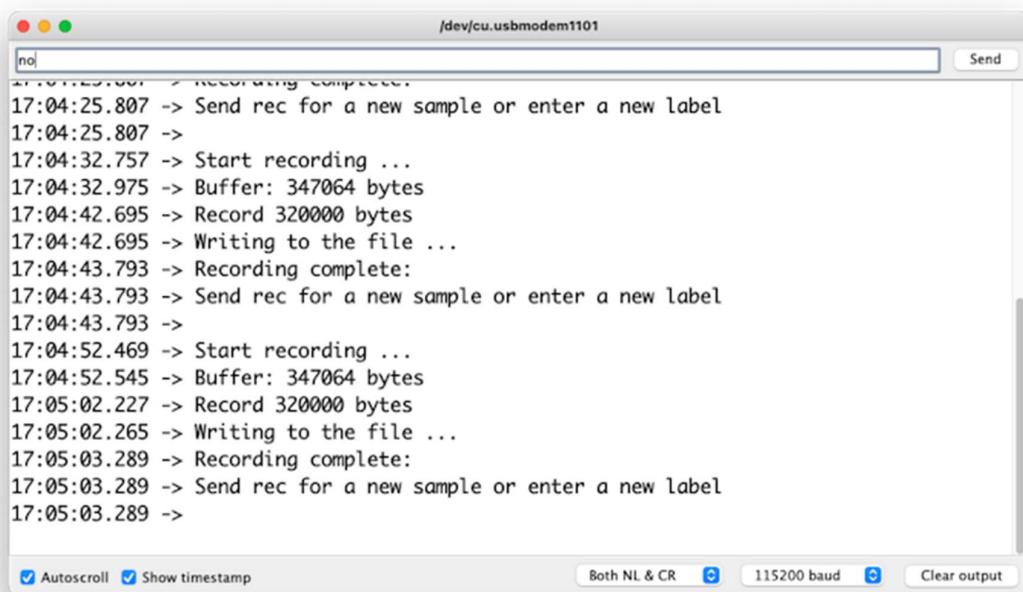


Invia l'etichetta (ad esempio, yes). Il programma attenderà un altro comando: rec



E il programma inizierà a registrare nuovi campioni ogni volta che viene inviato un comando rec. I file verranno salvati come yes.1.wav, yes.2.wav, yes.3.wav, ecc., finché non verrà

invia una nuova etichetta (ad esempio, no). In questo caso, si deve inviare il comando rec per ogni nuovo campione, che verrà salvato come no.1.wav, no.2.wav, no.3.wav, ecc.



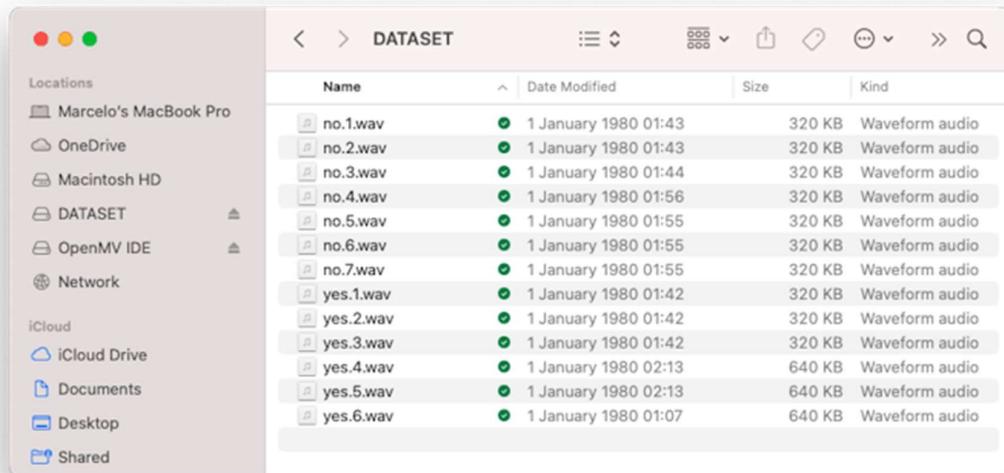
```

no] 17:04:25.807 -> Recording complete.
17:04:25.807 -> Send rec for a new sample or enter a new label
17:04:25.807 ->
17:04:32.757 -> Start recording ...
17:04:32.975 -> Buffer: 347064 bytes
17:04:42.695 -> Record 320000 bytes
17:04:42.695 -> Writing to the file ...
17:04:43.793 -> Recording complete:
17:04:43.793 -> Send rec for a new sample or enter a new label
17:04:43.793 ->
17:04:52.469 -> Start recording ...
17:04:52.545 -> Buffer: 347064 bytes
17:05:02.227 -> Record 320000 bytes
17:05:02.265 -> Writing to the file ...
17:05:03.289 -> Recording complete:
17:05:03.289 -> Send rec for a new sample or enter a new label
17:05:03.289 ->

```

Autoscroll Show timestamp Both NL & CR 115200 baud Clear output

Alla fine, otterremo i file salvati sulla scheda SD.

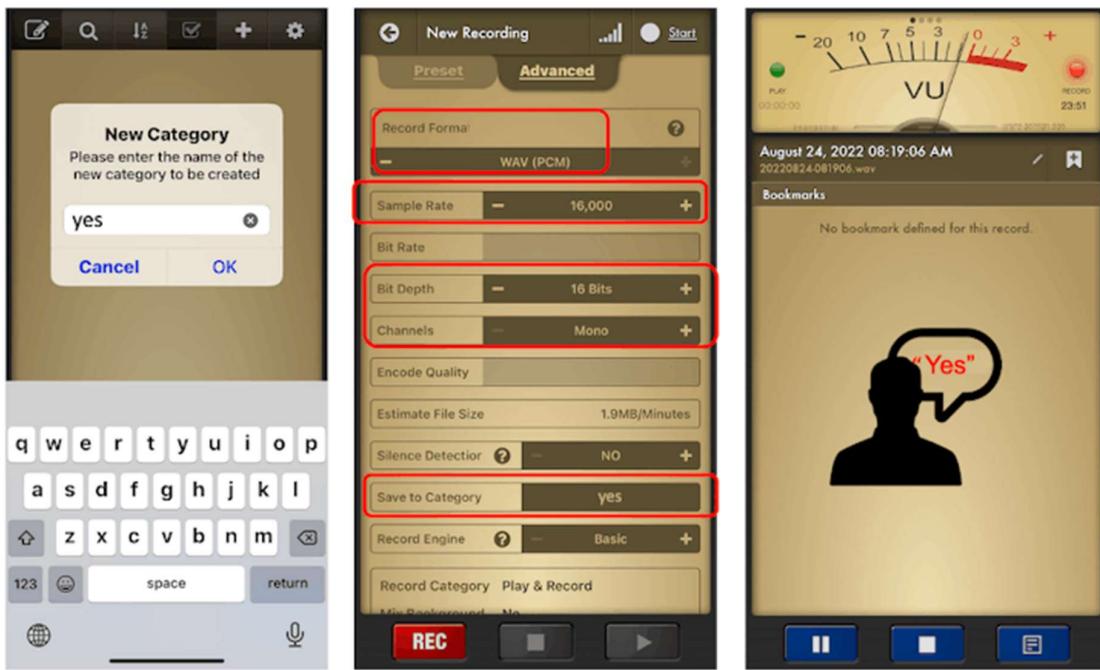


Name	Date Modified	Size	Kind
no.1.wav	1 January 1980 01:43	320 KB	Waveform audio
no.2.wav	1 January 1980 01:43	320 KB	Waveform audio
no.3.wav	1 January 1980 01:44	320 KB	Waveform audio
no.4.wav	1 January 1980 01:56	320 KB	Waveform audio
no.5.wav	1 January 1980 01:55	320 KB	Waveform audio
no.6.wav	1 January 1980 01:55	320 KB	Waveform audio
no.7.wav	1 January 1980 01:55	320 KB	Waveform audio
yes.1.wav	1 January 1980 01:42	320 KB	Waveform audio
yes.2.wav	1 January 1980 01:42	320 KB	Waveform audio
yes.3.wav	1 January 1980 01:42	320 KB	Waveform audio
yes.4.wav	1 January 1980 02:13	640 KB	Waveform audio
yes.5.wav	1 January 1980 02:13	640 KB	Waveform audio
yes.6.wav	1 January 1980 01:07	640 KB	Waveform audio

I file sono pronti per essere caricati su Edge Impulse Studio

App di Acquisizione dei Dati Audio (offline)

In alternativa, si può anche usare il PC o lo smartphone per acquisire dati audio con una frequenza di campionamento di 16 KHz e una profondità di bit di 16 bit. Una buona app per questo è [Voice Recorder Pro](#) (IOS). Le registrazioni si devono salvare come file .wav e inviarle al computer.

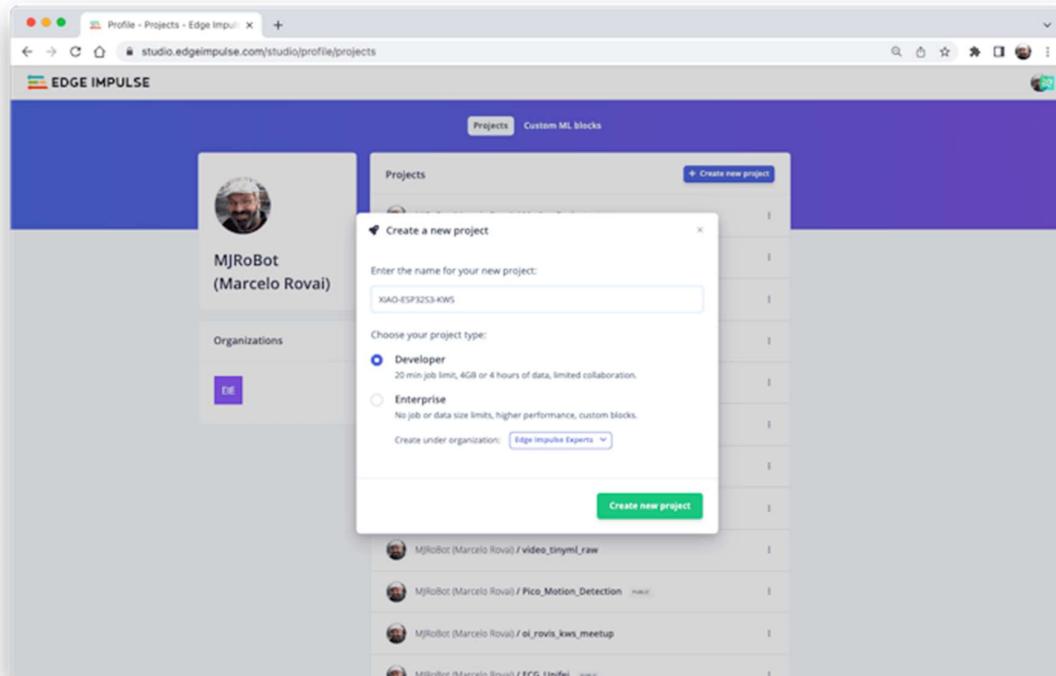


Notare che qualsiasi app, come [Audacity](#), può essere usata per la registrazione audio o anche il computer.

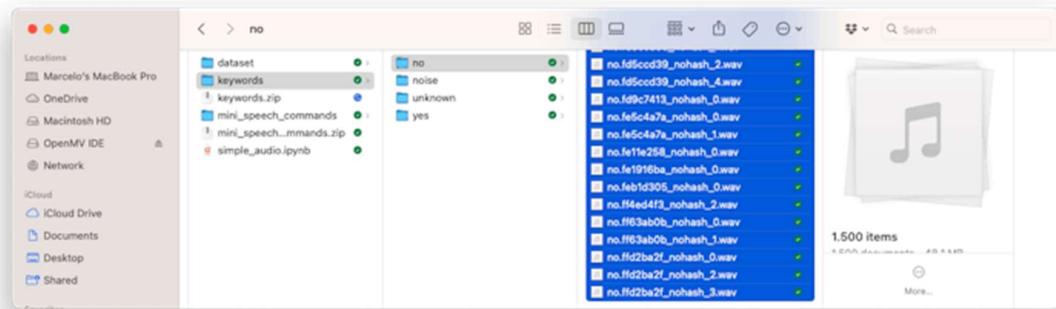
Modello di training con Edge Impulse Studio

Caricamento dei Dati

Quando il dataset grezzo è definito e raccolto (dataset di Pete + parole chiave registrate), dovremmo avviare un nuovo progetto in Edge Impulse Studio:



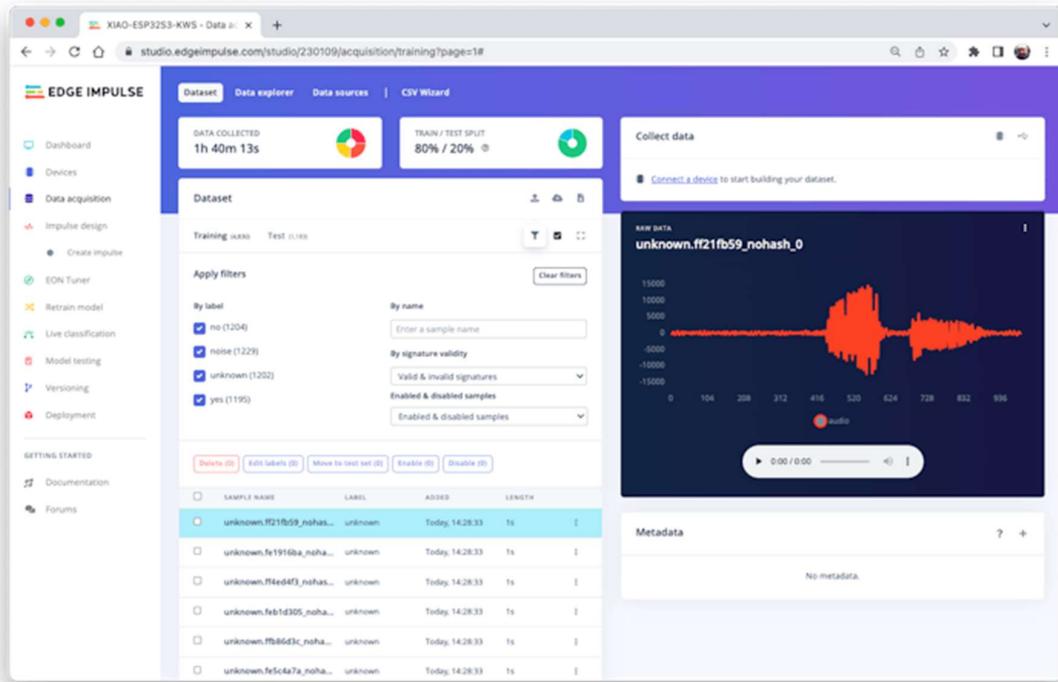
Una volta creato il progetto, selezionare lo strumento “Upload Existing Data” nella sezione “Acquisition section”. Si scelgono i file da caricare:



E si caricano nello Studio (si possono dividere automaticamente i dati in train/test). Ripetere per tutte le classi e tutti i dati grezzi.

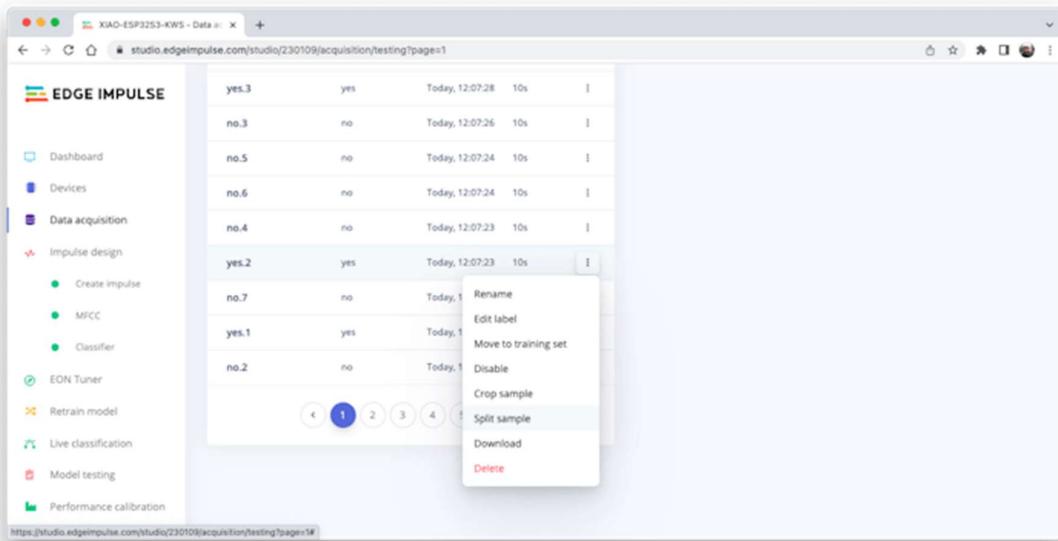
A screenshot of the Edge Impulse web interface. On the left, a sidebar lists project options like Dashboard, Devices, Data acquisition, and Model testing. The 'Data acquisition' section is active. In the center, a modal dialog titled 'Upload data' is open, showing a file selection area with 'Choose Files 1500 files' and an 'Upload output' section with a progress bar. The progress bar shows 'Uploading...' and a list of 1500 items being processed. Below the modal, a table lists five rows of uploaded files with columns for name, label, date, duration, and a delete icon. At the bottom right of the modal, it says 'Metadata' and 'No metadata.'

I campioni appariranno ora nella sezione “Data acquisition”.

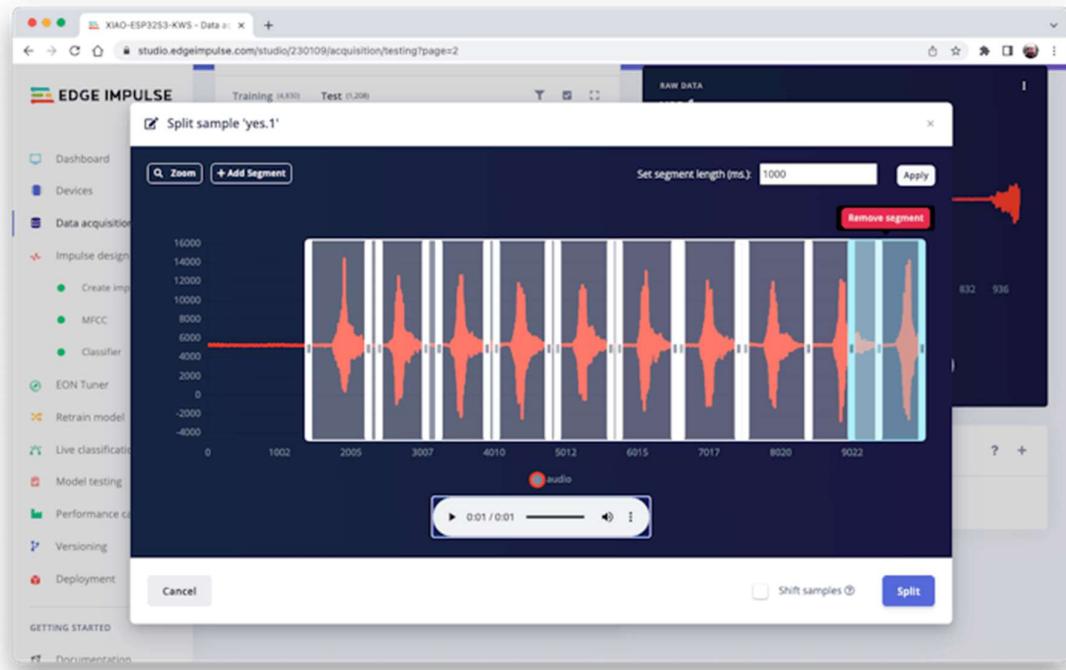


Tutti i dati sul dataset di Pete hanno una lunghezza di 1s, ma i campioni registrati nella sezione precedente hanno 10s e devono essere divisi in campioni da 1s per essere compatibili.

Cliccare sui tre punti dopo il nome del campione e selezionare “Split sample”.



Una volta all'interno dello strumento, dividere i dati in record da 1 secondo. Se necessario, aggiungere o rimuovere segmenti:



Questa procedura deve essere ripetuta per tutti i campioni.

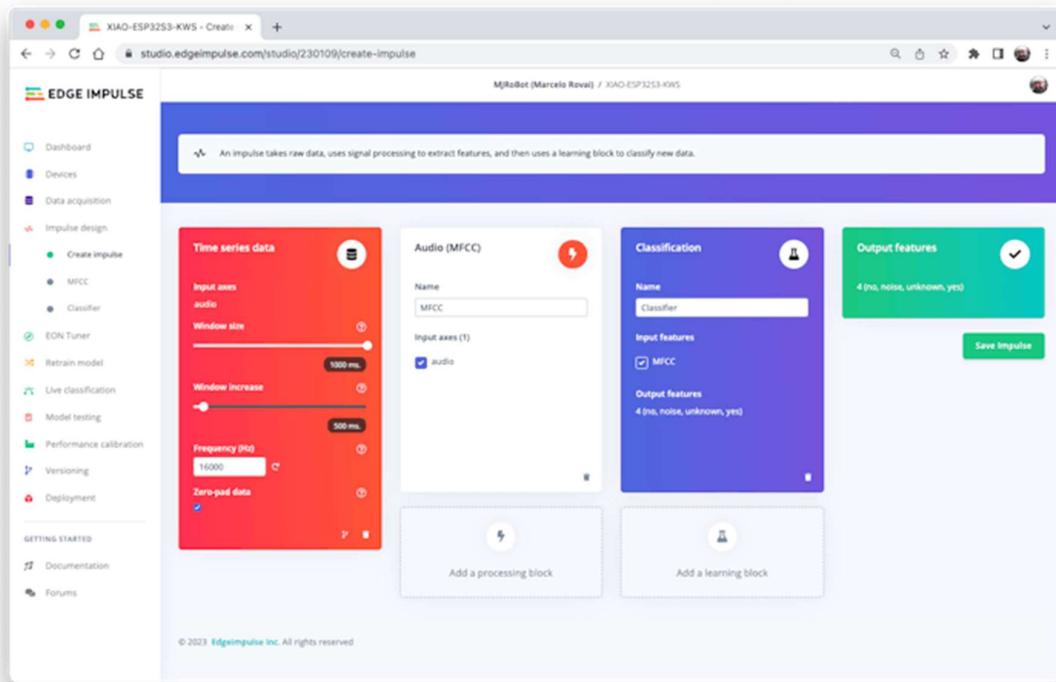
Nota: Per file audio più lunghi (minuti), prima si dividono in segmenti da 10 secondi e poi usa di nuovo lo strumento per ottenere le divisioni finali da 1 secondo.

Supponiamo di non dividere automaticamente i dati in train/test durante il caricamento. In tal caso, possiamo farlo manualmente (utilizzando il menù a tre punti, spostando i campioni singolarmente) o utilizzando Perform Train / Test Split su Dashboard - Danger Zone.

Possiamo facoltativamente controllare tutti i dataset utilizzando la scheda Data Explorer.

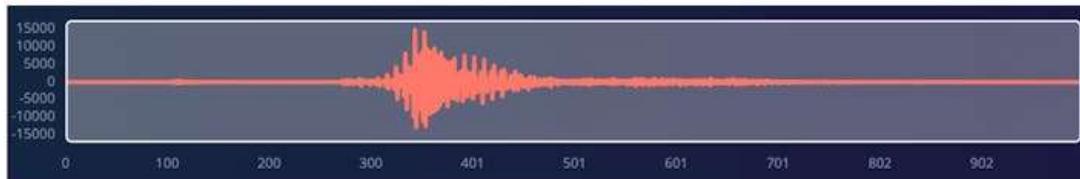
Creazione di Impulse (Pre-Process / Definizione del Modello)

Un impulse prende dati grezzi, usa l'elaborazione del segnale per estrarre le feature e poi usa un blocco di apprendimento per classificare nuovi dati.

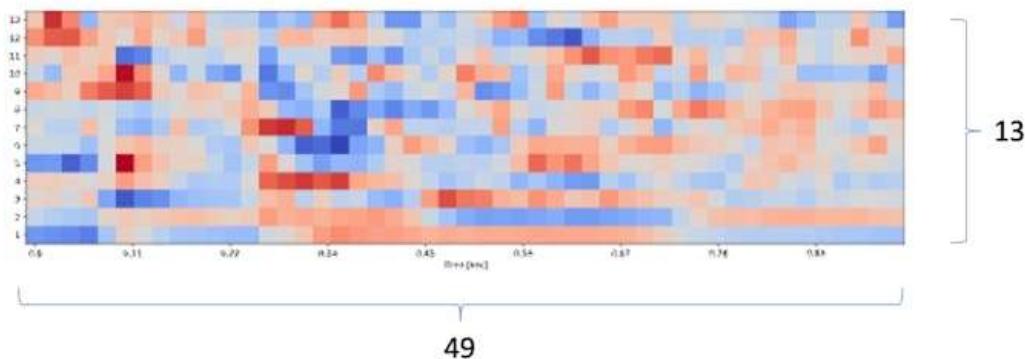


Per prima cosa, prenderemo i dati con una finestra di 1 secondo, aumentando i dati, facendo scorrere quella finestra ogni 500 ms. Notare che è impostata l'opzione Zero-pad data. È essenziale riempire con zeri i campioni inferiori a 1 secondo (in alcuni casi, si è ridotta la finestra di 1000 ms sullo strumento di divisione per evitare rumori e picchi). Ogni campione audio di 1 secondo dovrebbe essere pre-elaborato e convertito in un'immagine (ad esempio, $13 \times 49 \times 1$). Useremo MFCC, che estrae le caratteristiche dai segnali audio usando **Mel Frequency Cepstral Coefficients**, che sono ottimi per la voce umana.

Raw data → 16,000 features



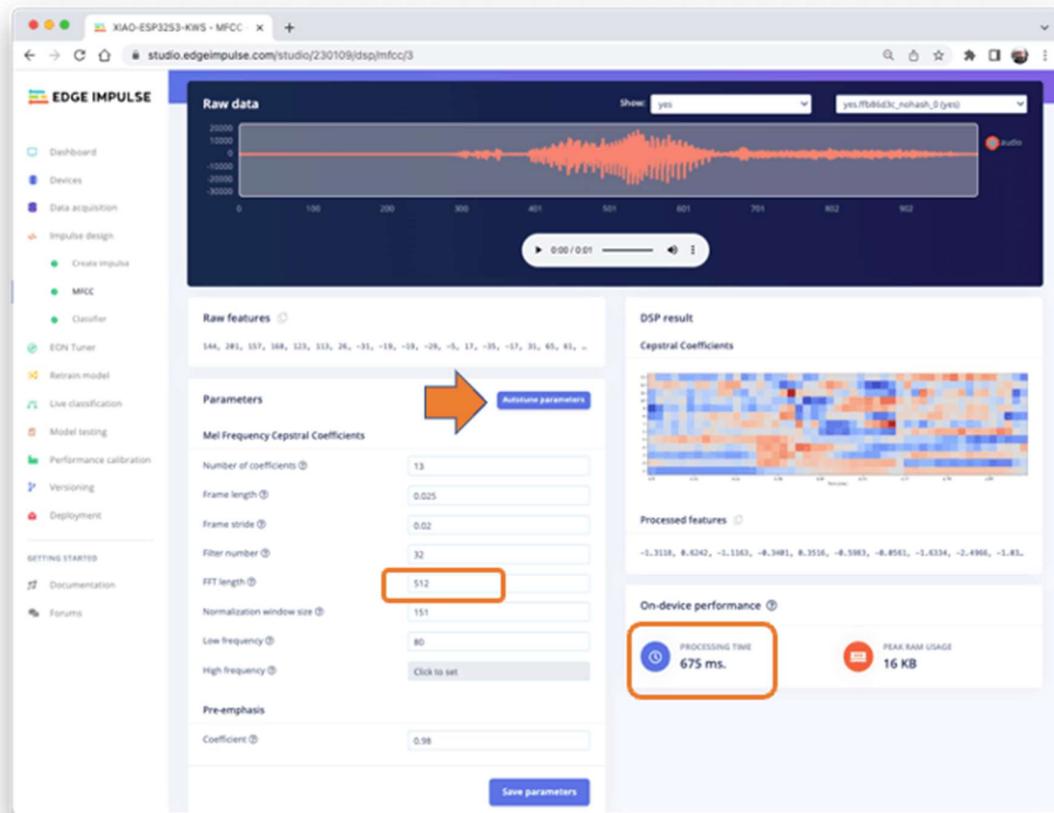
Processed features → 637 features (13×49)



Successivamente, selezioniamo KERAS per la classificazione e costruiamo il nostro modello da zero eseguendo la classificazione delle immagini tramite la rete neurale convoluzionale).

Pre-elaborazione (MFCC)

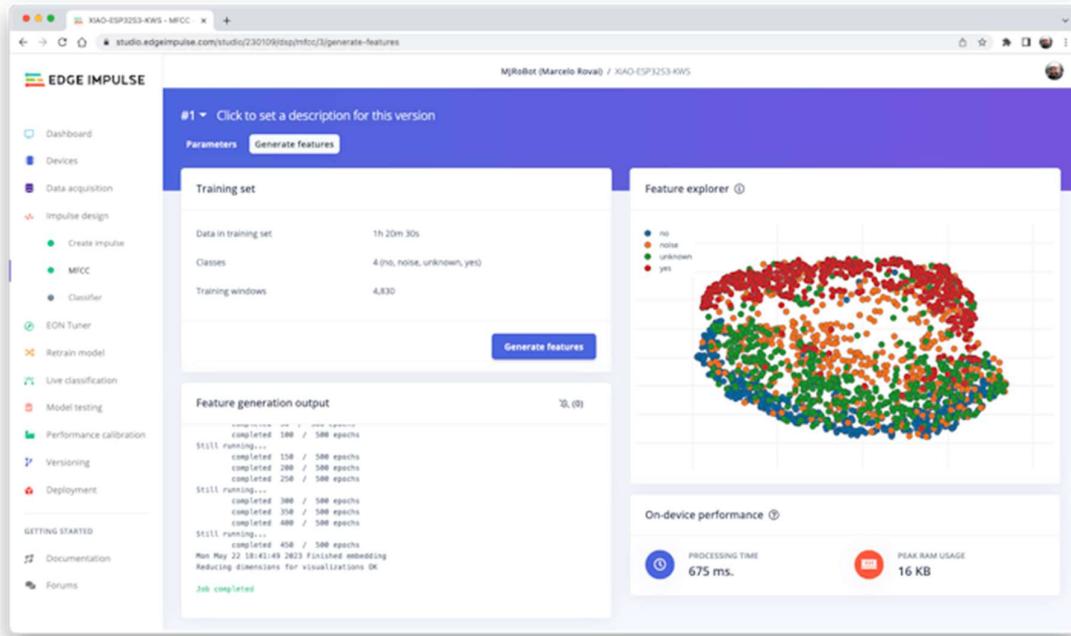
Il passo successivo è creare le immagini da addestrare nella fase successiva: Possiamo mantenere i valori dei parametri di default o sfruttare l'opzione DSP Autotune parameters, cosa che faremo.



Il risultato non impiegherà molta memoria per pre-elaborare i dati (solo 16 KB). Tuttavia, il tempo di elaborazione stimato è elevato, 675 ms per un Espressif ESP-EYE (il riferimento più vicino disponibile), con un clock di 240 KHz (lo stesso del nostro dispositivo), ma con una CPU più piccola (XTensa LX6, rispetto alla LX7 sull'ESP32S). Il tempo di inferenza reale dovrebbe essere inferiore.

Supponiamo di dover ridurre il tempo di inferenza in seguito. In tal caso, dovremmo tornare alla fase di pre-elaborazione e, ad esempio, ridurre la lunghezza FFT a 256, modificare il numero di coefficienti o un altro parametro.

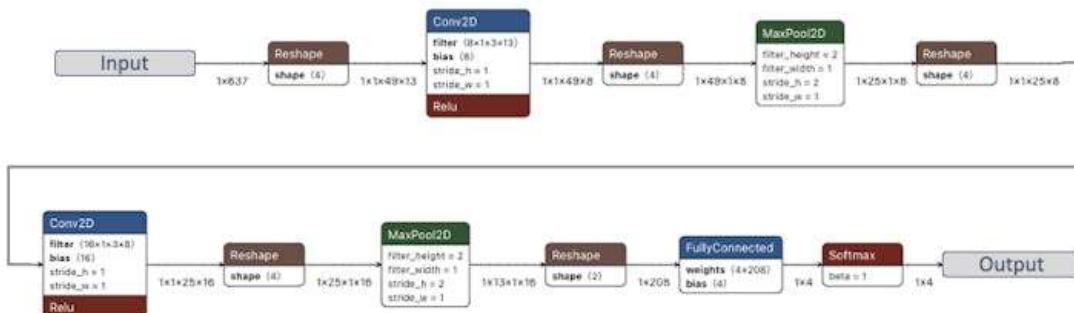
Per ora, manteniamo i parametri definiti dallo strumento Autotuning. Salviamo i parametri e generiamo le funzionalità.



Per andare oltre con la conversione di dati seriali temporali in immagini usando FFT, spettrogramma, ecc., si può giocare con questo CoLab: [Audio Raw Data Analysis](#).

Progettazione e Addestramento del Modello

Useremo un modello di Rete Neurale Convoluzionale (CNN). L'architettura di base è definita con due blocchi di Conv1D + MaxPooling (rispettivamente con 8 e 16 neuroni) e un Dropout di 0,25. E sull'ultimo layer, dopo aver appiattito quattro neuroni, uno per ogni classe:



Come iperparametri, avremo un Learning Rate di 0,005 e un modello che verrà addestrato per 100 epoche. Includeremo anche l'aumento dei dati, come un po' di rumore. Il risultato sembra OK:

Model

Model version: ② Quantized (int8) •

Last training performance (validation set)

ACCURACY 90.7% LOSS 0.25

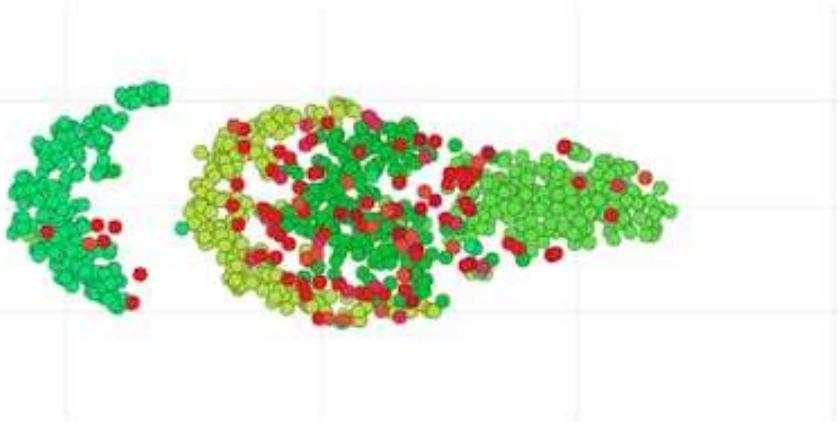
Confusion matrix (validation set)

	NO	NOISE	UNKNOWN	YES
NO	92.2%	0.8%	5.1%	1.6%
NOISE	0.4%	95.2%	4.0%	0.4%
UNKNOWN	10.2%	5.1%	82.0%	2.7%
YES	2.1%	0.4%	3.3%	94.3%
F1 SCORE	0.90	0.94	0.85	0.95

Data explorer (full training set) ②

Legend:

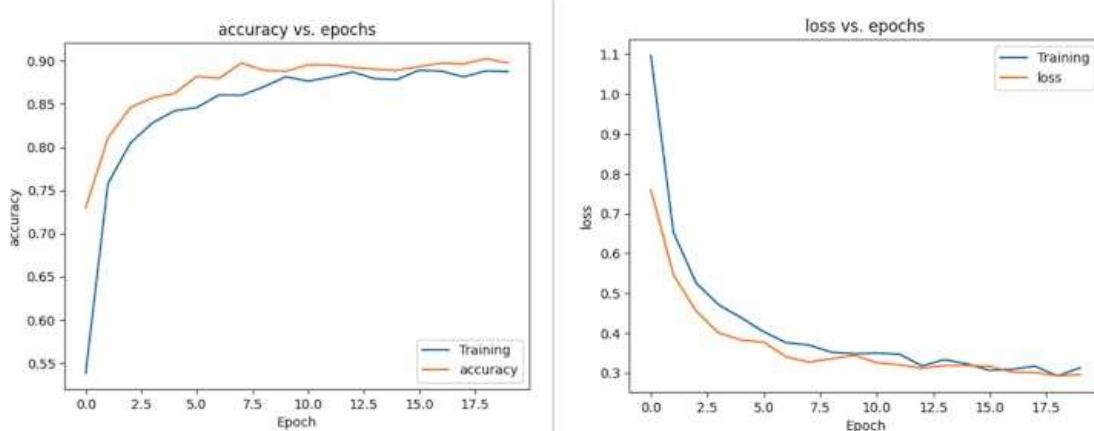
- no - correct
- noise - correct
- unknown - correct
- yes - correct
- no - incorrect
- noise - incorrect
- unknown - incorrect
- yes - incorrect



On-device performance ②

INFERENCING TIME: 6 ms. PEAK RAM USAGE: 3.7K FLASH USAGE: 27.1K

Per capire cosa sta succedendo “sotto il cofano”, si può scaricare il dataset ed eseguire un Jupyter Notebook giocando con il codice. Ad esempio, si può analizzare l’accuratezza per ogni epoca:



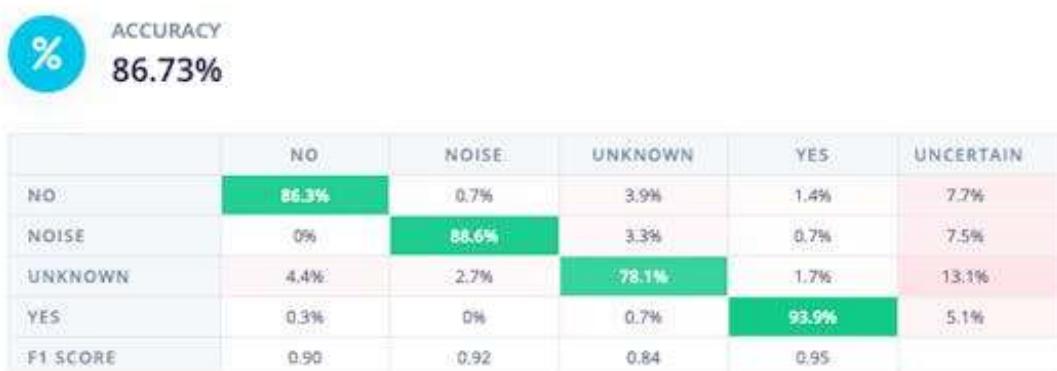
Questo CoLab Notebook può spiegare come si può andare oltre: [KWS Classifier Project - Looking “Under the hood](#)

Training/xiao_esp32s3_keyword_spotting_project_nn_classifier.ipynb”.

Test

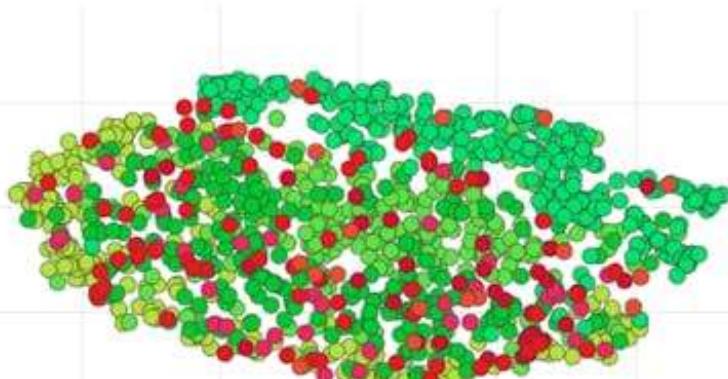
Testando il modello con i dati messi da parte prima dell’addestramento (Test Data), abbiamo ottenuto un’accuratezza di circa l’87%.

Model testing results



Feature explorer ②

- no - correct
- noise - correct
- unknown - correct
- yes - correct
- no - incorrect
- noise - incorrect
- unknown - incorrect
- yes - incorrect

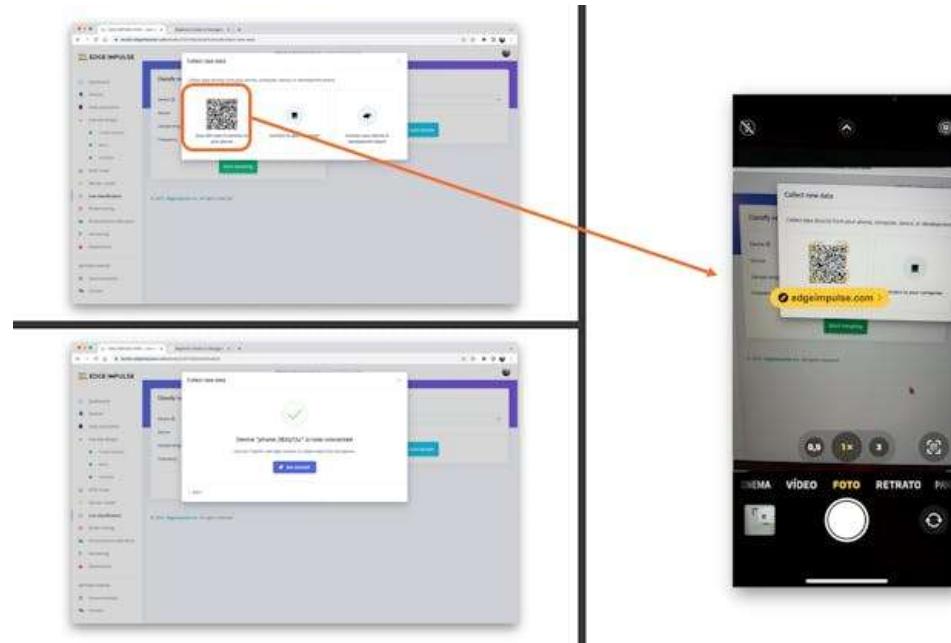


Esaminando il punteggio F1, possiamo vedere che per YES abbiamo ottenuto 0.95, un risultato eccellente una volta utilizzata questa parola chiave per “attivare” la nostra fase di post-elaborazione (accendere il LED integrato). Anche per NO, abbiamo ottenuto 0,90. Il risultato peggiore è per unknown, che è OK.

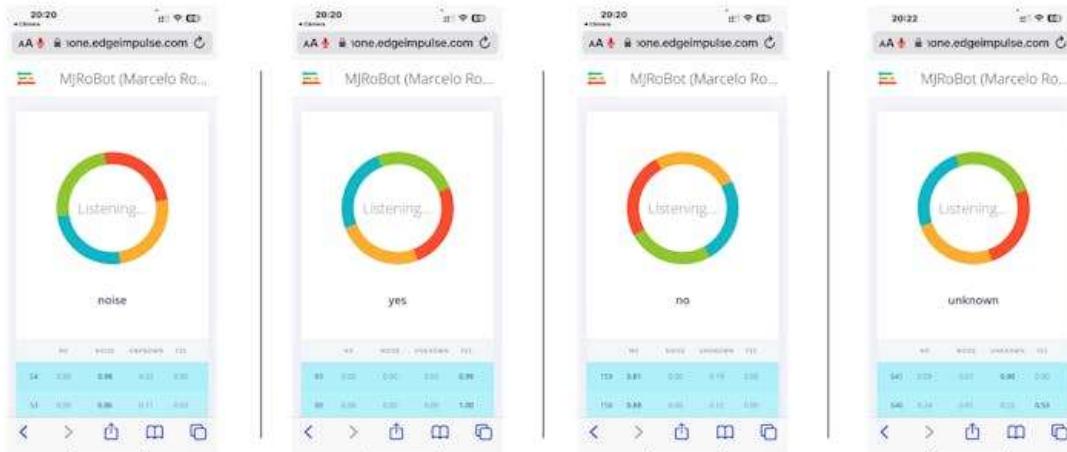
Possiamo procedere con il progetto, ma è possibile eseguire Live Classification utilizzando uno smartphone prima della distribuzione sul nostro dispositivo. Si va alla sezione Live Classification e si clicca su Connect a Development board:



Puntare il telefono sul codice a barre e selezionare il link.



Il telefono sarà connesso allo Studio. Selezionare l'opzione Classification sull'app e, quando è in esecuzione, iniziare a testare le parole chiave, confermando che il modello funziona con dati live e reali:



Distribuzione e Inferenza

Studio impacchetterà tutte le librerie necessarie, le funzioni di pre-elaborazione e i modelli addestrati, scaricandoli sul computer. Si deve selezionare l'opzione Arduino Library e, in basso, scegliere Quantized (Int8) e premere il pulsante Build.

The screenshot shows the TensorFlow.js Model Optimizer interface. At the top, it says "Configure your deployment". Below that, a note states: "You can deploy your impulse to any device. This makes the model run without an internet connection, minimizes latency, and runs with minimal power consumption. [Read more...](#)".

SELECTED DEPLOYMENT:

Arduino library
An Arduino library with examples that runs on most Arm-based Arduino development boards.

MODEL OPTIMIZATIONS:
Model optimizations can increase on-device performance but may reduce accuracy.

Enable EON™ Compiler Same accuracy, up to 50% less memory. [Learn more](#)

Quantized (Int8)	MFCC	CLASSIFIER	TOTAL
Selected ✓	675 ms.	6 ms.	681 ms.
LATENCY	675 ms.	6 ms.	681 ms.
RAM	13.6K	6.0K	15.6K
FLASH	-	49.9K	-
ACCURACY	-	-	-

Unoptimized (float32)	MFCC	CLASSIFIER	TOTAL
Select	675 ms.	31 ms.	706 ms.
LATENCY	675 ms.	31 ms.	706 ms.
RAM	15.6K	10.5K	15.6K
FLASH	-	53.2K	-
ACCURACY	-	-	-

To compare model accuracy, run model testing... **Run model testing**

Change target: **esp32** | [Change target...](#)

Build

Ora è il momento di un vero test. Faremo inferenze completamente scollegate da Studio. Modifichiamo uno degli esempi di codice ESP32 creati quando si distribuisce la libreria Arduino.

Nell'IDE Arduino, si va alla scheda File/Examples, si cerca il progetto e si seleziona esp32/esp32_microphone:



Questo codice è stato creato per il microfono integrato ESP-EYE, che dovrebbe essere adattato al nostro dispositivo.

Iniziare a modificare le librerie per gestire il bus I2S:

```

41 /* Includes -----
42 #include <XIAO-ESP32S3-KWS_inferencing.h>
43
44 #include "freertos/FreeRTOS.h"
45 #include "freertos/task.h"
46
47 #include "driver/i2s.h"
48

```

Da:

```
#include <I2S.h>
#define SAMPLE_RATE 16000U
#define SAMPLE_BITS 16
```

Inizializzare il microfono I2S in setup(), includendo le righe:

```

void setup()
{
...
    I2S.setAllPins(-1, 42, 41, -1, -1);
    if (!I2S.begin(PDM_MONO_MODE, SAMPLE_RATE, SAMPLE_BITS)) {
        Serial.println("Failed to initialize I2S!");
        while (1) ;
    }
...
}
```

Nella funzione static void capture_samples(void* arg), sostituire la riga 153 che legge i dati dal microfono I2S:

```

145 static void capture_samples(void* arg) {
146
147     const int32_t i2s_bytes_to_read = (uint32_t)arg;
148     size_t bytes_read = i2s_bytes_to_read;
149
150     while (record_status) {
151
152         /* read data at once from i2s */
153         i2s_read((i2s_port_t)1, (void*)sampleBuffer, i2s_bytes_to_read, &bytes_read, 100);
154

```

Da:

```
/* read data at once from i2s */
esp_i2s::i2s_read(esp_i2s::I2S_NUM_0,
                  (void*)sampleBuffer,
                  i2s_bytes_to_read,
                  &bytes_read, 100);
```

Nella funzione static bool microphone_inference_start(uint32_t n_samples), dovremmo commentare o eliminare le righe da 198 a 200, dove viene chiamata la funzione di inizializzazione del microfono. Ciò non è necessario perché il microfono I2S è già stato inizializzato durante setup().

```

186 static bool microphone_inference_start(uint32_t n_samples)
187 {
188     inference.buffer = (int16_t *)malloc(n_samples * sizeof(int16_t));
189
190     if(inference.buffer == NULL) {
191         return false;
192     }
193
194     inference.buf_count = 0;
195     inference.n_samples = n_samples;
196     inference.buf_ready = 0;
197
198 //    if (i2s_init(EI_CLASSIFIER_FREQUENCY)) {
199 //        ei_printf("Failed to start I2S!");
200 //    }
201

```

Infine, nella funzione static void microphone_inference_end(void), sostituire la riga 243:

```

241 static void microphone_inference_end(void)
242 {
243     i2s_deinit();
244     ei_free(inference.buffer);
245 }

```

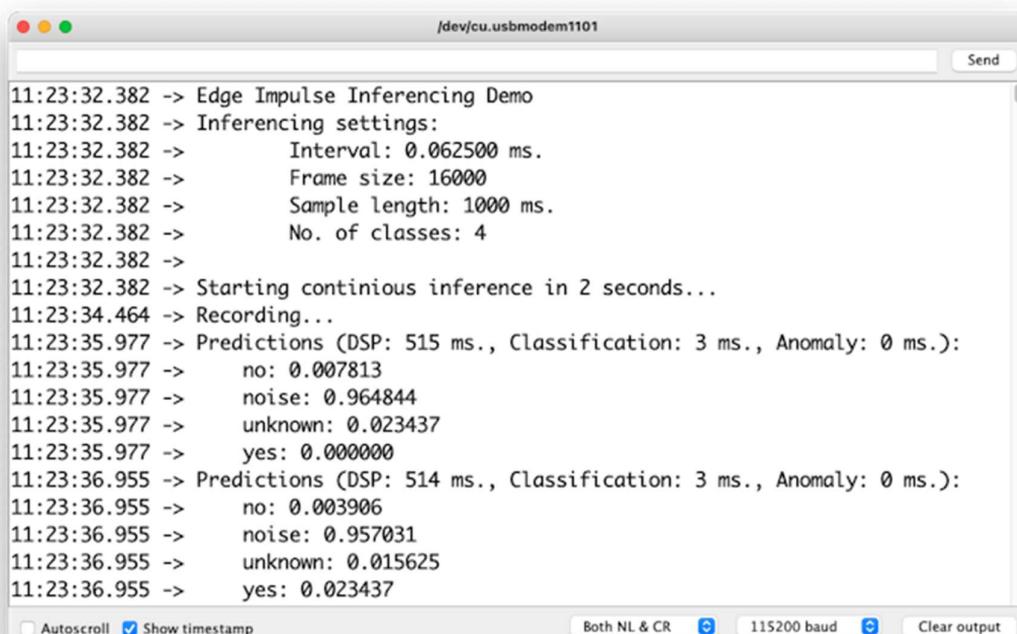
Da:

```

static void microphone_inference_end(void)
{
    free(sampleBuffer);
    ei_free(inference.buffer);
}

```

Il codice completo si trova tra i [progetti GitHub](#). Caricare lo sketch sulla bacheca e provare alcune inferenze reali:



Post-elaborazione

Ora che sappiamo che il modello funziona rilevando le nostre parole chiave, modifichiamo il codice per vedere il LED interno accendersi ogni volta che viene rilevato uno YES.

Si deve inizializzare il LED:

```
#define LED_BUILT_IN 21
...
void setup()
{
...
    pinMode(LED_BUILT_IN, OUTPUT); // Set the pin as output
    digitalWrite(LED_BUILT_IN, HIGH); //Turn off
...
}
```

E modifica la parte “// print the predictions” del codice precedente (su loop()):

```
int pred_index = 0;      // Initialize pred_index
float pred_value = 0;    // Initialize pred_value

// print the predictions
ei_printf("Predictions ");
ei_printf("(DSP: %d ms., Classification: %d ms., Anomaly: %d ms.)",
         result.timing.dsp, result.timing.classification, result.timing.anomaly);
ei_printf(": \n");
for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
    ei_printf("    %s: ", result.classification[ix].label);
    ei_printf_float(result.classification[ix].value);
    ei_printf("\n");

    if (result.classification[ix].value > pred_value) {
        pred_index = ix;
        pred_value = result.classification[ix].value;
    }
}

// show the inference result on LED
if (pred_index == 3){
    digitalWrite(LED_BUILT_IN, LOW); //Turn on
}
else{
    digitalWrite(LED_BUILT_IN, HIGH); //Turn off
}
```

Il codice completo si trova sul [GitHub del progetto](#). Caricare lo sketch sulla scheda e provare alcune inferenze reali:



L'idea è che il LED sarà ACCESO ogni volta che viene rilevata la parola chiave YES. Allo stesso modo, invece di accendere un LED, questo potrebbe essere un “trigger” per un dispositivo esterno, come abbiamo visto nell'introduzione.

Conclusione

The Seeed XIAO ESP32S3 Sense è un *tiny device gigante!* Tuttavia, è potente, affidabile, non costoso, a basso consumo e ha sensori adatti per essere utilizzato nelle applicazioni di apprendimento automatico embedded più comuni come visione e suono. Anche se Edge Impulse non supporta ufficialmente XIAO ESP32S3 Sense (ancora!), ci siamo resi conto che utilizzare Studio per la formazione e l'implementazione è semplice.

Nel [repository GitHub](#), si trova l'ultima versione di tutto il codice utilizzato in questo progetto e nei precedenti della serie XIAO ESP32S3.

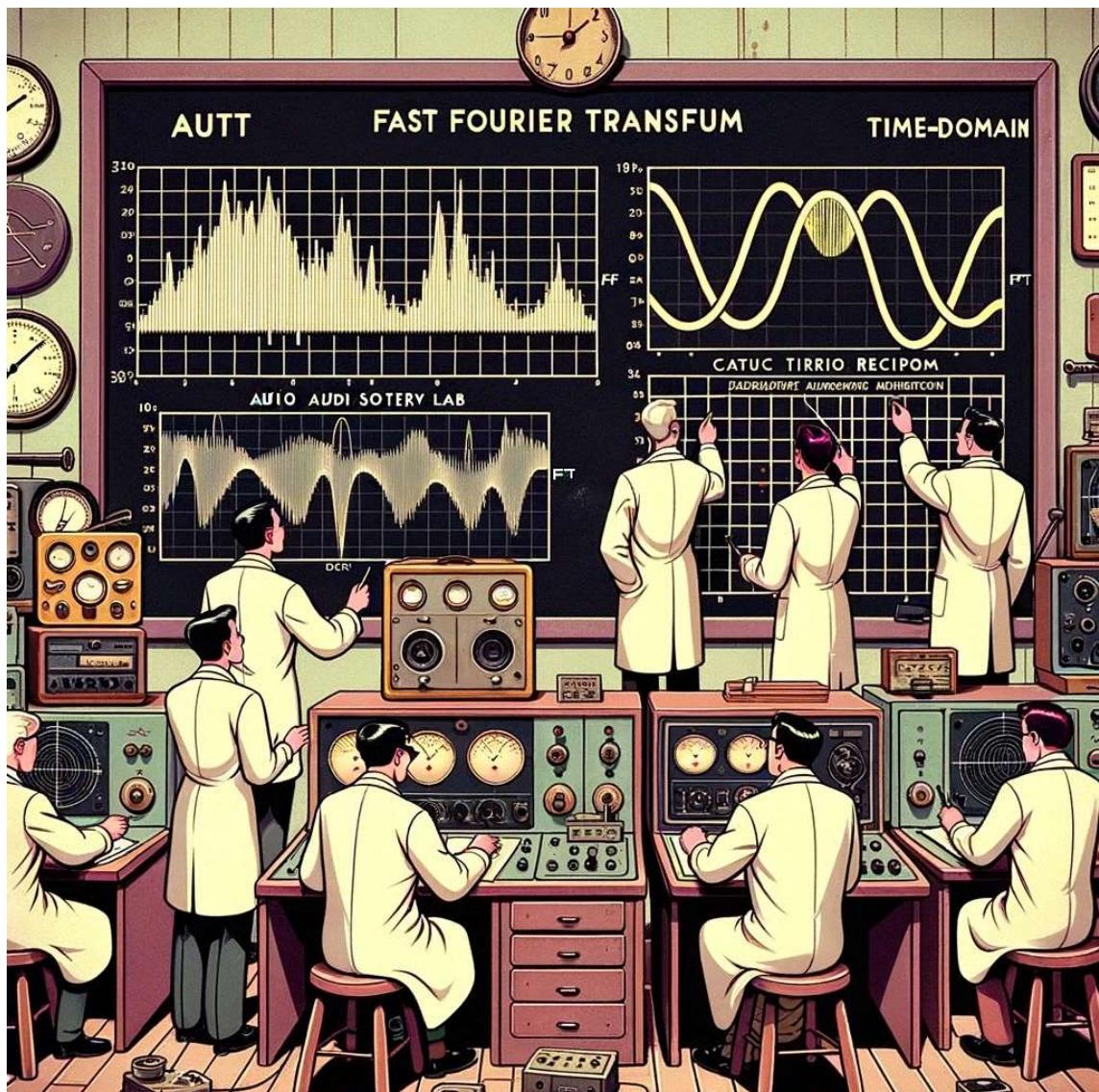
Prima di concludere, considerare che la classificazione dei suoni è più di una semplice voce. Ad esempio, si possono sviluppare progetti TinyML sul suono in diverse aree, come:

- **Sicurezza** (Rilevamento vetri rotti)
- **Industria** (Rilevamento di Anomalie)
- **Medicina** (Russare, Tosse, Malattie polmonari)
- **Natura** (Controllo alveari, suono degli insetti)

Risorse

- [Codici XIAO ESP32S3](#)
- [Sottoinsieme del Dataset dei Comandi Vocali di Google](#)
- [KWS MFCC Analysis Colab Notebook](#)
- [KWS CNN training Colab Notebook](#)
- [XIAO ESP32S3 Post-processing Code](#)
- [Progetto Edge Impulse](#)

Classificazione del Movimento e Rilevamento delle Anomalie

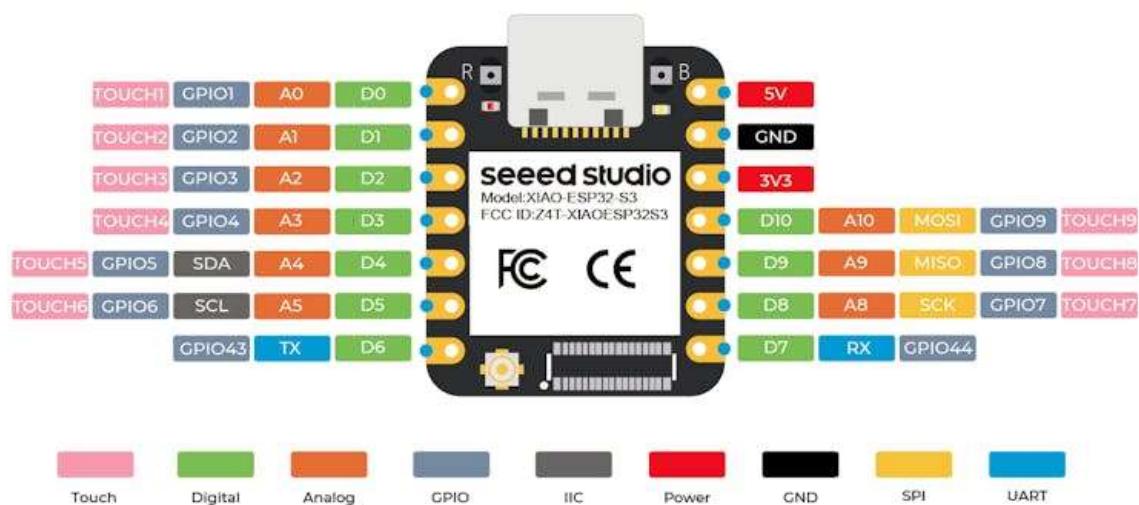


DALL-E prompt - Illustrazione in stile cartoon anni '50 ambientata in un laboratorio audio d'epoca. Scienziati, vestiti in abiti classici con camici bianchi, analizzano attentamente i dati audio su grandi lavagne. Le lavagne mostrano intricati grafici FFT (Fast Fourier Transform) e curve del dominio del tempo. Sono sparse in giro apparecchiature audio antiche, ma le rappresentazioni dei dati sono chiare e dettagliate, a indicare la loro attenzione all'analisi audio.

Panoramica

The XIAO ESP32S3 Sense, con la sua fotocamera e microfono integrati, è un dispositivo versatile. Ma cosa succede se c'è bisogno di aggiungere un altro tipo di sensore, come un IMU? Nessun problema! Una delle caratteristiche distintive di XIAO ESP32S3 sono i suoi molteplici pin che possono essere utilizzati come bus I2C (pin SDA/SCL), rendendolo una piattaforma adatta per l'integrazione di sensori.

FRONT



Installazione dell'IMU

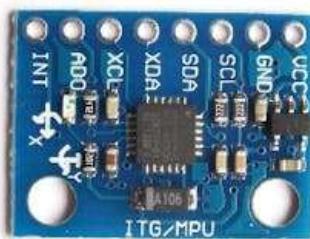
Quando si seleziona l'IMU, il mercato offre un'ampia gamma di dispositivi, ognuno con caratteristiche e capacità uniche. Si potrebbe scegliere, ad esempio, ADXL362 (3 assi), MAX21100 (6 assi), MPU6050 (6 assi), LIS3DHTR (3 assi) o LCM20600Seeed Grove— (6 assi), che fa parte dell'IMU 9DOF (lcm20600+AK09918). Questa varietà consente di adattare la scelta alle esigenze specifiche del progetto.

Per questo progetto, utilizzeremo un'IMU, la MPU6050 (o 6500), un'unità accelerometro/giroscopio a 6 assi a basso costo (meno di 2,00 USD).

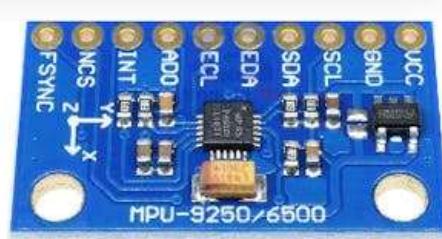
Alla fine del lab, commenteremo anche l'utilizzo dell'LCM20600.

MPU-6500 è un dispositivo di Motion Tracking a 6 assi che combina un giroscopio a 3 assi, un accelerometro a 3 assi e un Digital Motion Processor™ (DMP) in un piccolo package da 3x3x0,9 mm. È inoltre dotato di un FIFO da 4096 byte che può ridurre il traffico sull'interfaccia del bus seriale e ridurre il consumo energetico consentendo al processore di sistema di leggere i dati del sensore in modalità burst e quindi passare a una modalità a basso consumo.

Con il suo bus I2C dedicato al sensore, MPU-6500 accetta direttamente input da dispositivi I2C esterni. MPU-6500, con la sua integrazione a 6 assi, DMP on-chip e firmware di calibrazione runtime, consente ai produttori di eliminare la selezione, la qualificazione e l'integrazione a livello di sistema costose e complesse di dispositivi discreti, garantendo prestazioni di movimento ottimali per i consumatori. MPU-6500 è progettato anche per interfacciarsi con più sensori digitali non inerziali, come i sensori di pressione, sulla sua porta I2C ausiliaria.



MPU6050



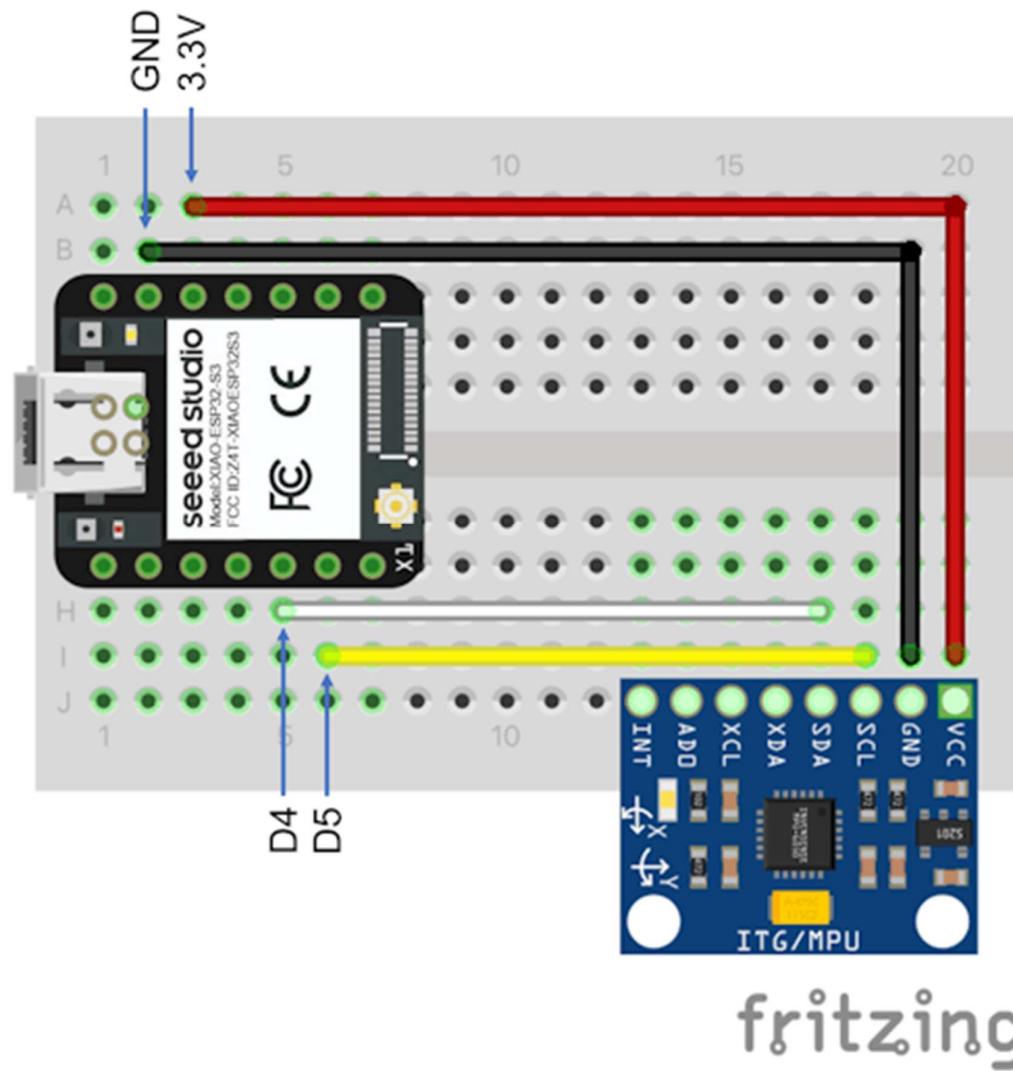
MPU6500

Di solito, le librerie disponibili sono per MPU6050, ma funzionano per entrambi i dispositivi.

Collegamento dell'HW

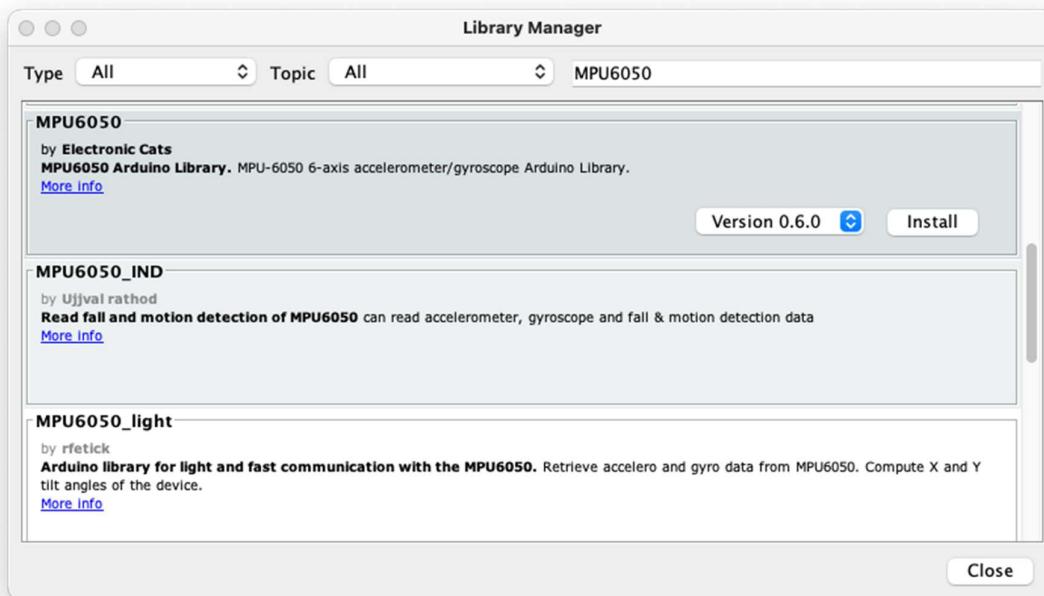
Collegare l'IMU allo XIAO secondo lo schema seguente:

- MPU6050 **SCL** → XIAO **D5**
- MPU6050 **SDA** → XIAO **D4**
- MPU6050 **VCC** → XIAO **3.3V**
- MPU6050 **GND** → XIAO **GND**



Installare la libreria

Si va su Arduino Library Manager e si digita MPU6050. Installare la versione più recente.



Scaricare lo sketch [MPU6050_Acc_Data_Acquisition.ino](#):

```
/*
 * Based on I2C device class (I2Cdev) Arduino sketch for MPU6050 class
 * by Jeff Rowberg <jeff@rowberg.net>
 * and Edge Impulse Data Forwarder Example (Arduino)
 * - https://docs.edgeimpulse.com/docs/cli-data-forwarder
 *
 * Developed by M.Rovai @11May23
 */

#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"

#define FREQUENCY_HZ      50
#define INTERVAL_MS        (1000 / (FREQUENCY_HZ + 1))
#define ACC_RANGE          1 // 0: -/+2G; 1: -/-4G

// convert factor g to m/s^2 ==> [-32768, +32767] ==> [-2g, +2g]
#define CONVERT_G_TO_MS2   (9.81/(16384.0/(1.+ACC_RANGE)))

static unsigned long last_interval_ms = 0;

MPU6050 imu;
int16_t ax, ay, az;

void setup() {
    Serial.begin(115200);

    // initialize device
    Serial.println("Initializing I2C devices...");
    Wire.begin();
    imu.initialize();
    delay(10);

    // // verify connection
    // if (imu.testConnection()) {
    //     Serial.println("IMU connected");
    // }
    // else {
    //     Serial.println("IMU Error");
    // }
    delay(300);

    // Set MCU 6050 Offset Calibration
}
```

```
imu.setXAccelOffset(-4732);
imu.setYAccelOffset(4703);
imu.setZAccelOffset(8867);
imu.setXGyroOffset(61);
imu.setYGyroOffset(-73);
imu.setZGyroOffset(35);

/* Set full-scale accelerometer range.
 * 0 = +/- 2g
 * 1 = +/- 4g
 * 2 = +/- 8g
 * 3 = +/- 16g
 */
imu.setFullScaleAccelRange(ACC_RANGE);
}

void loop() {

    if (millis() > last_interval_ms + INTERVAL_MS) {
        last_interval_ms = millis();

        // read raw accel/gyro measurements from device
        imu.getAcceleration(&ax, &ay, &az);

        // converting to m/s^2^
        float ax_m_s^2^ = ax * CONVERT_G_TO_MS2;
        float ay_m_s^2^ = ay * CONVERT_G_TO_MS2;
        float az_m_s^2^ = az * CONVERT_G_TO_MS2;

        Serial.print(ax_m_s^2^);
        Serial.print("\t");
        Serial.print(ay_m_s^2^);
        Serial.print("\t");
        Serial.println(az_m_s^2^);
    }
}
```

Alcuni commenti sul codice:

Notare che i valori generati dall'accelerometro e dal giroscopio hanno un intervallo: [-32768, +32767], quindi, ad esempio, se viene utilizzato l'intervallo di default dell'accelerometro, l'intervallo in G dovrebbe essere: [-2g, +2g]. Quindi, "1G" significa 16384.

Per la conversione in m/s², ad esempio, si può definire quanto segue:

```
#define CONVERT_G_TO_MS2 (9.81/16384.0)
```

Nel codice, è stata lasciata un'opzione (ACC_RANGE) da impostare su 0 (+/-2G) o 1 (+/- 4G).

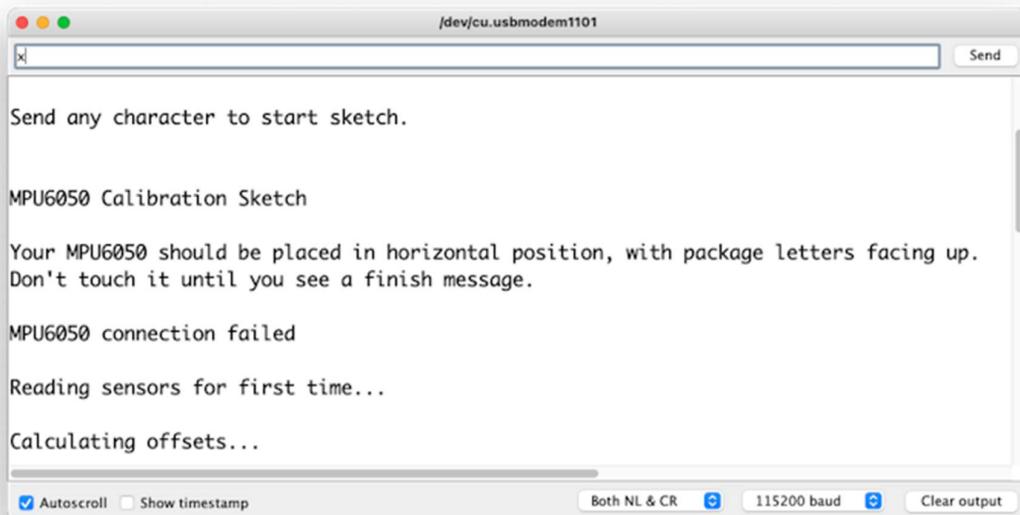
Useremo +/-4G; dovrebbe bastare per noi. In questo caso.

Acquisiremo i dati dell'accelerometro a una frequenza di 50 Hz e i dati di accelerazione saranno inviati alla porta seriale come metri al secondo quadrato (m/s²).

Quando si esegue il codice con l'IMU appoggiata sul tavolo, i dati dell'accelerometro mostrati sul monitor seriale dovrebbero essere circa 0.00, 0.00 e 9.81. Se i valori sono molto diversi, si deve calibrare l'IMU.

L'MCU6050 può essere calibrato usando lo sketch: [mcu6050-calibration.ino](#).

Eseguire il codice. Sul Serial Monitor verrà visualizzato quanto segue:



Inviare un carattere qualsiasi (nell'esempio sopra, "x") e la calibrazione dovrebbe iniziare.

Notare che c'è un messaggio MPU6050 di connessione fallita. Ignorare questo messaggio. Per qualche motivo, `imu.testConnection()` non restituisce un risultato corretto.

Alla fine, si riceveranno i valori di offset da utilizzare su tutti gli sketch:

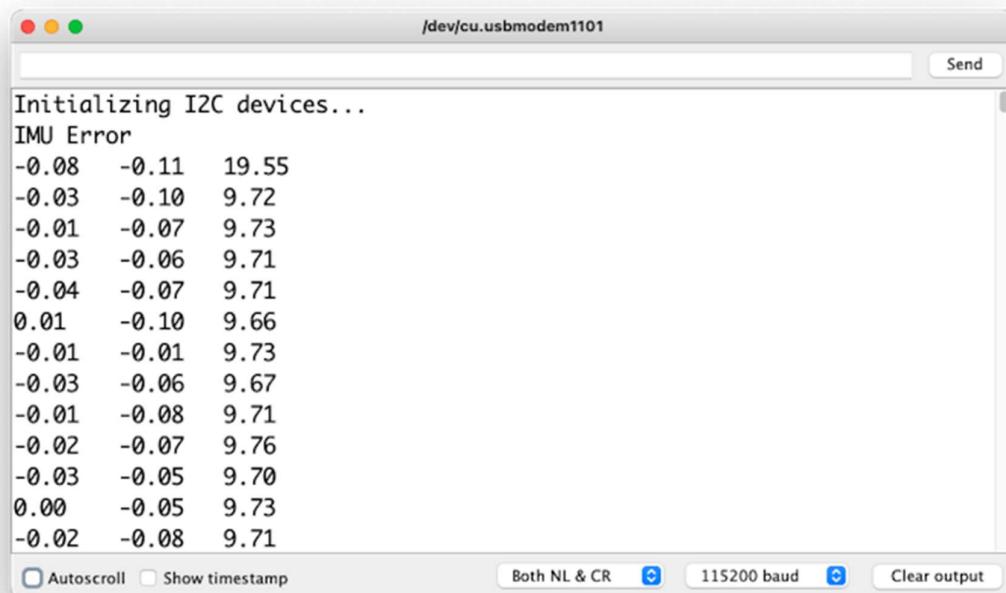


Prendere i valori e usarli nella configurazione:

```
//Set MCU 6050 OffSet Calibration
imu.setXAccelOffset(-4732);
imu.setYAccelOffset(4703);
imu.setZAccelOffset(8867);
imu.setXGyroOffset(61);
imu.setYGyroOffset(-73);
imu.setZGyroOffset(35);
```

Ora, eseguire lo sketch [MPU6050_Acc_Data_Acquisition.in](#):

Una volta eseguito lo sketch precedente, aprire il Serial Monitor:

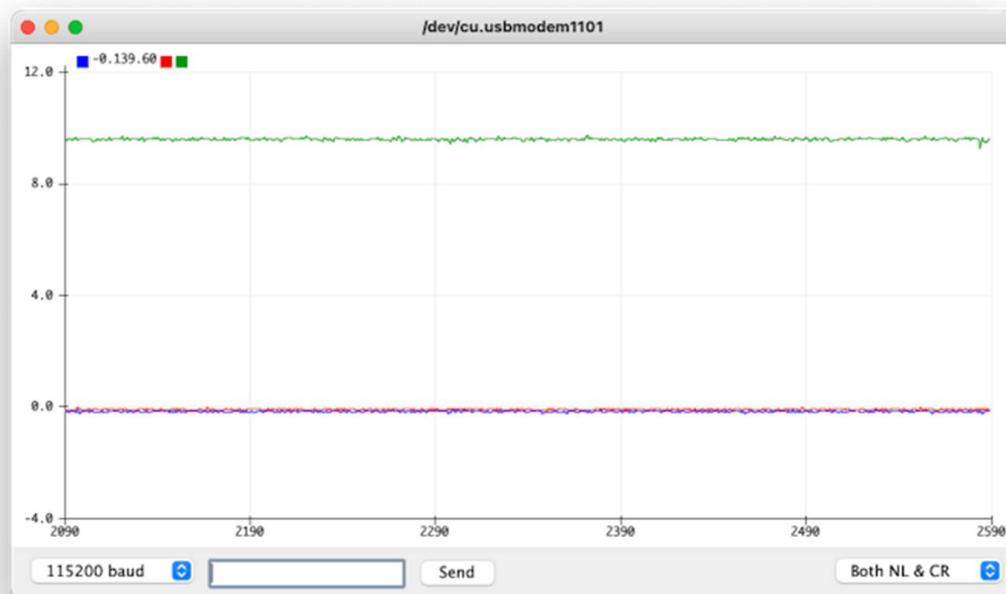


A screenshot of a terminal window titled "/dev/cu.usbmodem1101". The window displays a list of IMU error values. At the top, it says "Initializing I2C devices...". Below that, it says "IMU Error" followed by a series of 15 rows of data. Each row contains three values: -0.08, -0.11, and 19.55. The data is as follows:

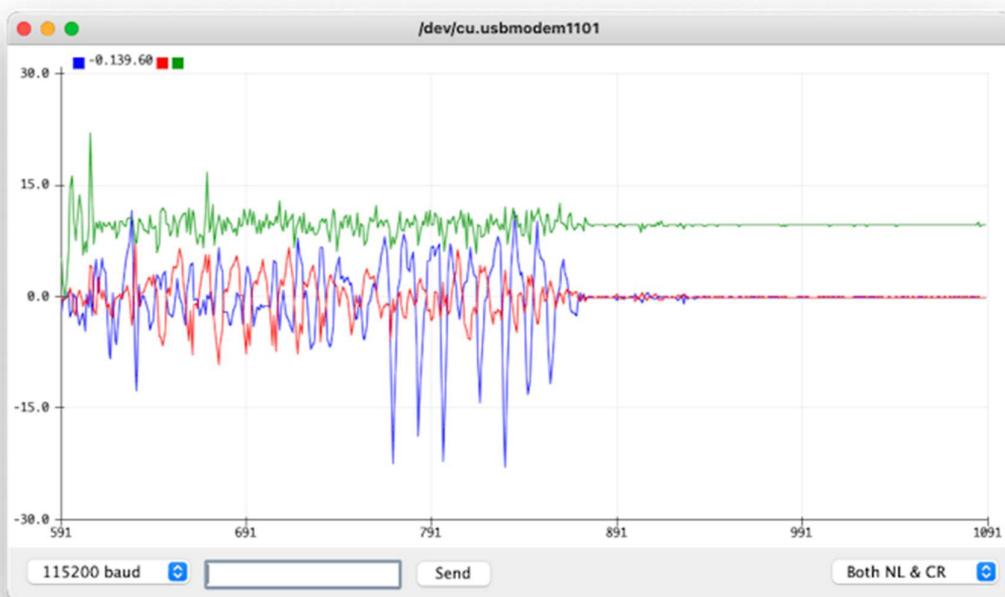
	-0.08	-0.11	19.55
1	-0.03	-0.10	9.72
2	-0.01	-0.07	9.73
3	-0.03	-0.06	9.71
4	-0.04	-0.07	9.71
5	0.01	-0.10	9.66
6	-0.01	-0.01	9.73
7	-0.03	-0.06	9.67
8	-0.01	-0.08	9.71
9	-0.02	-0.07	9.76
10	-0.03	-0.05	9.70
11	0.00	-0.05	9.73
12	-0.02	-0.08	9.71

At the bottom of the terminal window, there are several control buttons: "Autoscroll" (unchecked), "Show timestamp" (unchecked), "Both NL & CR" (selected), "115200 baud" (selected), and "Clear output".

Oppure controllare il Plotter:



Spostare il dispositivo sui tre assi. Si dovrebbe vedere la variazione sul Plotter:



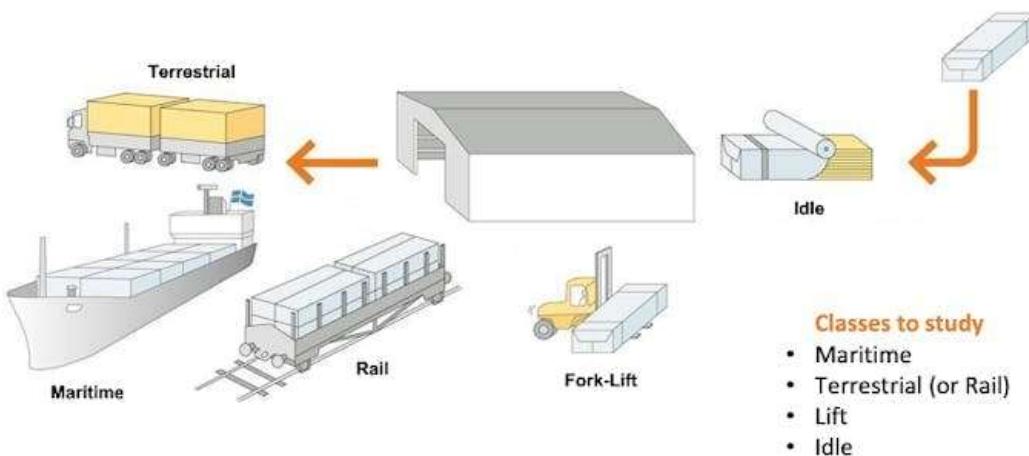
Il Progetto TinyML Motion Classification

Per il nostro lab, simuleremo sollecitazioni meccaniche nel trasporto. Il nostro problema sarà classificare quattro classi di movimento:

- **Maritime** (Pallet in navi)
- **Terrestrial** (Pallet in un camion o treno)
- **Lift** (Pallet movimentati da carrello elevatore)
- **Idle** (Pallet in magazzini)

Quindi, per iniziare, dovremmo raccogliere dati. Quindi, gli accelerometri forniranno i dati sul pallet (o sul contenitore).

Case Study: Mechanical Stresses in Transport



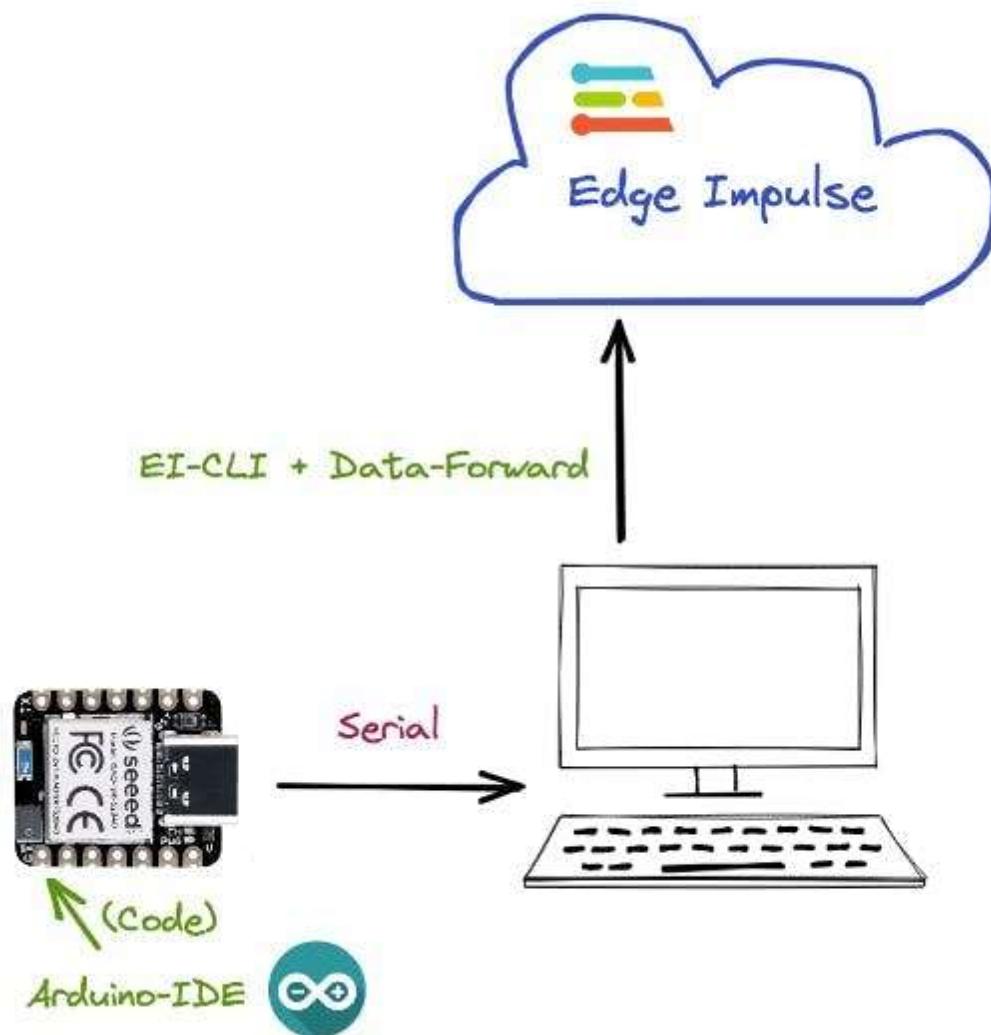
Dalle immagini sopra, possiamo vedere che i movimenti principalmente orizzontali dovrebbero essere associati alla “classe Terrestrial”, i movimenti verticali alla “classe Lift”, nessuna attività alla “classe Idle” e il movimento su tutti e tre gli assi alla classe [Maritime](#).

Collegamento del dispositivo a Edge Impulse

Per la raccolta dei dati, dovremmo prima collegare il dispositivo a Edge Impulse Studio, che verrà utilizzato anche per la pre-elaborazione dei dati, l'addestramento del modello, i test e la distribuzione.

Seguire le istruzioni [qui](#) per installare [Node.js](#) e Edge Impulse CLI sul computer.

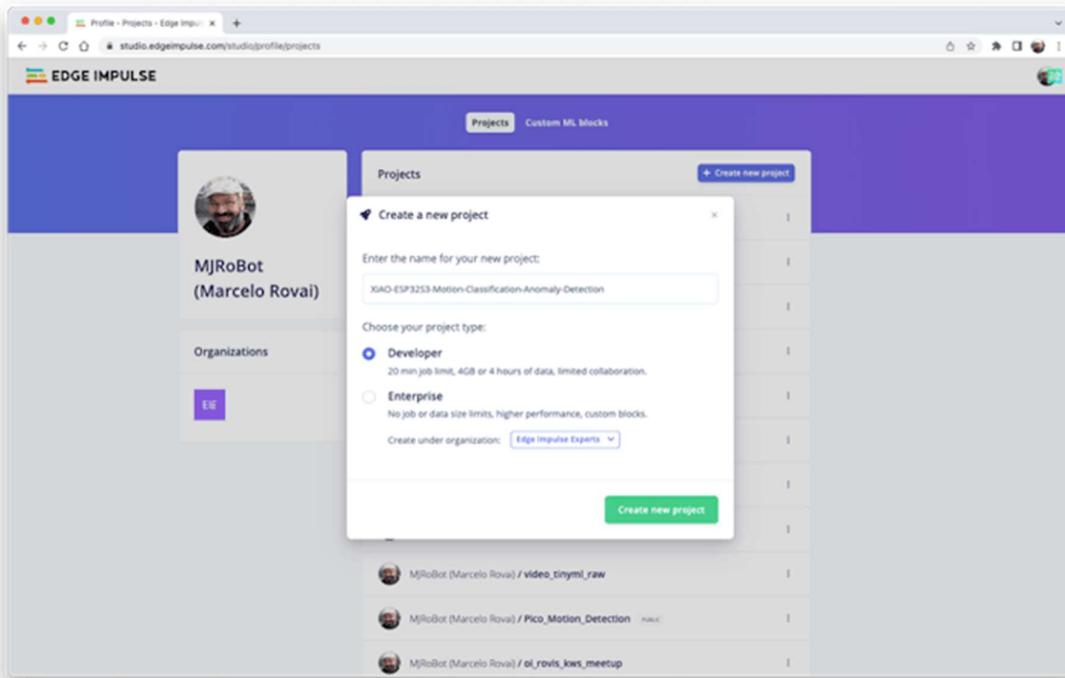
Dal momento che XIAO ESP32S3 non è più una scheda di sviluppo completamente supportata da Edge Impulse, dovremo, ad esempio, usare [CLI Data Forwarder](#) per acquisire dati dal nostro sensore e inviarli allo Studio, come mostrato in questo diagramma:



In alternativa, si possono acquisire i dati “offline”, memorizzarli su una scheda SD o inviarli al computer tramite Bluetooth o Wi-Fi. In questo [video](#), si possono imparare modi alternativi per inviare dati a Edge Impulse Studio.

Collegare il dispositivo alla porta seriale ed eseguire il codice precedente per catturare i dati IMU (Accelerometro), “stampandoli” sulla seriale. Questo consentirà a Edge Impulse Studio di “catturarli”.

Si va alla pagina Edge Impulse e si crea un progetto.



La lunghezza massima per un nome di libreria Arduino è di **63 caratteri**. Notare che Studio nominerà la libreria finale usando il nome del progetto e vi includerà “_inference”. Il nome scelto inizialmente non ha funzionato quando è stato provato a distribuire la libreria Arduino perché risultava in 64 caratteri. Quindi, lo si deve cambiare eliminando la parte “anomaly detection”.

Avviare **CLI Data Forwarder** sul terminale, immettendo (se è la prima volta) il seguente comando:

```
edge-impulse-data-forwarder --clean
```

Quindi, immettere le credenziali EI e scegli il progetto, le variabili e i nomi dei dispositivi:

```
marcelo_roval — node /usr/local/bin/edge-impulse-data-forwarder --clean — 125x26
(base) marcelo_roval@Marcelos-MacBook-Pro ~ %
(base) marcelo_roval@Marcelos-MacBook-Pro ~ % edge-impulse-data-forwarder --clean
Edge Impulse data forwarder v1.15.1
? What is your user name or e-mail address (edgeimpulse.com)? roval@mjrobot.org
? What is your password? [hidden]
Endpoints:
  Websocket: vss://remote-mgmt.edgeimpulse.com
  API: https://studio.edgeimpulse.com/v1
  Ingestion: https://ingestion.edgeimpulse.com

[SER] Connecting to /dev/tty.usbmodem101
[SER] Serial is connected (34:0:5::16::0:E::3E::2:C)
[WS ] Connecting to vss://remote-mgmt.edgeimpulse.com
[WS ] Connected to vss://remote-mgmt.edgeimpulse.com

? To which project do you want to connect this device? MJRobot (Marcelo Rovai) / XIAO-ESP3253-Motion-Classification-Anomaly-Detection
[SER] Detecting data frequency...
[SER] Detected data frequency: 51Hz
? 3 sensor axes detected (example values: [-0.15,-0.23,0.56]). What do you want to call them? Separate the names with ','
's: accx, accy, accz
? What name do you want to give this device? XIAO-ESP3253
[WS ] Device "XIAO-ESP3253" is now connected to project "XIAO-ESP3253-Motion-Classification-Anomaly-Detection"
[WS ] Go to https://studio.edgeimpulse.com/studio/226390/acquisition/training to build your machine learning model!
```

Si va al progetto EI e si verifica se il dispositivo è connesso (il punto dovrebbe essere verde):

Your devices

These are devices that are connected to the Edge Impulse remote management API, or have posted data to the Ingestion SDK.

NAME	ID	TYPE	SENSORS	REMOT...	LAST SEEN
XIAO-ESP32S3	34:8:5:18:8:E:3E::2:C	DATA_FORWARDER	Sensor with 3 axes (accX, acc...	●	Today, 17:24:59

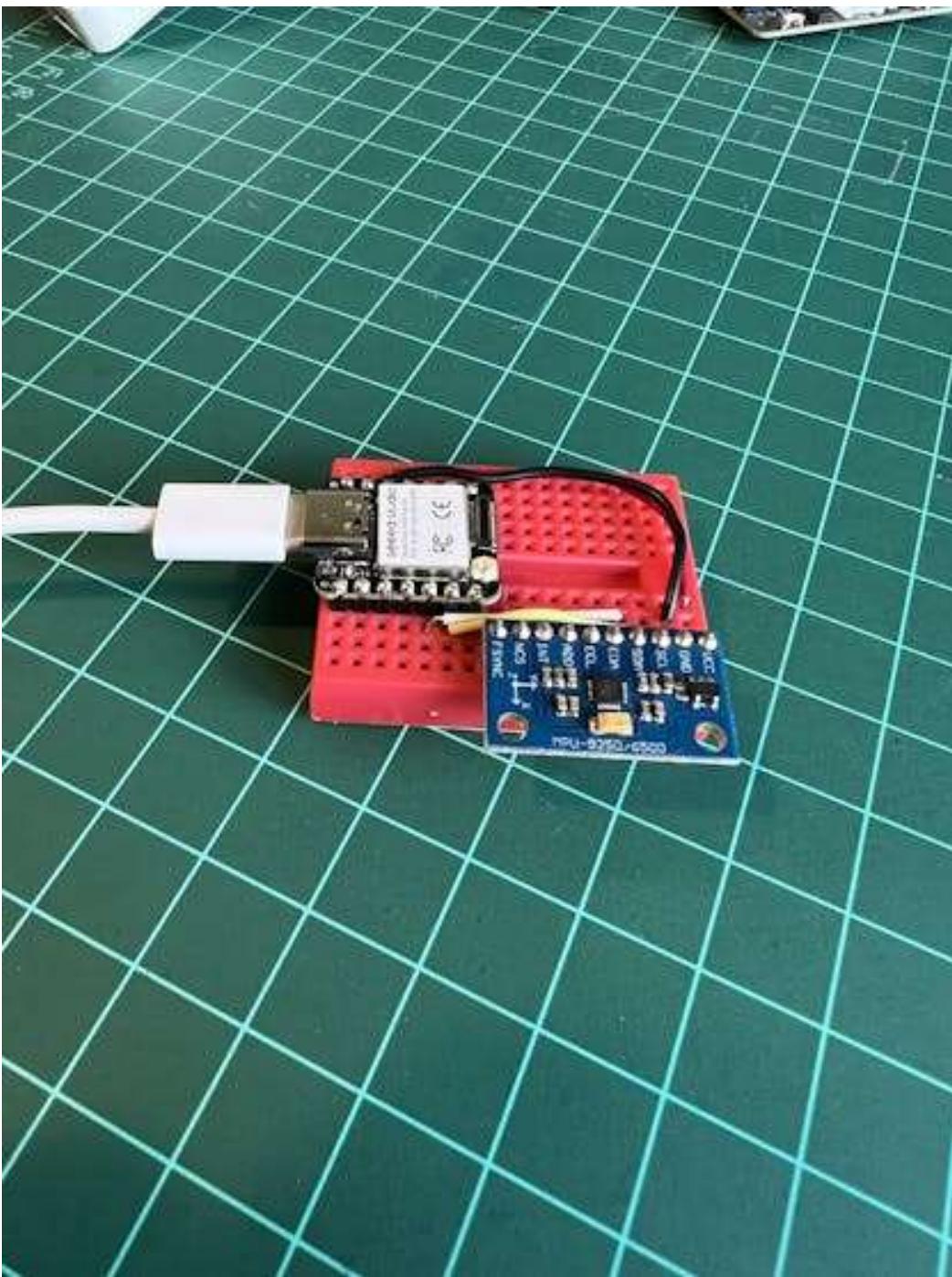
Raccolta Dati

Come discusso in precedenza, dovremmo catturare dati da tutte e quattro le classi "Transportation". Immaginiamo di avere un contenitore con un accelerometro incorporato:

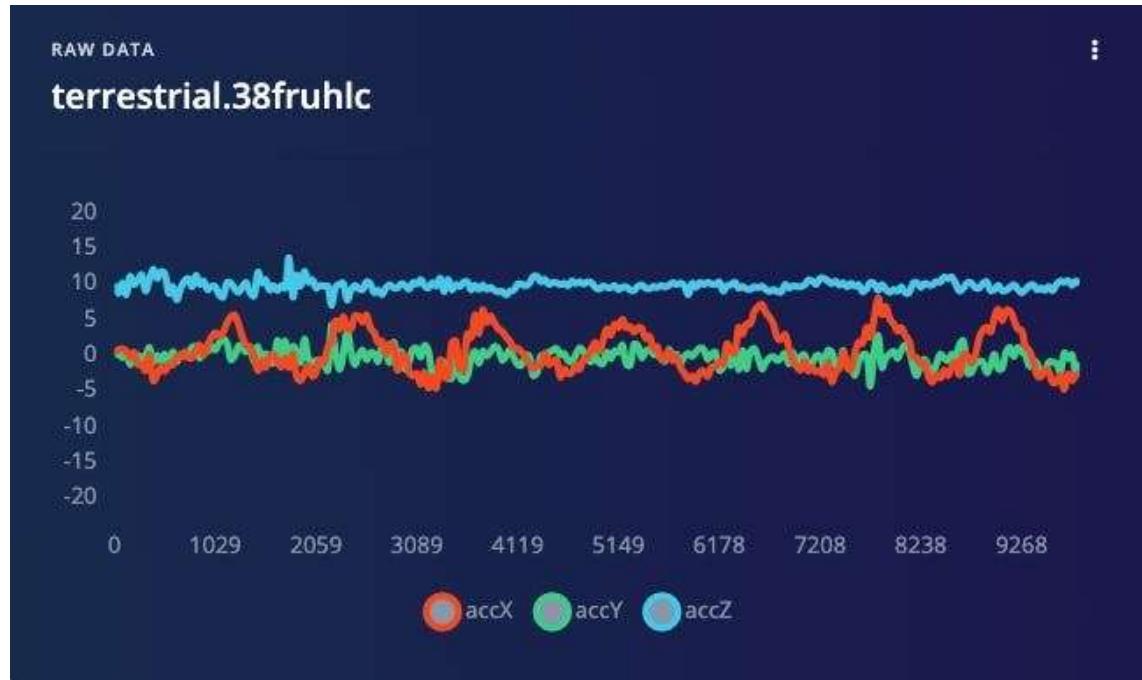


Ora immaginiamo che il contenitore sia su una barca, di fronte a un oceano in tempesta, su un camion, ecc.:

- **Maritime** (Pallet in navi)
 - Spostare lo XIAO in tutte le direzioni, simulando un movimento ondulatorio della barca.
- **Terrestrial** (Pallet in un camion o treno)
 - Spostare lo XIAO su una linea orizzontale.
- **Lift** (Pallet movimentati da carrello elevatore)
 - Spostare lo XIAO su una linea verticale.
- **Idle** (Pallet in magazzini)
 - Lasciare lo XIAO sul tavolo.



Di seguito è riportato un campione (dati grezzi) di 10 secondi:

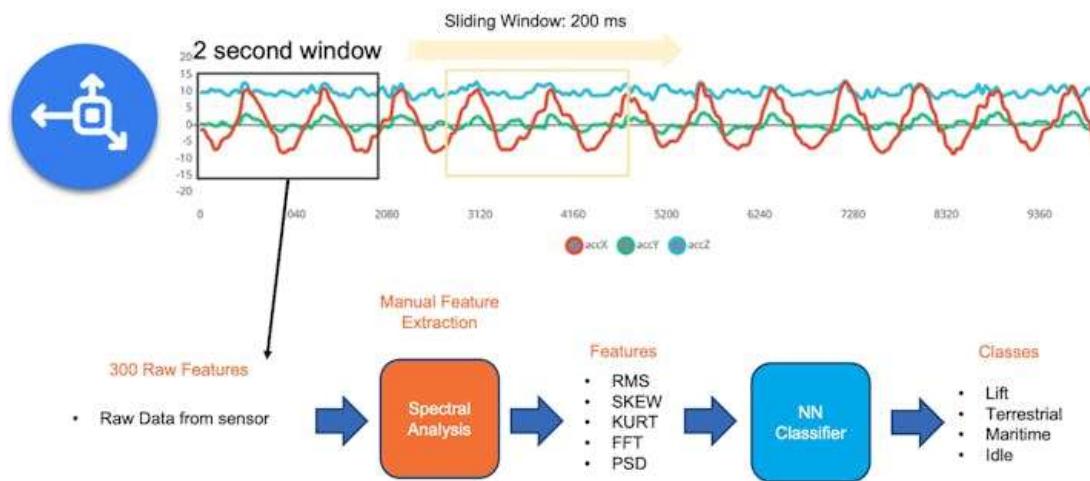


Ad esempio, si possono catturare 2 minuti (dodici campioni da 10 secondi ciascuno) per le quattro classi. Utilizzando i “3 punti” dopo ciascuno dei campioni, selezionarne 2, spostandoli per il set di test (o utilizzare lo strumento automatico Train/Test Split nella scheda Danger Zone della Dashboard). Di seguito, si possono vedere i dataset dei risultati:

SAMPLE NAME	LABEL	ADDED	LENGTH
terrestrial.json.40cm...	terrestrial	Today, 16:32:14	10s
terrestrial.json.40cm...	terrestrial	Today, 16:32:13	10s
lift.json.40cmahj.ing...	lift	Today, 16:32:13	10s
lift.json.40cmahj.ing...	lift	Today, 16:32:13	10s
terrestrial.json.40cm...	terrestrial	Today, 16:32:12	10s
terrestrial.json.40cm...	terrestrial	Today, 16:32:12	10s
terrestrial.json.40cm...	terrestrial	Today, 16:32:12	10s
maritime.json.40co0v...	maritime	Today, 16:32:11	10s
idle.json.40cm9phv.in...	idle	Today, 16:32:11	10s
lift.json.40cnq1hr.ing...	lift	Today, 16:32:11	10s
lift.json.40cnq1hr.ing...	lift	Today, 16:32:11	10s
idle.json.40cmdjs.in...	idle	Today, 16:32:10	10s

Pre-elaborazione dei Dati

Il tipo di dati grezzi catturato dall'accelerometro è una “serie temporale” e dovrebbe essere convertito in “dati tabulari”. Possiamo effettuare questa conversione utilizzando una finestra scorrevole sui dati campione. Ad esempio, nella figura sottostante,



Possiamo vedere 10 secondi di dati dell'accelerometro catturati con un “sample rate (SR)” [frequenza di campionamento] di 50 Hz. Una finestra di 2 secondi catturerà 300 dati (3 assi x 2 secondi x 50 campioni). Faremo scorrere questa finestra ogni 200 ms, creando un set di dati più grande in cui ogni istanza ha 300 feature grezze.

Si deve usare il miglior SR per il proprio caso, considerando il teorema di Nyquist, che afferma che un segnale periodico deve essere campionato a più del doppio della componente di frequenza più alta del segnale.

La pre-elaborazione dei dati è un’area impegnativa per l’apprendimento automatico embedded. Tuttavia, Edge Impulse aiuta a superare questo problema con la sua fase di pre-elaborazione dell’elaborazione del segnale digitale (DSP) e, più specificamente, le Spectral Feature.

Su Studio, questo dataset sarà l’input di un blocco Spectral Analysis, che è eccellente per analizzare il movimento ripetitivo, come i dati degli accelerometri. Questo blocco eseguirà un DSP (Digital Signal Processing), estraendo caratteristiche come “FFT” o “Wavelets”. Nel caso più comune, FFT, le **feature Time Domain Statistical** per asse/canale sono:

- RMS
- Skewness
- Kurtosis

E la **Frequency Domain Spectral features** per asse/canale sono:

- Spectral Power [Potenza spettrale]
- Skewness
- Kurtosis

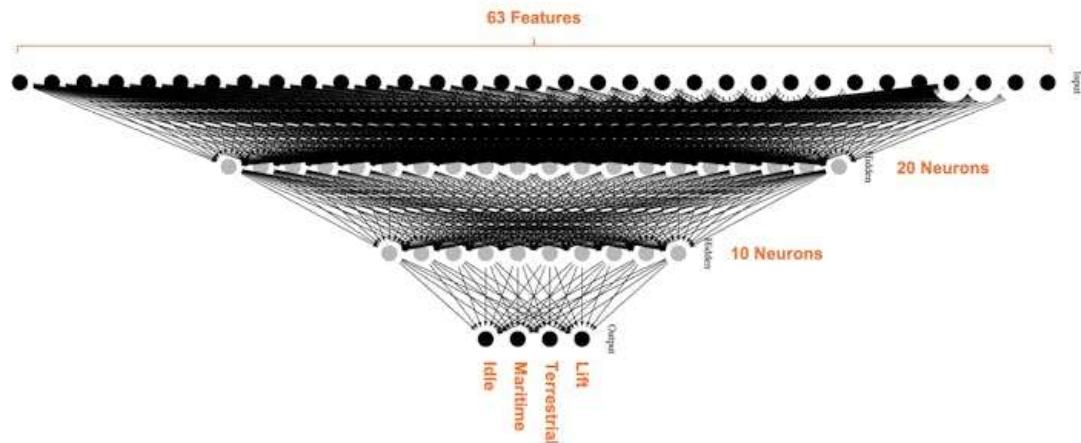
Ad esempio, per una lunghezza FFT di 32 punti, l’output risultante del blocco di analisi spettrale sarà di 21 feature per asse (un totale di 63 feature).

Quelle 63 feature saranno il tensore di input di un classificatore di reti neurali e il modello di rilevamento delle anomalie (K-Means).

Se ne può sapere di più esplorando il laboratorio [DSP Spectral Features](#)

Progettazione del Modello

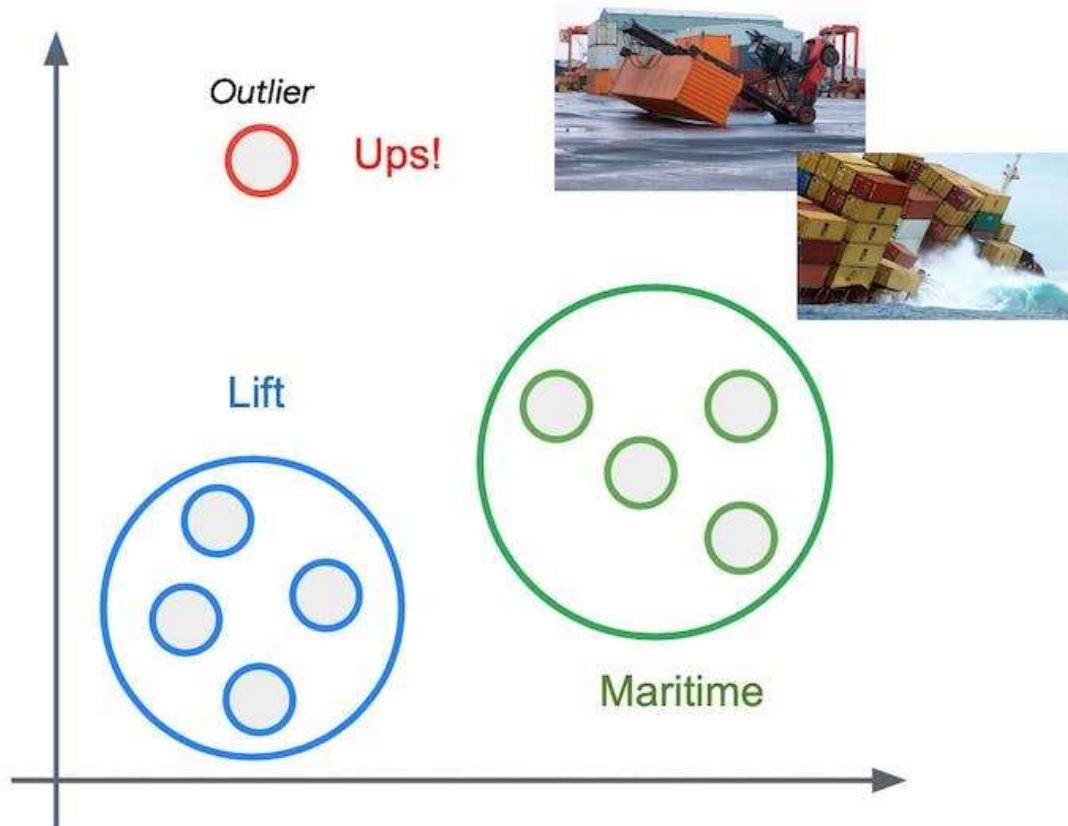
Il nostro classificatore sarà una Dense Neural Network (DNN) che avrà 63 neuroni sul suo layer di input, due layer nascosti con 20 e 10 neuroni e un layer di output con quattro neuroni (uno per ogni classe), come mostrato qui:



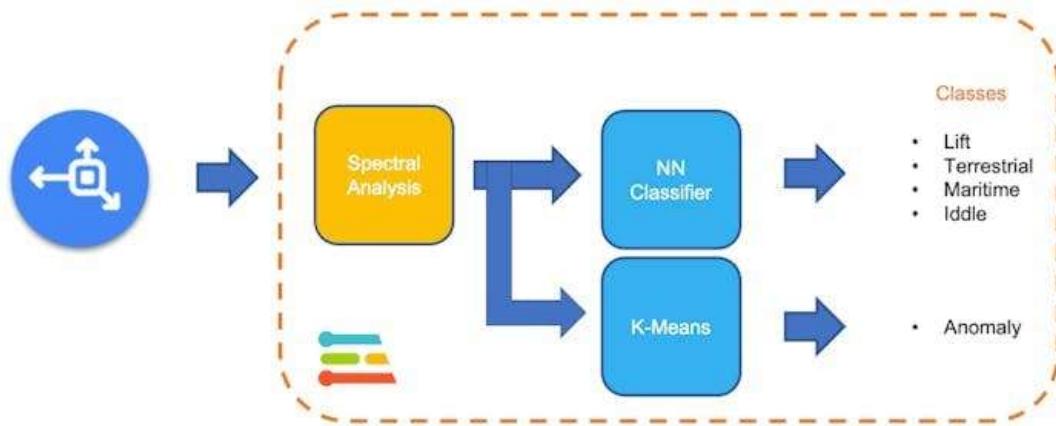
Impulse Design

Un impulso prende dati grezzi, usa l'elaborazione del segnale per estrarre le caratteristiche e poi usa un blocco di apprendimento per classificare nuovi dati.

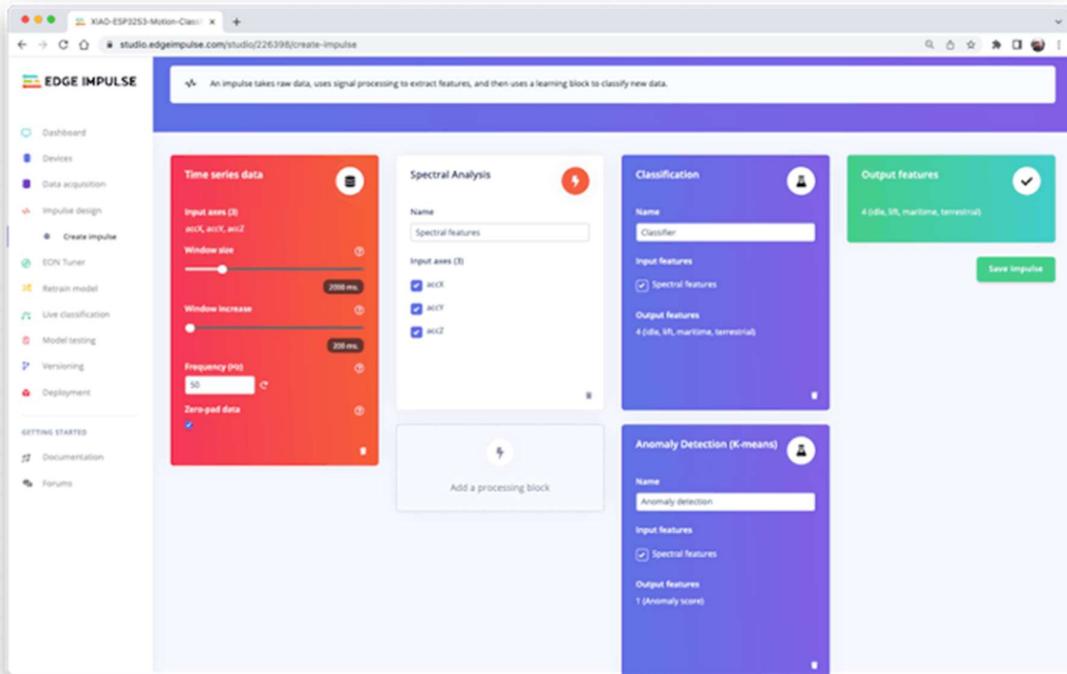
Sfruttiamo anche un secondo modello, il K-means, che può essere usato per le “Anomaly Detection” [rilevamento delle anomalie](#). Se immaginiamo di poter avere le nostre classi note come cluster, qualsiasi campione che non potrebbe adattarsi potrebbe essere un valore anomalo, un'anomalia (ad esempio, un container che rotola fuori da una nave in mare).



Immaginiamo il nostro XIAO che rotola o si muove capovolto, su un complemento di movimento diverso da quello addestrato

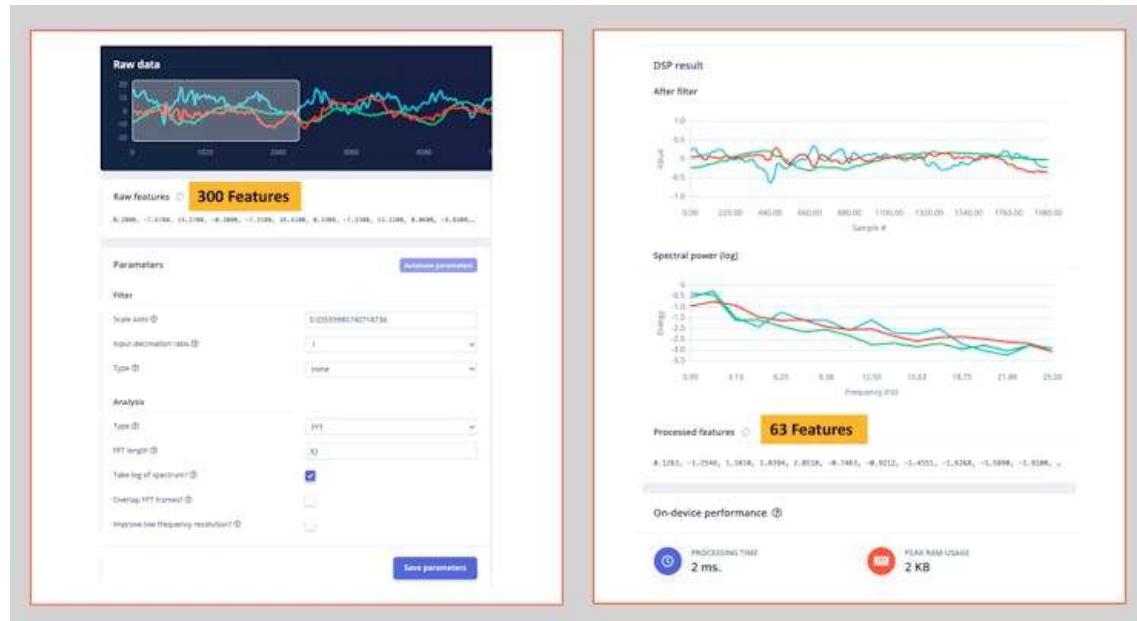


Di seguito è riportato il nostro progetto Impulse finale:



Generazione di feature

A questo punto del nostro progetto, abbiamo definito il metodo di pre-elaborazione e il modello progettato. Ora è il momento di terminare il lavoro. Per prima cosa, prendiamo i dati grezzi (tipo serie temporale) e convertiamoli in dati tabulari. Andiamo alla scheda Spectral Features e selezioniamo Save Parameters:



Nel menù in alto, selezionare l'opzione Generate Features e il pulsante Generate Features. Ogni dato della finestra di 2 secondi verrà convertito in un punto dati di 63 feature.

Feature Explorer mostrerà quei dati in 2D usando **UMAP**. Uniform Manifold Approximation and Projection (UMAP) è una tecnica di riduzione delle dimensioni che può essere usata per la visualizzazione in modo simile a t-SNE ma anche per la riduzione generale delle dimensioni non lineari.

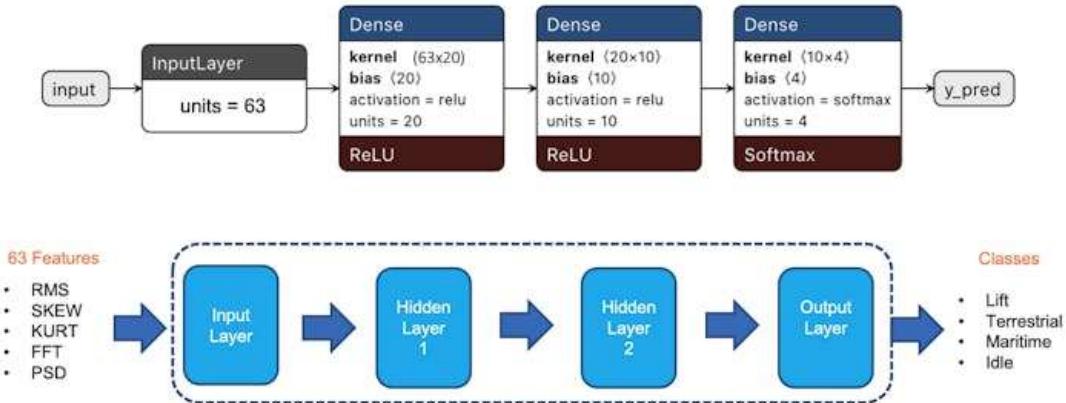
La visualizzazione consente di verificare che le classi presentino un'eccellente separazione, il che indica che il classificatore dovrebbe funzionare bene.



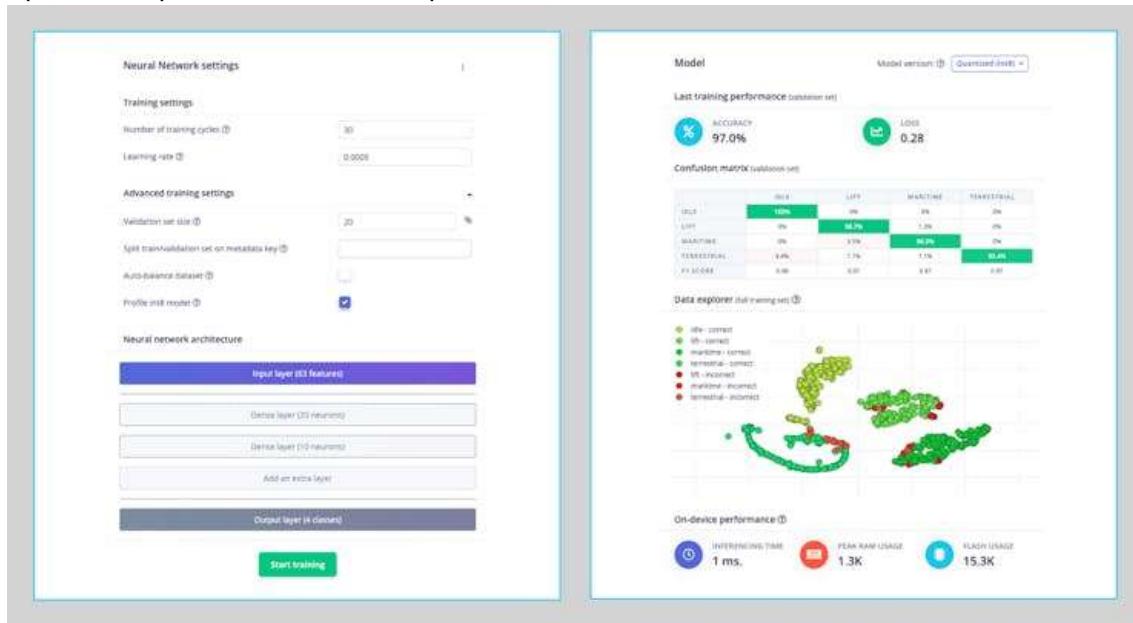
Facoltativamente, si può analizzare l'importanza relativa di ogni feature per una classe rispetto ad altre classi.

Training

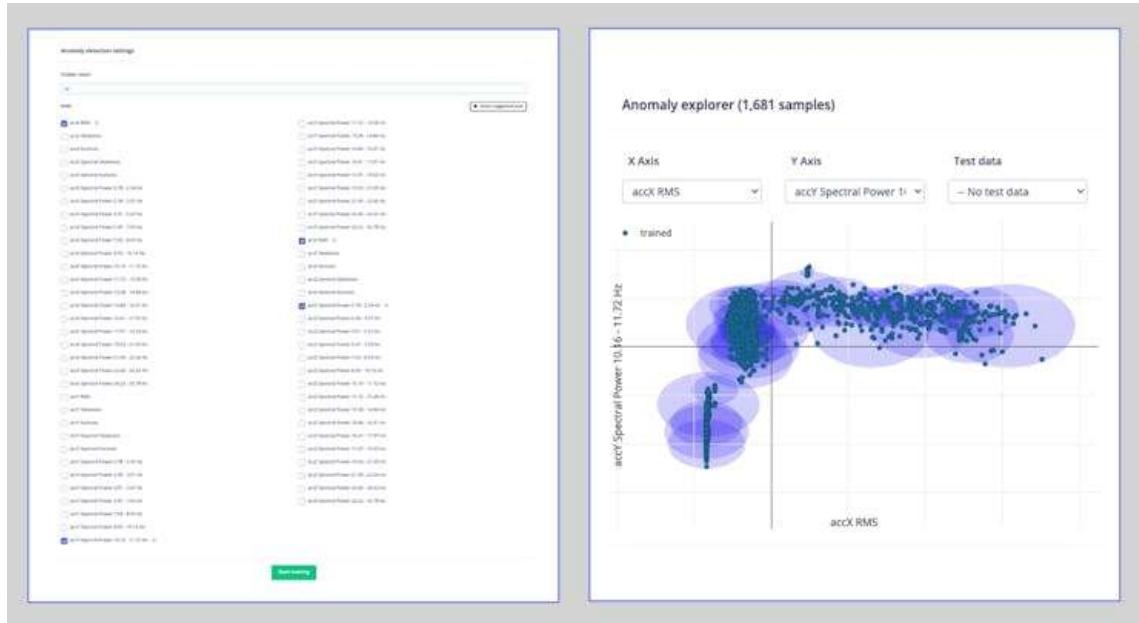
Il nostro modello ha quattro layer, come mostrato di seguito:



Come iperparametri, useremo un Learning Rate di 0,005 e il 20% di dati per la convalida per 30 epochhe. Dopo l'addestramento, possiamo vedere che l'accuratezza è del 97%.

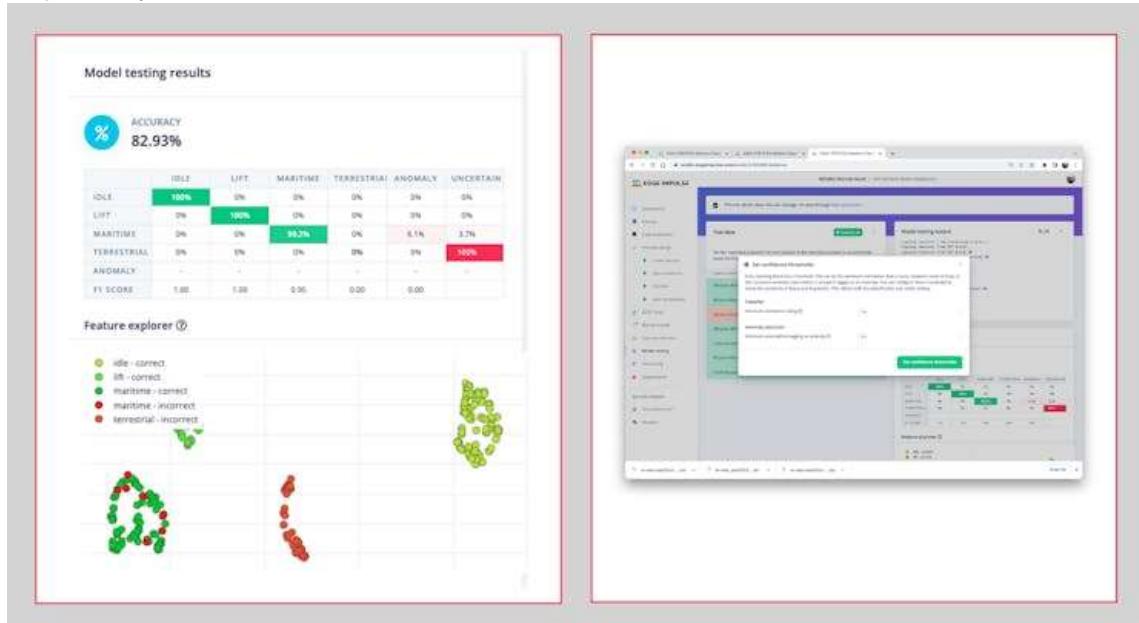


Per il rilevamento delle anomalie, dovremmo scegliere le feature suggerite che sono esattamente le più importanti nell'estrazione delle feature. Il numero di cluster sarà 32, come suggerito dallo Studio:

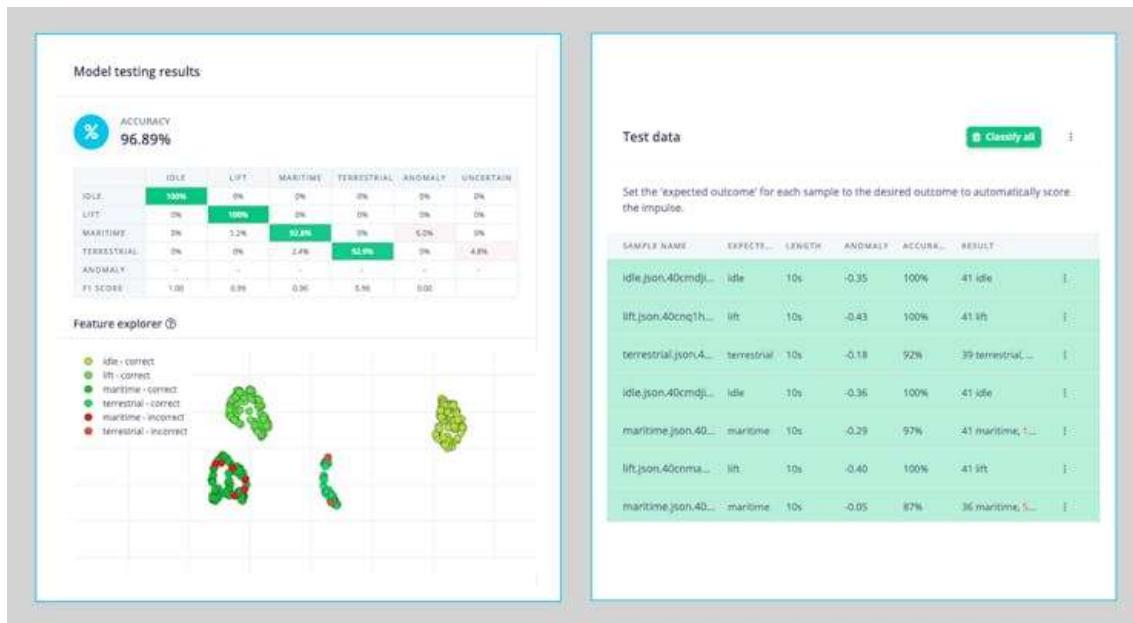


Test

Utilizzando il 20% dei dati lasciati indietro durante la fase di acquisizione dati, possiamo verificare come si comporterà il nostro modello con dati sconosciuti; se non al 100% (come previsto), il risultato non è stato così buono (8%), principalmente a causa della classe “terrestrial”. Una volta che abbiamo quattro classi (il cui output dovrebbe aggiungere 1,0), possiamo impostare una soglia inferiore affinché una classe sia considerata valida (ad esempio, 0,4):



Ora, l'accuratezza del test salirà al 97%.

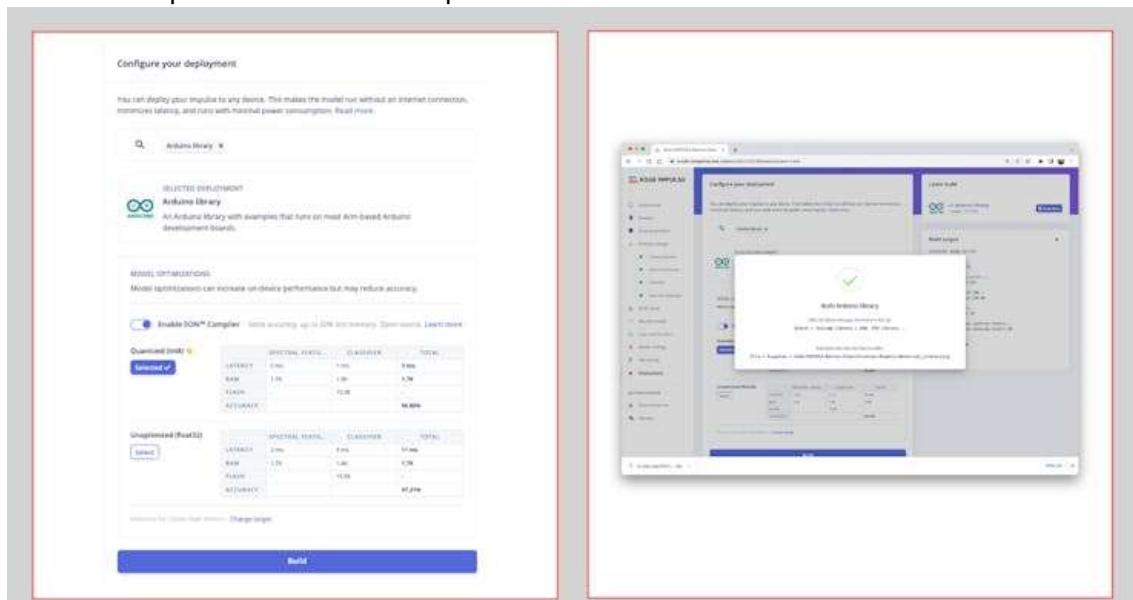


Si deve anche usare il dispositivo (che è ancora connesso allo Studio) ed eseguire una “Live Classification”.

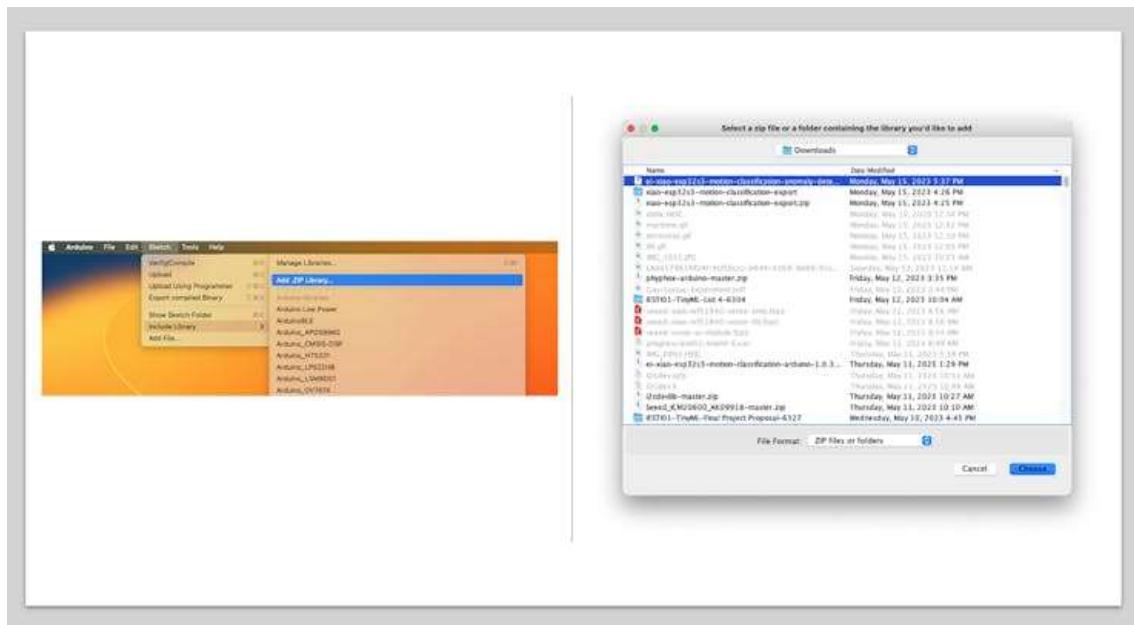
Notare che qui si cattureranno dati reali con il dispositivo e si caricheranno sullo Studio, dove verrà presa un’inferenza usando il modello addestrato (ma il modello NON è nel dispositivo).

Distribuzione

Adesso è il momento della magia! Studio impacchetterà tutte le librerie necessarie, le funzioni di pre-elaborazione e i modelli addestrati, scaricandoli sul computer. Si deve selezionare l’opzione “Arduino Library” e, in basso, scegliere Quantized (Int8) e Build. Verrà creato un file Zip e scaricato sul computer.



Sull’IDE Arduino, si va alla scheda Sketch, si seleziona l’opzione Add.ZIP Library e si sceglie il file.zip scaricato da Studio:



Inferenza

Ora è il momento di un vero test. Faremo inferenze completamente scollegate dallo Studio. Modifichiamo uno degli esempi di codice creati quando si distribuisce la libreria Arduino. Nell'IDE Arduino, si va alla scheda File/Examples e si cerca il progetto, e negli esempi, si seleziona nano_ble_sense_accelerometer:



Ovviamente, questa non è la propria board, ma possiamo far funzionare il codice con solo poche modifiche.

Ad esempio, all'inizio del codice, c'è la libreria relativa ad Arduino Sense IMU:

```
/* Includes ----- */
#include <XIAO-ESP32S3-Motion-Classification_inferencing.h>
#include <Arduino_LSM9DS1.h>
```

Sostituire la parte "include" con il codice relativo all'IMU:

```
#include <XIAO-ESP32S3-Motion-Classification_inferencing.h>
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"
```

Cambiare le Constant Defines

```
/* Constant defines ----- */
MPU6050 imu;
int16_t ax, ay, az;

#define ACC_RANGE          1 // 0: -/+2G; 1: +/-4G
#define CONVERT_G_TO_MS2   (9.81/(16384/(1.+ACC_RANGE)))
#define MAX_ACCEPTED_RANGE (2*9.81)+(2*9.81)*ACC_RANGE
```

Nella funzione di configurazione, avviare l'IMU per impostare i valori di offset e il range:

```
// initialize device
Serial.println("Initializing I2C devices...");
Wire.begin();
imu.initialize();
delay(10);

//Set MCU 6050 OffSet Calibration
imu.setXAccelOffset(-4732);
imu.setYAccelOffset(4703);
```

```

imu.setZAccelOffset(8867);
imu.setXGyroOffset(61);
imu.setYGyroOffset(-73);
imu.setZGyroOffset(35);

imu.setFullScaleAccelRange(ACC_RANGE);

```

Nella funzione loop, i buffer buffer[ix], buffer[ix + 1] e buffer[ix + 2] riceveranno i dati a 3 assi catturati dall'accelerometro. Nel codice originale, si ha la riga:

```
IMU.readAcceleration(buffer[ix], buffer[ix + 1], buffer[ix + 2]);
```

Modificarlo con questo blocco di codice:

```

imu.getAcceleration(&ax, &ay, &az);
buffer[ix + 0] = ax;
buffer[ix + 1] = ay;
buffer[ix + 2] = az;

```

Si deve cambiare l'ordine dei due blocchi di codice seguenti. Per prima cosa, si esegue la conversione in dati grezzi in “Meters per squared second (ms²)”, seguita dal test riguardante il range di accettazione massimo (che qui è in ms², ma su Arduino era in Gs):

```

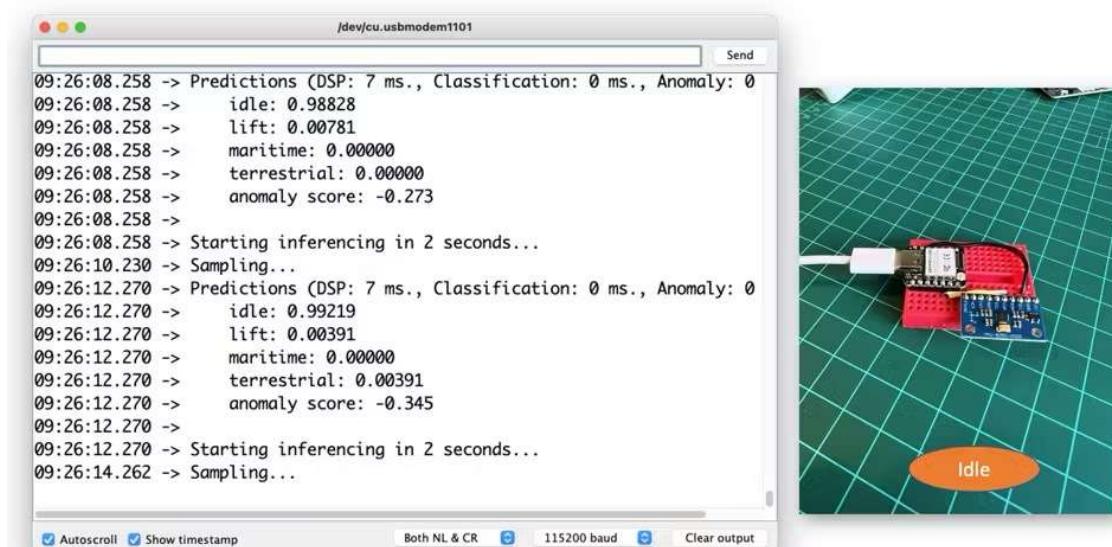
buffer[ix + 0] *= CONVERT_G_TO_MS2;
buffer[ix + 1] *= CONVERT_G_TO_MS2;
buffer[ix + 2] *= CONVERT_G_TO_MS2;

for (int i = 0; i < 3; i++) {
    if (fabs(buffer[ix + i]) > MAX_ACCEPTED_RANGE) {
        buffer[ix + i] = ei_get_sign(buffer[ix + i]) * MAX_ACCEPTED_RANGE;
    }
}

```

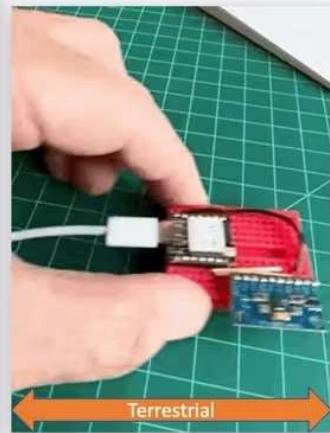
E questo è tutto! Ora si può caricare il codice sul dispositivo e procedere con le inferenze. Il codice completo è disponibile sul [GitHub del progetto](#).

Ora si devono provare i movimenti, osservando il risultato dell'inferenza di ogni classe sulle immagini:



```
/dev/cu.usbmodem1101
Send
09:28:30.557 -> Sampling...
09:28:32.559 -> Predictions (DSP: 7 ms., Classification: 0 ms., Anomaly: 0
09:28:32.559 ->     idle: 0.14844
09:28:32.559 ->     lift: 0.18359
09:28:32.559 ->     maritime: 0.20312
09:28:32.559 ->     terrestrial: 0.46484
09:28:32.559 ->     anomaly score: -0.123
09:28:32.559 ->
09:28:32.559 -> Starting inferencing in 2 seconds...
09:28:34.562 -> Sampling...
09:28:36.567 -> Predictions (DSP: 7 ms., Classification: 0 ms., Anomaly: 0
09:28:36.567 ->     idle: 0.16016
09:28:36.567 ->     lift: 0.17969
09:28:36.567 ->     maritime: 0.19922
09:28:36.567 ->     terrestrial: 0.45703
09:28:36.567 ->     anomaly score: -0.107
09:28:36.567 ->
09:28:36.567 -> Starting inferencing in 2 seconds...
09:28:36.567 ->

Autoscroll Show timestamp Both NL & CR 115200 baud Clear output
```



```
/dev/cu.usbmodem1101
Send
09:27:36.424 -> Predictions (DSP: 7 ms., Classification: 0 ms., Anomaly: 0
09:27:36.424 ->     idle: 0.00000
09:27:36.424 ->     lift: 0.98828
09:27:36.424 ->     maritime: 0.01172
09:27:36.424 ->     terrestrial: 0.00000
09:27:36.424 ->     anomaly score: -0.093
09:27:36.424 ->
09:27:36.424 -> Starting inferencing in 2 seconds...
09:27:38.432 -> Sampling...
09:27:40.446 -> Predictions (DSP: 7 ms., Classification: 0 ms., Anomaly: 0
09:27:40.446 ->     idle: 0.00000
09:27:40.446 ->     lift: 0.98828
09:27:40.446 ->     maritime: 0.01172
09:27:40.446 ->     terrestrial: 0.00000
09:27:40.446 ->     anomaly score: -0.203
09:27:40.446 ->
09:27:40.446 -> Starting inferencing in 2 seconds...
09:27:42.442 -> Sampling...
09:27:42.442 ->

Autoscroll Show timestamp Both NL & CR 115200 baud Clear output
```

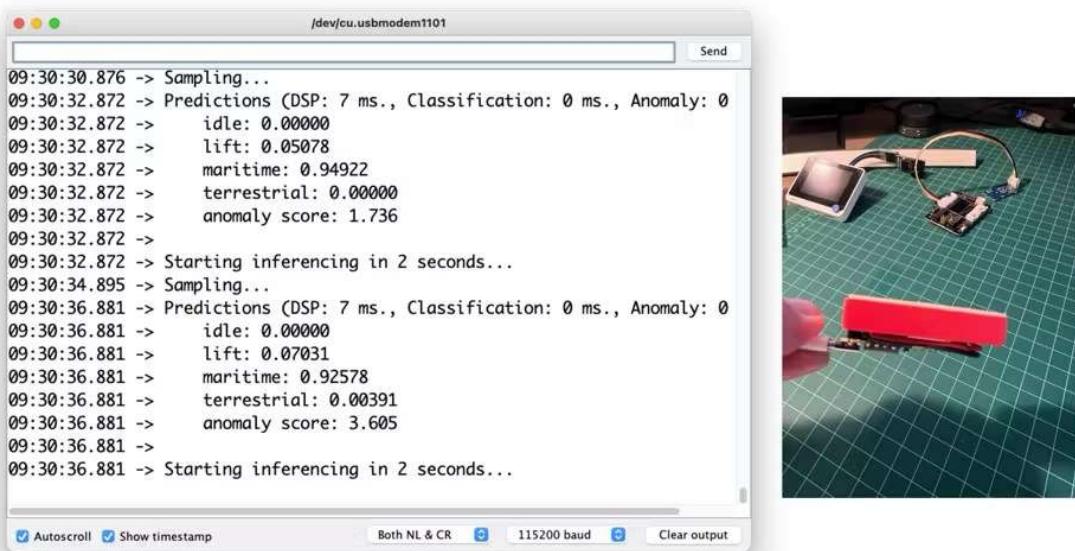


```
/dev/cu.usbmodem1101
Send
09:29:04.641 -> Predictions (DSP: 7 ms., Classification: 0 ms., Anomaly: 0
09:29:04.641 ->     idle: 0.00000
09:29:04.641 ->     lift: 0.02734
09:29:04.641 ->     maritime: 0.96875
09:29:04.641 ->     terrestrial: 0.00391
09:29:04.641 ->     anomaly score: 0.989
09:29:04.641 ->
09:29:04.641 -> Starting inferencing in 2 seconds...
09:29:06.628 -> Sampling...
09:29:08.690 -> Predictions (DSP: 7 ms., Classification: 0 ms., Anomaly: 0
09:29:08.690 ->     idle: 0.00000
09:29:08.690 ->     lift: 0.03906
09:29:08.690 ->     maritime: 0.92578
09:29:08.690 ->     terrestrial: 0.03516
09:29:08.690 ->     anomaly score: 0.697
09:29:08.690 ->
09:29:08.690 -> Starting inferencing in 2 seconds...
09:29:10.706 -> Sampling...
09:29:10.706 ->

Autoscroll Show timestamp Both NL & CR 115200 baud Clear output
```



E, naturalmente, qualche “anomalia”, ad esempio, capovolgere lo XIAO. Il punteggio anomalo sarà superiore a 1:



Conclusione

Per quanto riguarda l’IMU, questo progetto ha utilizzato l’MPU6050 a basso costo, ma potrebbe utilizzare anche altre IMU, ad esempio l’LCM20600 (6 assi), che fa parte del [Seeed Grove - IMU 9DOF \(lcm20600+AK09918\)](#). Si può sfruttare questo sensore, che ha integrato un connettore Grove, che può essere utile nel caso in cui si utilizza lo [XIAO con una scheda di estensione](#), come mostrato di seguito:



Si possono seguire le istruzioni [qui](#) per collegare l’IMU alla MCU. Notare solo che per usare l’accelerometro Grove ICM20600, è essenziale aggiornare i file **I2Cdev.cpp** e **I2Cdev.h** scaricabili dalla [libreria fornita da Seeed Studio](#). Per farlo, si sostituiscono entrambi i file da questo [link](#). Si trova uno sketch per testare l’IMU sul progetto GitHub: [accelerometer_test.ino](#).

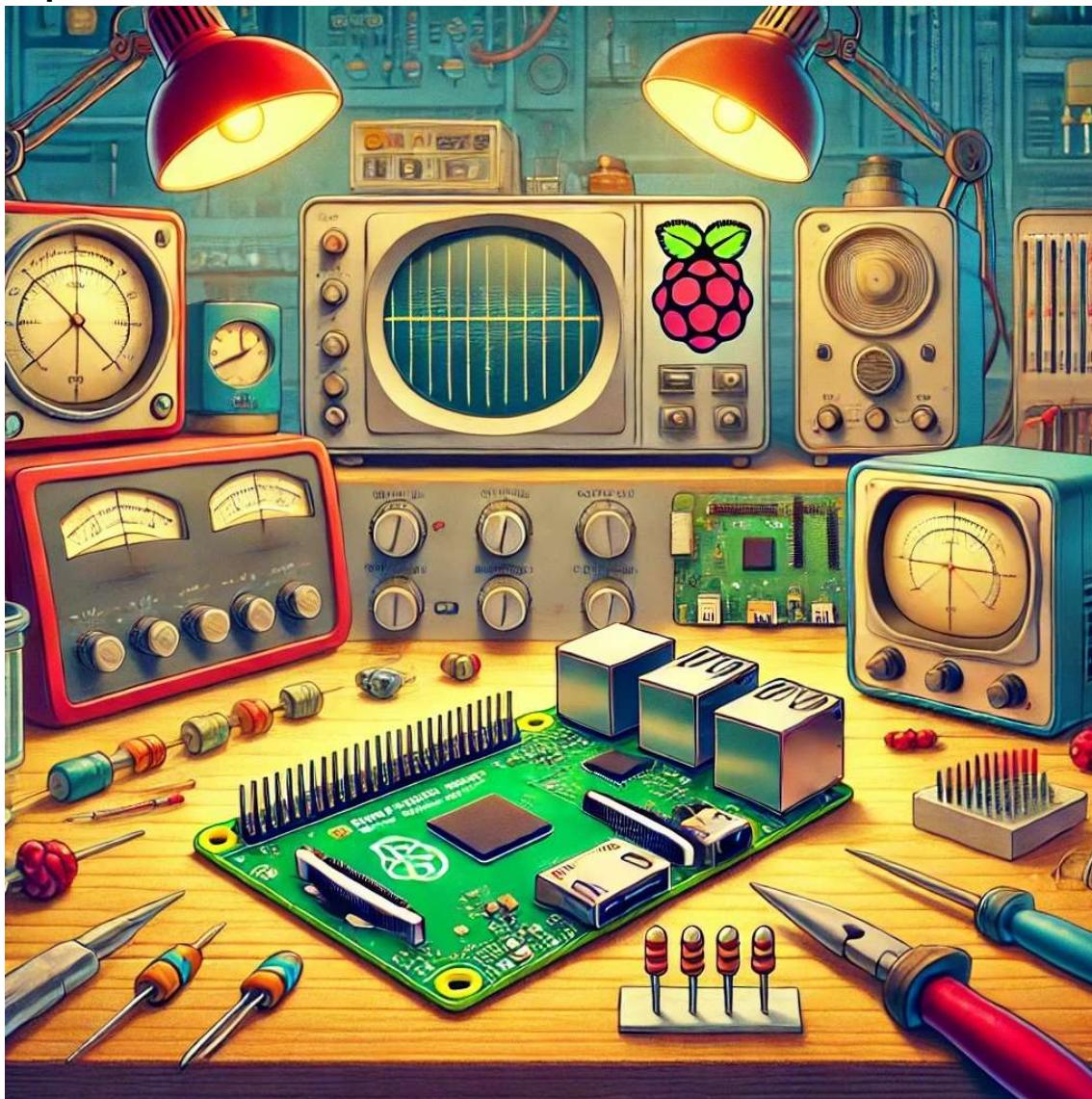
Nel repository GitHub del progetto, si trova l’ultima versione di tutto il codice e altri documenti: [XIAO-ESP32S3 - IMU](#).

Risorse

- [Codici XIAO ESP32S3](#)
- [Edge Impulse Spectral Features Block Colab Notebook](#)
- [Progetto Edge Impulse](#)

Raspberry Pi

Setup



DALL·E prompt - Un ambiente di laboratorio di elettronica ispirato agli anni '50, con uno stile cartoon. Il laboratorio dovrebbe avere apparecchiature vintage, grandi oscilloscopi, vecchie radio a valvole e grandi computer quadrati. La scheda Raspberry Pi 5 è esposta in modo prominente, mostrata accuratamente nelle sue dimensioni reali, simile a una carta di credito, su un banco da lavoro. La scheda Pi è circondata da classici strumenti da laboratorio come un saldatore, resistenze e fili. La scena generale dovrebbe essere vivace, con colori esagerati e dettagli giocosi caratteristici di un cartone animato. Non dovrebbero essere inclusi loghi o testo.

Questo capitolo guiderà attraverso la configurazione dei modelli Raspberry Pi Zero 2 W (*Raspi-Zero*) e Raspberry Pi 5 (*Raspi-5*). Tratteremo la configurazione hardware, l'installazione del sistema operativo, la configurazione iniziale e i test.

Le istruzioni generali per *Raspi-5* si applicano anche alle versioni più vecchie di Raspberry Pi, come *Raspi-3* e *Raspi-4*.

Panoramica

Il Raspberry Pi è un potente e versatile computer a scheda singola che è diventato uno strumento essenziale per gli ingegneri di varie discipline. Sviluppati dalla [Raspberry Pi Foundation](#), questi dispositivi compatti offrono una combinazione unica di convenienza,

potenza di calcolo e ampie capacità GPIO (General Purpose Input/Output), rendendoli ideali per la prototipazione, lo sviluppo di sistemi embedded e progetti di ingegneria avanzata.

Caratteristiche Principali

1. **Potenza di Calcolo:** Nonostante le dimensioni ridotte, Raspberry Pi offre notevoli capacità di elaborazione, con gli ultimi modelli dotati di processori ARM multi-core e fino a 8 GB di RAM.
2. **Interfaccia GPIO:** L'header GPIO a 40 pin consente l'interazione diretta con sensori, attuatori e altri componenti elettronici, facilitando i progetti di integrazione hardware-software.
3. **Ampia Connattività:** Wi-Fi, Bluetooth, Ethernet e più porte USB integrate consentono diversi progetti di comunicazione e networking.
4. **Accesso Hardware di Basso Livello:** Raspberry Pi fornisce accesso a interfacce come I2C, SPI e UART, consentendo un controllo dettagliato e la comunicazione con dispositivi esterni.
5. **Capacità in Tempo Reale:** Con una configurazione appropriata, Raspberry Pi può essere utilizzato per applicazioni soft in tempo reale, rendendolo adatto per sistemi di controllo e attività di elaborazione del segnale.
6. **Efficienza Energetica:** Il basso consumo energetico consente progetti alimentati a batteria e a basso consumo energetico, in particolare in modelli come Pi Zero.

Modelli Raspberry Pi (trattati in questo libro)

1. **Raspberry Pi Zero 2 W (Raspi-Zero):**
 - Ideale per: sistemi embedded compatti
 - Specifiche principali: CPU single-core da 1 GHz (ARM Cortex-A53), 512 MB di RAM, consumo energetico minimo
2. **Raspberry Pi 5 (Raspi-5):**
 - Ideale per: applicazioni più esigenti come edge computing, computer vision e applicazioni edgeAI, inclusi LLM.
 - Specifiche principali: CPU quad-core da 2,4 GHz (ARM Cortex A-76), fino a 8 GB di RAM, interfaccia PCIe per espansioni

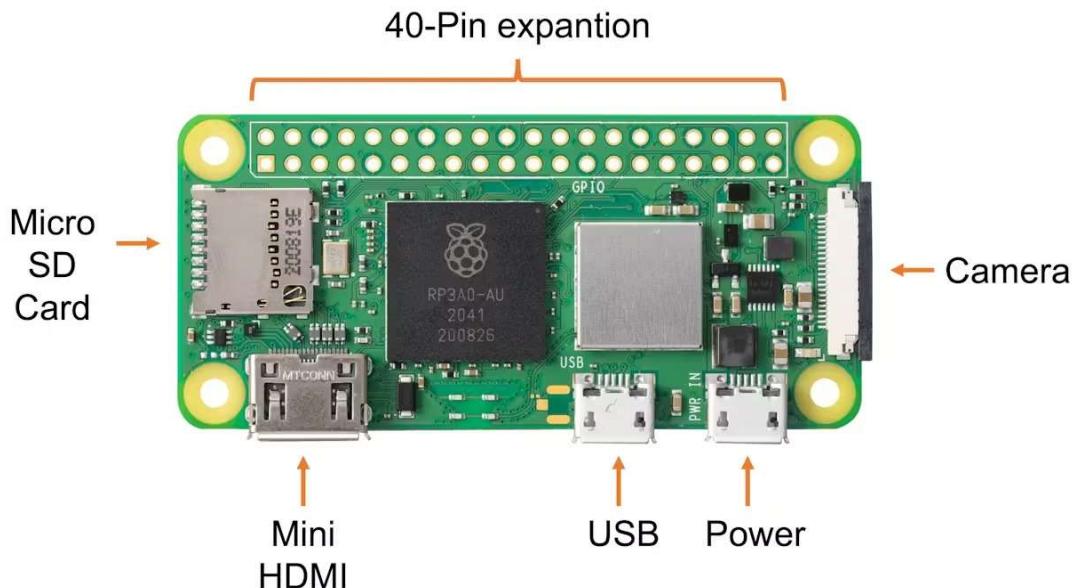
Applicazioni di Ingegneria

1. **Progettazione di Sistemi Embedded:** Sviluppo e prototipi di sistemi embedded per applicazioni reali.
2. **IoT e Dispositivi in Rete:** Creazione di dispositivi interconnessi ed esplorazione di protocolli come MQTT, CoAP e HTTP/HTTPS.
3. **Sistemi di Controllo:** Implementazione di loop di controllo feedback, controller PID e interfaccia per attuatori.
4. **Visione Artificiale e IA:** Utilizzo di librerie come OpenCV e TensorFlow Lite per l'elaborazione delle immagini e l'apprendimento automatico in edge.
5. **Acquisizione e Analisi dei Dati:** Raccolta dati dai sensori, analisi in tempo reale e creazione di sistemi di registrazione dei dati.
6. **Robotica:** Creazione di controller per robot, algoritmi di pianificazione del movimento e interfaccie con driver di motori.
7. **Elaborazione del Segnale:** Analisi del segnale in tempo reale, filtraggio e applicazioni DSP.
8. **Sicurezza di Rete:** Imposta VPN, firewall ed esplora i test di penetrazione della rete.

Questo tutorial guiderà nell'impostazione dei modelli Raspberry Pi più comuni, consentendo di iniziare rapidamente il progetto di apprendimento automatico. Tratteremo la configurazione hardware, l'installazione del sistema operativo e la configurazione iniziale, concentrando sulla preparazione del Pi per le applicazioni di apprendimento automatico.

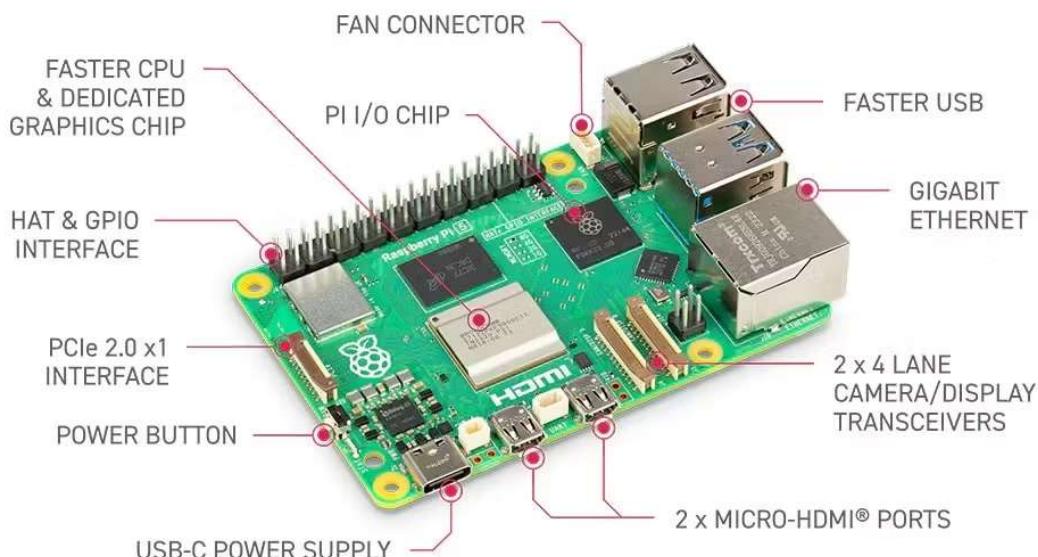
Panoramica Hardware

Raspberry Pi Zero 2W



- **Processore**: CPU Arm Cortex-A53 quad-core a 64 bit da 1 GHz
- **RAM**: SDRAM da 512 MB
- **Wireless**: LAN wireless 802.11 b/g/n da 2,4 GHz, Bluetooth 4.2, BLE
- **Porte**: Mini HDMI, micro USB OTG, connettore per fotocamera CSI-2
- **Alimentazione**: 5 V tramite porta micro USB

Raspberry Pi 5



- **Processore**:
 - Pi 5: CPU Arm Cortex-A76 quad-core a 64 bit @ 2,4 GHz
 - Pi 4: SoC Cortex-A72 quad-core (ARM v8) a 64 bit @ 1,5 GHz
- **RAM**: opzioni da 2 GB, 4 GB o 8 GB (8 GB consigliati per attività di IA)
- **Wireless**: Wireless 802.11ac dual-band, Bluetooth 5.0

- **Porte:** 2 porte micro HDMI, 2 porte USB 3.0, 2 porte USB 2.0, porta fotocamera CSI, porta display DSI
- **Alimentazione:** 5 V CC tramite connettore USB-C (3 A)

Nei laboratori, useremo nomi diversi per riferirci al Raspberry: Raspi, Raspi-5, Raspi-Zero, ecc. Di solito, si usa Raspi quando le istruzioni o i commenti si applicano a tutti i modelli.

Installazione del Sistema Operativo

Il sistema operativo (SO)

Un sistema operativo (SO) è un software fondamentale che gestisce le risorse hardware e software del computer, fornendo servizi standard per i programmi per computer. È il software principale che gira su un computer, fungendo da intermediario tra hardware e software applicativo. Il SO gestisce la memoria, i processi, i driver dei dispositivi, i file e i protocolli di sicurezza del computer.

1. Funzioni chiave:

- Gestione dei processi: Assegnazione del tempo della CPU a diversi programmi
- Gestione della memoria: Assegnazione e liberazione della memoria in base alle necessità
- Gestione del file system: Organizzazione e monitoraggio di file e directory
- Gestione dei dispositivi: Comunicazione con dispositivi hardware collegati
- Interfaccia utente: Fornitura di un modo per gli utenti di interagire con il computer

2. Componenti:

- Kernel: Il nucleo del sistema operativo che gestisce le risorse hardware
- Shell: L'interfaccia utente per interagire con il sistema operativo
- File system: Organizzazione e gestione dell'archiviazione dei dati
- Device driver: Software che consente al sistema operativo di comunicare con l'hardware

Il Raspberry Pi esegue una versione specializzata di Linux progettata per sistemi embedded. Questo sistema operativo, in genere una variante di Debian chiamata Raspberry Pi OS (in precedenza Raspbian), è ottimizzato per l'architettura basata su ARM del Pi e per le risorse limitate.

L'ultima versione di Raspberry Pi OS è basata su [Debian Bookworm](#).

Caratteristiche principali:

1. Leggero: Progettato per funzionare in modo efficiente sull'hardware del Pi.
2. Versatile: Supporta un'ampia gamma di applicazioni e linguaggi di programmazione.
3. Open source: Consente la personalizzazione e i miglioramenti guidati dalla comunità.
4. Supporto GPIO: Consente l'interazione con sensori e altri hardware tramite i pin del Pi.
5. Aggiornamenti regolari: Costantemente migliorato per prestazioni e sicurezza.

Embedded Linux sul Raspberry Pi fornisce un sistema operativo completo in un pacchetto compatto, rendendolo ideale per progetti che vanno da semplici dispositivi IoT ad applicazioni di apprendimento automatico edge più complesse. La sua compatibilità con gli strumenti e le librerie Linux standard lo rende una potente piattaforma per lo sviluppo e la sperimentazione.

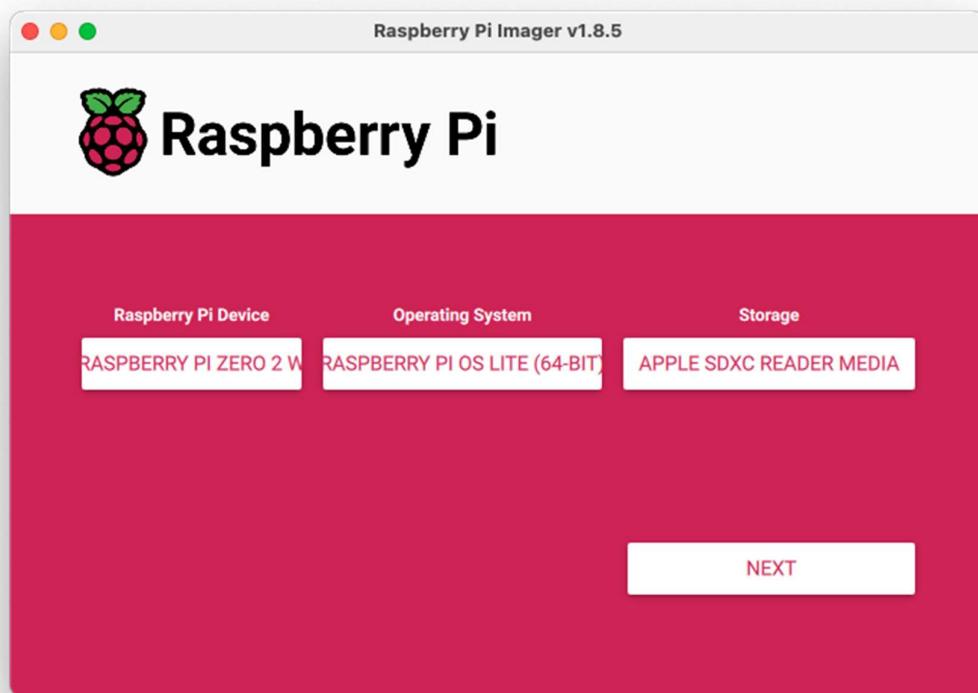
Installazione

Per usare Raspberry Pi, avremo bisogno di un sistema operativo. Di default, Raspberry Pi verifica la presenza di un sistema operativo su qualsiasi scheda SD inserita nello slot, quindi dovremmo installare un sistema operativo usando [Raspberry Pi Imager](#).

Raspberry Pi Imager è uno strumento per scaricare e scrivere immagini su *macOS*, *Windows* e *Linux*. Include molte immagini di sistemi operativi popolari per Raspberry Pi. Useremo Imager anche per preconfigurare credenziali e impostazioni di accesso remoto.

Seguire i passaggi per installare il sistema operativo nel Raspi.

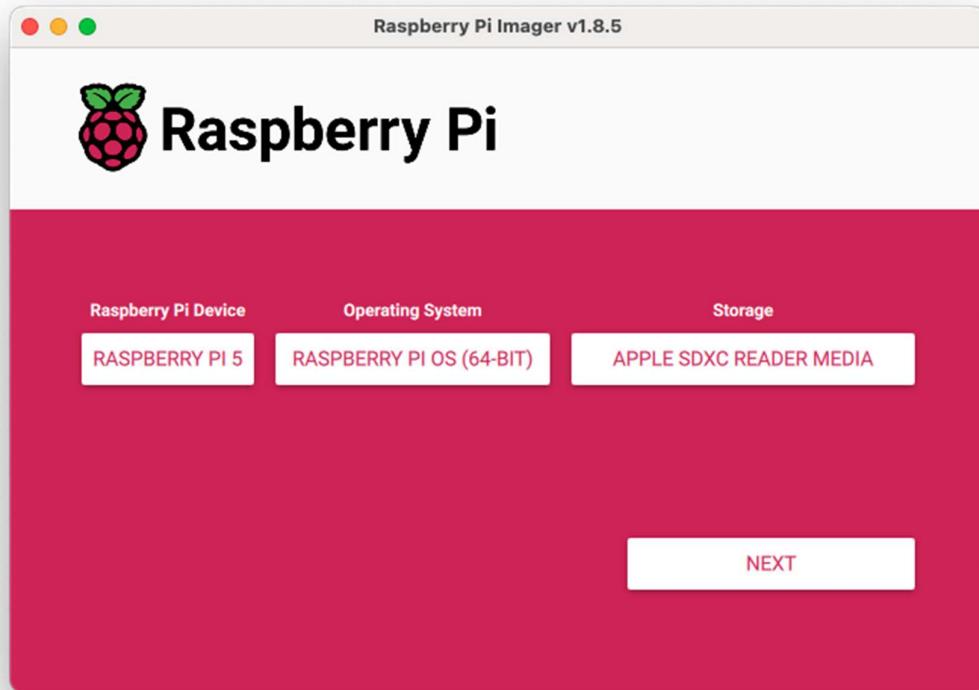
1. [Scaricare](#) e installare Raspberry Pi Imager sul computer.
2. Inserire una scheda microSD nel computer (si consiglia una scheda SD da 32 GB).
3. Aprire Raspberry Pi Imager e selezionare il modello di Raspberry Pi.
4. Scegliere il sistema operativo appropriato:
 - **For Raspi-Zero:** Ad esempio, si può selezionare: [Raspberry Pi OS Lite \(64-bit\)](#).



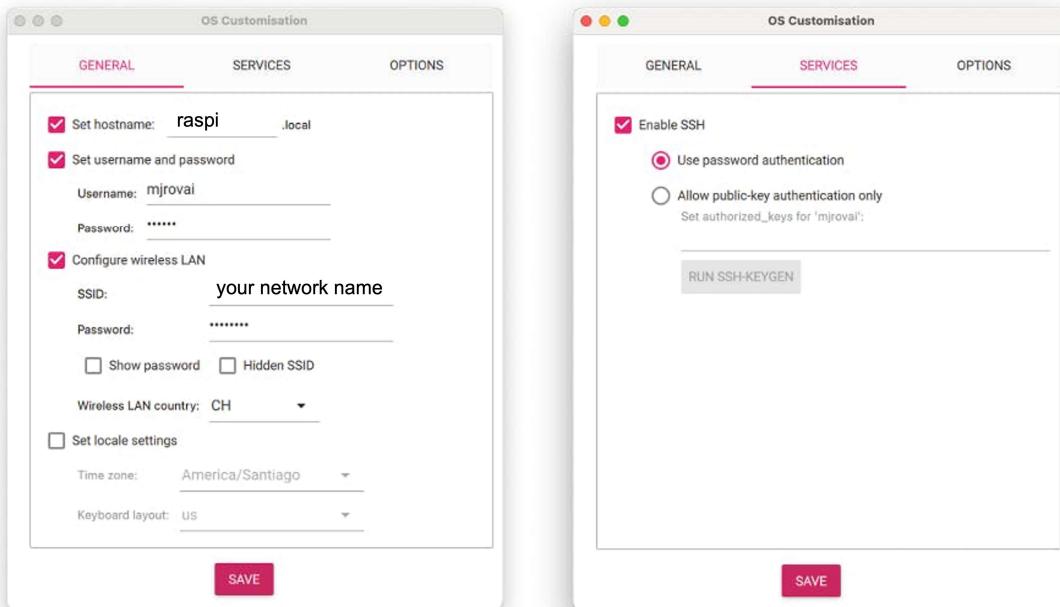
img

Grazie alla ridotta SDRAM (512 MB), il sistema operativo consigliato per Raspi-Zero è la versione a 32 bit. Tuttavia, per eseguire alcuni modelli di apprendimento automatico, come YOLOv8 di Ultralitics, dovremmo usare la versione a 64 bit. Sebbene Raspi-Zero possa eseguire un *desktop*, sceglieremo la versione LITE (senza Desktop) per ridurre la RAM necessaria per il normale funzionamento.

- Per **Raspi-5**: possiamo selezionare la versione completa a 64 bit, che include un *desktop*: [Raspberry Pi OS \(64-bit\)](#)



5. Selezionare la scheda microSD come dispositivo di archiviazione.
6. Cliccare su `Next` e poi sull'icona dell'`ingranaggio` per accedere alle opzioni avanzate.
7. Impostare `hostname`, `nome utente` e `password` di Raspi, configurare il `WiFi` e abilitare `SSH` (molto importante!)



8. Scrivere l'immagine sulla scheda microSD.

Negli esempi qui, useremo nomi host diversi a seconda del dispositivo utilizzato: raspi, raspi-5, raspi-Zero, ecc. Sarebbe utile se lo si sostituisse con quello in uso.

Configurazione Iniziale

1. Inserire la scheda microSD nel Raspberry Pi.

2. Collegare l'alimentazione per avviare il Raspberry Pi.
3. Attendere il completamento del processo di boot iniziale (potrebbero volerci alcuni minuti).

I comandi Linux più comuni da usare con Raspi si trovano [qui](#) o [qui](#).

Accesso Remoto

Accesso SSH

Il modo più semplice per interagire con Raspi-Zero è tramite SSH (“Headless”). Si può usare un terminale (MAC/Linux), [PuTTY](#) (Windows) o qualsiasi altro.

1. Trovare l'indirizzo IP del Raspberry Pi (ad esempio, controllando il router).
2. Sul computer, aprire un terminale e connettersi tramite SSH:

```
ssh username@[raspberry_pi_ip_address]
```

In alternativa, se non si ha l'indirizzo IP, si può provare quanto segue:

```
ssh username@hostname.local
```

per esempio, ssh mjrovai@rpi-5.local, ssh mjrovai@raspi.local, ecc.

```
marcelo_rovai — mjrovai@rpi-5: ~ — ssh mjrovai@rpi-5.local — 103x24
Last login: Thu May  9 14:54:09 on ttys000
(base) marcelo_rovai@Marcelos-MacBook-Pro ~ % ssh mjrovai@rpi-5.local
The authenticity of host 'rpi-5.local (fde3:6154:baa3:1:a1d:2f29:d5a4:8fea)' can't be established.
ED25519 key fingerprint is SHA256:3gMwIEapFZSgHroNr5bLw6dpTiBiH54mYtXwaqn6VVU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'rpi-5.local' (ED25519) to the list of known hosts.
mjrovai@rpi-5.local's password:
Linux rpi-5 6.6.20+rpi-2712 #1 SMP PREEMPT Debian 1:6.6.20-1+rpi1 (2024-03-07) aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Mar 15 15:12:24 2024
mjrovai@rpi-5: ~ $
```

img

Quando si vede il prompt:

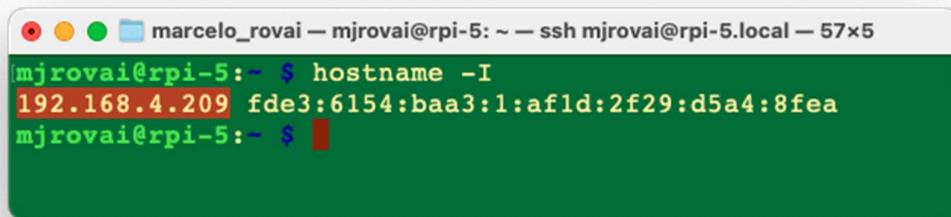
```
mjrovai@rpi-5:~ $
```

Significa che si sta interagendo da remoto con il Raspi. È una buona norma aggiornare/migliorare il sistema regolarmente. Per questo, si deve eseguire:

```
sudo apt-get update
sudo apt upgrade
```

Si deve confermare l'indirizzo IP Raspi. Sul terminale, si può usare:

```
hostname -I
```



```
marcelo_rovai — mjrovai@rpi-5: ~ — ssh mjrovai@rpi-5.local — 57x5
[mjrovai@rpi-5: ~ $ hostname -I
192.168.4.209 fde3:6154:baa3:1:af1d:2f29:d5a4:8fea
mjrovai@rpi-5: ~ $ ]
```

Per spegnere il Raspi tramite terminale:

Per spegnere il Raspberry Pi, ci sono idee migliori che staccare semplicemente il cavo di alimentazione. Questo perché il Raspi potrebbe ancora scrivere dati sulla scheda SD, nel qual caso il semplice spegnimento potrebbe causare la perdita di dati o, peggio ancora, una scheda SD danneggiata.

Per uno spegnimento di sicurezza, usare la riga di comando:

```
sudo shutdown -h now
```

Per evitare possibili perdite di dati e danneggiamenti della scheda SD, prima di rimuovere l'alimentazione, si deve attendere qualche secondo dopo lo spegnimento affinché il LED del Raspberry Pi smetta di lampeggiare e si spenga. Una volta che il LED si spegne, è sicuro spegnere.

Trasferire file tra il Raspi e un computer

Il trasferimento di file tra il Raspi e il nostro computer principale può essere effettuato tramite una chiavetta USB, direttamente sul terminale (con scp) o un programma FTP sulla rete.

Utilizzo del Protocollo Secure Copy Protocol (scp):

Copiare i file sul Raspberry Pi

Creiamo un file di testo sul nostro computer, ad esempio, `test.txt`.



Si può usare qualsiasi editor di testo. Nello stesso terminale, un'opzione è `nano`.

Per copiare il file `test.txt` dal personal computer alla cartella home di un utente sul Raspberry Pi, si esegue il seguente comando dalla directory contenente `test.txt`, sostituendo il segnaposto `<username>` con il nome utente che si usa per accedere al Raspberry Pi e il segnaposto `<pi_ip_address>` con l'indirizzo IP del Raspberry Pi:

```
$ scp test.txt <username>@<pi_ip_address>:~/
```

Notare che `~/` significa che sposteremo il file nella ROOT del Raspi. Si può scegliere qualsiasi cartella nel Raspi. Ma si deve creare la cartella prima di eseguire `scp`, poiché `scp` non lo fa automaticamente.

Ad esempio, trasferiamo il file `test.txt` nella ROOT dell'Raspi-zero, che ha un IP di

`192.168.4.210`:

```
scp test.txt mjrovai@192.168.4.210:~/
```

```
90-LAB-RaspberryPi -- zsh -- 106x7
(base) marcelo_rovai@Marcelos-MacBook-Pro 90-LAB-RaspberryPi % scp test.txt mjrovai@192.168.4.210:~/
mjrovai@192.168.4.210's password:
test.txt
100%   31      3.3KB/s  00:00
(base) marcelo_rovai@Marcelos-MacBook-Pro 90-LAB-RaspberryPi %
```

Abbiamo usato un profilo diverso per differenziare i terminali. L'azione di cui sopra avviene **sul computer**. Ora, andiamo sul Raspi (utilizzando SSH) e controlliamo se il file è lì:

```
marcelo_rovai — mjrovai@raspi-zero: ~ — ssh mjrovai@192.168.4.210 — 62x5
[mjrovai@raspi-zero: ~ $ ls
test.txt
mjrovai@raspi-zero: ~ $ ]
```

Copiare i file dal Raspberry Pi

Per copiare un file chiamato `test.txt` dalla directory home di un utente su un Raspberry Pi alla directory corrente su un altro computer, si esegue il seguente comando **sul computer host**:

```
$ scp <username>@<pi_ip_address>:myfile.txt .
```

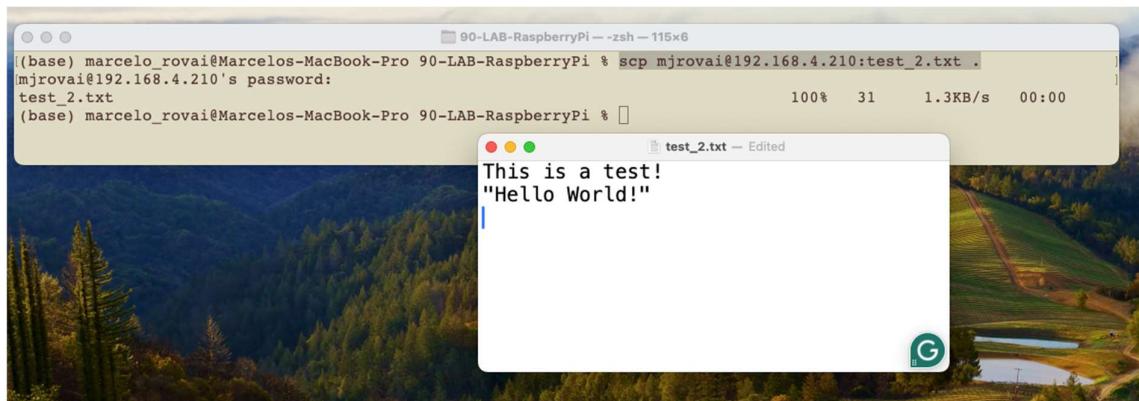
Per esempio:

Sul Raspi, creiamo una copia del file con un altro nome:

```
cp test.txt test_2.txt
```

E sul computer host (in questo caso, un Mac)

```
scp mjrovai@192.168.4.210:test_2.txt .
```

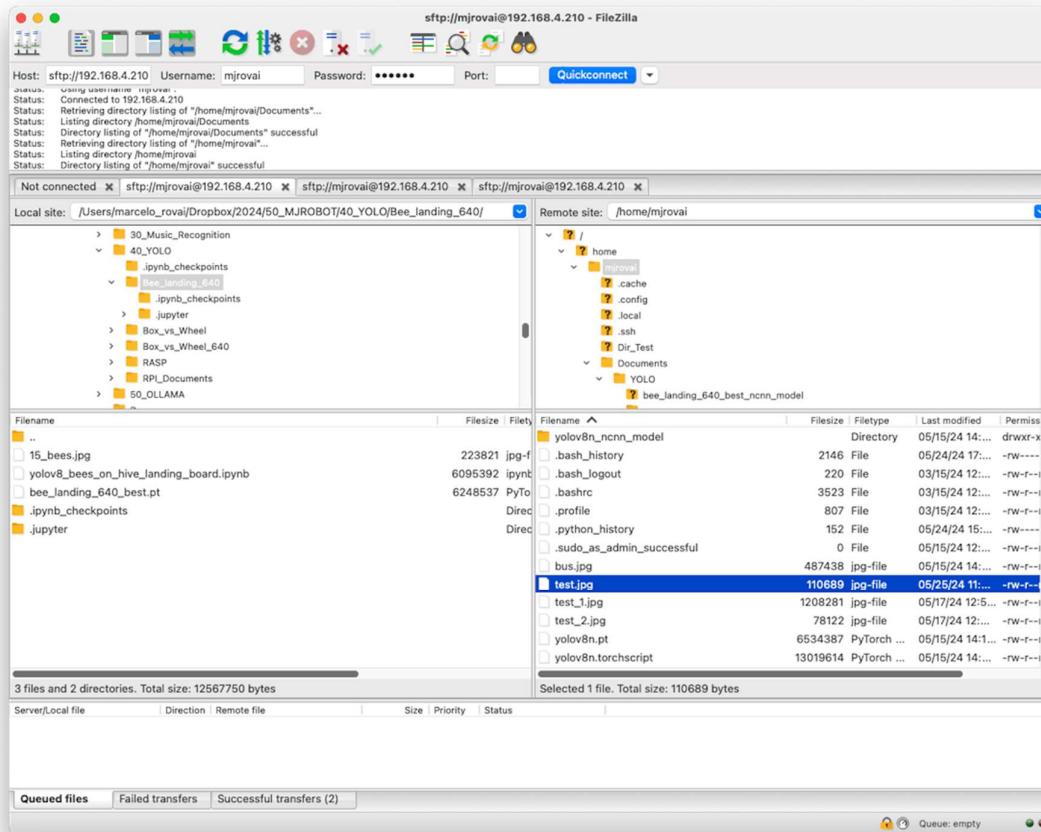


Trasferimento di file tramite FTP

È anche possibile trasferire file tramite FTP, come [FileZilla FTP Client](#). Seguire le istruzioni, installare il programma per il proprio sistema operativo desktop e usare l'indirizzo IP Raspi come Host. Per esempio:

```
sftp://192.168.4.210
```

e inserire nome utente e password di Raspi. Premendo Quickconnect si apriranno due finestre, una per il desktop del computer host (destra) e un'altra per il Raspi (sinistra).



Aumentare la memoria SWAP

Utilizzando `htop`, un visualizzatore di processi interattivo multiplataforma, si possono facilmente monitorare, in tempo reale, le risorse in esecuzione sul Raspi, come l'elenco dei processi, le CPU in esecuzione e la memoria utilizzata. Per lanciare `htop`, si immette il comando sul terminale:

```
htop
```

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
1023	mjrovai	20	0	7512	3072	2432	R	0.7	0.7	0:00.29	htop
1	root	20	0	164M	11312	8416	S	0.0	2.6	0:04.54	/sbin/init
217	root	20	0	49888	14464	13568	S	0.0	3.4	0:00.90	/lib/systemd/systemd-journald
242	root	20	0	26308	6784	4224	S	0.0	1.6	0:01.05	/lib/systemd/systemd-udevd
331	systemd-ti	20	0	90580	6656	5888	S	0.0	1.6	0:00.59	/lib/systemd/systemd-timesyncd
394	systemd-ti	20	0	90580	6656	5888	S	0.0	1.6	0:00.01	/lib/systemd/systemd-timesyncd
396	avahi	20	0	7488	3200	2816	S	0.0	0.7	0:07.44	avahi-daemon: running [raspi-zero]
397	root	20	0	6692	2432	2304	S	0.0	0.6	0:00.04	/usr/sbin/cron -f
399	messagebus	20	0	8792	4096	3328	S	0.0	1.0	0:01.73	/usr/bin/dbus-daemon --system --a
405	polkitd	20	0	229M	6784	6016	S	0.0	1.6	0:00.22	/usr/lib/polkit-1/polkitd --no-de
413	avahi	20	0	7180	1300	1024	S	0.0	0.3	0:00.00	avahi-daemon: chroot helper
416	root	20	0	25404	7168	6272	S	0.0	1.7	0:00.66	/lib/systemd/systemd-logind
420	nobody	20	0	5828	2304	2176	S	0.0	0.5	0:00.13	/usr/sbin/thd --triggers /etc/tri
452	polkitd	20	0	229M	6784	6016	S	0.0	1.6	0:00.00	/usr/lib/polkit-1/polkitd --no-de
454	root	20	0	256M	18816	15872	S	0.0	4.4	0:03.38	/usr/sbin/NetworkManager --no-dae
457	polkitd	20	0	229M	6784	6016	S	0.0	1.6	0:00.07	/usr/lib/polkit-1/polkitd --no-de
461	root	20	0	17176	10368	9088	S	0.0	2.4	0:00.25	/sbin/wpa_supplicant -u -s -O DIR
471	root	20	0	243M	11264	9600	S	0.0	2.6	0:00.46	/usr/sbin/ModemManager

Per quanto riguarda la memoria, tra i dispositivi della famiglia Raspberry Pi, il Raspi-Zero ha la quantità più piccola di SRAM (500 MB), rispetto a una selezione da 2 GB a 8 GB sui Raspi 4 o 5. Per qualsiasi Raspi, è possibile aumentare la memoria disponibile per il sistema con "Swap". La memoria di swap, nota anche come spazio di swap, è una tecnica utilizzata nei sistemi operativi dei computer per archiviare temporaneamente i dati dalla RAM (Random Access Memory) sulla scheda SD quando la RAM fisica è completamente utilizzata. Ciò consente al sistema operativo (SO) di continuare a funzionare anche quando la RAM è piena, il che può prevenire crash o rallentamenti del sistema.

La memoria di swap avvantaggia i dispositivi con RAM limitata, come il Raspi-Zero.

Aumentare lo swap può aiutare a eseguire applicazioni o processi più impegnativi, ma è essenziale bilanciare questo con il potenziale impatto sulle prestazioni dell'accesso frequente al disco.

Per default, la memoria SWAP (Swp) di Rapi-Zero è di soli 100 MB, il che è molto poco per l'esecuzione di alcune applicazioni di apprendimento automatico più complesse ed esigenti (ad esempio, YOLO). Aumentiamola a 2 MB:

Per prima cosa, si disattiva lo swap-file:

```
sudo dphys-swapfile swapoff
```

Poi, si deve aprire e modificare il file `/etc/dphys-swapfile`. Per farlo, useremo nano:

```
sudo nano /etc/dphys-swapfile
```

Cercare la variabile **CONF_SWAPSIZE** (il valore predefinito è 200) e aggiornarla a **2000**:

```
CONF_SWAPSIZE=2000
```

E salvare il file.

Quindi, riattivare lo swapfile e riavviare Raspi-zero:

```
sudo dphys-swapfile setup
sudo dphys-swapfile swapon
sudo reboot
```

Quando il dispositivo viene riavviato (so deve entrare di nuovo con SSH), ci si accorgerà che il valore massimo della memoria di swap mostrato in alto è ora qualcosa come 2 GB (nel nostro caso, 1.95 GB).

Per mantenere in esecuzione *htop*, si deve aprire un'altra finestra del terminale per interagire continuamente con il Raspi.

Installazione di una Fotocamera

Il Raspi è un dispositivo eccellente per applicazioni di visione artificiale; è necessaria una fotocamera. Possiamo installare una webcam USB standard sulla porta micro-USB utilizzando un adattatore USB OTG (Raspi-Zero e Raspi-5) oppure un modulo telecamera collegato alla porta Raspi CSI (Camera Serial Interface).

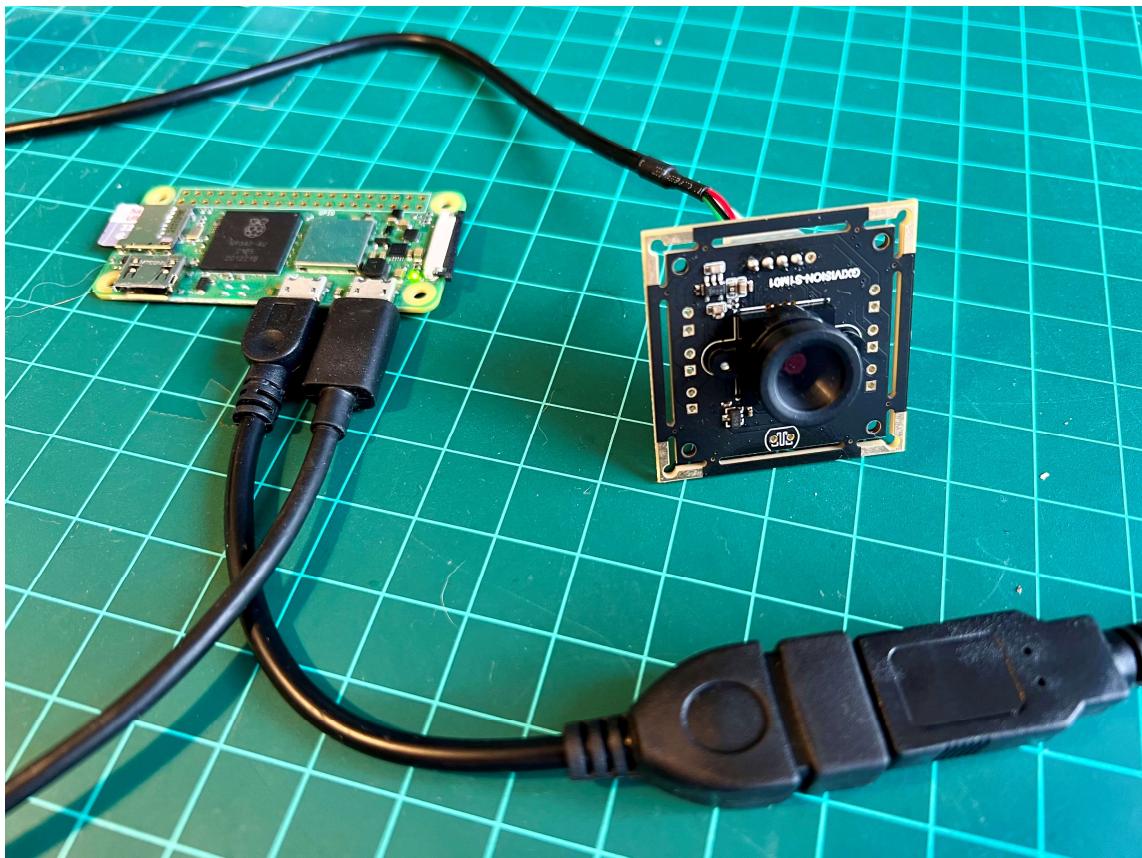
Le webcam USB hanno generalmente una qualità inferiore rispetto ai moduli fotocamera che si collegano alla porta CSI. Inoltre, non possono essere controllate utilizzando i comandi `raspistill` e `raspivid` nel terminale o il pacchetto di registrazione `picamera` in Python. Tuttavia, potrebbero esserci dei motivi per collegare una telecamera USB al Raspberry Pi, ad esempio perché è molto più facile configurare più telecamere con un singolo Raspberry Pi, cavi lunghi o semplicemente perché si ha una telecamera del genere a portata di mano.

Installazione di una Webcam USB

1. Spegnere il Raspi:

```
sudo shutdown -h now
```

2. Collegare la webcam USB (modulo telecamera USB 30fps, 1280x720) al Raspi (in questo esempio, si usa il Raspi-Zero, ma le istruzioni funzionano per tutti i Raspi).



3. Riaccendere ed eseguire SSH
4. Per verificare che la fotocamera USB è riconosciuta, eseguire:

```
lsusb
```

Si dovrebbe vedere la fotocamera elencata nell'output.

```
mjrovai@raspi-zero:~$ lsusb
Bus 001 Device 003: ID 0c45:1915 Microdia USB 2.0 Camera
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
mjrovai@raspi-zero:~$ 
mjrovai@raspi-zero:~$ 
```

5. Per scattare una foto di prova con la fotocamera USB, si usa:

fswebcam test_image.jpg

Questo salverà un’immagine denominata “test_image.jpg” nella directory corrente.

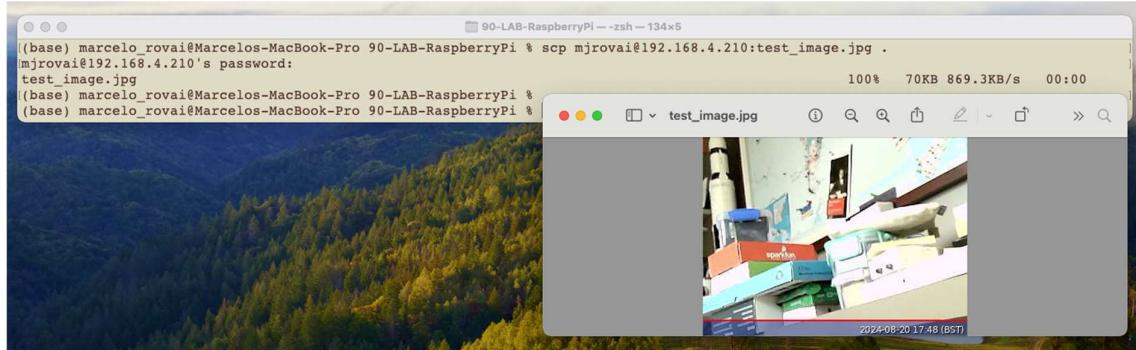
```
mjrovai@raspi-zero:~$ fswebcam test_image.jpg
--- Opening /dev/video0...
Trying source module v4l2...
/dev/video0 opened.
No input was specified, using the first.
Adjusting resolution from 384x288 to 320x240.
--- Capturing frame...
Captured frame in 0.00 seconds.
--- Processing captured image...
Fontconfig warning: ignoring UTF-8: not a valid region tag
Writing JPEG image to 'test_image.jpg'.
[mjrovai@raspi-zero:~$ ls
Documents test.txt test_2.txt test_image.jpg
mjrovai@raspi-zero:~$ 
```

6. Dato che stiamo usando SSH per connetterci al Rapsi, dobbiamo trasferire l’immagine al computer principale in modo da poterla visualizzare. Possiamo usare FileZilla o SCP per questo:

Si apre un terminale **sul computer host** e si esegue:

scp mjrovai@raspi-zero.local:~/test_image.jpg .

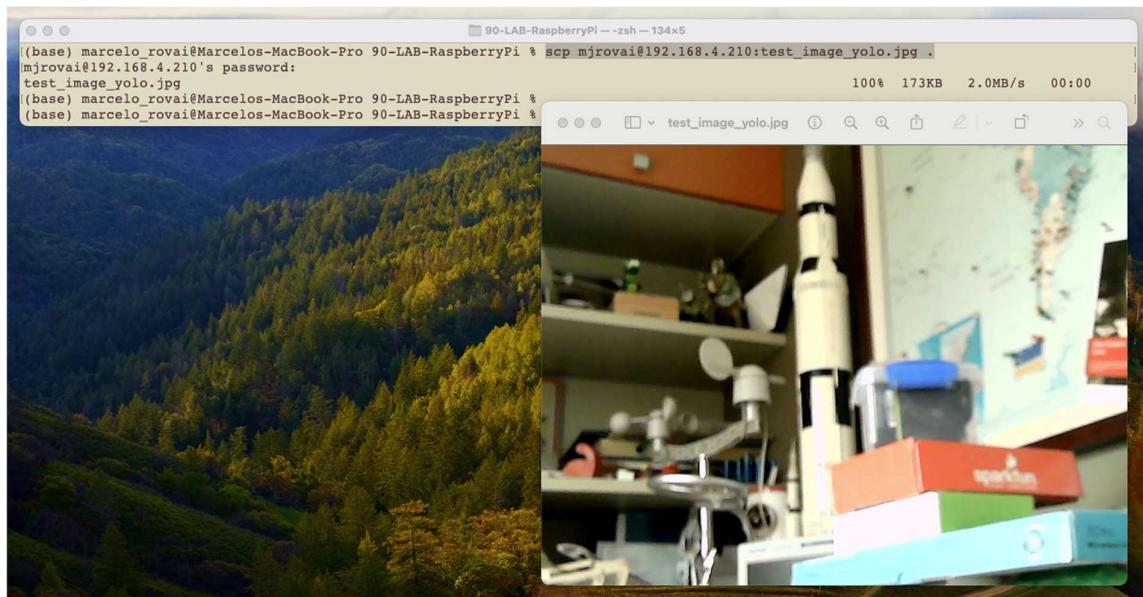
Sostituire “mjrovai” col nome utente e “raspi-zero” con il nome host di Pi.



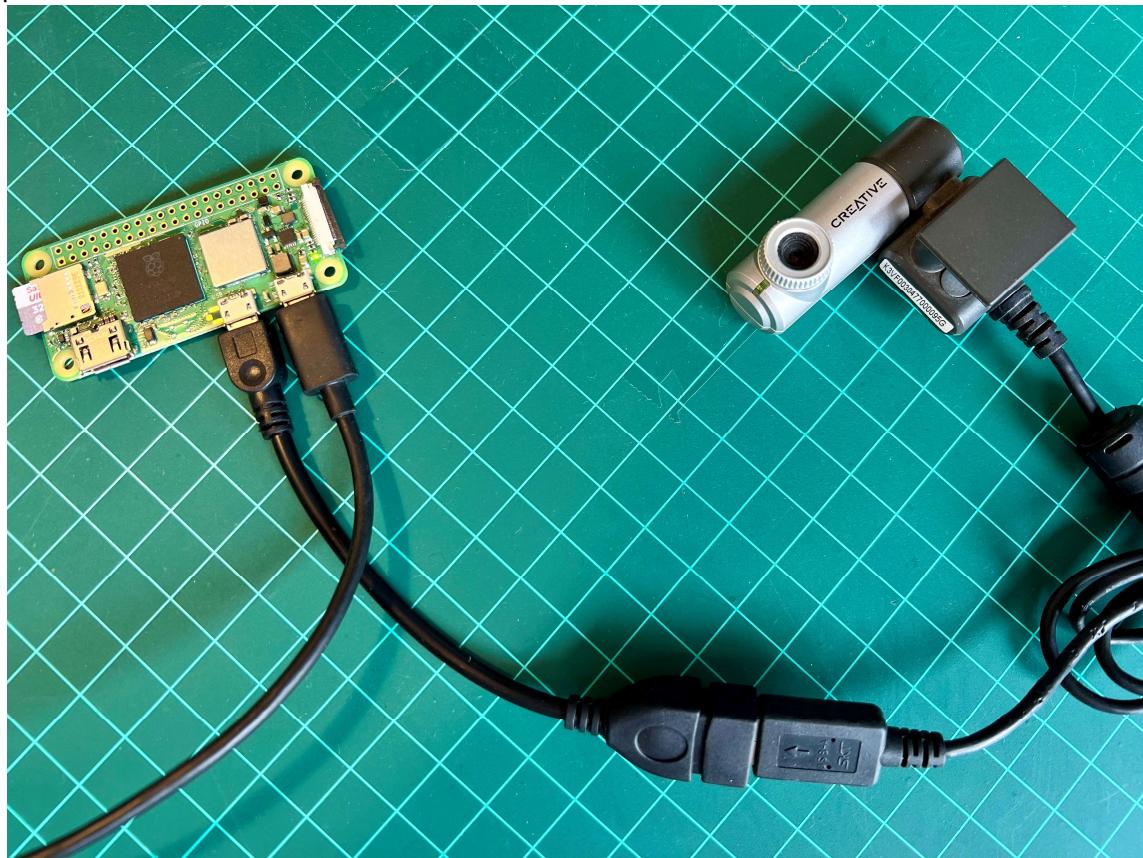
7. Se la qualità dell’immagine non è soddisfacente, si possono regolare varie impostazioni; ad esempio, definire una risoluzione adatta a YOLO (640x640):

fswebcam -r 640x640 --no-banner test_image_yolo.jpg

Questo cattura un’immagine ad alta risoluzione senza il banner di default.



Si può anche usare una normale webcam USB:



Verificandola con `lsusb`

```
mjrovai@raspi-zero:~$ lsusb
Bus 001 Device 002: ID 041e:401f Creative Technology, Ltd Webcam Notebook [PD1171]
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
mjrovai@raspi-zero:~$
```

Streaming Video

Per lo streaming video (che richiede più risorse), possiamo installare e usare mjpg-streamer:

Per prima cosa, si installa Git:

```
sudo apt install git
```

Ora, dovremmo installare le dipendenze necessarie per mjpg-streamer, clonare il repository e procedere con l'installazione:

```
sudo apt install cmake libjpeg62-turbo-dev
git clone https://github.com/jacksonliam/mjpg-streamer.git
cd mjpg-streamer/mjpg-streamer-experimental
make
sudo make install
```

Quindi avviare lo streaming con:

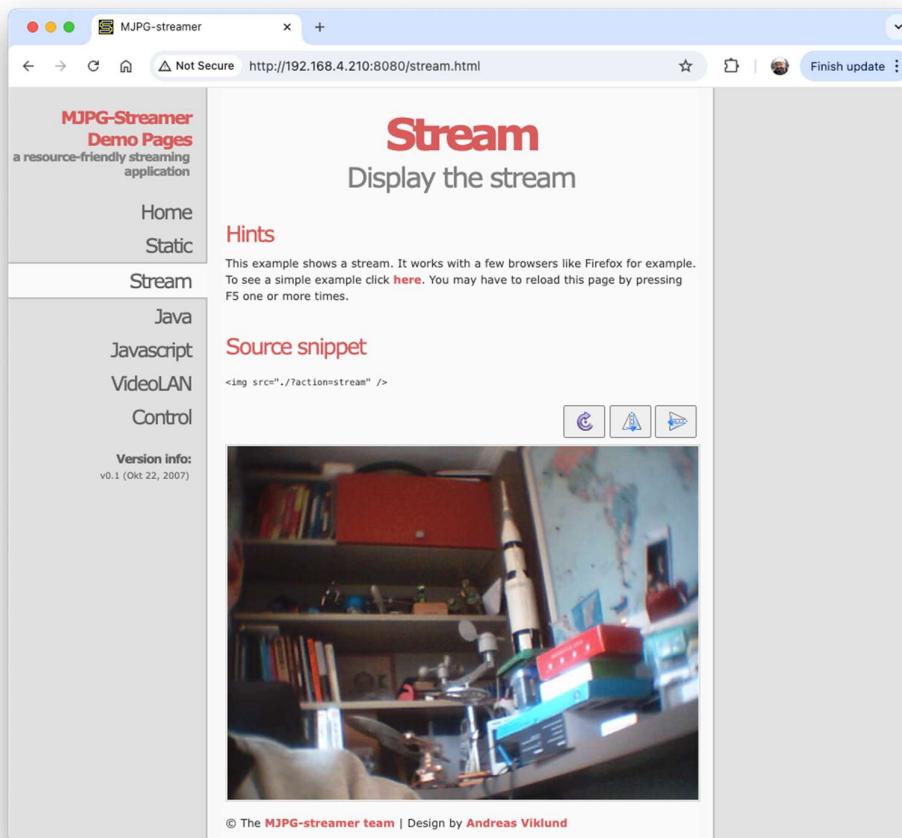
```
mjpg_streamer -i "input_uvc.so" -o "output_http.so -w ./www"
```

Possiamo poi accedere allo streaming aprendo un browser Web e andando su:

http://<your_pi_ip_address>:8080. In questo caso: <http://192.168.4.210:8080>

Dovremmo vedere una pagina web con opzioni per visualizzare lo stream. Cliccare su k che dice "Stream" o provare ad accedere a:

http://<raspberry_pi_ip_address>:8080/?action=stream



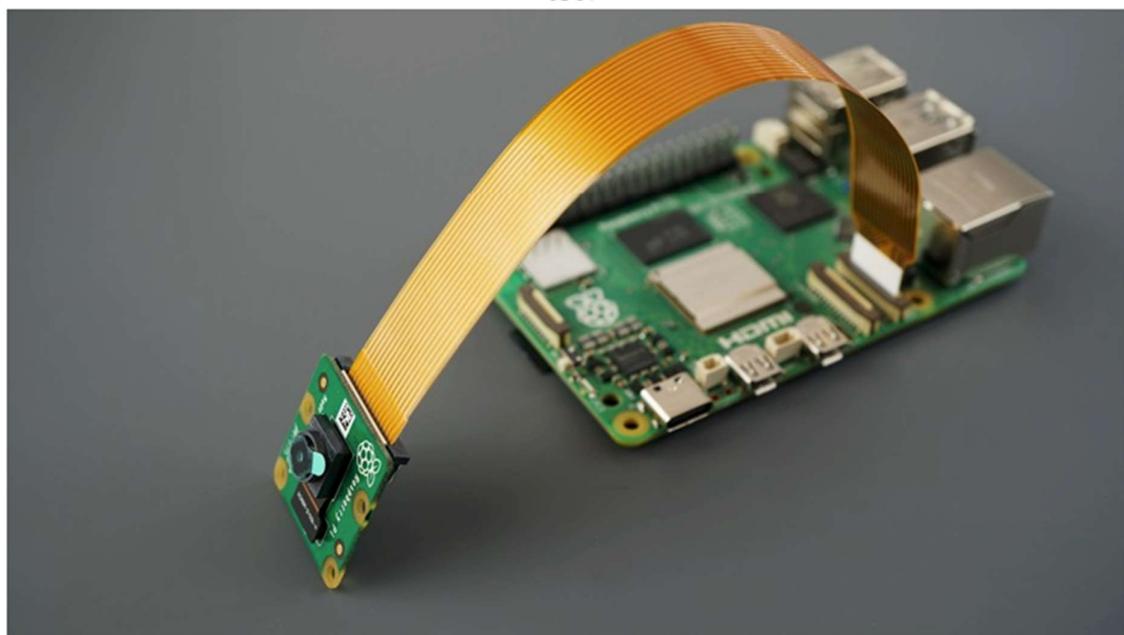
Installazione di un Modulo Fotocamera sulla porta CSI

Ora ci sono diversi moduli fotocamera Raspberry Pi. Il modello originale da 5 megapixel è stato [rilasciato](#) nel 2013, seguito da un [8-megapixel Camera Module 2](#) che è stato rilasciato nel 2016. L'ultimo modello di fotocamera è il [12-megapixel Camera Module 3](#), rilasciato nel 2023.

La fotocamera originale da 5 MP (**Arducam OV5647**) non è più disponibile da Raspberry Pi, ma può essere trovata da diversi fornitori alternativi. Di seguito è riportato un esempio di tale fotocamera su un Raspi-Zero.



Ecco un altro esempio di un modulo fotocamera v2, dotato di un sensore **Sony IMX219** da 8 megapixel:



Qualsiasi modulo fotocamera funzionerà sul Raspberry Pi, ma per questo è necessario aggiornare il file `configuration.txt`:

```
sudo nano /boot/firmware/config.txt
```

In fondo al file, ad esempio, per usare la fotocamera Arducam OV5647 da 5 MP, si deve aggiungere la riga:

```
dtoverlay=ov5647, cam0
```

Oppure per il modulo v2, che ha la fotocamera Sony IMX219 da 8 MP:

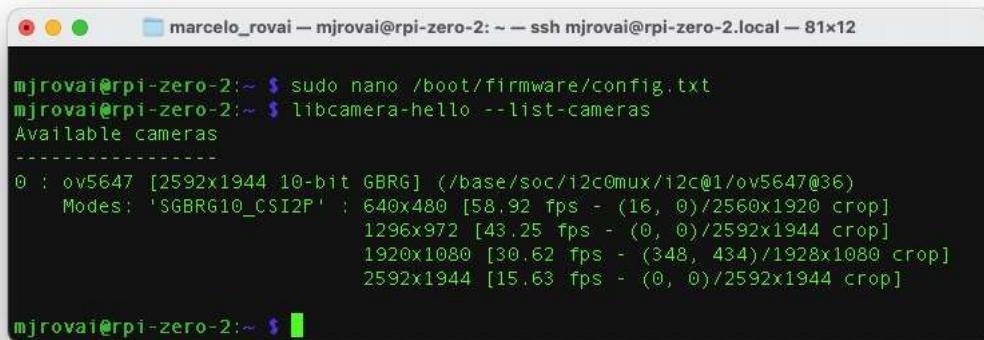
```
dtoverlay=imx219,cam0
```

Salvare il file (CTRL+O [INVIO] CRTL+X) e riavviare il Raspi:

Sudo reboot

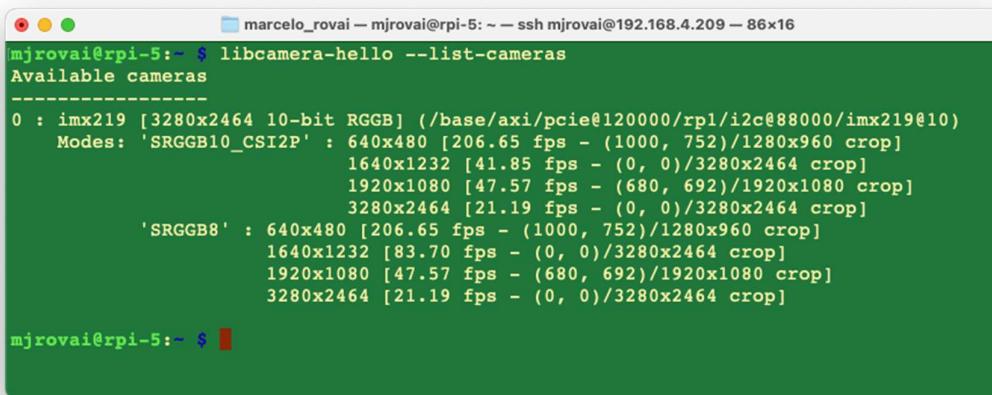
Dopo l'avvio, si può vedere se la fotocamera è elencata:

```
libcamera-hello --list-cameras
```



```
mjrovai@rpi-zero-2:~ $ sudo nano /boot/firmware/config.txt
mjrovai@rpi-zero-2:~ $ libcamera-hello --list-cameras
Available cameras
-----
0 : ov5647 [2592x1944 10-bit GBRG] (/base/soc/i2c0mux/i2c@1/ov5647@36)
    Modes: 'SGBRG10_CSI2P' : 640x480 [58.92 fps - (16, 0)/2560x1920 crop]
            1296x972 [43.25 fps - (0, 0)/2592x1944 crop]
            1920x1080 [30.62 fps - (348, 434)/1928x1080 crop]
            2592x1944 [15.63 fps - (0, 0)/2592x1944 crop]

mjrovai@rpi-zero-2:~ $
```



```
mjrovai@rpi-5:~ $ libcamera-hello --list-cameras
Available cameras
-----
0 : imx219 [3280x2464 10-bit RGGB] (/base/axi/pcie@120000/rp1/i2c@88000/imx219@10)
    Modes: 'SRGGB10_CSI2P' : 640x480 [206.65 fps - (1000, 752)/1280x960 crop]
            1640x1232 [41.85 fps - (0, 0)/3280x2464 crop]
            1920x1080 [47.57 fps - (680, 692)/1920x1080 crop]
            3280x2464 [21.19 fps - (0, 0)/3280x2464 crop]
    'SRGGB8' : 640x480 [206.65 fps - (1000, 752)/1280x960 crop]
            1640x1232 [83.70 fps - (0, 0)/3280x2464 crop]
            1920x1080 [47.57 fps - (680, 692)/1920x1080 crop]
            3280x2464 [21.19 fps - (0, 0)/3280x2464 crop]

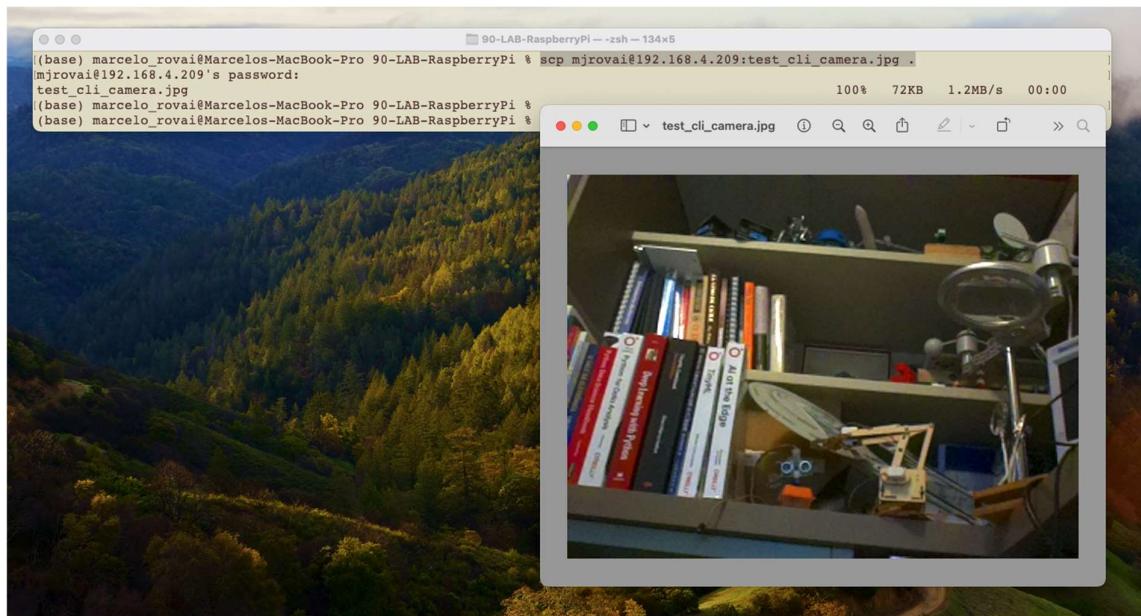
mjrovai@rpi-5:~ $
```

libcamera è una libreria software open source che supporta i sistemi di telecamere direttamente dal sistema operativo Linux sui processori Arm. Riduce al minimo il codice proprietario in esecuzione sulla GPU Broadcom.

Catturiamo un'immagine jpeg con una risoluzione di 640 x 480 per il test e salviamola in un file denominato `test_cli_camera.jpg`

```
rpicam-jpeg --output test_cli_camera.jpg --width 640 --height 480
```

se vogliamo vedere il file salvato, dovremmo usare `ls -f`, che elenca tutto il contenuto della directory corrente in formato lungo. Come prima, possiamo usare `scp` per visualizzare l'immagine:

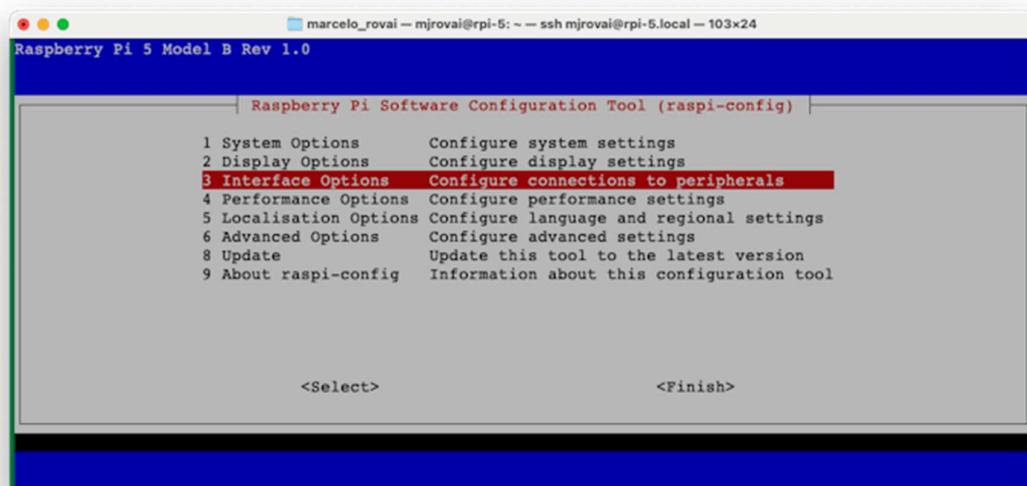


Esecuzione del desktop Raspi da remoto

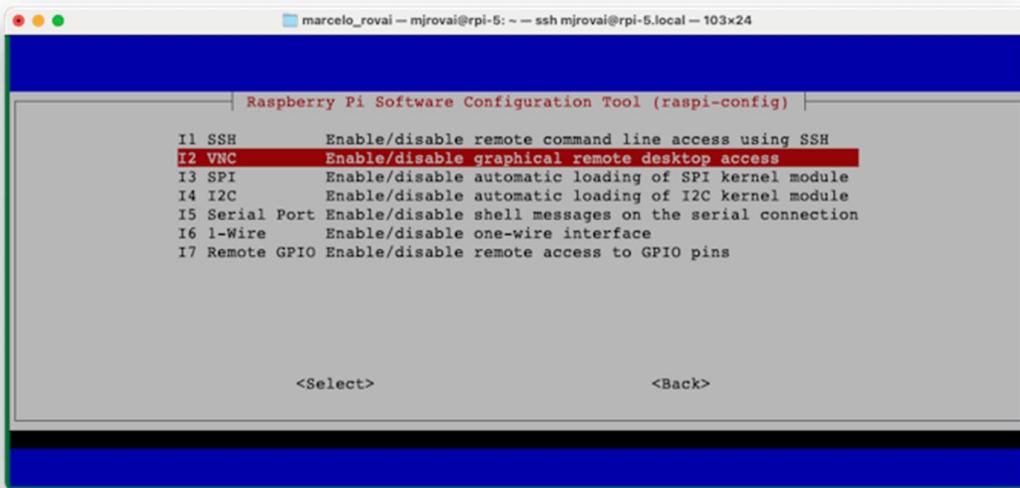
Sebbene abbiamo interagito principalmente con il Raspberry Pi utilizzando comandi del terminale tramite SSH, possiamo accedere all'intero ambiente desktop grafico da remoto se abbiamo installato il sistema operativo Raspberry Pi completo (ad esempio, [Raspberry Pi OS \(64-bit\)](#)). Questo può essere particolarmente utile per le attività che traggono vantaggio da un'interfaccia visuale. Per abilitare questa funzionalità, dobbiamo configurare un server VNC (Virtual Network Computing) sul Raspberry Pi. Ecco come fare:

1. Abilitare il server VNC:

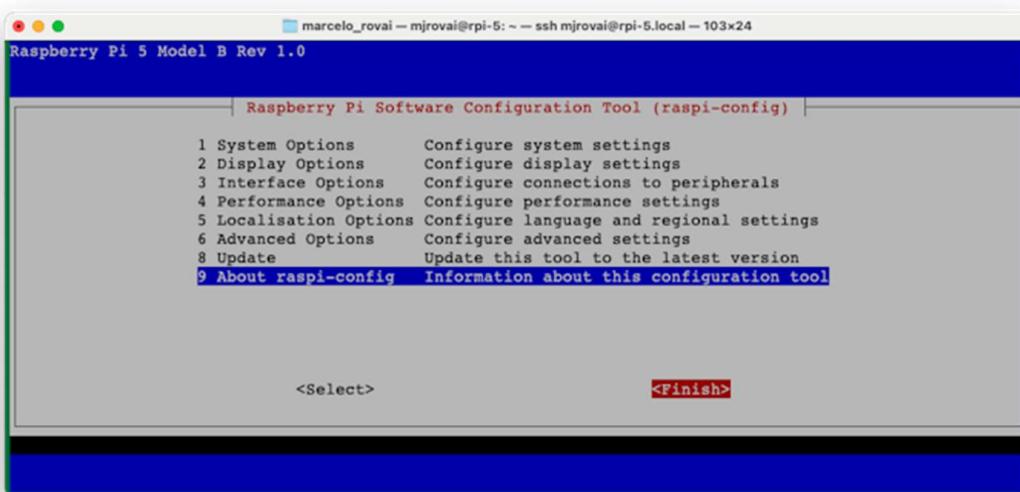
- Connettersi al Raspberry Pi tramite SSH.
- Esegui lo strumento di configurazione Raspberry Pi immettendo: `bash sudo raspi-config`
- Andare su `Interface Options` utilizzando i tasti freccia.



- Selezionare `VNC` e `Yes` per abilitare server VNC.



- Uscire dallo strumento di configurazione, salvando le modifiche quando richiesto.

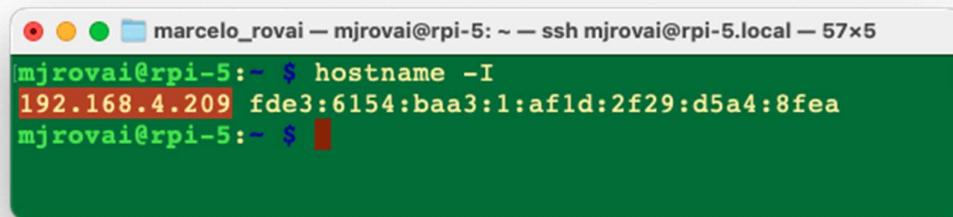


2. Installare un VNC Viewer sul computer:

- Scaricare e installare un'applicazione di visualizzazione VNC sul computer principale. Le opzioni più diffuse includono RealVNC Viewer, TightVNC o VNC Viewer di RealVNC. Installeremo [VNC Viewer](#) di RealVNC.

3. Una volta installato, si deve confermare l'indirizzo IP Raspi. Ad esempio, sul terminale, si può usare:

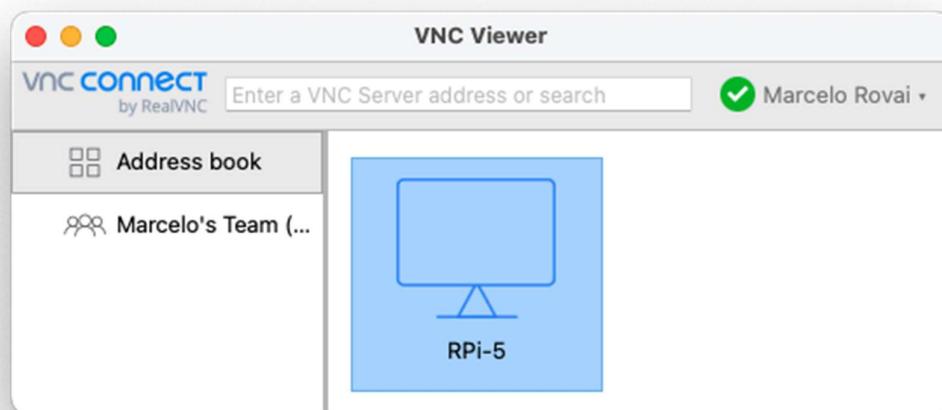
```
hostname -I
```



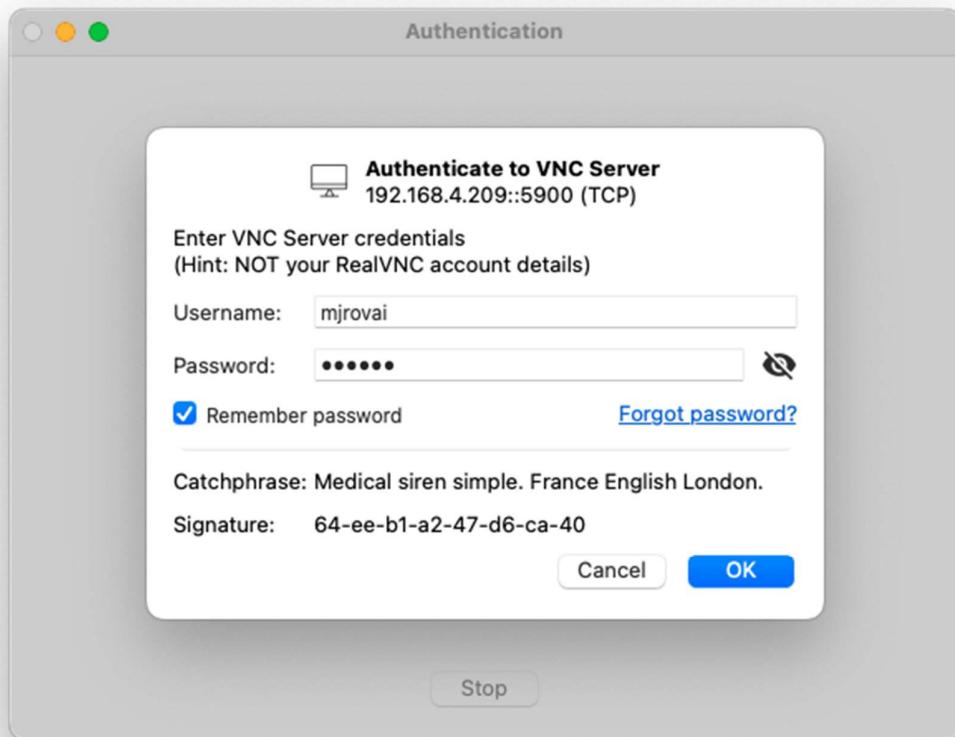
```
marcelo_rovai — mjrovai@rpi-5: ~ — ssh mjrovai@rpi-5.local — 57x5
[mjrovai@rpi-5:~] $ hostname -I
192.168.4.209 fde3:6154:baa3:1:af1d:2f29:d5a4:8fea
mjrovai@rpi-5:~] $
```

4. Connetersi al Raspberry Pi:

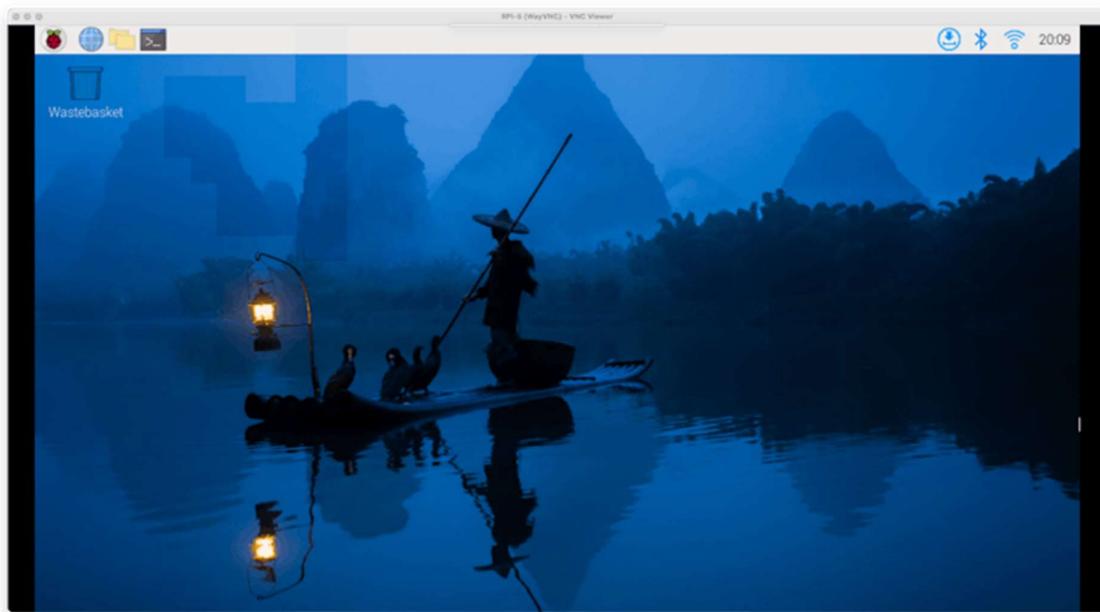
- Aprire l'applicazione di visualizzazione VNC.



- Inserire l'indirizzo IP e il nome host del Raspberry Pi.
- Quando richiesto, inserire il nome utente e la password del Raspberry Pi.

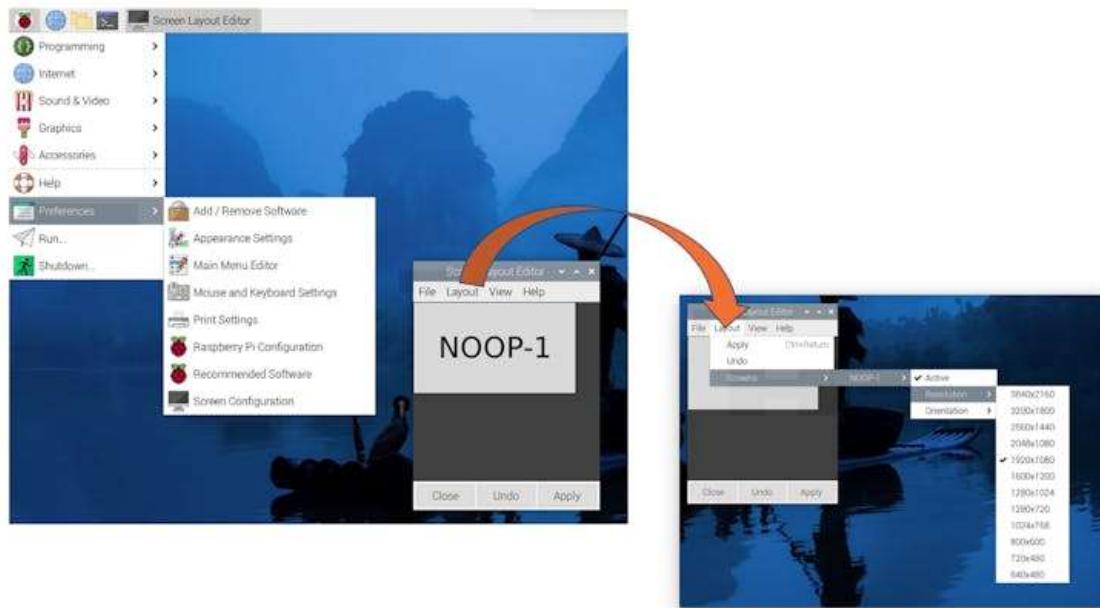


5. Il Raspberry Pi 5 Desktop dovrebbe apparire sul monitor del computer.



6. Regolare le Impostazioni dello Schermo (se necessario):

- Una volta connessi, regolare la risoluzione dello schermo per una visualizzazione ottimale. Questo può essere fatto tramite le impostazioni desktop del Raspberry Pi o modificando il file config.txt.
- Facciamolo tramite le impostazioni desktop. Raggiungere il menù (l'icona Raspberry nell'angolo in alto a sinistra) e selezionare la migliore definizione dello schermo per il proprio monitor:



Aggiornamento e Installazione del Software

1. Aggiorna il sistema:

```
sudo apt update && sudo apt upgrade -y
```

2. Installare il software essenziale:

```
sudo apt install python3-pip -y
```

3. Abilita pip per i progetti Python:

```
sudo rm /usr/lib/python3.11/EXTERNALLY-MANAGED
```

Considerazioni Specifiche del Modello

Raspberry Pi Zero (Raspi-Zero)

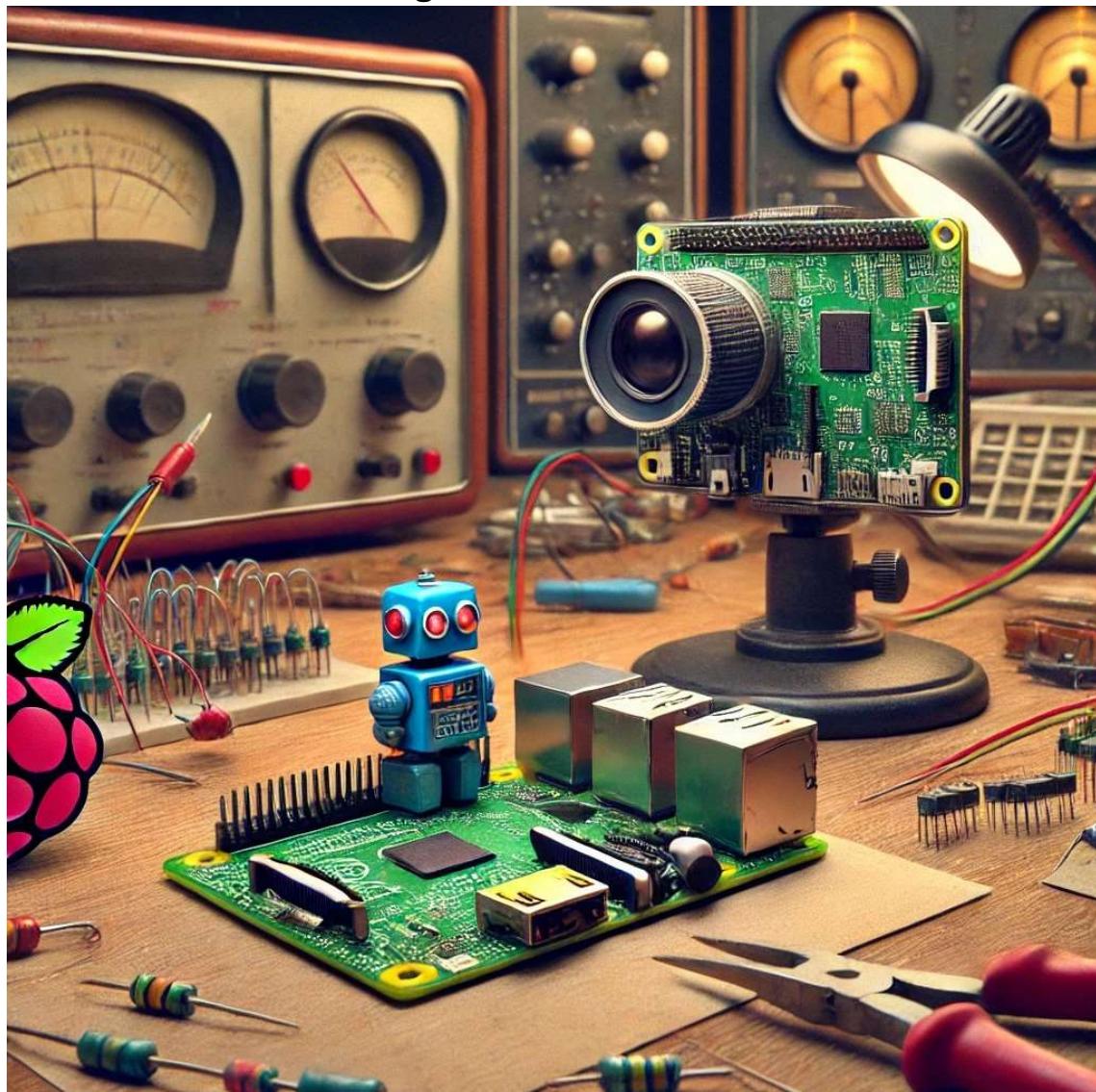
- Potenza di elaborazione limitata, ideale per progetti leggeri
- Per risparmiare risorse è meglio utilizzare una configurazione headless (SSH).
- Valutare l'aumento dello spazio di swap per attività che richiedono molta memoria.
- Può essere utilizzato per Image Classification e Object Detection Labs ma non per LLM (SLM).

Raspberry Pi 4 o 5 (Raspi-4 o Raspi-5)

- Adatto per progetti più impegnativi, tra cui IA e apprendimento automatico.
- Può eseguire l'intero ambiente desktop senza problemi.
- Raspi-4 può essere utilizzato per Image Classification e Object Detection Labs ma non funzionerà bene con LLM (SLM).
- Per Raspi-5, prendere in considerazione l'utilizzo di un dissipatore attivo per la gestione della temperatura durante attività intensive, come nel laboratorio LLM (SLM).

Ricordarsi di adattare i requisiti del progetto in base allo specifico modello di Raspberry Pi in uso. Raspi-Zero è ottimo per progetti a basso consumo e con vincoli di spazio, mentre i modelli Raspi-4 o 5 sono più adatti per attività più intensive dal punto di vista computazionale.

Classificazione delle Immagini



DALL-E prompt - Un'immagine di copertina per un capitolo "Classificazione delle immagini" in un tutorial Raspberry Pi, progettata nello stesso stile vintage da laboratorio di elettronica anni '50 delle copertine precedenti. La scena dovrebbe presentare un Raspberry Pi collegato a un modulo fotocamera, con la fotocamera che cattura una foto del piccolo robot blu fornito dall'utente. Il robot dovrebbe essere posizionato su un banco da lavoro, circondato da classici strumenti da laboratorio come saldatori, resistenze e fili. Lo sfondo del laboratorio dovrebbe includere apparecchiature d'epoca come oscilloscopi e radio a valvole, mantenendo il tocco dettagliato e nostalgico dell'epoca. Non dovrebbero essere inclusi testo o loghi.

Panoramica

La classificazione delle immagini è un'attività fondamentale nella visione artificiale che comporta la categorizzazione di un'immagine in una delle diverse classi predefinite. È una pietra angolare dell'intelligenza artificiale, che consente alle macchine di interpretare e comprendere le informazioni visive in un modo che imita la percezione umana.

La classificazione delle immagini si riferisce all'assegnazione di un'etichetta o di una categoria a un'intera immagine in base al suo contenuto visivo. Questa attività è fondamentale nella visione artificiale e ha numerose applicazioni in vari settori. L'importanza della classificazione delle immagini risiede nella sua capacità di automatizzare le attività di comprensione visiva che altrimenti richiederebbero l'intervento umano.

Applicazioni in Scenari del Mondo Reale

La classificazione delle immagini ha trovato la sua strada in numerose applicazioni del mondo reale, rivoluzionando vari settori:

- Sanità: Assistenza nell'analisi delle immagini mediche, come l'identificazione di anomalie nelle radiografie o nelle risonanze magnetiche.
- Agricoltura: Monitoraggio della salute delle colture e rilevamento delle malattie delle piante tramite immagini aeree.
- Automotive: Abilitazione di sistemi avanzati di assistenza alla guida e veicoli autonomi per riconoscere segnali stradali, pedoni e altri veicoli.
- Vendita al Dettaglio: Potenziamento delle capacità di ricerca visiva e sistemi di gestione automatizzata dell'inventario.
- Sicurezza e Sorveglianza: Potenziamento dei sistemi di rilevamento delle minacce e riconoscimento facciale.
- Monitoraggio Ambientale: Analisi delle immagini satellitari per studi su deforestazione, pianificazione urbana e cambiamenti climatici.

Vantaggi dell'Esecuzione della Classificazione su Dispositivi Edge come Raspberry Pi

L'implementazione della classificazione delle immagini su dispositivi edge come Raspberry Pi offre diversi vantaggi interessanti:

1. Bassa latenza: L'elaborazione delle immagini in locale elimina la necessità di inviare dati ai server cloud, riducendo significativamente i tempi di risposta.
2. Funzionalità Offline: La classificazione può essere eseguita senza una connessione Internet, rendendola adatta ad ambienti remoti o con problemi di connettività.
3. Privacy e Sicurezza: I dati sensibili delle immagini rimangono sul dispositivo locale, affrontando i problemi di privacy dei dati e i requisiti di conformità.
4. Efficacia in Termini di Costi: Elimina la necessità di costose risorse di cloud computing, in particolare per attività di classificazione continue o ad alto volume.
5. Scalabilità: Consente architetture di elaborazione distribuite in cui più dispositivi possono funzionare in modo indipendente o in una rete.
6. Efficienza Energetica: I modelli ottimizzati su hardware dedicato possono essere più efficienti dal punto di vista energetico rispetto alle soluzioni basate su cloud, il che è fondamentale per applicazioni alimentate a batteria o remote.
7. Personalizzazione: L'implementazione di modelli specializzati o aggiornati di frequente, su misura per casi d'uso specifici, è più gestibile.

Possiamo creare soluzioni di visione artificiale più reattive, sicure ed efficienti sfruttando la potenza di dispositivi edge come Raspberry Pi per la classificazione delle immagini. Questo approccio apre nuove possibilità per integrare l'elaborazione visiva intelligente in varie applicazioni e ambienti.

Nelle sezioni seguenti, esploreremo come implementare e ottimizzare la classificazione delle immagini su Raspberry Pi, sfruttando questi vantaggi per creare sistemi di visione artificiale potenti ed efficienti.

Impostazione dell'Ambiente

Aggiornamento di Raspberry Pi

Innanzitutto, assicurarsi che il Raspberry Pi sia aggiornato:

```
sudo apt update  
sudo apt upgrade -y
```

Installazione delle Librerie Richieste

Installare le librerie necessarie per l'elaborazione delle immagini e l'apprendimento automatico:

```
sudo apt install python3-pip
sudo rm /usr/lib/python3.11/EXTERNALLY-MANAGED
pip3 install --upgrade pip
```

Impostazione di un Ambiente Virtuale (Facoltativo ma Consigliato)

Creare un ambiente virtuale per gestire le dipendenze:

```
python3 -m venv ~/tflite
source ~/tflite/bin/activate
```

Installazione di TensorFlow Lite

Siamo interessati a eseguire **inferenza**, ovvero l'esecuzione di un modello TensorFlow Lite su un dispositivo per effettuare previsioni basate sui dati di input. Per eseguire un'inferenza con un modello TensorFlow Lite, dobbiamo eseguirla tramite un **interprete**. L'interprete

TensorFlow Lite è progettato per essere snello e veloce. L'interprete utilizza un ordinamento grafico statico e un allocatore di memoria personalizzato (meno dinamico) per garantire un carico, inizializzazione ed esecuzione latenza minimi.

Utilizzeremo il **runtime TensorFlow Lite** per Raspberry Pi, una libreria semplificata per l'esecuzione di modelli di apprendimento automatico su dispositivi mobili e embedded, senza includere tutti i pacchetti TensorFlow.

```
pip install tflite_runtime --no-deps
```

La wheel installata: tflite_runtime-2.14.0-cp311-cp311-manylinux_2_34_aarch64.whl

Installazione di Librerie Python Aggiuntive

Installare le librerie Python richieste per l'uso con Image Classification:

Se è installata un'altra versione di Numpy, disinstallarla prima.

```
pip3 uninstall numpy
```

Installare la versione 1.23.2, che è compatibile con tflite_runtime.

```
pip3 install numpy==1.23.2
pip3 install Pillow matplotlib
```

Creazione di una directory di lavoro:

Se si lavora su Raspi-Zero con il sistema operativo minimo (No Desktop), non si ha un albero di directory user-pre-defined (lo si può verificare con `ls`). Quindi, creiamone uno:

```
mkdir Documents
cd Documents/
mkdir TFLITE
cd TFLITE/
mkdir IMG_CLASS
cd IMG_CLASS
mkdir models
cd models
```

Su Raspi-5, /Documents dovrebbe esserci.

Ottenere un Modello di Classificazione delle Immagini Pre-addestrato:

Un modello pre-addestrato appropriato è fondamentale per una classificazione delle immagini di successo su dispositivi con risorse limitate come Raspberry Pi. **MobileNet** è progettato per applicazioni di visione mobile e embedded con un buon equilibrio tra accuratezza e velocità. Versioni: MobileNetV1, MobileNetV2, MobileNetV3. Scarichiamo la V2:

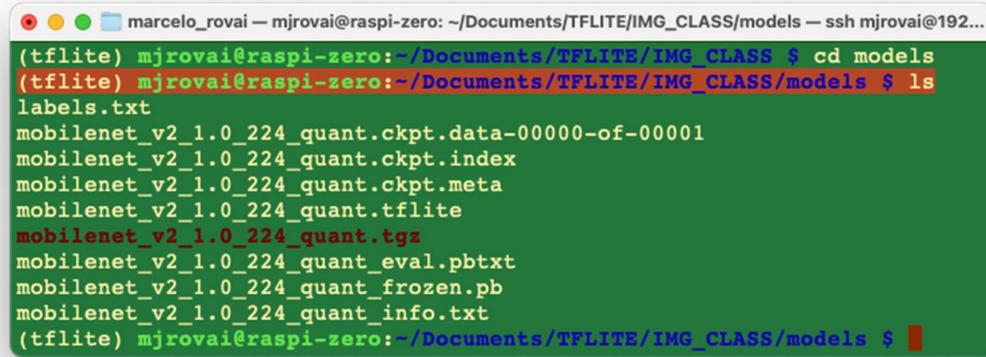
```
wget https://storage.googleapis.com/download.tensorflow.org/models/tflite_11_05_08/mobilenet_v2_1.0_224_quant.tgz
```

```
tar xzf mobilenet_v2_1.0_224_quant.tgz
```

Prelevarne le **etichette**:

```
wget https://github.com/Mjrovai/EdgeML-with-Raspberry-Pi/blob/main/IMG_CLASS/models/labels.txt
```

Alla fine, si dovrebbero avere i modelli nella sua directory:



```
marcelo_rovai — mjrovai@raspi-zero: ~/Documents/TFLITE/IMG_CLASS/models — ssh mjrovai@192.168.4.210:~$ cd models
(tflite) mjrovai@raspi-zero:~/Documents/TFLITE/IMG_CLASS/models $ ls
labels.txt
mobilenet_v2_1.0_224_quant.ckpt.data-00000-of-00001
mobilenet_v2_1.0_224_quant.ckpt.index
mobilenet_v2_1.0_224_quant.ckpt.meta
mobilenet_v2_1.0_224_quant.tflite
mobilenet_v2_1.0_224_quant.tgz
mobilenet_v2_1.0_224_quant_eval.pbtxt
mobilenet_v2_1.0_224_quant_frozen.pb
mobilenet_v2_1.0_224_quant_info.txt
(tflite) mjrovai@raspi-zero:~/Documents/TFLITE/IMG_CLASS/models $
```

Ci serviranno solo il modello `mobilenet_v2_1.0_224_quant.tflite` e `labels.txt`. Si possono eliminare gli altri file.

Impostazione di Jupyter Notebook (Facoltativo)

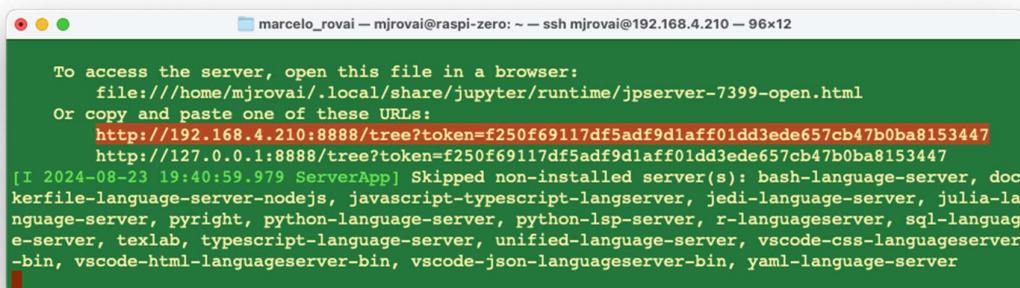
Se si preferisce usare Jupyter Notebook per lo sviluppo:

```
pip3 install jupyter
jupyter notebook --generate-config
```

Per eseguire Jupyter Notebook, si lancia il comando (cambiare l'indirizzo IP per il proprio):

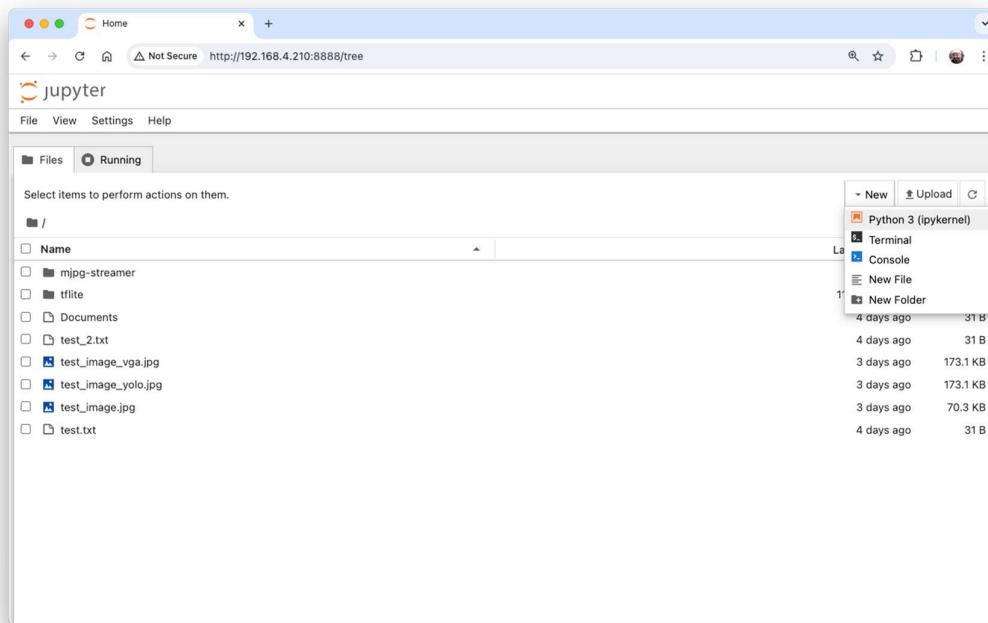
```
jupyter notebook --ip=192.168.4.210 --no-browser
```

Sul terminale, si può vedere l'indirizzo URL locale per aprire il notebook:



```
marcelo_rovai — mjrovai@raspi-zero: ~ — ssh mjrovai@192.168.4.210 — 96x12
To access the server, open this file in a browser:
  file:///home/mjrovai/.local/share/jupyter/runtime/jpserver-7399-open.html
Or copy and paste one of these URLs:
  http://192.168.4.210:8888/tree?token=f250f69117df5adf9d1aff01dd3ede657cb47b0ba8153447
  http://127.0.0.1:8888/tree?token=f250f69117df5adf9d1aff01dd3ede657cb47b0ba8153447
[I 2024-08-23 19:40:59.979 ServerApp] Skipped non-installed server(s): bash-language-server, dockerfile-language-server-nodejs, javascript-typescript-langserver, jedi-language-server, julia-language-server, pyright, python-language-server, python-lsp-server, r-languageserver, sql-languageserver, texlab, typescript-language-server, unified-language-server, vscode-css-languageserver-bin, vscode-html-languageserver-bin, vscode-json-languageserver-bin, yaml-language-server
```

Vi si può accedere da un altro dispositivo inserendo l'indirizzo IP del Raspberry Pi e il token fornito in un browser Web (il token lo si può copiare dal terminale).



Definire la directory di lavoro nel Raspi e creare un nuovo notebook Python 3.

Verifica della Configurazione

Testare la configurazione eseguendo un semplice script Python:

```
import tensorflow.python.platform
import numpy as np
from PIL import Image

print("NumPy:", np.__version__)
print("Pillow:", Image.__version__)

# Try to create a TFLite Interpreter
model_path = "./models/mobilenet_v2_1.0_224_quant.tflite"
interpreter = tensorflow.python.framework.tensorflow_pywrap_tensorflow.TF_NewInterpreter(model_path)
interpreter.allocate_tensors()
print("TFLite Interpreter created successfully!")
```

Si può creare lo script Python usando nano sul terminale, salvandolo con **CTRL+O + ENTER + CTRL+X**

```
GNU nano 7.2          setup_test.py *
import tensorflow.python.platform
import numpy as np
from PIL import Image

print("NumPy:", np.__version__)
print("Pillow:", Image.__version__)

# Try to create a TFLite Interpreter
model_path = "./models/mobilenet_v2_1.0_224_quant.tflite"
interpreter = tensorflow.python.framework.tensorflow_pywrap_tensorflow.TF_NewInterpreter(model_path)
interpreter.allocate_tensors()
print("TFLite Interpreter created successfully!")

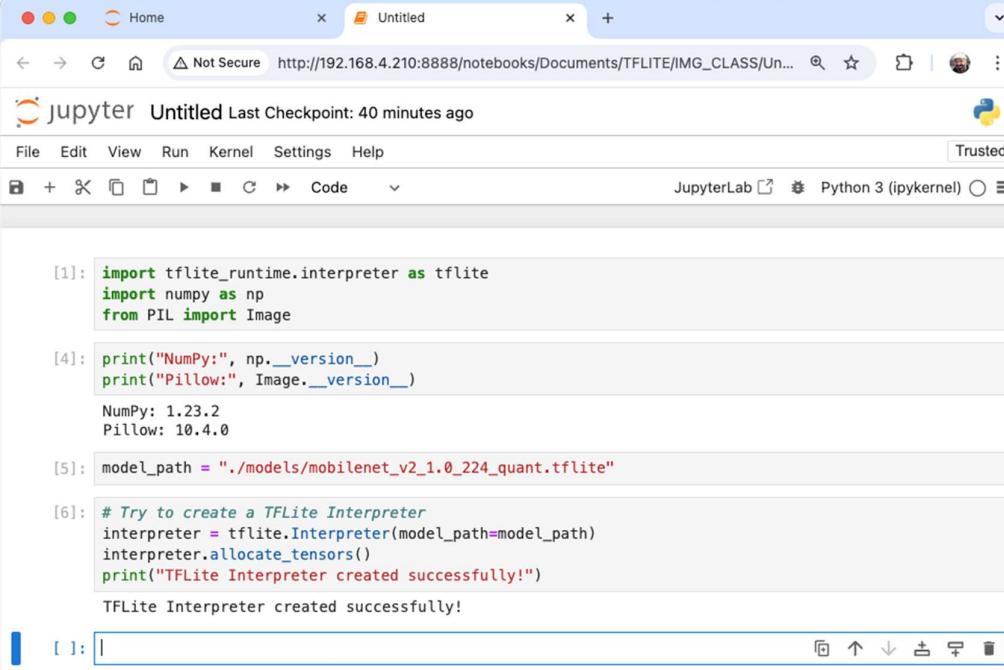
^G Help      ^O Write Out ^W Where Is ^K Cut      ^T Execute
^X Exit      ^R Read File ^\ Replace   ^U Paste    ^J Justify
```

Ed eseguirlo col comando:



```
(tflite) mjrovai@raspi-zero:~ $ cd Documents/TFLITE/IMG_CLASS/
(tflite) mjrovai@raspi-zero:~/Documents/TFLITE/IMG_CLASS $ python3 setup_test.py
NumPy: 1.23.2
Pillow: 10.4.0
TFLite Interpreter created successfully!
(tflite) mjrovai@raspi-zero:~/Documents/TFLITE/IMG_CLASS $
```

Oppure lo si può lanciare direttamente sul [Notebook](#):



```
[1]: import tensorflow.lite_runtime.interpreter as tflite
import numpy as np
from PIL import Image

[4]: print("NumPy:", np.__version__)
print("Pillow:", Image.__version__)

NumPy: 1.23.2
Pillow: 10.4.0

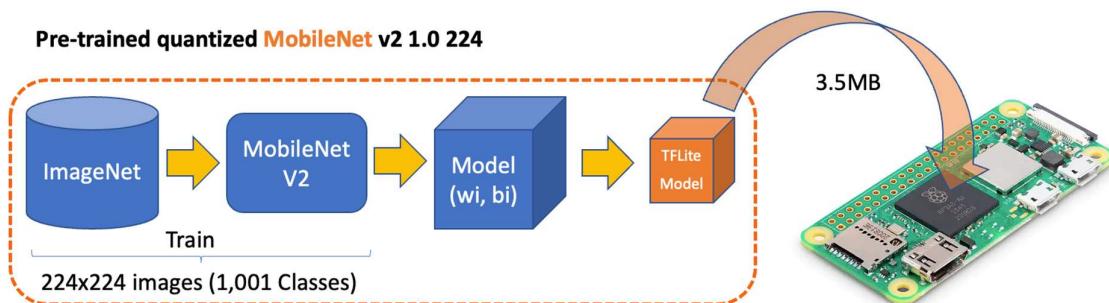
[5]: model_path = "./models/mobilenet_v2_1.0_224_quant.tflite"

[6]: # Try to create a TFLite Interpreter
interpreter = tflite.Interpreter(model_path=model_path)
interpreter.allocate_tensors()
print("TFLite Interpreter created successfully!")

TFLite Interpreter created successfully!
```

Fare inferenze con Mobilenet V2

Nell'ultima sezione, abbiamo impostato l'ambiente, incluso il download di un modello pre-addestrato popolare, Mobilenet V2, addestrato sulle immagini 224x224 di ImageNet (1,2 milioni) per 1.001 classi (1.000 categorie di oggetti più 1 sfondo). Il modello è stato convertito in un formato TensorFlow Lite compatto da 3,5 MB, rendendolo adatto allo spazio di archiviazione e alla memoria limitati di un Raspberry Pi.



Apriamo un nuovo [notebook](#) per seguire tutti i passaggi per classificare un'immagine:
Importare le librerie necessarie:

```
import time
import numpy as np
import matplotlib.pyplot as plt
```

```
from PIL import Image
import tensorflow as tf
```

Caricare il modello TFLite e allocare i tensori:

```
model_path = "./models/mobilenet_v2_1.0_224_quant.tflite"
interpreter = tf.Interpreter(model_path=model_path)
interpreter.allocate_tensors()
```

Ottenere i tensori di input e output.

```
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

Input details ci daranno informazioni su come il modello dovrebbe essere alimentato con un'immagine. Il profilo di (1, 224, 224, 3) ci informa che un'immagine con dimensioni (224x224x3) dovrebbe essere inserita una alla volta (Batch Dimension: 1).

input_details

```
[{'name': 'input',
 'index': 171,
 'shape': array([ 1, 224, 224, 3], dtype=int32), ← Input Image Shape
 'shape_signature': array([ 1, 224, 224, 3], dtype=int32),
 'dtype': numpy.uint8,
 'quantization': (0.0078125, 128),
 'quantization_parameters': {'scales': array([0.0078125], dtype=float32),
 'zero_points': array([128], dtype=int32),
 'quantized_dimension': 0},
 'sparsity_parameters': {}}]
```

Gli **output details** mostrano che l'inferenza risulterà in un array di 1.001 valori interi. Tali valori derivano dalla classificazione dell'immagine, dove ogni valore è la probabilità che quella specifica etichetta sia correlata all'immagine.

output_details

```
[{'name': 'output',
 'index': 172,
 'shape': array([ 1, 1001], dtype=int32), ← Output model
 'shape_signature': array([ 1, 1001], dtype=int32),
 'dtype': numpy.uint8,
 'quantization': (0.09889253973960876, 58),
 'quantization_parameters': {'scales': array([0.09889254], dtype=float32),
 'zero_points': array([58], dtype=int32),
 'quantized_dimension': 0},
 'sparsity_parameters': {}}]
```

Esaminiamo anche il **dtype** dei dettagli di input del modello

```
input_dtype = input_details[0]['dtype']
input_dtype
dtype('uint8')
```

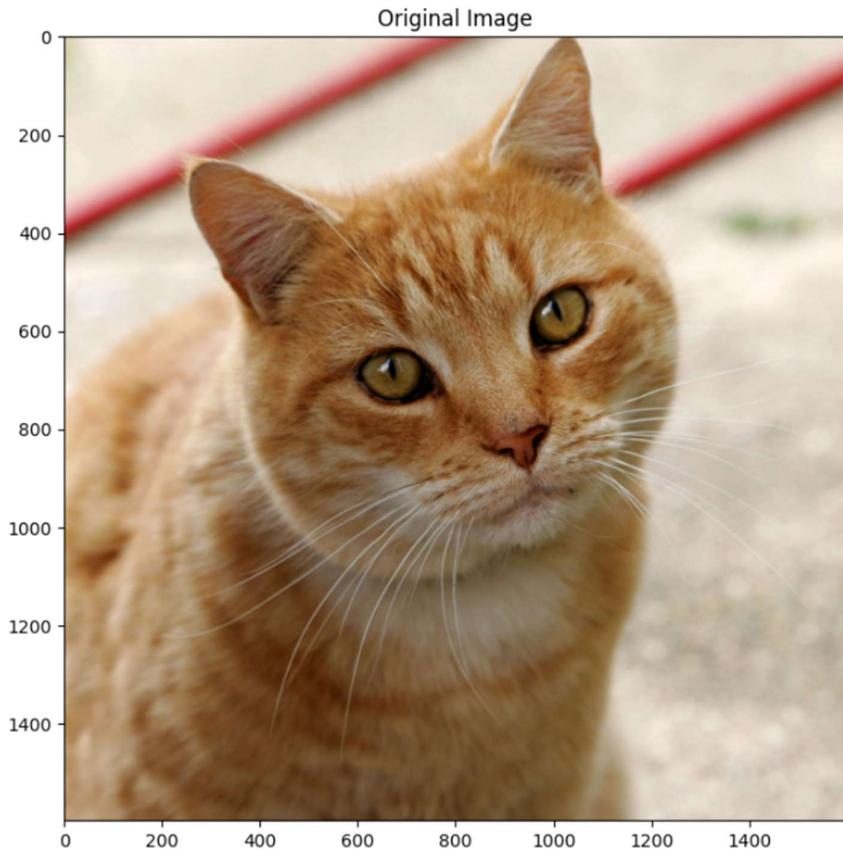
Questo mostra che l'immagine di input dovrebbe essere composta da pixel grezzi (0 - 255). Prendiamo un'immagine di prova. La si può trasferire dal computer o scaricarne una per testarla. Per prima cosa creiamo una cartella nella nostra directory di lavoro:

```
mkdir images
cd images
wget https://upload.wikimedia.org/wikipedia/commons/3/3a/Cat03.jpg
```

Carichiamo e visualizziamo l'immagine:

```
# Load the image
img_path = "./images/Cat03.jpg"
img = Image.open(img_path)

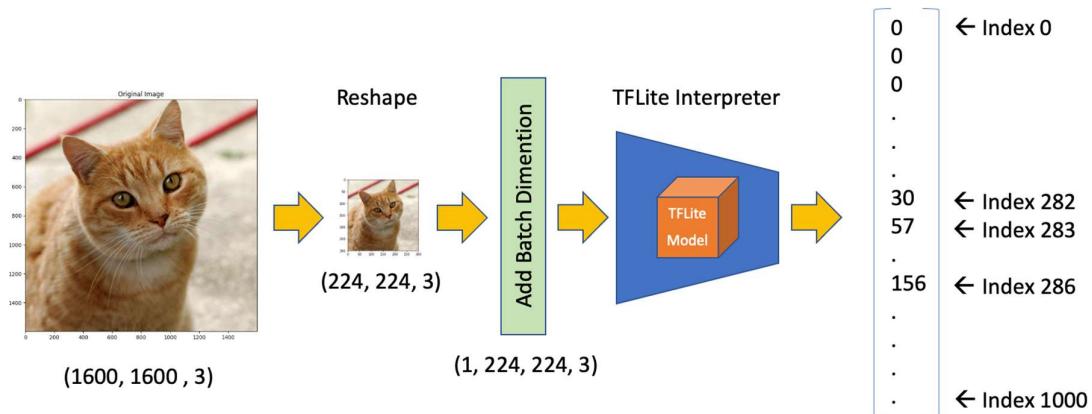
# Display the image
plt.figure(figsize=(8, 8))
plt.imshow(img)
plt.title("Original Image")
plt.show()
```



Possiamo vedere la dimensione dell'immagine eseguendo il comando:

```
width, height = img.size
```

Questo ci mostra che l'immagine è RGB con una larghezza di 1600 e un'altezza di 1600 pixel. Quindi, per usare il nostro modello, dovremmo rimodellarlo in (224, 224, 3) e aggiungere una dimensione batch di 1, come definito nei dettagli di input: (1, 224, 224, 3). Il risultato dell'inferenza, come mostrato nei dettagli di output, sarà un array con una dimensione di 1001, come mostrato di seguito:



Quindi, rimodelliamo l'immagine, aggiungiamo la dimensione batch e vediamo il risultato:

```
img = img.resize((input_details[0]['shape'][1], input_details[0]['shape'][2]))
input_data = np.expand_dims(img, axis=0)
input_data.shape
```

La forma di `input_data` è come previsto: (1, 224, 224, 3)

Confermiamo il `dtype` dei dati di input:

```
input_data.dtype
dtype('uint8')
```

Il `dtype` dei dati di input è 'uint8', che è compatibile con il `dtype` previsto per il modello.

Utilizzando `input_data`, eseguiamo l'interprete e otteniamo le previsioni (output):

```
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()
predictions = interpreter.get_tensor(output_details[0]['index'])[0]
```

La previsione è un array con 1001 elementi. Otteniamo i primi 5 indici in cui i loro elementi hanno valori elevati:

```
top_k_results = 5
top_k_indices = np.argsort(predictions)[::-1][:top_k_results]
top_k_indices
```

`top_k_indices` è un array con 5 elementi: `array([283, 286, 282])`

Quindi, 283, 286, 282, 288 e 479 sono le classi più probabili dell'immagine. Avendo l'indice, dobbiamo trovare a quale classe è assegnato (ad esempio auto, gatto o cane). Il file di testo scaricato con il modello ha un'etichetta associata a ciascun indice da 0 a 1.000. Usiamo una funzione per caricare il file .txt come un elenco:

```
def load_labels(filename):
    with open(filename, 'r') as f:
        return [line.strip() for line in f.readlines()]
```

E otteniamo l'elenco, stampando le etichette associate agli indici:

```
labels_path = "./models/labels.txt"
labels = load_labels(labels_path)

print(labels[286])
print(labels[283])
print(labels[282])
print(labels[288])
print(labels[479])
```

Di conseguenza abbiamo:

```
Egyptian cat
tiger cat
tabby
lynx
carton
```

Almeno i quattro indici principali sono correlati ai felini. Il contenuto di `prediction` è la probabilità associata a ciascuna delle etichette. Come abbiamo visto nei dettagli dell'output, quei valori sono quantizzati e dovrebbero essere dequantizzati e applicare la softmax.

```
scale, zero_point = output_details[0]['quantization']
dequantized_output = (predictions.astype(np.float32) - zero_point) * scale
exp_output = np.exp(dequantized_output - np.max(dequantized_output))
probabilities = exp_output / np.sum(exp_output)
```

Stampiamo le prime 5 probabilità:

```
print(probabilities[286])
print(probabilities[283])
print(probabilities[282])
print(probabilities[288])
print(probabilities[479])
0.27741462
0.3732285
0.16919471
0.10319158
0.023410844
```

Per chiarezza, creiamo una funzione per mettere in relazione le etichette con le probabilità:

```
for i in range(top_k_results):
    print("\t{:20}: {}%".format(
        labels[top_k_indices[i]],
        (int(probabilities[top_k_indices[i]]*100))))
```

tiger cat	: 37%
Egyptian cat	: 27%
tabby	: 16%
lynx	: 10%
carton	: 2%

Definire una funzione generale di Classificazione delle Immagini

Creiamo una funzione generale per dare un'immagine come input e otteniamo le prime 5 classi possibili:

```

def image_classification(img_path, model_path, labels, top_k_results=5):
    # load the image
    img = Image.open(img_path)
    plt.figure(figsize=(4, 4))
    plt.imshow(img)
    plt.axis('off')

    # Load the TFLite model
    interpreter = tflite.Interpreter(model_path=model_path)
    interpreter.allocate_tensors()

    # Get input and output tensors
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()

    # Preprocess
    img = img.resize((input_details[0]['shape'][1],
                      input_details[0]['shape'][2]))
    input_data = np.expand_dims(img, axis=0)

    # Inference on Raspi-Zero
    interpreter.set_tensor(input_details[0]['index'], input_data)
    interpreter.invoke()

    # Obtain results and map them to the classes
    predictions = interpreter.get_tensor(output_details[0]['index'])[0]

    # Get indices of the top k results
    top_k_indices = np.argsort(predictions)[-1:top_k_results]

    # Get quantization parameters
    scale, zero_point = output_details[0]['quantization']

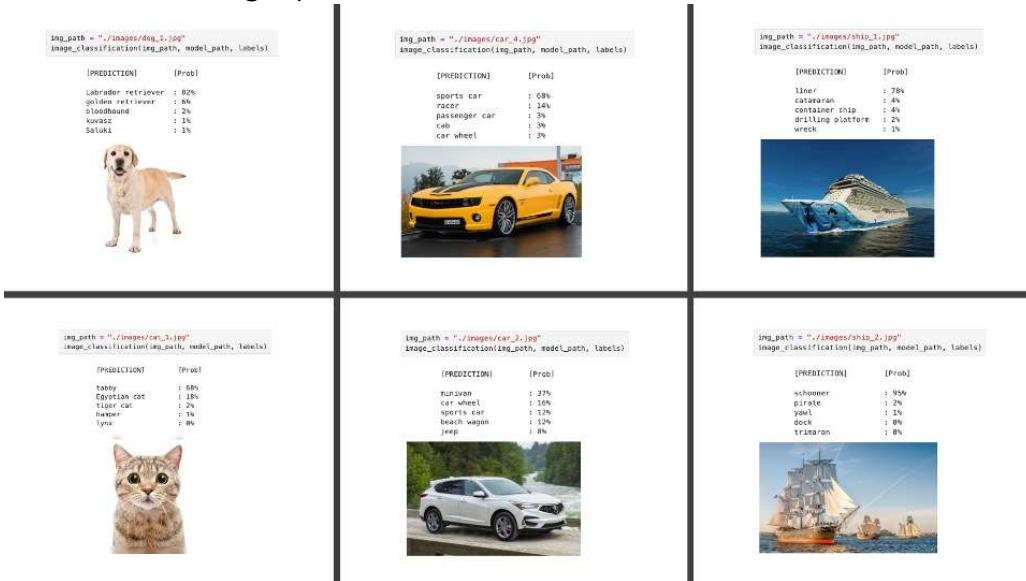
    # Dequantize the output and apply softmax
    dequantized_output = (predictions.astype(np.float32) - zero_point) * scale
    exp_output = np.exp(dequantized_output - np.max(dequantized_output))
    probabilities = exp_output / np.sum(exp_output)

    print("\n\t[PREDICTION] [Prob]\n")
    for i in range(top_k_results):
        print("\t{:20}: {:.%}.".format(
            labels[top_k_indices[i]],
            (int(probabilities[top_k_indices[i]]*100)))))

    print(img_path)
    image_classification(img_path, model_path, labels)

```

E caricando alcune immagini per i test, abbiamo:



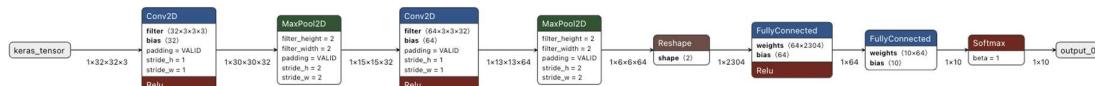
Test con un modello addestrato da zero

Addestriamo un modello TFLite da zero. Per questo, si può seguire il Notebook: [CNN to classify Cifar-10 dataset](#)

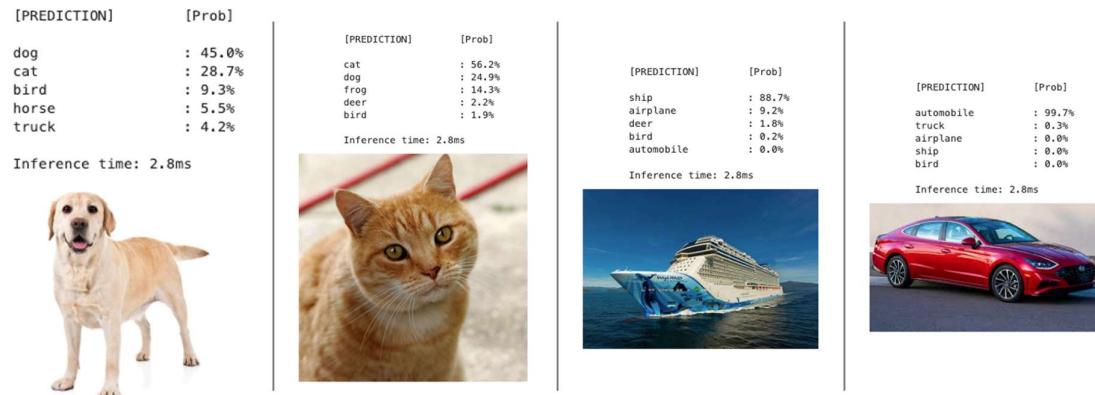
Nel notebook, abbiamo addestrato un modello utilizzando il dataset CIFAR10, che contiene 60.000 immagini da 10 classi di CIFAR (*airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck*). CIFAR ha immagini a colori 32x32 (3 canali colore) in cui gli oggetti non sono centrati e possono avere l'oggetto con uno sfondo, come gli aerei che potrebbero avere un cielo nuvoloso dietro di loro! In breve, immagini piccole ma reali.

Il modello addestrato dalla CNN (*cifar10_model.keras*) aveva una dimensione di 2,0 MB.

Utilizzando *TFLite Converter*, il modello *cifar10.tflite* è diventato di 674 MB (circa 1/3 della dimensione originale).



Sul notebook [Cifar 10 - Image Classification on a Raspi with TFLite](#) (che può essere eseguito sul Raspi), possiamo seguire gli stessi passaggi che abbiamo fatto con *mobilenet_v2_1.0_224_quant.tflite*. Di seguito sono riportati esempi di immagini che utilizzano la *General Function for Image Classification* su un Raspi-Zero, come mostrato nell'ultima sezione.



Installing Picamera2

[Picamera2](#), una libreria Python per interagire con la fotocamera del Raspberry Pi, è basata sullo stack della fotocamera *libcamera* e la Raspberry Pi Foundation la mantiene. La libreria Picamera2 è supportata su tutti i modelli Raspberry Pi, dal Pi Zero al RPi 5. È già installata in tutto il sistema Raspi, ma dovremmo renderla accessibile all'interno dell'ambiente virtuale.

- Per prima cosa, attivare l'ambiente virtuale se non è già attivato:

```
source ~/tfelite/bin/activate
```

- Ora, creiamo un file .pth nell'ambiente virtuale per aggiungere il percorso del sistema del pacchetto sul sito:

```
echo "/usr/lib/python3/dist-packages" > $VIRTUAL_ENV/lib/python3.11/
site-packages/system_site_packages.pth
```

Nota: Se la versione di Python è diversa, sostituire `python3.11` con la versione appropriata.

- Dopo aver creato questo file, provare a importare picamera2 in Python:

```
python3
>>> import picamera2
>>> print(picamera2.__file__)
```

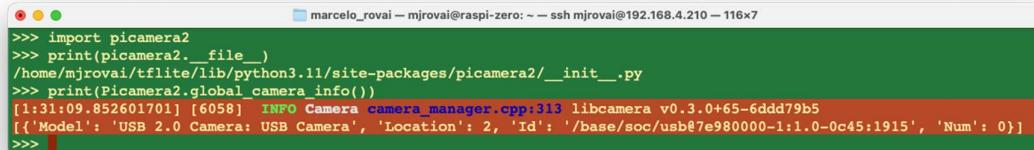
Il codice sopra mostrerà la posizione del file del modulo `picamera2` stesso, dimostrando che la libreria è accessibile dall'ambiente.

```
/home/mjrovai/tfelite/lib/python3.11/site-packages/picamera2/_init_.py
```

È anche possibile elencare le telecamere disponibili nel sistema:

```
>>> print(Picamera2.global_camera_info())
```

Nel nostro caso, con una USB installata, si è ottenuto:



```
>>> import picamera2
>>> print(picamera2._file__)
/home/mjrovai/tflite/lib/python3.11/site-packages/picamera2/_init__.py
>>> print(picamera2.global_camera_info())
[1:31:09.852601701] [6058] INFO Camera camera_manager.cpp:313 libcamera v0.3.0+65-6dd79b5
[{'Model': 'USB 2.0 Camera: USB Camera', 'Location': 2, 'Id': '/base/soc/usb@7e980000-1:1.0-0c45:1915', 'Num': 0}]
>>>
```

Ora che abbiamo confermato che picamera2 funziona nell'ambiente con un indice 0, proviamo un semplice script Python per catturare un'immagine dalla fotocamera USB:

```
from picamera2 import Picamera2
import time

# Initialize the camera
picam2 = Picamera2() # default is index 0

# Configure the camera
config = picam2.create_still_configuration(main={"size": (640, 480)})
picam2.configure(config)

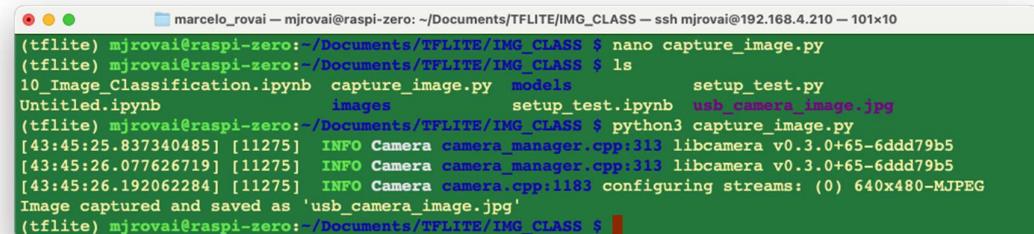
# Start the camera
picam2.start()

# Wait for the camera to warm up
time.sleep(2)

# Capture an image
picam2.capture_file("usb_camera_image.jpg")
print("Image captured and saved as 'usb_camera_image.jpg'")

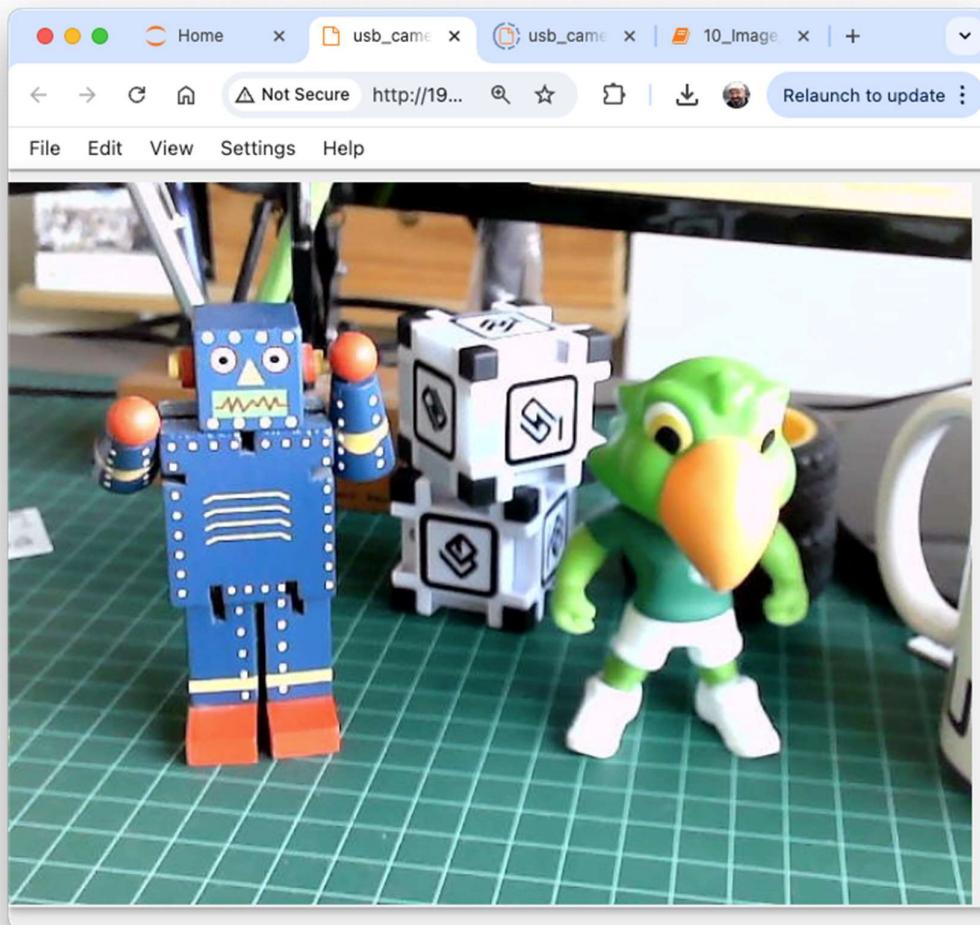
# Stop the camera
picam2.stop()
```

Utilizzare l'editor di testo Nano, Jupyter Notebook o qualsiasi altro editor. Salvarlo come script Python (ad esempio, `capture_image.py`) ed eseguilo. Questo dovrebbe catturare un'immagine dalla fotocamera e salvarla come “`usb_camera_image.jpg`” nella stessa directory dello script.



```
(tflite) mjrovai@raspi-zero:~/Documents/TFLITE/IMG_CLASS $ nano capture_image.py
(tflite) mjrovai@raspi-zero:~/Documents/TFLITE/IMG_CLASS $ ls
10_Image_Classification.ipynb capture_image.py models setup_test.py
Untitled.ipynb images setup_test.ipynb usb_camera_image.jpg
(tflite) mjrovai@raspi-zero:~/Documents/TFLITE/IMG_CLASS $ python3 capture_image.py
[43:45:25.837340485] [11275] INFO Camera camera_manager.cpp:313 libcamera v0.3.0+65-6dd79b5
[43:45:26.077626719] [11275] INFO Camera camera_manager.cpp:313 libcamera v0.3.0+65-6dd79b5
[43:45:26.192062284] [11275] INFO Camera camera.cpp:1183 configuring streams: (0) 640x480-MJPEG
Image captured and saved as 'usb_camera_image.jpg'
(tflite) mjrovai@raspi-zero:~/Documents/TFLITE/IMG_CLASS $
```

Se Jupyter è aperto, si può vedere l'immagine catturata sul computer. Altrimenti, si trasferisce il file dal Raspi al computer.



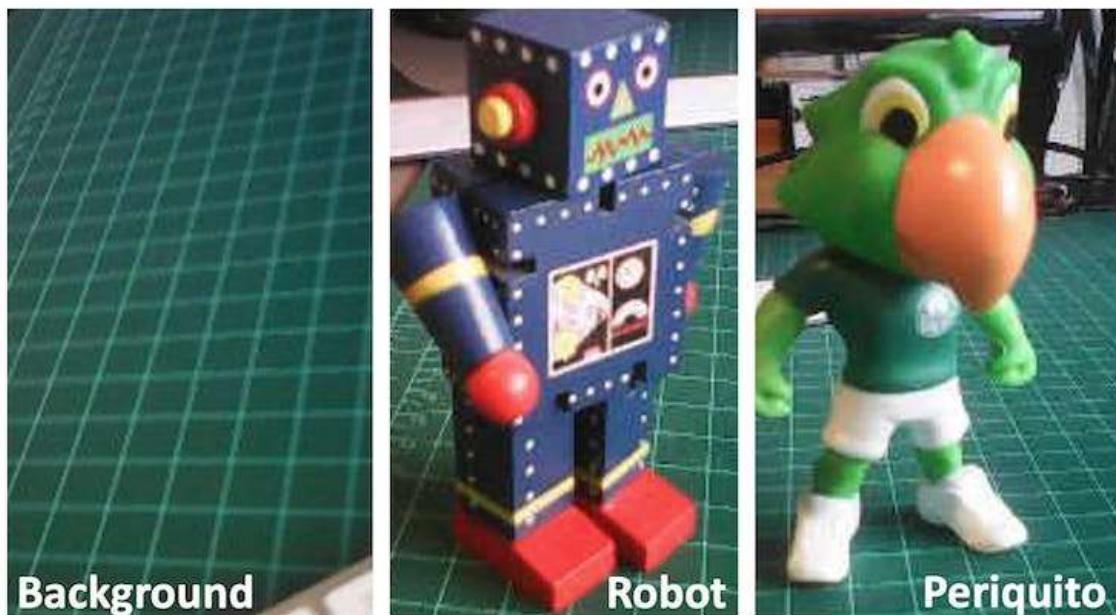
Se si sta lavorando con un Raspi-5 con un intero desktop, si può aprire il file direttamente sul dispositivo.

Progetto di Classificazione delle Immagini

Ora, svilupperemo un progetto completo di Classificazione delle Immagini utilizzando Edge Impulse Studio. Come abbiamo fatto con Movilinet V2, il modello TFLite addestrato e convertito verrà utilizzato per l'inferenza.

L'Obiettivo

Il primo passo in qualsiasi progetto ML è definire il suo obiettivo. In questo caso, è rilevare e classificare due oggetti specifici presenti in un'immagine. Per questo progetto, utilizzeremo due piccoli giocattoli: un robot e un piccolo pappagallo brasiliano (chiamato Periquito). Raccoglieremo anche immagini di un *background* in cui questi due oggetti sono assenti.



Raccolta Dati

Una volta definito l'obiettivo del nostro progetto di apprendimento automatico, il passaggio successivo, e più cruciale, è la raccolta del dataset. Possiamo utilizzare un telefono per l'acquisizione delle immagini, ma qui utilizzeremo il Raspi. Impostiamo un semplice server Web sul nostro Raspberry Pi per visualizzare le immagini QVGA (320 x 240) acquisite in un browser.

1. Per prima cosa, installiamo Flask, un framework Web leggero per Python:

```
pip3 install flask
```

2. Creiamo un nuovo script Python che combina l'acquisizione delle immagini con un server Web. Lo chiameremo `get_img_data.py`:

```
from flask import Flask, Response, render_template_string, request, redirect, url_for
from picamera2 import Picamera2
import io
import threading
import time
import os
import signal

app = Flask(__name__)

# Global variables
base_dir = "dataset"
picam2 = None
frame = None
frame_lock = threading.Lock()
capture_counts = {}
current_label = None
shutdown_event = threading.Event()

def initialize_camera():
    global picam2
    picam2 = Picamera2()
    config = picam2.create_preview_configuration(main={"size": (320, 240)})
    picam2.configure(config)
    picam2.start()
    time.sleep(2) # Wait for camera to warm up

def get_frame():
    global frame
    while not shutdown_event.is_set():
        stream = io.BytesIO()
        picam2.capture_file(stream, format='jpeg')
        with frame_lock:
            frame = stream.getvalue()
```

```

        time.sleep(0.1) # Adjust as needed for smooth preview

def generate_frames():
    while not shutdown_event.is_set():
        with frame_lock:
            if frame is not None:
                yield (b'--frame\r\n'
                       b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
        time.sleep(0.1) # Adjust as needed for smooth streaming

def shutdown_server():
    shutdown_event.set()
    if picam2:
        picam2.stop()
    # Give some time for other threads to finish
    time.sleep(2)
    # Send SIGINT to the main process
    os.kill(os.getpid(), signal.SIGINT)

@app.route('/', methods=['GET', 'POST'])
def index():
    global current_label
    if request.method == 'POST':
        current_label = request.form['label']
        if current_label not in capture_counts:
            capture_counts[current_label] = 0
        os.makedirs(os.path.join(base_dir, current_label), exist_ok=True)
    return redirect(url_for('capture_page'))
return render_template_string('''
<!DOCTYPE html>
<html>
<head>
    <title>Dataset Capture - Label Entry</title>
</head>
<body>
    <h1>Enter Label for Dataset</h1>
    <form method="post">
        <input type="text" name="label" required>
        <input type="submit" value="Start Capture">
    </form>
</body>
</html>
''')

@app.route('/capture')
def capture_page():
    return render_template_string('''
<!DOCTYPE html>
<html>
<head>
    <title>Dataset Capture</title>
    <script>
        var shutdownInitiated = false;
        function checkShutdown() {
            if (!shutdownInitiated) {
                fetch('/check_shutdown')
                    .then(response => response.json())
                    .then(data => {
                        if (data.shutdown) {
                            shutdownInitiated = true;
                            document.getElementById('video-feed').src = '';
                            document.getElementById('shutdown-message')
                                .style.display = 'block';
                        }
                    })
            }
        }
        setInterval(checkShutdown, 1000); // Check every second
    </script>
</head>
<body>
'''
```

```
<h1>Dataset Capture</h1>
<p>Current Label: {{ label }}</p>
<p>Images captured for this label: {{ capture_count }}</p>

<div id="shutdown-message" style="display: none; color: red;">
    Capture process has been stopped. You can close this window.
</div>
<form action="/capture_image" method="post">
    <input type="submit" value="Capture Image">
</form>
<form action="/stop" method="post">
    <input type="submit" value="Stop Capture"
    style="background-color: #ff6666;">
</form>
<form action="/" method="get">
    <input type="submit" value="Change Label"
    style="background-color: #ffff66;">
</form>
</body>
</html>
'', label=current_label, capture_count=capture_counts.get(current_label, 0))

@app.route('/video_feed')
def video_feed():
    return Response(generate_frames(),
                    mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/capture_image', methods=['POST'])
def capture_image():
    global capture_counts
    if current_label and not shutdown_event.is_set():
        capture_counts[current_label] += 1
        timestamp = time.strftime("%Y%m%d-%H%M%S")
        filename = f'image_{timestamp}.jpg'
        full_path = os.path.join(base_dir, current_label, filename)

        picam2.capture_file(full_path)

    return redirect(url_for('capture_page'))

@app.route('/stop', methods=['POST'])
def stop():
    summary = render_template_string('''
        <!DOCTYPE html>
        <html>
        <head>
            <title>Dataset Capture - Stopped</title>
        </head>
        <body>
            <h1>Dataset Capture Stopped</h1>
            <p>The capture process has been stopped. You can close this window.</p>
            <p>Summary of captures:</p>
            <ul>
                {% for label, count in capture_counts.items() %}
                    <li>{{ label }}: {{ count }} images</li>
                {% endfor %}
            </ul>
        </body>
        </html>
    ''', capture_counts=capture_counts)

    # Start a new thread to shutdown the server
    threading.Thread(target=shutdown_server).start()

    return summary

@app.route('/check_shutdown')
def check_shutdown():
    return {'shutdown': shutdown_event.is_set()}

if __name__ == '__main__':
    initialize_camera()
```

```
threading.Thread(target=get_frame, daemon=True).start()
app.run(host='0.0.0.0', port=5000, threaded=True)
```

3. Eseguire questo script:

```
```bash
python3 get_img_data.py
```

```

4. Accedere all'interfaccia web:

- Sul Raspberry Pi stesso (se si ha una GUI): Aprire un browser web e andare su `http://localhost:5000`

- Da un altro dispositivo sulla stessa rete: Aprire un browser web e andare su `http://<raspberry_pi_ip>:5000` (Sostituire `<raspberry_pi_ip>` con l'indirizzo IP del Raspberry Pi). Per esempio: `http://192.168.4.210:5000/`

Questo script Python crea un'interfaccia basata sul web per catturare e organizzare set di dati di immagini usando un Raspberry Pi e la sua fotocamera. È utile per progetti di apprendimento automatico che richiedono dati di immagini etichettati.

Caratteristiche Principali:

- **Interfaccia Web**: Accessibile da qualsiasi dispositivo sulla stessa rete del Raspberry Pi.
- **Anteprima Telecamera in Tempo Reale**: Mostra un feed in tempo reale dalla telecamera.
- **Sistema di Etichettatura**: Consente agli utenti di immettere etichette per diverse categorie di immagini.
- **Archiviazione Organizzata**: Salva automaticamente le immagini in sottodirectory specifiche per etichetta.
- **Contatori per Etichetta**: Tiene traccia di quante immagini vengono acquisite per ogni etichetta.
- **Statistiche di Riepilogo**: Fornisce un riepilogo delle immagini acquisite quando si interrompe il processo di acquisizione.

Componenti Principali:

- **Applicazione Web Flask**: Gestisce il routing e serve l'interfaccia Web.
- **Integrazione di Picamera2**: Controlla la telecamera Raspberry Pi.
- **Threaded Frame Capture**: Assicura un'anteprima live fluida.
- **File Management**: Organizza le immagini catturate in directory etichettate.

Funzioni Chiave:

- `initialize_camera()`: Imposta l'istanza Picamera2.
- `get_frame()`: Cattura continuamente i frame per l'anteprima live.
- `generate_frames()`: Genera i frame per il feed video live.
- `shutdown_server()`: Imposta l'evento di arresto, arresta la telecamera e arresta il server Flask.
- `index()`: Gestisce la pagina di input dell'etichetta.
- `capture_page()`: Visualizza l'interfaccia di acquisizione principale.
- `video_feed()`: Mostra un'anteprima live per posizionare la telecamera.
- `capture_image()`: Salva un'immagine con l'etichetta corrente.
- `stop()`: Arresta il processo di acquisizione e visualizza un riepilogo.

Flusso di Utilizzo:

- Avviare lo script sul Raspberry Pi.
- Accedere all'interfaccia web da un browser.
- Inserire un'etichetta per le immagini da catturare e premere 'Start Capture'.

```
![] (images/png/enter_label.png)
```

- Utilizzare l'anteprima live per posizionare la telecamera.
- Cliccare 'Capture Image' per salvare le immagini sotto l'etichetta corrente.

```
![] (images/png/capture.png)
```

- Cambiare le etichette come necessario per le diverse categorie, selezionando 'Change Label'.
- Cliccare 'Stop Capture' al termine per vedere un riepilogo.

```
![] (images/png/stop.png)
```

Note Tecniche:

- Lo script usa il threading per gestire la cattura di frame e il web serving simultanei.

- Le immagini vengono salvate con timestamp nei nomi dei file per renderle uniche.
- L'interfaccia web è reattiva e accessibile da dispositivi mobili.

Possibilità di Personalizzazione:

- Regolare la risoluzione dell'immagine nella funzione `initialize_camera()`. Qui abbiamo utilizzato QVGA (320X240).
- Modificare i modelli HTML per un aspetto diverso.
- Aggiungere ulteriori passaggi di elaborazione o analisi delle immagini nella funzione `capture_image()`.

Numero di campioni sul Dataset:

Si ottengono circa 60 immagini da ciascuna categoria (`periquito`, `robot` e `background`). Provare ad acquisire con diverse angolazioni, sfondi e condizioni di luce. Sul Raspi, finiremo con una cartella denominata `dataset`, che contiene 3 sottocartelle *periquito*, *robot* e *background*. una per ogni classe di immagini.

Si può usare `Filezilla` per trasferire il dataset creato sul computer principale.

Addestramento del modello con Edge Impulse

Useremo Edge Impulse Studio per addestrare il modello. Si va nella [Pagina di Edge Impulse] (<https://edgeimpulse.com/>), si inseriscono le credenziali e si crea un nuovo progetto:

![] (images/png/new-proj-ei.png)

> Qui si può clonare un progetto simile: [Raspi - Img Class] (<https://studio.edgeimpulse.com/public/510251/live>).

Dataset

Esamineremo quattro passaggi principali usando EI Studio (o Studio). Questi passaggi sono fondamentali per preparare il nostro modello per l'uso sul Raspi: Dataset, Impulse, Test e Deploy (sul dispositivo Edge, in questo caso, il Raspi).

> Per quanto riguarda il Dataset, è essenziale sottolineare che il nostro Dataset originale, acquisito con il Raspi, sarà suddiviso in *Training*, *Validation* e *Test*. Il Test Set sarà separato dall'inizio e riservato per l'uso solo nella fase di Test dopo l'addestramento. Il Validation Set sarà utilizzato durante l'addestramento.

Su Studio, seguire i passaggi per caricare i dati acquisiti:

1. Si va nella scheda `Data acquisition` e nella sezione `UPLOAD DATA`, si caricano i file dal computer nelle categorie scelte.
2. Lasciare a Studio la suddivisione del dataset originale in *train and test* e scegliere l'etichetta a riguardo
3. Ripetere la procedura per tutte e tre le classi. Alla fine, si vedranno i "raw data" in Studio:

![] (images/png/data-Aquisition.png)

Studio consente di esplorare i dati, mostrando una vista completa di tutti i quelli nel progetto. Si possono cancellare, ispezionare o modificare le etichette cliccando sui singoli elementi di dati. Nel nostro caso, un progetto semplice, i dati sembrano OK.

![] (images/png/data-esplorer.png)

Il Progetto Impulse

In questa fase, dovremmo definire come:

- Pre-elaborare i nostri dati, il che consiste nel ridimensionare le singole immagini e determinare la `color depth` [profondità di colore] da utilizzare (sia RGB che in scala di grigi) e
- Specificare un Modello. In questo caso, sarà `Transfer Learning (Images)` a mettere a punto un modello di classificazione delle immagini MobileNet V2 pre-addestrato sui nostri dati. Questo metodo funziona bene anche con set di dati di immagini relativamente piccoli (circa 180 immagini nel nostro caso).

Transfer Learning con MobileNet offre un approccio semplificato all'addestramento del modello, che è particolarmente utile per ambienti con risorse limitate e progetti con dati e etichettati limitati. MobileNet, noto per la sua architettura leggera, è un modello pre-ad-

destrato che ha già appreso funzionalità preziose da un ampio set di dati (ImageNet).

```
![] (images/jpeg/model_1.jpg)
```

Sfruttando queste funzionalità apprese, possiamo addestrare un nuovo modello per il compito specifico con meno dati e risorse computazionali e raggiungere un'accuratezza competitiva.

```
![] (images/jpeg/model_2.jpg)
```

Questo approccio riduce significativamente i tempi di addestramento e i costi computazionali, rendendolo ideale per la prototipazione rapida e l'implementazione su dispositivi embedded in cui l'efficienza è fondamentale.

Si va alla scheda Impulse Design e si crea l'*impulse*, definendo una dimensione dell'immagine di 160x160 e schiacciandola (forma quadrata, senza ritaglio). Si seleziona Image e i blocchi Transfer Learning. Si salva l'Impulse.

```
![] (images/png/impulse.png)
```

Pre-elaborazione delle immagini

Tutte le immagini QVGA/RGB565 in ingresso verranno convertite in 76.800 feature (160x160x3).

```
![] (images/png/preproc.png)
```

Premere `Save parameters` e selezionare `Generate features` nella scheda successiva.

Progettazione del modello

MobileNet è una famiglia di reti neurali convoluzionali efficienti progettate per applicazioni di visione mobile e embedded. Le caratteristiche principali di MobileNet sono:

1. Leggero: Ottimizzato per dispositivi mobili e sistemi embedded con risorse di calcolo limitate.
2. Velocità: Tempi di inferenza rapidi, adatti per applicazioni in tempo reale.
3. Precisione: Mantiene una buona accuratezza nonostante le dimensioni compatte.

[MobileNetV2] (<https://arxiv.org/abs/1801.04381>), introdotto nel 2018, migliora l'architettura MobileNet originale. Le caratteristiche principali includono:

1. Residui Invertiti: Le strutture residue invertite vengono utilizzate quando vengono create connessioni di scelta rapida tra layer di colli di bottiglia sottili.
2. Colli di Bottiglia Lineari: Rimuove le non linearità nei layer stretti per impedire la distruzione delle informazioni.
3. Convoluzioni Separabili in Profondità: Continua a utilizzare questa efficiente operazione da MobileNetV1.

Nel nostro progetto, faremo un `Transfer Learning` con `MobileNetV2 160x160 1.0`, il che significa che le immagini utilizzate per l'addestramento (e l'inferenza futura) dovrebbero avere una *input Size* [dimensione di input] di 160x160 pixel e un *Width Multiplier* [moltiplicatore di larghezza] di 1.0 (larghezza completa, non ridotta). Questa configurazione bilancia tra dimensione del modello, velocità e accuratezza.

Training del Modello

Un'altra preziosa tecnica di apprendimento profondo è il **Data Augmentation**. Il "data augmentation" migliora l'accuratezza dei modelli di apprendimento automatico creando dati artificiali aggiuntivi. Un sistema di data augmentation apporta piccole modifiche casuali ai dati di training durante l'addestramento (ad esempio capovolgendo, ritagliando o ruotando le immagini).

Guardando internamente, qui si può vedere come Edge Impulse implementa una policy di data Augmentation sui dati:

```
``` python
Implements the data augmentation policy
def augment_image(image, label):
 # Flips the image randomly
 image = tf.image.random_flip_left_right(image)

 # Increase the image size, then randomly crop it down to
 # the original dimensions
```

```

resize_factor = random.uniform(1, 1.2)
new_height = math.floor(resize_factor * INPUT_SHAPE[0])
new_width = math.floor(resize_factor * INPUT_SHAPE[1])
image = tf.image.resize_with_crop_or_pad(image, new_height, new_width)
image = tf.image.random_crop(image, size=INPUT_SHAPE)

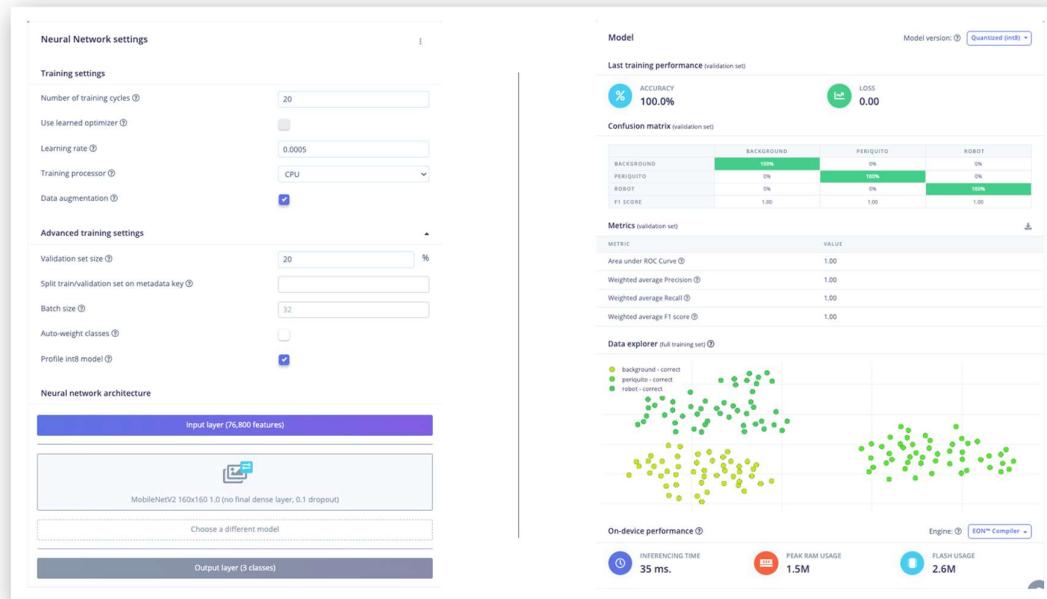
Vary the brightness of the image
image = tf.image.random_brightness(image, max_delta=0.2)

return image, label

```

L'esposizione a queste variazioni durante l'addestramento può aiutare a impedire al modello di prendere scorciatoie "memorizzando" indizi superficiali nei dati di addestramento, il che significa che potrebbe riflettere meglio i pattern profondi in esame nel set di dati.

L'ultimo layer denso del nostro modello avrà 0 neuroni con un dropout del 10% per prevenire il sovraccarico. Ecco il risultato del Training:



Il risultato è eccellente, con una latenza ragionevole di 35 ms (per un Raspi-4), che dovrebbe tradursi in circa 30 fps (fotogrammi al secondo) durante l'inferenza. Un Raspi-Zero dovrebbe essere più lento e il Raspi-5 più veloce.

### **Compromesso: Accuratezza contro Velocità**

Se è necessaria un'inferenza più rapida, dovremmo addestrare il modello usando alfa più piccoli (0.35, 0.5 e 0.75) o persino ridurre le dimensioni dell'immagine in ingresso, a discapito dell'accuratezza. Tuttavia, ridurre le dimensioni dell'immagine in ingresso e diminuire l'alfa (moltiplicatore di larghezza) può accelerare l'inferenza per MobileNet V2, ma hanno compromessi diversi. Confrontiamoli:

1. Riduzione delle Dimensioni dell'Immagine in Ingresso:

Pro:

- Riduce significativamente il costo computazionale su tutti i layer.
- Riduce l'utilizzo della memoria.
- Spesso fornisce un aumento sostanziale della velocità.

Contro:

- Potrebbe ridurre la capacità del modello di rilevare piccole caratteristiche o dettagli fini.
- Può avere un impatto significativo sulla precisione, specialmente per le attività che richiedono un riconoscimento a grana fine.

2. Riduzione di Alpha (Moltiplicatore di Larghezza):

Pro:

- Riduce il numero di parametri e calcoli nel modello.
- Mantiene la risoluzione di input originale, preservando potenzialmente più dettagli.
- Può fornire un buon equilibrio tra velocità e precisione.

Contro:

- Potrebbe non accelerare l'inferenza in modo così drastico come la riduzione delle dimensioni di input.
- Può ridurre la capacità del modello di apprendere caratteristiche complesse.

Confronto:

1. Impatto sulla Velocità:
  - La riduzione delle dimensioni di input spesso fornisce un aumento di velocità più sostanziale perché riduce i calcoli in modo quadratico (dimezzando sia la larghezza che l'altezza si riducono i calcoli di circa il 75%).
  - La riduzione di Alpha fornisce una riduzione più lineare nei calcoli.
2. Impatto sulla Precisione:
  - La riduzione delle dimensioni di input può avere un impatto significativo sulla precisione, specialmente quando si rilevano piccoli oggetti o dettagli fini.
  - La riduzione di alpha tende ad avere un impatto più graduale sulla precisione.
3. Architettura del Modello:
  - La modifica delle dimensioni di input non altera l'architettura del modello.
  - La modifica di alpha modifica la struttura del modello riducendo il numero di canali in ogni layer.

Raccomandazione:

1. Se l'applicazione non richiede il rilevamento di piccoli dettagli e può tollerare una certa perdita di accuratezza, ridurre le dimensioni di input è spesso il modo più efficace per accelerare l'inferenza.
2. Ridurre l'alfa potrebbe essere preferibile se mantenere la capacità di rilevare dettagli fini è fondamentale o se c'è bisogno di un compromesso più equilibrato tra velocità e accuratezza.
3. Per ottenere risultati migliori, si devono sperimentare entrambi:
  - Provare MobileNet V2 con dimensioni di input come 160x160 o 92x92
  - Sperimentare con valori alfa come 1.0, 0.75, 0.5 o 0.35.
4. Eseguire sempre il benchmark delle diverse configurazioni sull'hardware specifico e con il particolare set di dati per trovare l'equilibrio ottimale per il caso d'uso.

Ricordarsi che la scelta migliore dipende dai requisiti specifici di accuratezza, velocità e dalla natura delle immagini con cui si sta lavorando. Spesso vale la pena sperimentare combinazioni per trovare la configurazione ottimale per il particolare caso d'uso.

### **Test del Modello**

Ora, si deve prendere il set di dati all'inizio del progetto ed eseguire il modello addestrato usandolo come input. Di nuovo, il risultato è eccellente (92,22%).

### **Distribuzione del modello**

Come abbiamo fatto nella sezione precedente, possiamo distribuire il modello addestrato come .tflite e usare Raspi per eseguirlo usando Python.

Nella scheda Dashboard, si va su Transfer learning model (int8 quantized) e si clicca sull'icona di download:

Download block output			
TITLE	TYPE	SIZE	
Image training data	NPY file	150 windows	
Image training labels	NPY file	150 windows	
Image testing data	NPY file	36 windows	
Image testing labels	NPY file	36 windows	
Transfer learning model	TensorFlow Lite (float32)	9 MB	
Transfer learning model	TensorFlow Lite (int8 quantized)	3 MB	
Transfer learning model	Model evaluation metrics (JSON file)	5 KB	
Transfer learning model	TensorFlow SavedModel	8 MB	
Transfer learning model	Keras h5 model	8 MB	

Scarichiamo anche la versione float32 per il confronto

Trasferire il modello dal computer al Raspi (./models), ad esempio, usando FileZilla. Catturare, inoltre, alcune immagini per l'inferenza (./images).

Importare le librerie necessarie:

```
import time
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import tensorflow.lite_runtime.interpreter as tflite
```

Definire i path e le etichette:

```
img_path = "./images/robot.jpg"
model_path = "./models/ei-raspi-img-class-int8-quantized-model.tflite"
labels = ['background', 'periquito', 'robot']
```

Notare che i modelli addestrati su Edge Impulse Studio produrranno valori con indice 0, 1, 2, ecc., dove le etichette effettive seguiranno un ordine alfabetico.

Caricare il modello, allocare i tensori e ottenere i dettagli dei tensori di input e output:

```
Load the TFLite model
interpreter = tflite.Interpreter(model_path=model_path)
interpreter.allocate_tensors()

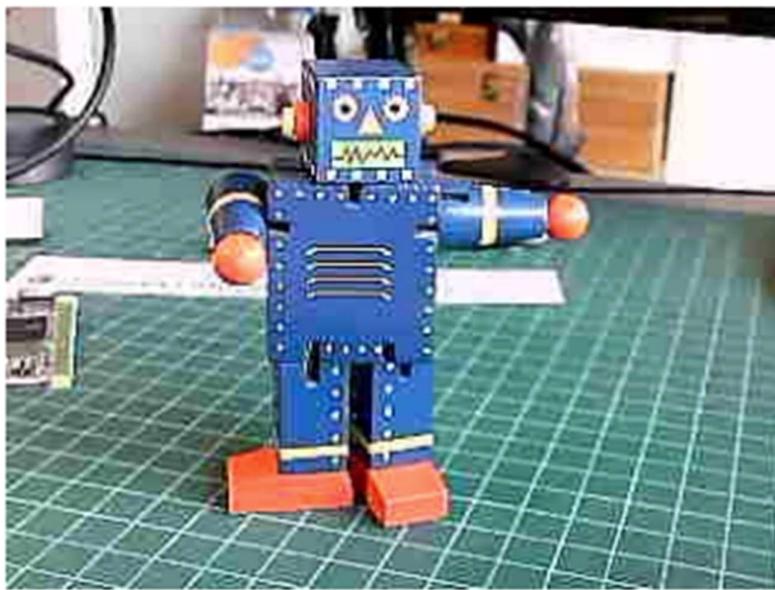
Get input and output tensors
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

Una differenza importante da notare è che il `dtype` dei dettagli di input del modello è ora `int8`, il che significa che i valori di input vanno da -128 a +127, mentre ogni pixel della nostra immagine va da 0 a 255. Ciò significa che dovremmo pre-elaborare l'immagine per farla corrispondere. Possiamo controllare qui:

```
input_dtype = input_details[0]['dtype']
input_dtype
numpy.int8
```

Quindi, apriamo l'immagine e mostriamola:

```
img = Image.open(img_path)
plt.figure(figsize=(4, 4))
plt.imshow(img)
plt.axis('off')
plt.show()
```



Ed eseguiamo la pre-elaborazione:

```
scale, zero_point = input_details[0]['quantization']
img = img.resize((input_details[0]['shape'][1],
 input_details[0]['shape'][2]))
img_array = np.array(img, dtype=np.float32) / 255.0
img_array = (img_array / scale + zero_point).clip(-128, 127).astype(np.int8)
input_data = np.expand_dims(img_array, axis=0)
```

Controllando i dati di input, possiamo verificare che il tensore di input è compatibile con quanto previsto dal modello:

```
input_data.shape, input_data.dtype
((1, 160, 160, 3), dtype('int8'))
```

Adesso è il momento di effettuare l'inferenza. Calcoliamo anche la latenza del modello:

```
Inference on Raspi-Zero
start_time = time.time()
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()
end_time = time.time()
inference_time = (end_time - start_time) * 1000 # Convert to milliseconds
print ("Inference time: {:.1f}ms".format(inference_time))
```

Il modello impiegherà circa 125 ms per eseguire l'inferenza nel Raspi-Zero, che dura 3 o 4 volte più a lungo di un Raspi-5.

Ora possiamo ottenere le etichette di output e le probabilità. È anche importante notare che il modello addestrato su Edge Impulse Studio ha un softmax nel suo output (diverso dal Movilenet V2 originale) e dovremmo usare l'output grezzo del modello come "probabilità".

```
Obtain results and map them to the classes
predictions = interpreter.get_tensor(output_details[0]['index'])[0]

Get indices of the top k results
top_k_results=3
top_k_indices = np.argsort(predictions)[::-1][:top_k_results]

Get quantization parameters
scale, zero_point = output_details[0]['quantization']

Dequantize the output
dequantized_output = (predictions.astype(np.float32) - zero_point) * scale
probabilities = dequantized_output
```

```

print("\n\t[PREDICTION] [Prob]\n")
for i in range(top_k_results):
 print("\t{:20}: {:.2f}%".format(
 labels[top_k_indices[i]],
 probabilities[top_k_indices[i]] * 100))

```

[PREDICTION]	[Prob]
robot	: 99.61%
periquito	: 0.00%
background	: 0.00%

Modifichiamo la funzione creata in precedenza in modo da poter gestire diversi tipi di modelli:

```

def image_classification(img_path, model_path, labels, top_k_results=3,
 apply_softmax=False):
 # Load the image
 img = Image.open(img_path)
 plt.figure(figsize=(4, 4))
 plt.imshow(img)
 plt.axis('off')

 # Load the TFLite model
 interpreter = tflite.Interpreter(model_path=model_path)
 interpreter.allocate_tensors()

 # Get input and output tensors
 input_details = interpreter.get_input_details()
 output_details = interpreter.get_output_details()

 # Preprocess
 img = img.resize((input_details[0]['shape'][1],
 input_details[0]['shape'][2]))

 input_dtype = input_details[0]['dtype']

 if input_dtype == np.uint8:
 input_data = np.expand_dims(np.array(img), axis=0)
 elif input_dtype == np.int8:
 scale, zero_point = input_details[0]['quantization']
 img_array = np.array(img, dtype=np.float32) / 255.0
 img_array = (img_array / scale + zero_point).clip(-128, 127).astype(np.int8)
 input_data = np.expand_dims(img_array, axis=0)
 else: # float32
 input_data = np.expand_dims(np.array(img, dtype=np.float32), axis=0) / 255.0

 # Inference on Raspi-Zero
 start_time = time.time()
 interpreter.set_tensor(input_details[0]['index'], input_data)
 interpreter.invoke()
 end_time = time.time()
 inference_time = (end_time - start_time) * 1000 # Convert to milliseconds

 # Obtain results
 predictions = interpreter.get_tensor(output_details[0]['index'])[0]

 # Get indices of the top k results
 top_k_indices = np.argsort(predictions)[::-1][:top_k_results]

 # Handle output based on type
 output_dtype = output_details[0]['dtype']
 if output_dtype in [np.int8, np.uint8]:
 # Dequantize the output
 scale, zero_point = output_details[0]['quantization']
 predictions = (predictions.astype(np.float32) - zero_point) * scale

 if apply_softmax:
 # Apply softmax

```

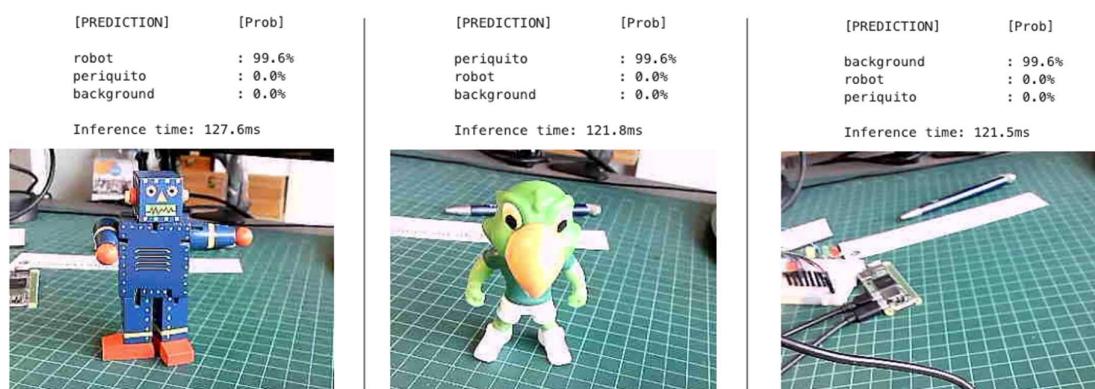
```

 exp_preds = np.exp(predictions - np.max(predictions))
 probabilities = exp_preds / np.sum(exp_preds)
else:
 probabilities = predictions

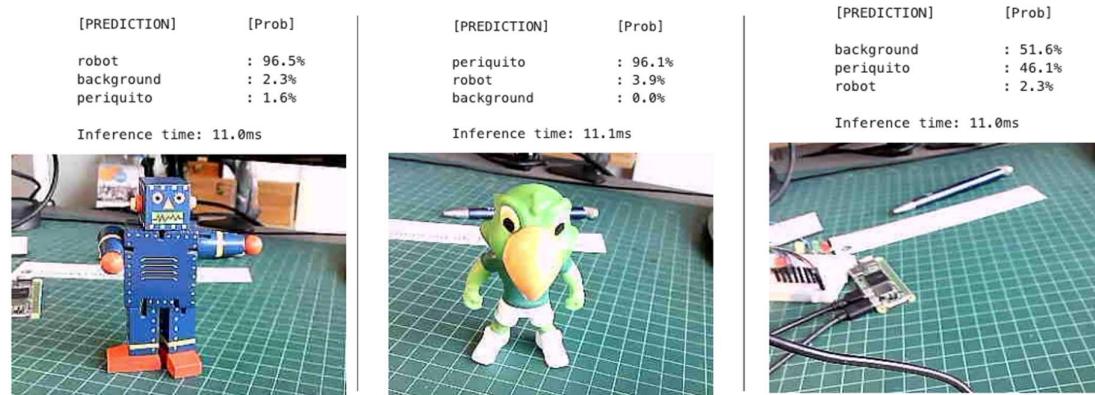
print("\n\t[PREDICTION] [Prob]\n")
for i in range(top_k_results):
 print("\t{:20}: {:.1f}%".format(
 labels[top_k_indices[i]],
 probabilities[top_k_indices[i]] * 100))
print ("Inference time: {:.1f}ms".format(inference_time))

```

E lo si testa con immagini diverse e con il modello quantizzato int8 (**160x160 alpha =1.0**).



Scarichiamo un modello più piccolo, come quello addestrato per il [Nicla Vision Lab](#) (int8 quantized model, 96x96, alpha = 0.1), come test. Possiamo usare la stessa funzione:



Il modello ha perso un po' di accuratezza, ma è ancora OK dato che non cerca molti dettagli.

Per quanto riguarda la latenza, siamo circa **dieci volte più veloci** su Raspi-Zero.

## Classificazione delle Immagini in Tempo Reale

Sviluppiamo un'app per catturare immagini con la fotocamera USB in tempo reale, mostrandone la classificazione.

Utilizzando nano sul terminale, salvare il codice sottostante, come `img_class_live_infer.py`.

```

from flask import Flask, Response, render_template_string, request, jsonify
from picamera2 import Picamera2
import io
import threading
import time
import numpy as np
from PIL import Image
import tflite_runtime.interpreter as tflite
from queue import Queue

app = Flask(__name__)

Global variables
picam2 = None

```

```
frame = None
frame_lock = threading.Lock()
is_classifying = False
confidence_threshold = 0.8
model_path = "./models/ei-raspi-img-class-int8-quantized-model.tflite"
labels = ['background', 'periquito', 'robot']
interpreter = None
classification_queue = Queue(maxsize=1)

def initialize_camera():
 global picam2
 picam2 = Picamera2()
 config = picam2.create_preview_configuration(main={"size": (320, 240)})
 picam2.configure(config)
 picam2.start()
 time.sleep(2) # Wait for camera to warm up

def get_frame():
 global frame
 while True:
 stream = io.BytesIO()
 picam2.capture_file(stream, format='jpeg')
 with frame_lock:
 frame = stream.getvalue()
 time.sleep(0.1) # Capture frames more frequently

def generate_frames():
 while True:
 with frame_lock:
 if frame is not None:
 yield (b'--frame\r\n'
 b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

def load_model():
 global interpreter
 if interpreter is None:
 interpreter = tflite.Interpreter(model_path=model_path)
 interpreter.allocate_tensors()
 return interpreter

def classify_image(img, interpreter):
 input_details = interpreter.get_input_details()
 output_details = interpreter.get_output_details()

 img = img.resize((input_details[0]['shape'][1],
 input_details[0]['shape'][2]))
 input_data = np.expand_dims(np.array(img), axis=0) \
 .astype(input_details[0]['dtype'])

 interpreter.set_tensor(input_details[0]['index'], input_data)
 interpreter.invoke()

 predictions = interpreter.get_tensor(output_details[0]['index'])[0]
 # Handle output based on type
 output_dtype = output_details[0]['dtype']
 if output_dtype in [np.int8, np.uint8]:
 # Dequantize the output
 scale, zero_point = output_details[0]['quantization']
 predictions = (predictions.astype(np.float32) - zero_point) * scale
 return predictions

def classification_worker():
 interpreter = load_model()
 while True:
 if is_classifying:
 with frame_lock:
 if frame is not None:
 img = Image.open(io.BytesIO(frame))
 predictions = classify_image(img, interpreter)
 max_prob = np.max(predictions)
 if max_prob >= confidence_threshold:
 label = labels[np.argmax(predictions)]
 else:
```

```

 label = 'Uncertain'
 classification_queue.put({'label': label,
 'probability': float(max_prob)})
 time.sleep(0.1) # Adjust based on your needs

@app.route('/')
def index():
 return render_template_string('''
 <!DOCTYPE html>
 <html>
 <head>
 <title>Image Classification</title>
 <script>
 src="https://code.jquery.com/jquery-3.6.0.min.js">
 </script>
 <script>
 function startClassification() {
 $.post('/start');
 $('#startBtn').prop('disabled', true);
 $('#stopBtn').prop('disabled', false);
 }

 function stopClassification() {
 $.post('/stop');
 $('#startBtn').prop('disabled', false);
 $('#stopBtn').prop('disabled', true);
 }

 function updateConfidence() {
 var confidence = $('#confidence').val();
 $.post('/update_confidence', {confidence: confidence});
 }

 function updateClassification() {
 $.get('/get_classification', function(data) {
 $('#classification').text(data.label + ': ' +
 data.probability.toFixed(2));
 });
 }
 </script>
 </head>
 <body>
 <h1>Image Classification</h1>

 <button id="startBtn" onclick="startClassification()">
 Start Classification
 </button>
 <button id="stopBtn" onclick="stopClassification()" disabled>
 Stop Classification
 </button>

 <label for="confidence">Confidence Threshold:</label>
 <input type="number" id="confidence" name="confidence" min="0"
 max="1" step="0.1" value="0.8" onchange="updateConfidence()">

 <div id="classification">Waiting for classification...</div>
 </body>
 </html>
''')

@app.route('/video_feed')
def video_feed():
 return Response(generate_frames(),
 mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/start', methods=['POST'])
def start_classification():
 global is_classifying
 is_classifying = True
 return '', 204

```

```

@app.route('/stop', methods=['POST'])
def stop_classification():
 global is_classifying
 is_classifying = False
 return '', 204

@app.route('/update_confidence', methods=['POST'])
def update_confidence():
 global confidence_threshold
 confidence_threshold = float(request.form['confidence'])
 return '', 204

@app.route('/get_classification')
def get_classification():
 if not is_classifying:
 return jsonify({'label': 'Not classifying', 'probability': 0})
 try:
 result = classification_queue.get_nowait()
 except Queue.Empty:
 result = {'label': 'Processing', 'probability': 0}
 return jsonify(result)

if __name__ == '__main__':
 initialize_camera()
 threading.Thread(target=get_frame, daemon=True).start()
 threading.Thread(target=classification_worker, daemon=True).start()
 app.run(host='0.0.0.0', port=5000, threaded=True)

```

Sul terminale lanciare:

```
python3 img_class_live_infer.py
```

E accedere all'interfaccia web:

- Sul Raspberry Pi stesso (se si ha una GUI): si apre un browser web e si va su `http://localhost:5000`
- Da un altro dispositivo sulla stessa rete: aprire un browser web e andare su `http://<raspberry_pi_ip>:5000` (Sostituire `<raspberry_pi_ip>` con l'indirizzo IP del Raspberry Pi). Per esempio: `http://192.168.4.210:5000/`

Ecco alcuni screenshot dell'app in esecuzione su un desktop esterno



Qui si può vedere l'app in esecuzione su YouTube:

<https://www.youtube.com/watch?v=o1QsQrpCMw4>

Il codice crea un'applicazione web per la classificazione delle immagini in tempo reale utilizzando un Raspberry Pi, il suo modulo fotocamera e un modello TensorFlow Lite. L'applicazione utilizza Flask per fornire un'interfaccia web in cui è possibile visualizzare il feed della fotocamera e vedere i risultati della classificazione in tempo reale.

### Componenti Chiave:

1. **Applicazione Web Flask:** Fornisce l'interfaccia utente e gestisce le richieste.
2. **PiCamera2:** Cattura le immagini dal modulo fotocamera Raspberry Pi.
3. **TensorFlow Lite:** Esegue il modello di classificazione delle immagini.

4. **Threading:** Gestisce le operazioni simultanee per prestazioni fluide.

#### Caratteristiche Principali:

- Visualizzazione feed telecamera live
- Classificazione immagini in tempo reale
- Soglia di confidenza regolabile
- Avvia/Arresta classificazione su richiesta

#### Struttura del Codice:

##### 1. Importazioni e Setup:

- Flask per applicazione web
- PiCamera2 per controllo telecamera
- TensorFlow Lite per l'inferenza
- Threading e Queue per operazioni concorrenti

##### 2. Variabili Globali:

- Gestione telecamera e frame
- Controllo classificazione
- Informazioni modello ed etichetta

##### 3. Funzioni della Telecamera:

- `initialize_camera()`: Imposta PiCamera2
- `get_frame()`: Cattura continuamente frame
- `generate_frames()`: Genera frame per il feed web

##### 4. Funzioni del Modello:

- `load_model()`: Carica il modello TFLite
- `classify_image()`: Esegue l'inferenza su una singola immagine

##### 5. Worker per la Classificazione:

- Gira in un thread separato
- Classifica continuamente i frame quando è attivo
- Aggiorna una coda con i risultati più recenti

##### 6. Route di Flask:

- `/`: Serve la pagina HTML principale
- `/video_feed`: Trasmette il feed della telecamera
- `/start and /stop`: Controlla la classificazione
- `/update_confidence`: Regola la soglia di confidenza
- `/get_classification`: Restituisce l'ultimo risultato di classificazione

##### 7. Template HTML:

- Visualizza il feed della telecamera e la classificazione dei risultati
- Fornisce controlli per avviare/arrestare e regolare le impostazioni

##### 8. Esecuzione Principale:

- Inizializza la fotocamera e avvia i thread necessari
- Esegue l'applicazione Flask

#### Concetti Chiave:

1. **Operazioni Concorrenti:** Utilizzo di thread per gestire l'acquisizione e la classificazione della telecamera separatamente dal server Web.
2. **Aggiornamenti in Tempo Reale:** Aggiornamenti frequenti dei risultati della classificazione senza ricaricare la pagina.

3. **Riutilizzo del Modello:** Caricamento del modello TFLite una volta e il riutilizzo per l'efficienza.
4. **Configurazione Flessibile:** Consente agli utenti di regolare la soglia di confidenza al volo.

#### Uso:

1. Assicurarsi che tutte le dipendenze siano installate.
2. Eseguire lo script su un Raspberry Pi con un modulo telecamera.
3. Accedere all'interfaccia Web da un browser utilizzando l'indirizzo IP del Raspberry Pi.
4. Avviare la classificazione e regolare le impostazioni in base alle esigenze.

#### Conclusione:

La classificazione delle immagini è emersa come un'applicazione potente e versatile dell'apprendimento automatico, con implicazioni significative per vari campi, dall'assistenza sanitaria al monitoraggio ambientale. Questo capitolo ha dimostrato come implementare un sistema di classificazione delle immagini robusto su dispositivi edge come Raspi-Zero e Raspi-5, mostrando il potenziale per l'intelligenza in tempo reale sul dispositivo.

Abbiamo esplorato l'intera pipeline di un progetto di classificazione delle immagini, dalla raccolta dati e dall'addestramento del modello tramite Edge Impulse Studio all'implementazione e all'esecuzione di inferenze su un Raspi. Il processo ha evidenziato diversi punti chiave:

1. L'importanza di una corretta raccolta dati e pre-elaborazione per l'addestramento efficace dei modelli.
2. La potenza dell'apprendimento tramite trasferimento, che ci consente di sfruttare modelli pre-addestrati come MobileNet V2 per un addestramento efficiente con dati limitati.
3. I compromessi tra accuratezza del modello e velocità di inferenza, particolarmente cruciali per i dispositivi edge.
4. L'implementazione della classificazione in tempo reale tramite un'interfaccia basata sul Web, che dimostra applicazioni pratiche.

La capacità di eseguire questi modelli su dispositivi edge come Raspi apre numerose possibilità per applicazioni IoT, sistemi autonomi e soluzioni di monitoraggio in tempo reale. Consente una latenza ridotta, una migliore privacy e il funzionamento in ambienti con connettività limitata.

Come abbiamo visto, anche con i vincoli computazionali dei dispositivi edge, è possibile ottenere risultati impressionanti in termini di accuratezza e velocità. La flessibilità di regolare i parametri del modello, come le dimensioni di input e i valori alfa, consente una messa a punto precisa per soddisfare requisiti di progetto specifici.

Guardando al futuro, il campo dell'intelligenza artificiale edge e della classificazione delle immagini continua a evolversi rapidamente. I progressi nelle tecniche di compressione dei modelli, nell'accelerazione hardware e nelle architetture di reti neurali più efficienti promettono di espandere ulteriormente le capacità dei dispositivi edge nelle attività di visione artificiale.

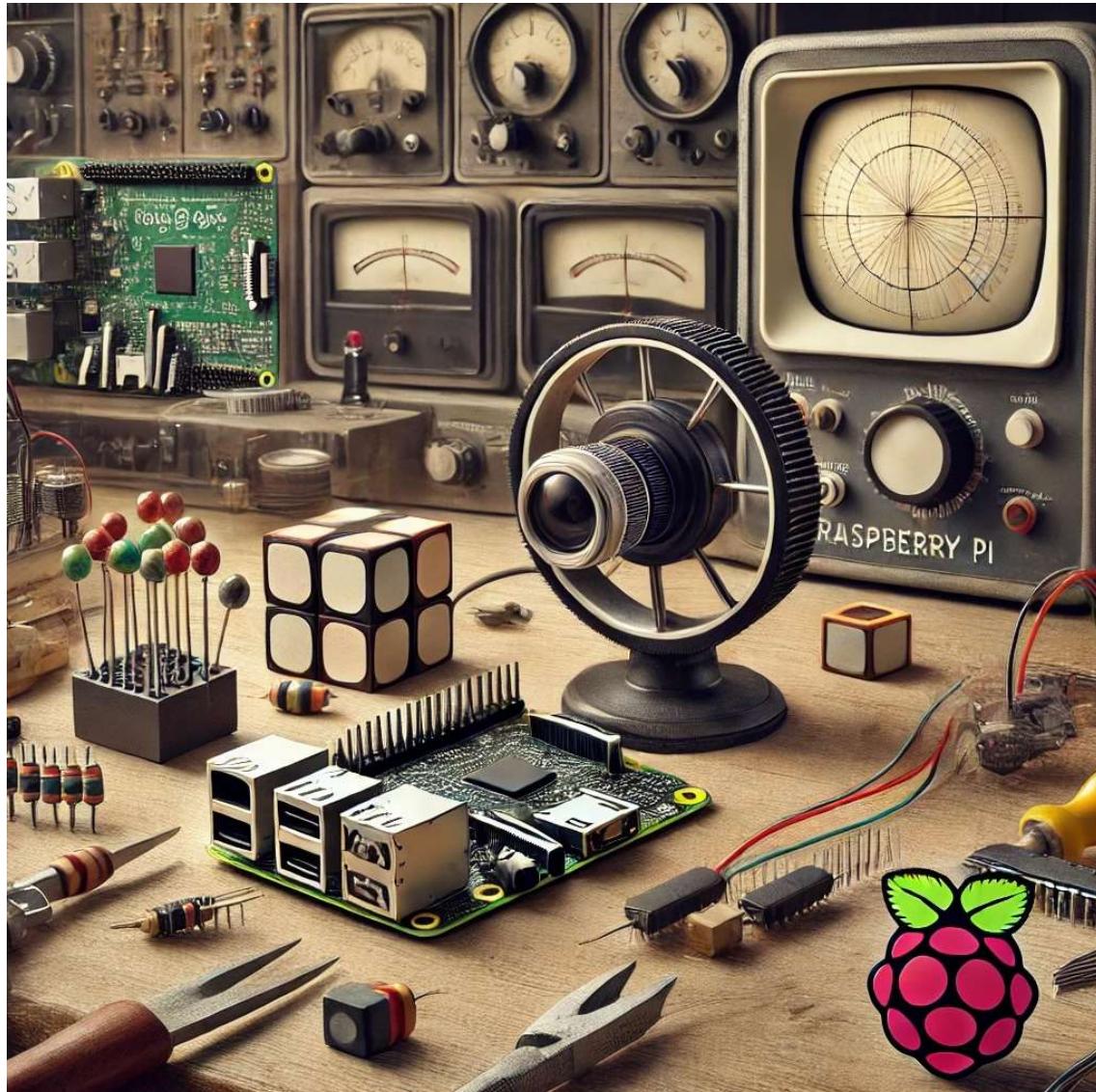
Questo progetto funge da base per applicazioni di visione artificiale più complesse e incoraggia un'ulteriore esplorazione nell'entusiasmante mondo dell'intelligenza artificiale edge e dell'IoT. Che si tratti di automazione industriale, applicazioni per la casa intelligente o monitoraggio ambientale, le competenze e i concetti trattati qui forniscono un solido punto di partenza per un'ampia gamma di progetti innovativi.

#### Risorse

- [Esempio di Dataset](#)

- Impostazione del Notebook di Prova su un Raspi
- Notebook di Classificazione delle Immagini su un Raspi
- CNN per classificare il dataset Cifar-10 su CoLab
- Cifar 10 - Classificazione delle Immagini su un Raspi
- Script Python
- Progetto Edge Impulse

## Rilevamento degli Oggetti



DALL·E prompt - Immagine di copertina per un capitolo "Rilevamento oggetti" in un tutorial su Raspberry Pi, progettata nello stesso stile da laboratorio di elettronica degli anni '50 delle copertine precedenti. La scena dovrebbe presentare in primo piano ruote e cubi, simili a quelli forniti dall'utente, posizionati su un banco da lavoro. Un Raspberry Pi con un modulo fotocamera collegato dovrebbe catturare un'immagine di questi oggetti. Circondare la scena con classici strumenti da laboratorio come saldatori, resistori e fili. Lo sfondo del laboratorio dovrebbe includere apparecchiature d'epoca come oscilloscopi e radio a valvole, mantenendo il tocco dettagliato e nostalgico dell'epoca. Non dovrebbero essere inclusi testo o loghi.

## Panoramica

Sulla base della nostra esplorazione della classificazione delle immagini, ora rivolgiamo la nostra attenzione a un'attività di visione artificiale più avanzata: il rilevamento degli oggetti. Mentre la classificazione delle immagini assegna una singola etichetta a un'intera immagine, il rilevamento degli oggetti va oltre identificando e localizzando più oggetti all'interno di una

singola immagine. Questa capacità apre molte nuove applicazioni e sfide, in particolare nell'edge computing e nei dispositivi IoT come Raspberry Pi.

Il rilevamento degli oggetti combina le attività di classificazione e localizzazione. Non solo determina quali oggetti sono presenti in un'immagine, ma individua anche le loro posizioni, ad esempio disegnando riquadri di delimitazione attorno a essi. Questa complessità aggiunta rende il rilevamento degli oggetti uno strumento più potente per comprendere le scene visive, ma richiede anche modelli e tecniche di training più sofisticati.

Nell'IA edge, dove lavoriamo con risorse computazionali limitate, l'implementazione di modelli di rilevamento degli oggetti efficienti diventa cruciale. Le sfide che abbiamo affrontato con la classificazione delle immagini, ovvero bilanciare le dimensioni del modello, la velocità di inferenza e l'accuratezza, sono amplificate nel rilevamento degli oggetti.

Tuttavia, i vantaggi sono anche più significativi, poiché il rilevamento degli oggetti consente un'analisi dei dati visivi più sfumata e dettagliata.

Alcune applicazioni del rilevamento di oggetti su dispositivi edge includono:

1. Sistemi di sorveglianza e sicurezza
2. Veicoli autonomi e droni
3. Controllo industriale della qualità
4. Monitoraggio della fauna selvatica
5. Applicazioni di realtà aumentata

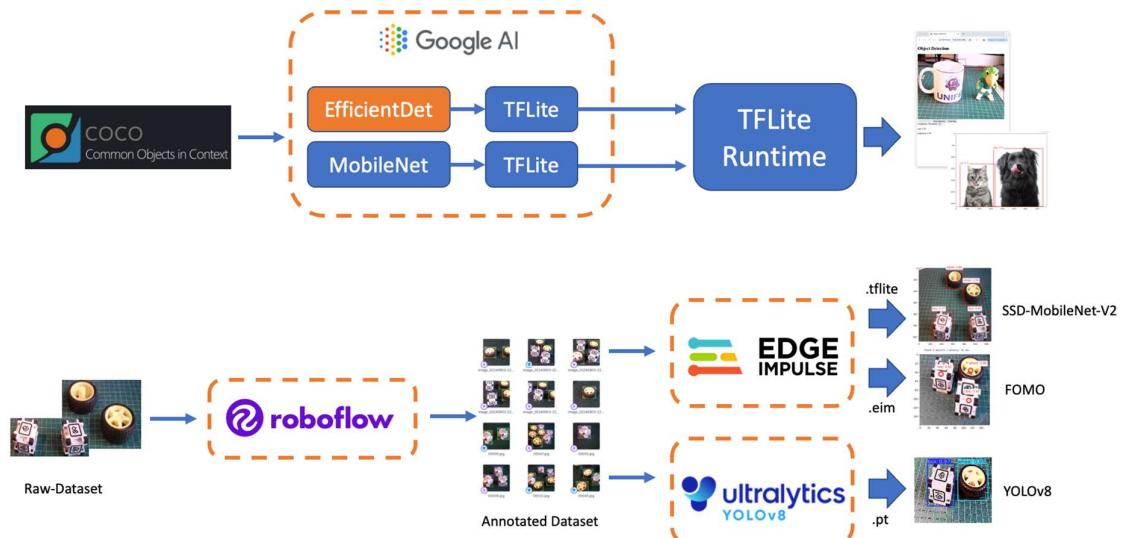
Mettendo le mani nel rilevamento di oggetti, ci baseremo sui concetti e sulle tecniche esplorate nella classificazione delle immagini. Esamineremo le architetture di rilevamento di oggetti più diffuse progettate per l'efficienza, come:

- “Single Stage Detectors” [Rilevatori a stadio singolo], come MobileNet ed EfficientDet,
- FOMO (Faster Objects, More Objects), e
- YOLO (You Only Look Once).

Per saperne di più sui modelli di rilevamento di oggetti, seguire il tutorial [A Gentle Introduction to Object Recognition With Deep Learning](#).

Esploreremo quei modelli di rilevamento di oggetti utilizzando

- TensorFlow Lite Runtime (ora modificato in [LiteRT](#)),
- Edge Impulse Linux Python SDK e
- Ultralytics



In questo laboratorio, tratteremo i fondamenti del rilevamento degli oggetti e le sue differenze rispetto alla classificazione delle immagini. Impareremo anche come addestrare, mettere a

punto, testare, ottimizzare e distribuire le architetture di rilevamento degli oggetti più diffuse utilizzando un dataset creato da zero.

### Fondamenti dell'Object Detection

Il rilevamento degli oggetti si basa sulle fondamenta della classificazione delle immagini, ma ne amplia notevolmente le capacità. Per comprendere il rilevamento degli oggetti, è fondamentale innanzitutto riconoscere le sue principali differenze rispetto alla classificazione delle immagini:

#### Classificazione delle Immagini vs. Rilevamento degli Oggetti

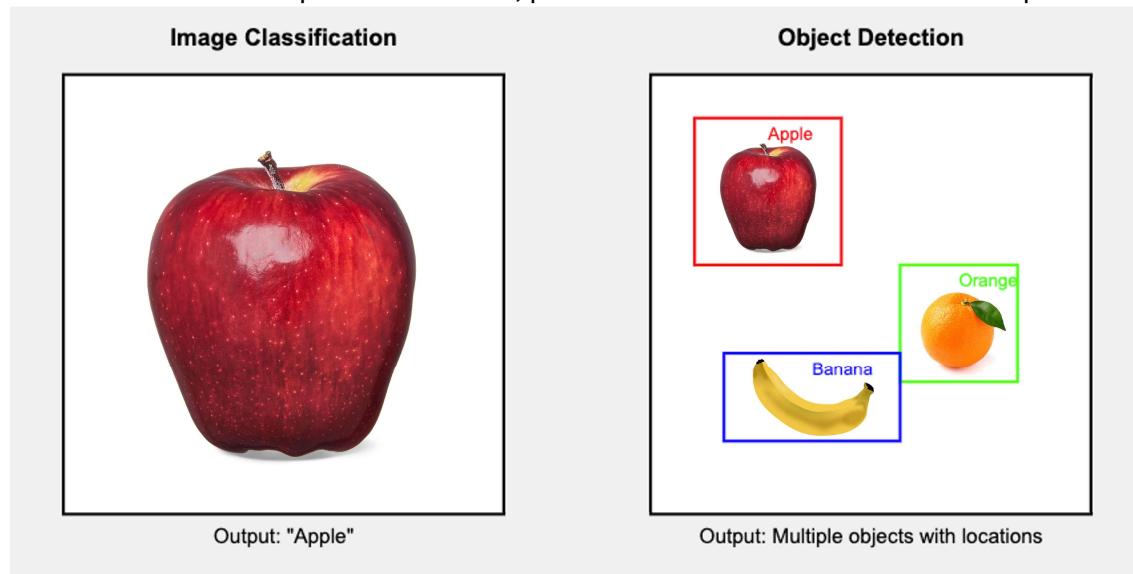
##### Classificazione delle Immagini:

- Assegna una singola etichetta a un'intera immagine
- Risponde alla domanda: "Qual è l'oggetto o la scena principale di questa immagine?"
- Genera una singola previsione di classe per l'intera immagine

##### Rilevamento degli oggetti:

- Identifica e individua più oggetti all'interno di un'immagine
- Risponde alle domande: "Quali oggetti sono presenti in questa immagine e dove si trovano?"
- Genera più previsioni, ciascuna composta da un'etichetta di classe e un riquadro di delimitazione

Per visualizzare questa differenza, prendiamo in considerazione un esempio:



Questo diagramma illustra la differenza critica: la classificazione delle immagini fornisce un'unica etichetta per l'intera immagine, mentre il rilevamento degli oggetti identifica più oggetti, le loro classi e le loro posizioni all'interno dell'immagine.

### Componenti Chiave del Rilevamento degli Oggetti

I sistemi di rilevamento degli oggetti sono in genere costituiti da due componenti principali:

1. Localizzazione degli Oggetti: Questo componente identifica dove si trovano gli oggetti nell'immagine. In genere genera riquadri di delimitazione, regioni rettangolari che racchiudono ogni oggetto rilevato.
2. Classificazione degli Oggetti: Questo componente determina la classe o la categoria di ogni oggetto rilevato, in modo simile alla classificazione delle immagini ma applicato a ogni regione localizzata.

## Sfide nel Rilevamento degli Oggetti

Il rilevamento degli oggetti presenta diverse sfide oltre a quelle della classificazione delle immagini:

- Oggetti multipli: Un'immagine può contenere più oggetti di varie classi, dimensioni e posizioni.
- Scale variabili: Gli oggetti possono apparire a diverse dimensioni all'interno dell'immagine.
- Occlusione: Gli oggetti possono essere parzialmente nascosti o sovrapposti.
- Disordine di sfondo: Distinguere gli oggetti da sfondi complessi può essere difficile.
- Prestazioni in tempo reale: Molte applicazioni richiedono tempi di inferenza rapidi, soprattutto su dispositivi edge.

## Approcci al Rilevamento di Oggetti

Esistono due approcci principali al rilevamento di oggetti:

1. Rilevatori a due stadi: Prima propongono le regioni di interesse e poi classificano ciascuna regione. Esempi includono R-CNN e le sue varianti (Fast R-CNN, Faster R-CNN).
2. Rilevatori a stadio singolo: Questi prevedono bounding box (o centroidi) e probabilità di classe in un passaggio “forward” [in avanti] della rete. Esempi includono YOLO (You Only Look Once), EfficientDet, SSD (Single Shot Detector) e FOMO (Faster Objects, More Objects). Questi sono spesso più veloci e più adatti per dispositivi edge come Raspberry Pi.

## Metriche di Valutazione

Il rilevamento degli oggetti utilizza metriche diverse rispetto alla classificazione delle immagini:

- **Intersection over Union (IoU)**: Misura la sovrapposizione tra bounding box previsti e “ground truth” [reali].
- **Mean Average Precision (mAP)**: Combina precisione e richiamo in tutte le classi e soglie IoU.
- **Frames Per Second (FPS)**: Misura la velocità di rilevamento, fondamentale per le applicazioni in tempo reale su dispositivi edge.

## Panoramica dei Modelli di Rilevamento degli Oggetti Pre-Addestrati

Come abbiamo visto nell'introduzione, data un'immagine o un flusso video, un modello di rilevamento degli oggetti può identificare quale, di un set noto di oggetti, potrebbe essere presente e fornire informazioni sulle loro posizioni all'interno dell'immagine.

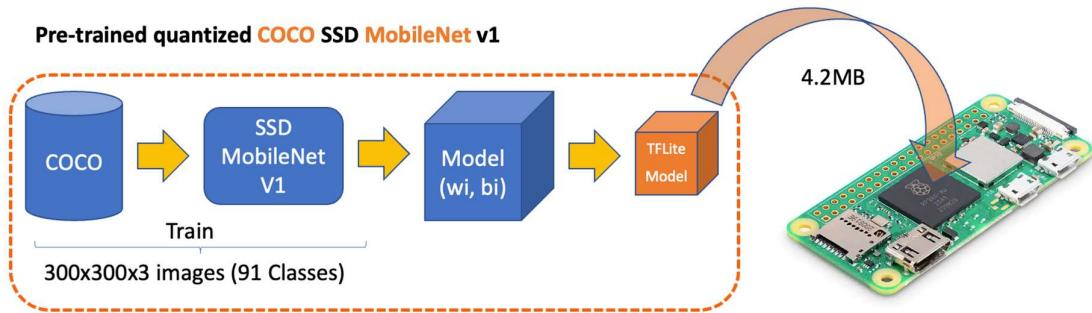
Si possono testare alcuni modelli comuni online visitando [Object Detection - MediaPipe Studio](#)

Su [Kaggle](#), possiamo trovare i modelli tflite pre-addestrati più comuni da usare con Raspi, [ssd\\_mobilenet\\_v1](#) e [EfficientDet](#). Quei modelli sono stati addestrati sul dataset COCO (Common Objects in Context), con oltre 200.000 immagini etichettate in 91 categorie. Scaricare i modelli e caricali nella cartella `./models` in Raspi.

In alternativa, si possono trovare i modelli e le etichette COCO su [GitHub](#).

Per la prima parte di questo laboratorio, ci concentreremo su un modello SSD-Mobilenet V1 300x300 pre-addestrato e lo confronteremo con EfficientDet-lite0 320x320, anch'esso addestrato utilizzando il set di dati COCO 2017. Entrambi i modelli sono stati convertiti in un formato TensorFlow Lite (4,2 MB per SSD Mobilenet e 4,6 MB per EfficientDet).

SSD-Mobilenet V2 o V3 è consigliato per progetti di “transfer learning”, ma una volta che il modello TFLite V1 sarà disponibile al pubblico, lo useremo per questa panoramica.



### Impostazione dell'Ambiente TFLite

Dovremo confermare i passaggi eseguiti nell'ultimo laboratorio pratico, Image Classification, come segue:

- Aggiornamento di Raspberry Pi
- Installazione delle Librerie Richieste
- Impostazione di un Ambiente Virtuale (Facoltativo ma Consigliato)

---

source ~/tflite/bin/activate

---

- Installazione di TensorFlow Lite Runtime
- Installazione di Additional Python Libraries (all'interno dell'ambiente)

### Creazione di una Directory di Lavoro:

Considerando che abbiamo creato la cartella `Documents/TFLITE` nell'ultimo Lab, creiamo ora le cartelle specifiche per questo lab di rilevamento degli oggetti:

---

```
cd Documents/TFLITE/
mkdir OBJ_DETECT
cd OBJ_DETECT
mkdir images
mkdir models
cd models
```

---

### Inferenza e Post-Elaborazione

Apriamo un nuovo `notebook` per seguire tutti i passaggi per rilevare oggetti su un'immagine:

Importare le librerie necessarie:

---

```
import time
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import tensorflow as tf
```

---

Caricare il modello TFLite e allocare i tensori:

---

```
model_path = "./models/ssd-mobilenet-v1-tflite-default-v1.tflite"
interpreter = tf.lite.Interpreter(model_path=model_path)
interpreter.allocate_tensors()
```

---

Ottenerne i tensori di input e output.

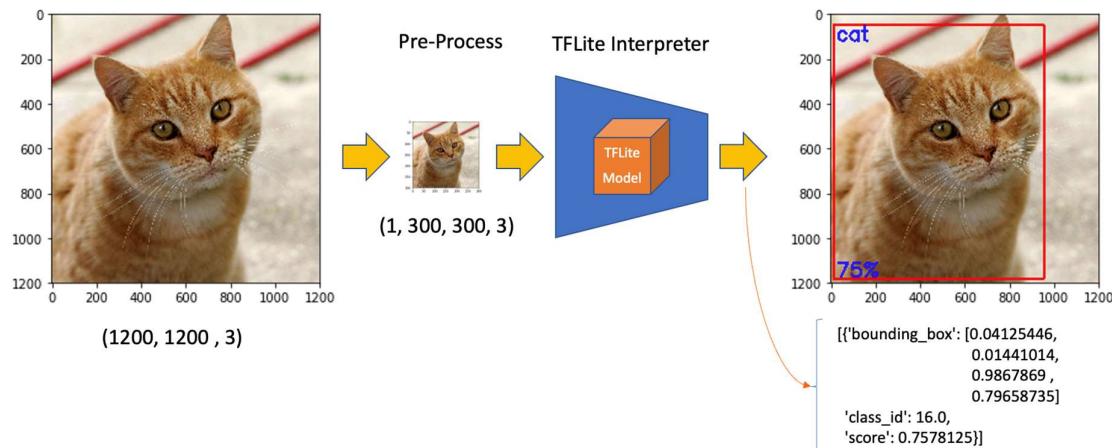
---

```
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

---

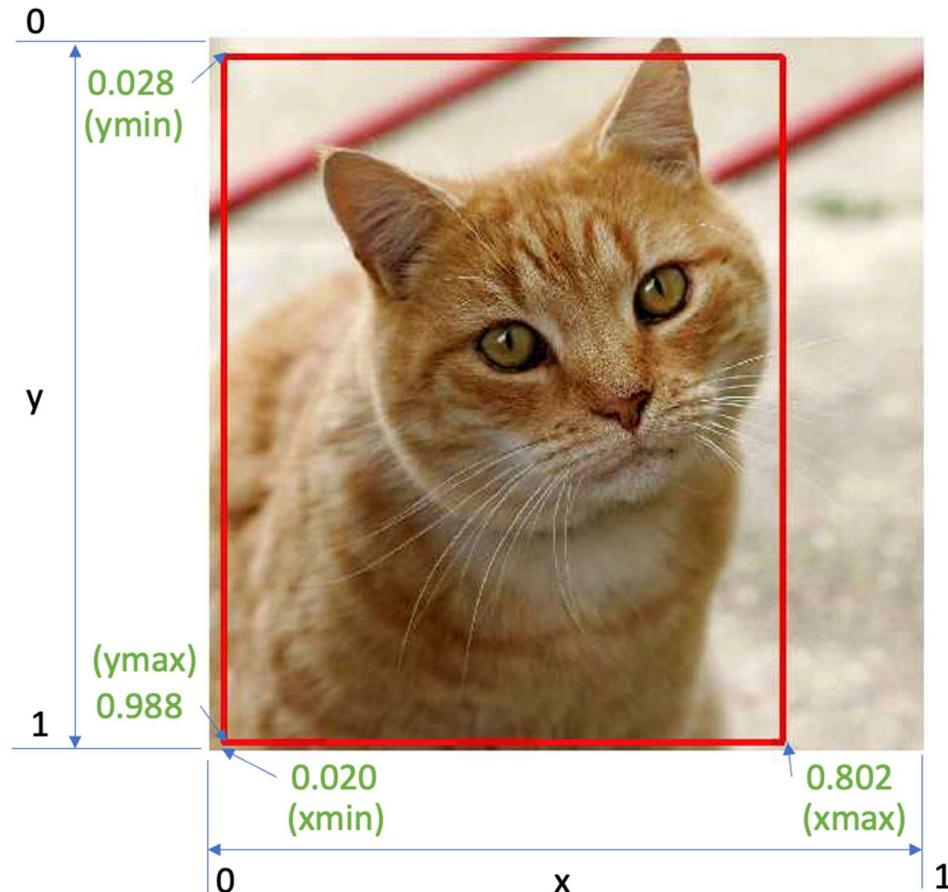
**Input details** ci informeranno su come il modello dovrebbe essere alimentato con un'immagine. La forma di `(1, 300, 300, 3)` con un `dtype` di `uint8` ci dice che un'immagine non normalizzata (intervallo di valori pixel da 0 a 255) con dimensioni (300x300x3) dovrebbe essere inserita una alla volta (Batch Dimension: 1).

Gli **output details** includono non solo le etichette (“classes”) e le probabilità (“scores”), ma anche la posizione relativa della finestra dei bounding box (“box”) su dove si trova l'oggetto nell'immagine e il numero di oggetti rilevati (“num\_detections”). I dettagli di output ci dicono anche che il modello può rilevare un **massimo di 10 oggetti** nell'immagine.



Quindi, per l'esempio precedente, utilizzando la stessa immagine di gatto utilizzata col *Lab di Classificazione Immagine* alla ricerca dell'output, abbiamo una **probabilità del 76%** di aver trovato un oggetto con un **ID classe di 16** su un'area delimitata da un **bounding box** di **[0.028011084, 0.020121813, 0.9886069, 0.802299]**. Questi quattro numeri sono correlati a `ymin`, `xmin`, `ymax` e `xmax`, le coordinate del box.

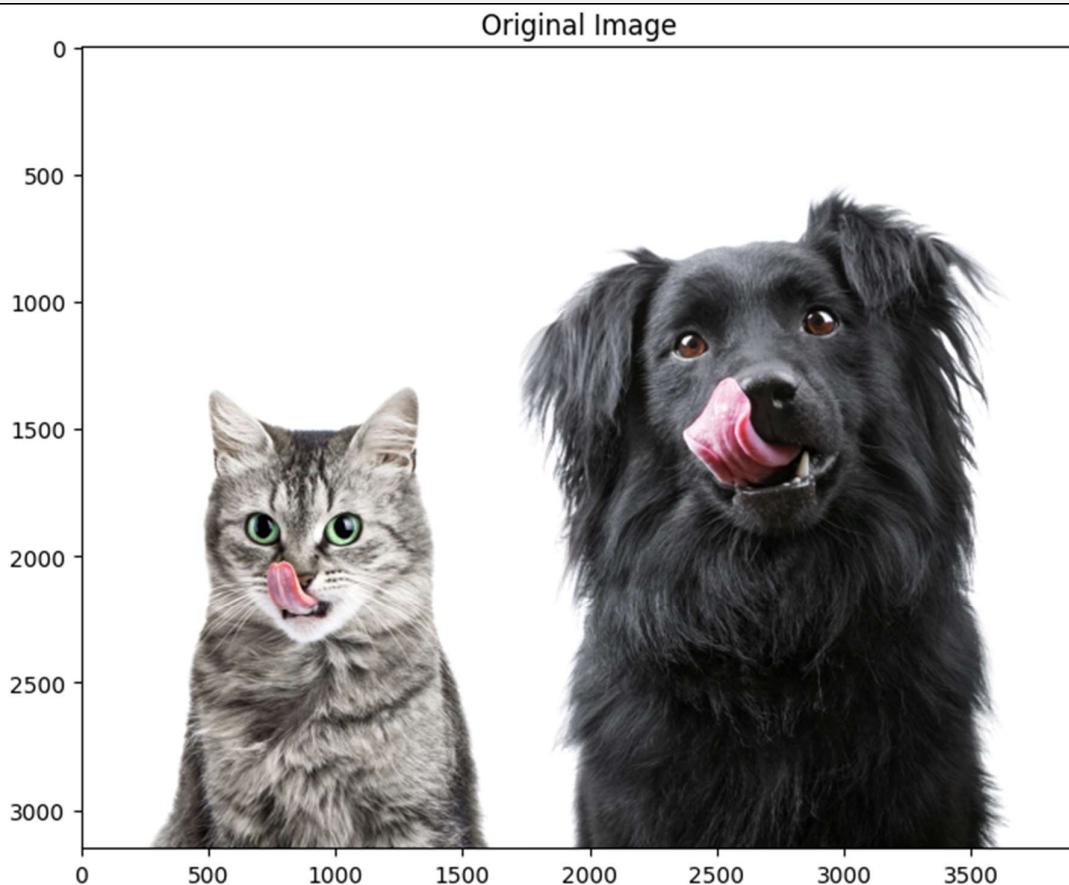
Considerando che **y** va dall'alto (`ymin`) al basso (`ymax`) e **x** va da sinistra (`xmin`) a destra (`xmax`), abbiamo, di fatto, le coordinate dell'angolo superiore/sinistro e di quello inferiore/destro. Con entrambi i bordi e conoscendo la forma dell'immagine, è possibile disegnare un rettangolo attorno all'oggetto, come mostrato nella figura sottostante:



Successivamente, dovremmo scoprire cosa significa un ID di classe uguale a 16. Aprendo il file `coco_labels.txt`, come un elenco, ogni elemento ha un indice associato e ispezionando l'indice 16, otteniamo, come previsto, `cat`. La probabilità è il valore restituito dal punteggio. Carichiamo ora alcune immagini con più oggetti su di esse per il test.

```
img_path = "./images/cat_dog.jpeg"
orig_img = Image.open(img_path)

Display the image
plt.figure(figsize=(8, 8))
plt.imshow(orig_img)
plt.title("Original Image")
plt.show()
```



In base ai dettagli di input, pre-elaboriamo l'immagine, modificandone la forma ed espandendone le dimensioni:

```
img = orig_img.resize((input_details[0]['shape'][1],
 input_details[0]['shape'][2]))
input_data = np.expand_dims(img, axis=0)
input_data.shape, input_data.dtype
```

La nuova forma `input_data` è `(1, 300, 300, 3)` con un `dtype` di `uint8`, che è compatibile con quanto previsto dal modello.

Utilizzando `input_data`, eseguiamo l'interprete, misuriamo la latenza e otteniamo l'output:

```
start_time = time.time()
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()
end_time = time.time()
inference_time = (end_time - start_time) * 1000 # Convert to milliseconds
print ("Inference time: {:.1f}ms".format(inference_time))
```

Con una latenza di circa 800 ms, possiamo ottenere 4 output distinti:

```
boxes = interpreter.get_tensor(output_details[0]['index'])[0]
classes = interpreter.get_tensor(output_details[1]['index'])[0]
scores = interpreter.get_tensor(output_details[2]['index'])[0]
num_detections = int(interpreter.get_tensor(output_details[3]['index'])[0])
```

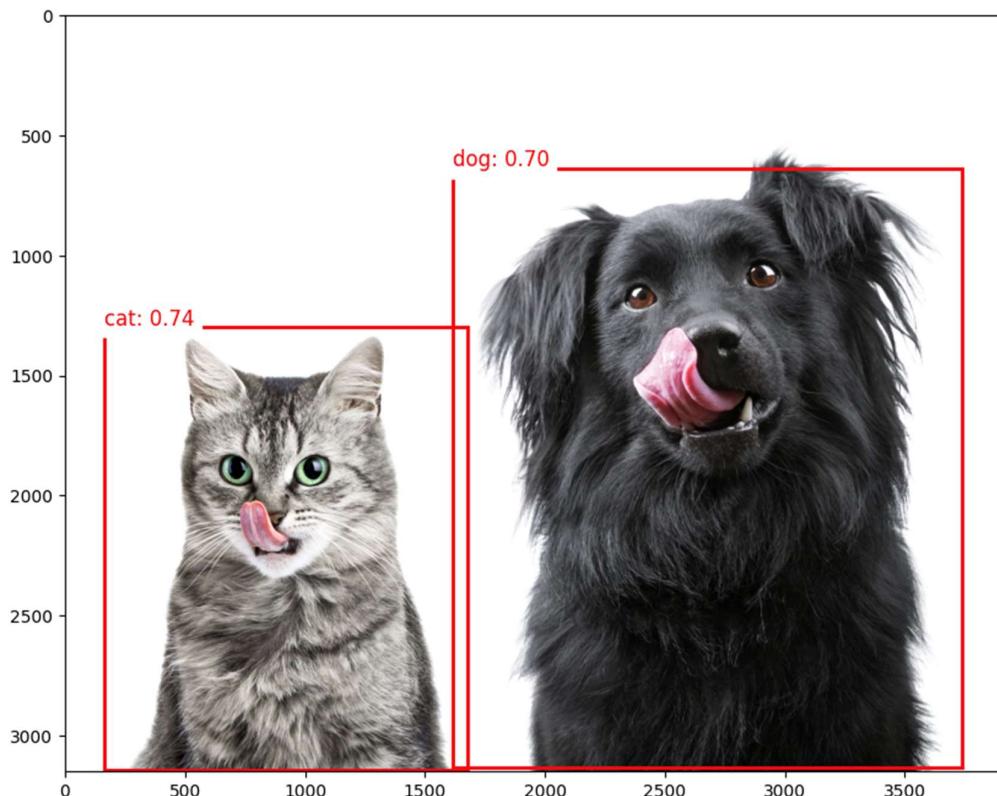
Da una rapida occhiata, possiamo vedere che il modello ha rilevato 2 oggetti con un punteggio superiore a 0,5:

```
for i in range(num_detections):
 if scores[i] > 0.5: # Confidence threshold
 print(f"Object {i}:")
 print(f" Bounding Box: {boxes[i]}")
 print(f" Confidence: {scores[i]}")
 print(f" Class: {classes[i]}")

Object 0:
 Bounding Box: [0.4125163 0.04130688 0.997076 0.42888364]
 Confidence: 0.73828125
 Class: 16.0
Object 1:
 Bounding Box: [0.20249811 0.41268167 0.99390197 0.95425284]
 Confidence: 0.69921875
 Class: 17.0
```

E possiamo anche visualizzare i risultati:

```
plt.figure(figsize=(12, 8))
plt.imshow(orig_img)
for i in range(num_detections):
 if scores[i] > 0.5: # Adjust threshold as needed
 ymin, xmin, ymax, xmax = boxes[i]
 (left, right, top, bottom) = (xmin * orig_img.width,
 xmax * orig_img.width,
 ymin * orig_img.height,
 ymax * orig_img.height)
 rect = plt.Rectangle((left, top), right-left, bottom-top,
 fill=False, color='red', linewidth=2)
 plt.gca().add_patch(rect)
 class_id = int(classes[i])
 class_name = labels[class_id]
 plt.text(left, top-10, f'{class_name}: {scores[i]:.2f}',
 color='red', fontsize=12, backgroundcolor='white')
```



### ***EfficientDet***

EfficientDet non è tecnicamente un modello SSD (Single Shot Detector), ma condivide alcune somiglianze e si basa su idee provenienti da SSD e altre architetture di rilevamento degli oggetti:

1. EfficientDet:

- Sviluppato dai ricercatori di Google nel 2019
- Utilizza EfficientNet come rete backbone
- Utilizza una nuova “bi-directional feature pyramid network (BiFPN)” [rete piramidale di feature bidirezionale]
- Utilizza il ridimensionamento composto per ridimensionare in modo efficiente la rete backbone e i componenti di rilevamento degli oggetti.

2. Somiglianze con SSD:

- Entrambi sono rilevatori a stadio singolo, il che significa che eseguono la localizzazione e la classificazione degli oggetti in un singolo passaggio “forward” [in avanti].
- Entrambi utilizzano mappe di feature multiscala per rilevare oggetti a diverse scale.

3. Differenze principali:

- Backbone: SSD in genere utilizza VGG o MobileNet, mentre EfficientDet utilizza EfficientNet.
- Fusione di funzionalità: SSD utilizza una semplice piramide di funzionalità, mentre EfficientDet utilizza il più avanzato BiFPN.
- Metodo di ridimensionamento: EfficientDet introduce il ridimensionamento composto per tutti i componenti della rete

4. Vantaggi di EfficientDet:

- In genere, raggiunge migliori compromessi tra accuratezza ed efficienza rispetto a SSD e molti altri modelli di rilevamento di oggetti.
- Un ridimensionamento più flessibile consente una famiglia di modelli con diversi compromessi tra dimensioni e prestazioni.

Sebbene EfficientDet non sia un modello SSD, può essere visto come un’evoluzione delle architetture di rilevamento a fase singola, che incorpora tecniche più avanzate per migliorare efficienza e precisione. Quando si utilizza EfficientDet, possiamo aspettarci strutture di output simili a SSD (ad esempio, bounding box e punteggi di classe).

Su GitHub, si trova un altro [notebook](#) che esplora il modello EfficientDet che abbiamo realizzato con SSD MobileNet.

### **Progetto di Rilevamento di Oggetti**

Ora, svilupperemo un progetto completo di classificazione delle immagini dalla raccolta dei dati all’addestramento e all’implementazione. Come abbiamo fatto con il progetto di classificazione delle immagini, il modello addestrato e convertito verrà utilizzato per l’inferenza.

Utilizzeremo lo stesso set di dati per addestrare 3 modelli: SSD-MobileNet V2, FOMO e YOLO.

### **L’Obiettivo**

Tutti i progetti di Machine Learning devono iniziare con un obiettivo. Supponiamo di trovarci in una struttura industriale e di dover ordinare e contare **ruote** e **scatole** speciali.



In altre parole, dovremmo eseguire una classificazione multi-etichetta, in cui ogni immagine può avere tre classi:

- Background (nessun oggetto)
- Box [Scatola]
- Wheel [Ruota]

### Raccolta Dati Grezzi

Una volta definito l'obiettivo del nostro progetto di apprendimento automatico, il passaggio successivo, e più cruciale, è la raccolta del dataset. Possiamo utilizzare un telefono, il Raspberry Pi o un mix per creare il set di dati grezzi (senza etichette). Usiamo la semplice app Web sul nostro Raspberry Pi per visualizzare le immagini QVGA (320 x 240) catturate in un browser.

Da GitHub, si prende lo script Python [get\\_img\\_data.py](#) e lo si apre nel terminale:

```
python3 get_img_data.py
```

Accedere all'interfaccia web:

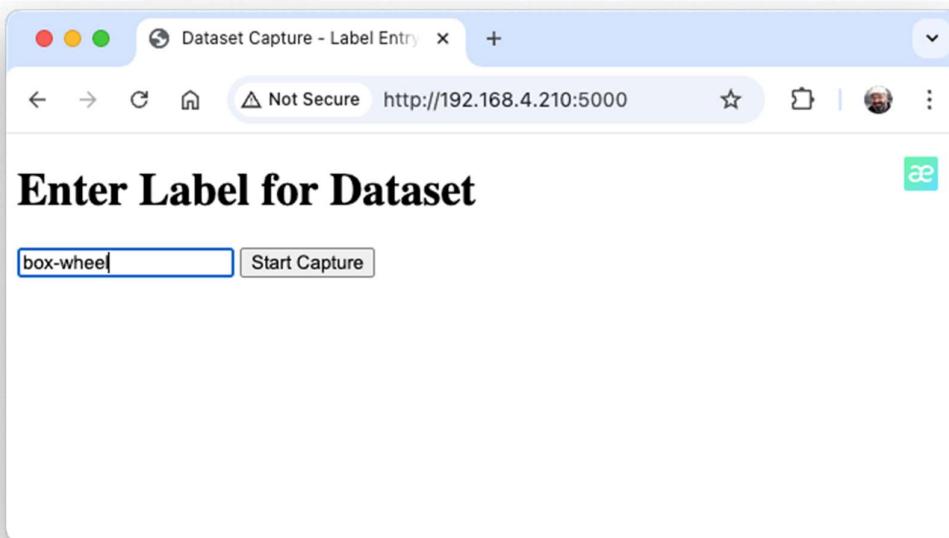
- Sul Raspberry Pi stesso (se si ha una GUI): si apre un browser web e si va [SU http://localhost:5000](http://localhost:5000)
- Da un altro dispositivo sulla stessa rete: aprire un browser web e andare su [http://<raspberry\\_pi\\_ip>:5000](http://<raspberry_pi_ip>:5000) (Sostituire <raspberry\_pi\_ip> con l'indirizzo IP del Raspberry Pi). Per esempio: <http://192.168.4.210:5000/>

```
mjrovai@raspi-zero:~$ source ~/tf-lite/bin/activate
(mjrovai@raspi-zero:~$ cd Documents/TFLITE/OBJ_DETECT
(mjrovai@raspi-zero:~/Documents/TFLITE/OBJ_DETECT$ python3 get_img_data.py
[29:05:34.506637970] [26857] INFO Camera camera_manager.cpp:313 libcamera v0.3.0+65-6ddd79b5
[29:05:34.731419984] [26857] INFO Camera camera_manager.cpp:313 libcamera v0.3.0+65-6ddd79b5
[29:05:34.841600314] [26857] INFO Camera camera.cpp:1183 configuring streams: (0) 320x240-MJPEG
 * Serving Flask app 'get_img_data'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.4.210:5000
Press CTRL+C to quit
```

Lo script

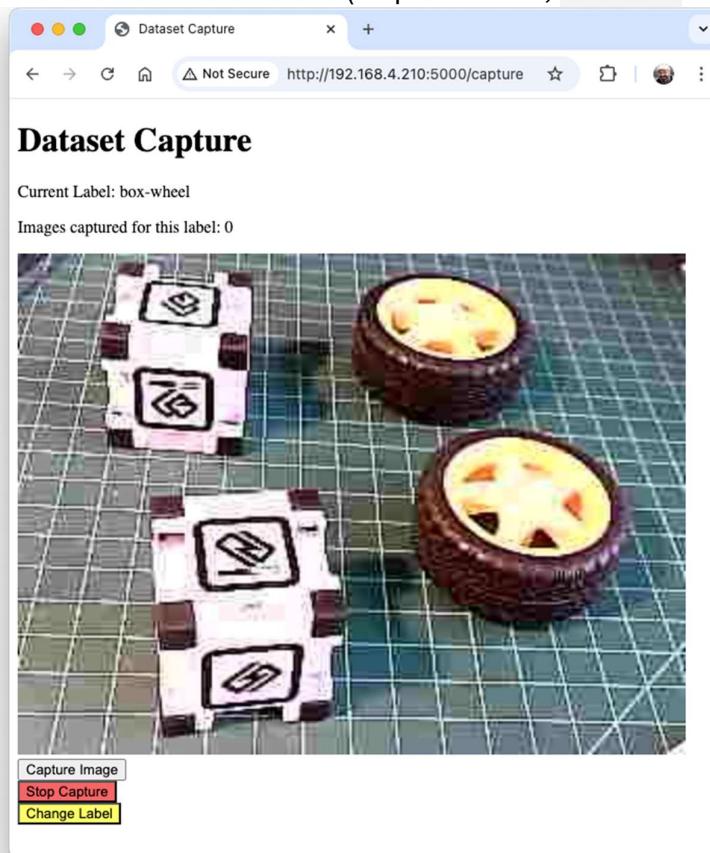
Python crea un'interfaccia basata sul Web per catturare e organizzare set di dati di immagini utilizzando un Raspberry Pi e la sua fotocamera. È utile per progetti di apprendimento automatico che richiedono dati di immagini etichettati o meno, come nel nostro caso.

Accedere all'interfaccia Web da un browser, inserire un'etichetta generica per le immagini da catturare e premere Start Capture.



Notare che le immagini da catturare avranno più etichette che dovranno essere definite in seguito.

Utilizzare l'anteprima live per posizionare la fotocamera e cliccare su **Capture Image** per salvare le immagini sotto l'etichetta corrente (in questo caso, **box-wheel**).



Quando abbiamo abbastanza immagini, possiamo premere **Stop Capture**. Le immagini catturate vengono salvate nella cartella dataset/box-wheel:

```
(tflite) mjrovai@raspi-zero:~/Documents/TFLITE/OBJ_DETECT/dataset -- ssh mjrovai@192.168.4.210 - 102x11
Untitled.ipynb dataset get_img_data.py images models
(tflite) mjrovai@raspi-zero:~/Documents/TFLITE/OBJ_DETECT $ cd dataset
(tflite) mjrovai@raspi-zero:~/Documents/TFLITE/OBJ_DETECT/dataset $ ls
box-wheel
(tflite) mjrovai@raspi-zero:~/Documents/TFLITE/OBJ_DETECT/dataset $ ls box-wheel
image_20240903-224513.jpg image_20240903-224516.jpg image_20240903-224530.jpg
image_20240903-224512.jpg image_20240903-224533.jpg image_20240903-224535.jpg
image_20240903-224458.jpg image_20240903-224520.jpg image_20240903-224532.jpg
image_20240903-224504.jpg image_20240903-224524.jpg
(tflite) mjrovai@raspi-zero:~/Documents/TFLITE/OBJ_DETECT/dataset $
```

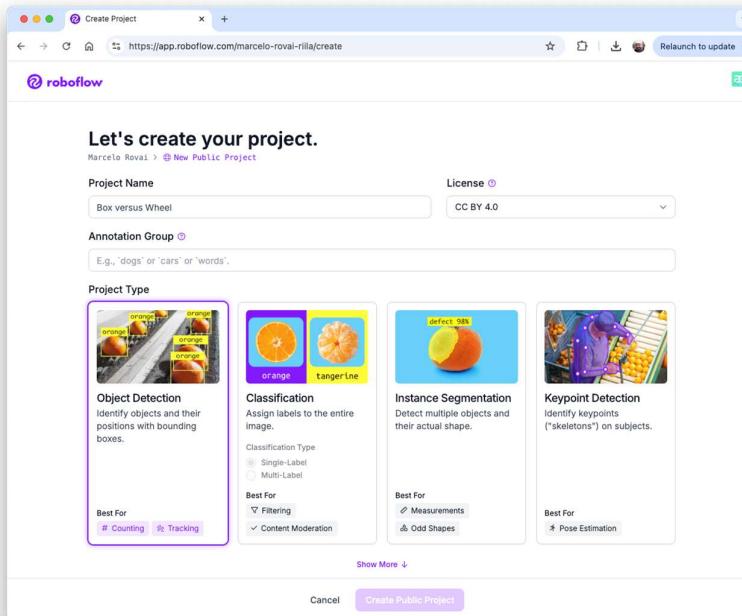
Acquisire circa 60 immagini. Provare ad acquisire con diverse angolazioni, sfondi e condizioni di luce. Filezilla può trasferire il dataset raw creato sul computer principale.

### ***Etichettatura dei Dati***

Il passaggio successivo in un progetto di Object Detect è creare un dataset etichettato. Dovremmo etichettare le immagini del dataset raw, creando riquadri di delimitazione attorno agli oggetti di ogni immagine (box e ruota). Possiamo usare strumenti di etichettatura come [LabelImg](#), [CVAT](#), [Roboflow](#), o persino [Edge Impulse Studio](#). Dopo aver esplorato lo strumento Edge Impulse in altri laboratori, usiamo Roboflow qui.

Stiamo usando Roboflow (versione gratuita) qui per due motivi principali. 1) Possiamo avere un auto-labeler e 2) Il dataset annotato è disponibile in diversi formati e può essere utilizzato sia su Edge Impulse Studio (lo useremo per MobileNet V2 e FOMO train) sia su CoLab (YOLOv8 train), ad esempio. Avendo il dataset annotato su Edge Impulse (account gratuito), non è possibile utilizzarlo per il training su altre piattaforme.

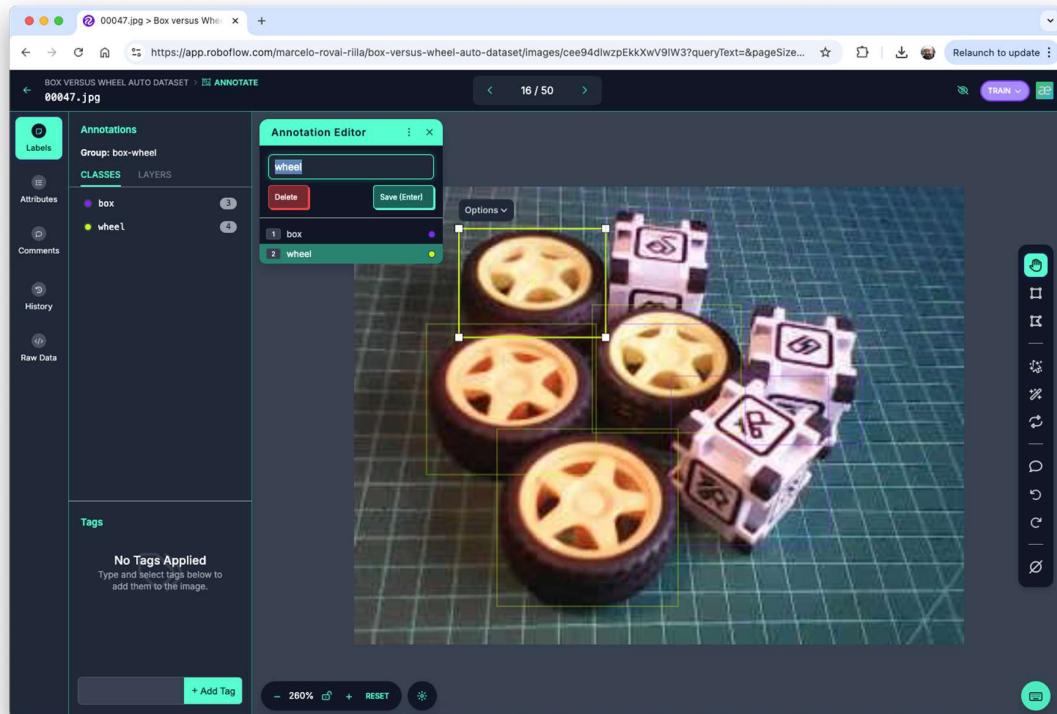
Dovremmo caricare il dataset grezzo su [Roboflow](#). Creare un account gratuito lì e avviare un nuovo progetto, ad esempio (“box-versus-wheel”).



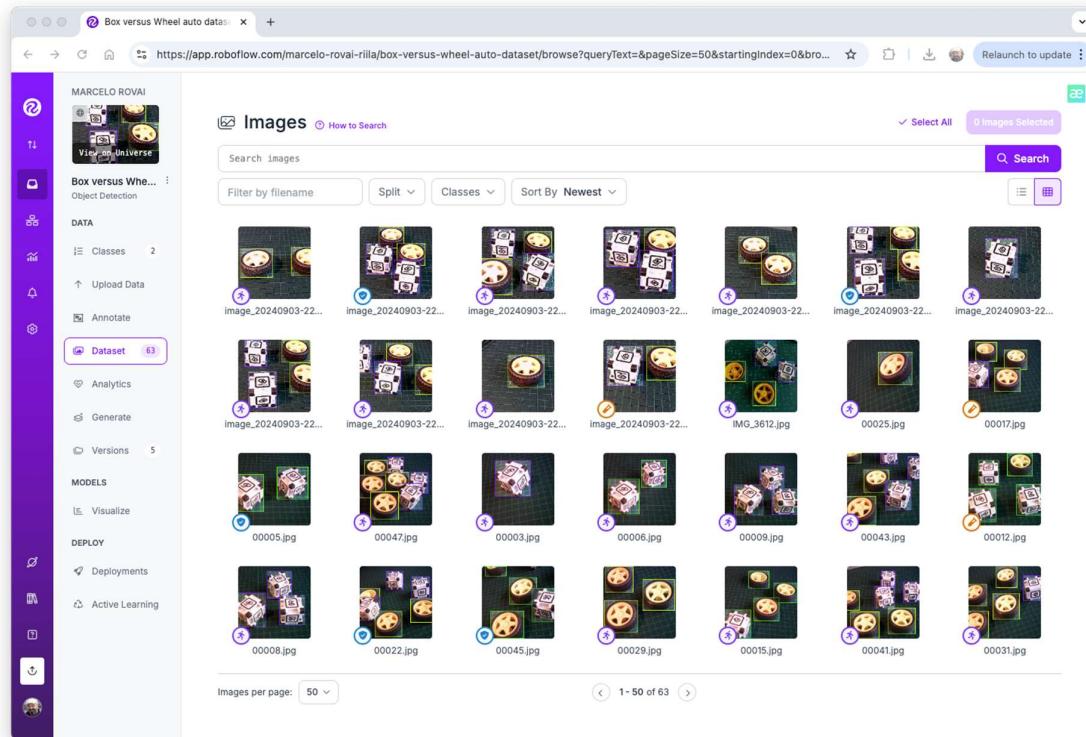
Non entreremo nei dettagli del processo Roboflow dato che sono disponibili molti tutorial.

## Annotazione

Una volta creato il progetto e caricato il dataset, si devono effettuare le annotazioni utilizzando lo strumento “Auto-Label”. Notare che si può anche caricare immagini con solo uno sfondo, che dovrebbero essere salvate senza annotazioni.



Una volta annotate tutte le immagini, si devono dividere in training, validation e testing.

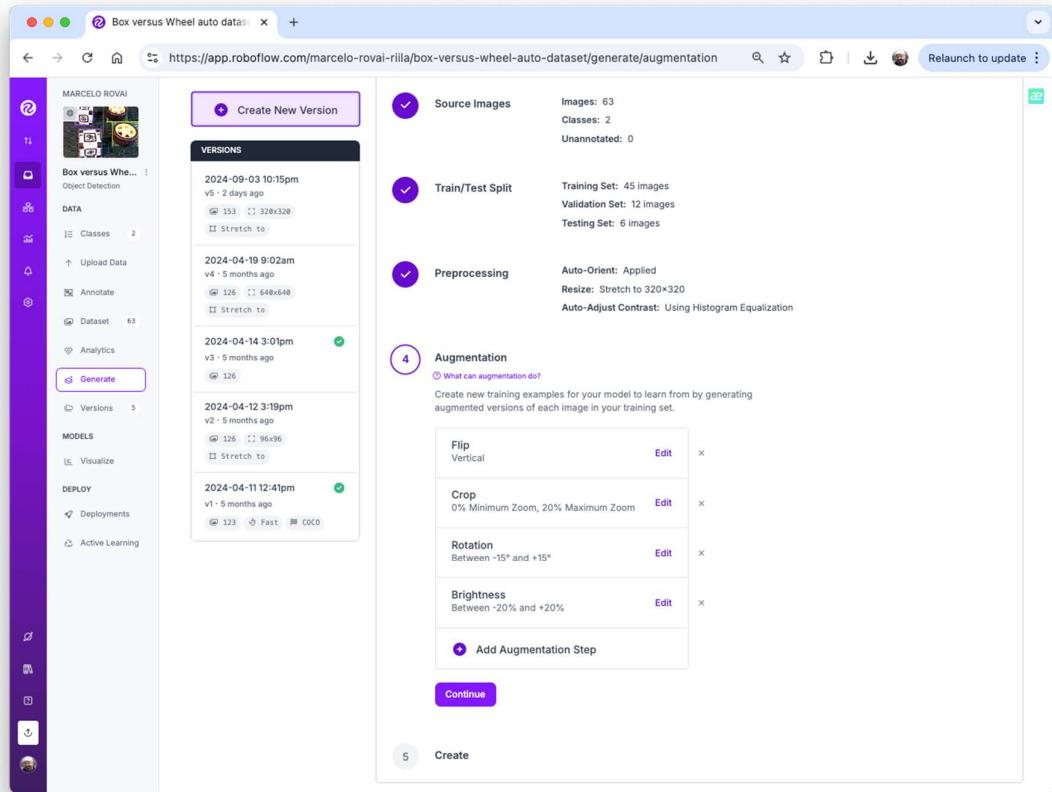


## Pre-elaborazione dei Dati

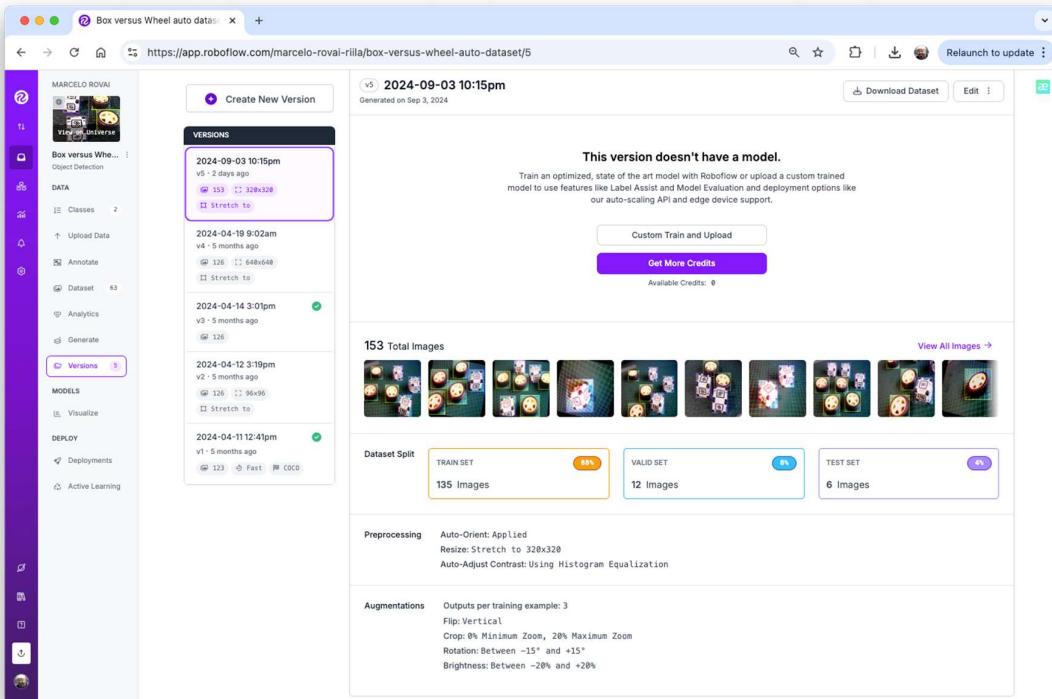
L'ultimo passaggio con il dataset è la pre-elaborazione per generare una versione finale per il training. Ridimensioniamo tutte le immagini a 320x320 e generiamo versioni aumentate di

ogni immagine (augmentation) per creare nuovi esempi di training da cui il nostro modello può imparare.

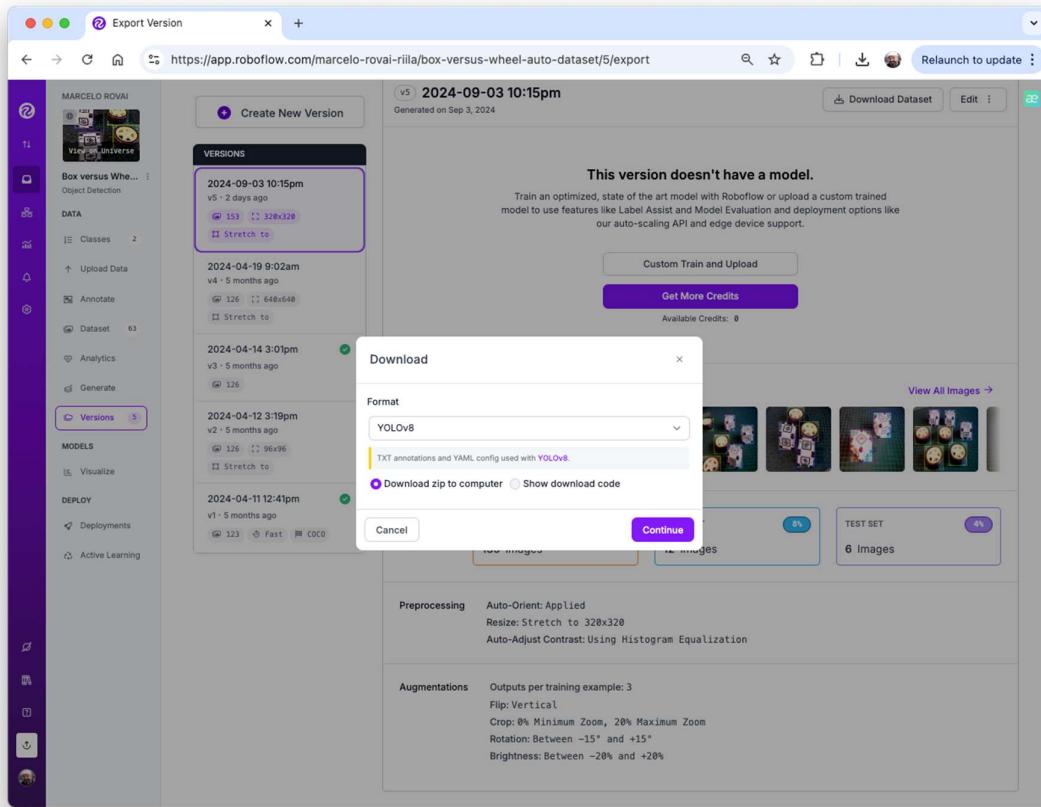
Per l'augmentation, ruoteremo le immagini (+/-15°), ritaglieremo e varieremo la luminosità e l'esposizione.



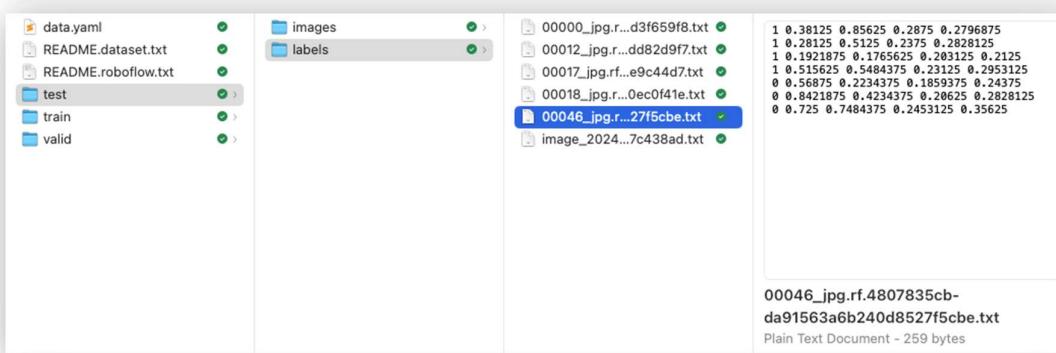
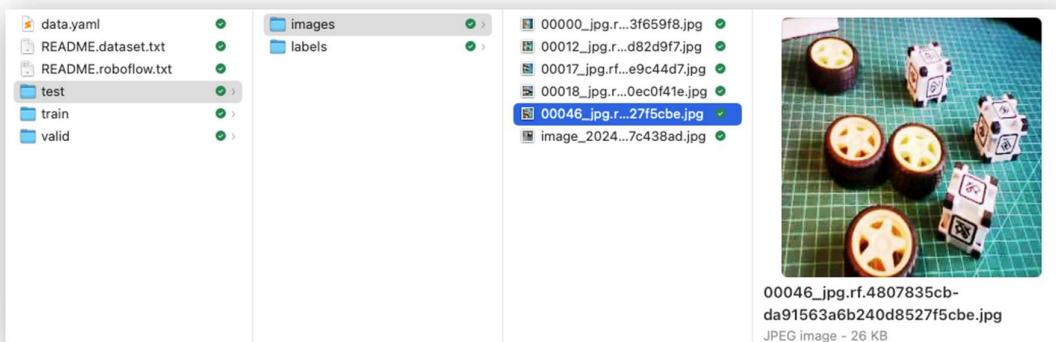
Alla fine del processo avremo 153 immagini.



Ora si deve esportare il dataset annotato in un formato che Edge Impulse, Ultralitics e altri framework/strumenti possano comprendere, ad esempio YOLOv8. Scarichiamo una versione compressa del dataset sul nostro desktop.



Qui è possibile rivedere come è stato strutturato il dataset



Ci sono 3 cartelle separate, una per ogni suddivisione (`train/test/valid`). Per ognuna di esse ci sono 2 sottocartelle, `images` e `labels`. Le immagini sono archiviate come `image_id.jpg` e `image_id.txt`, dove “`image_id`” è univoco per ogni immagine.

Il formato del file delle etichette sarà `class_id coordinate del riquadro di delimitazione`, dove nel nostro caso `class_id` sarà `0` per `box` e `1` per `wheel`. L’ID numerico (0, 1, 2...) seguirà l’ordine alfabetico del nome della classe.

Il file `data.yaml` contiene informazioni sul set di dati come i nomi delle classi (`names: ['box', 'wheel']`) seguendo il formato YOLO.

E questo è tutto! Siamo pronti per iniziare l’addestramento utilizzando Edge Impulse Studio (come faremo nel passaggio successivo), Ultralytics (come faremo quando discuteremo di YOLO) o persino l’addestramento da zero su CoLab (come abbiamo fatto col dataset Cifar-10 nel laboratorio di classificazione delle immagini).

Il dataset pre-elaborato si trova sul [sito Roboflow](#), o qui:

## Addestramento di un Modello SSD MobileNet su Edge Impulse Studio

Si va su [Edge Impulse Studio](#), si inseriscono le proprie credenziali in **Login** (o si crea un account) e si avvia un nuovo progetto.

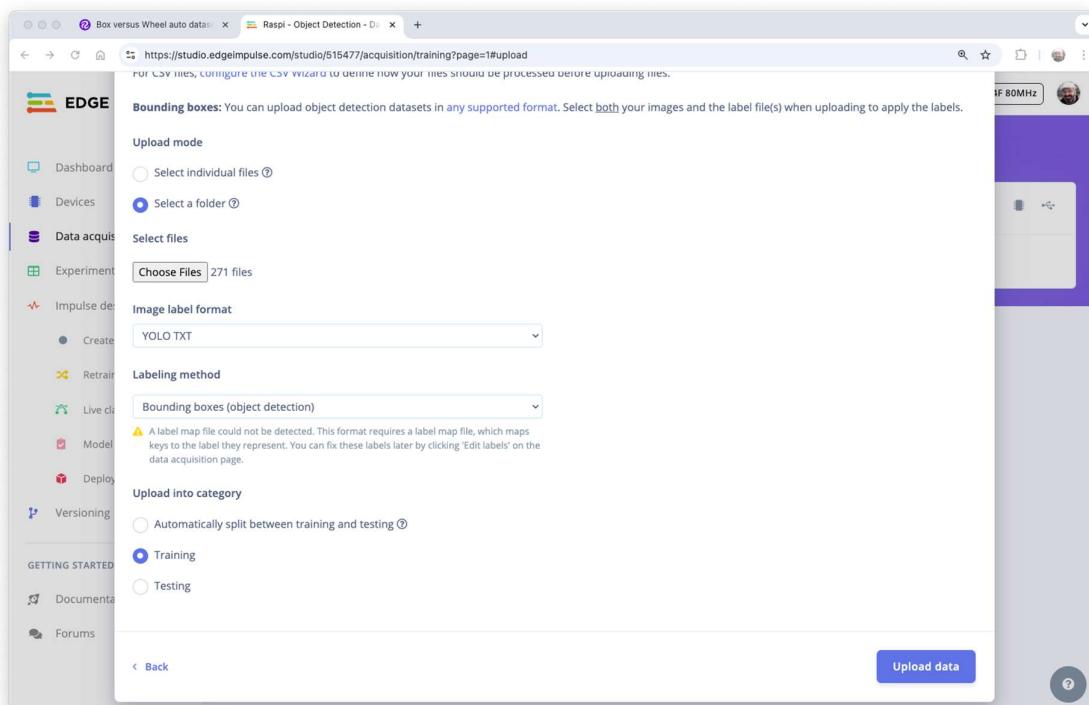
Qui, è possibile clonare il progetto sviluppato per questo laboratorio pratico: [Raspi - Object Detection](#).

Nella scheda `Dashboard` del progetto, si va in basso e su **Project info** e per “Labeling method” si seleziona `Bounding boxes (object detection)`

### **Caricamento dei dati annotati**

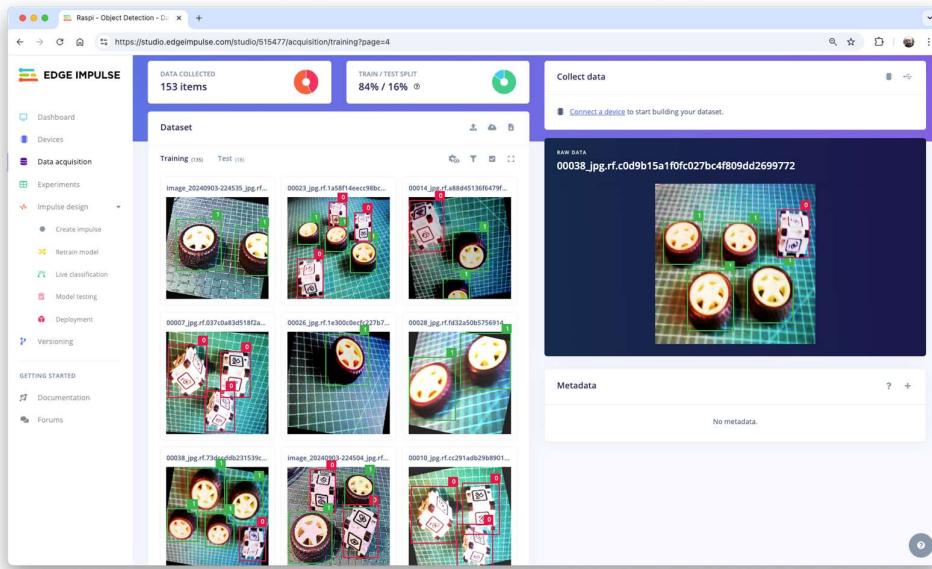
Su Studio, si va alla scheda `Data acquisition` e nella sezione `UPLOAD DATA` si carica dal computer il set di dati non elaborato.

Possiamo usare l’opzione `Select a folder`, scegliendo, ad esempio, la cartella `train` nel computer, che contiene due sottocartelle, `images` e `labels`. Selezionare `Image label format`, “YOLO TXT”, caricare nella categoria `Training` e premere `Upload data`.



Ripetere il processo per i dati di test (caricare entrambe le cartelle, test e convalida). Alla fine del processo di caricamento, si ottiene il set di dati annotato di 153 immagini suddivise in `train/test` (84%/16%).

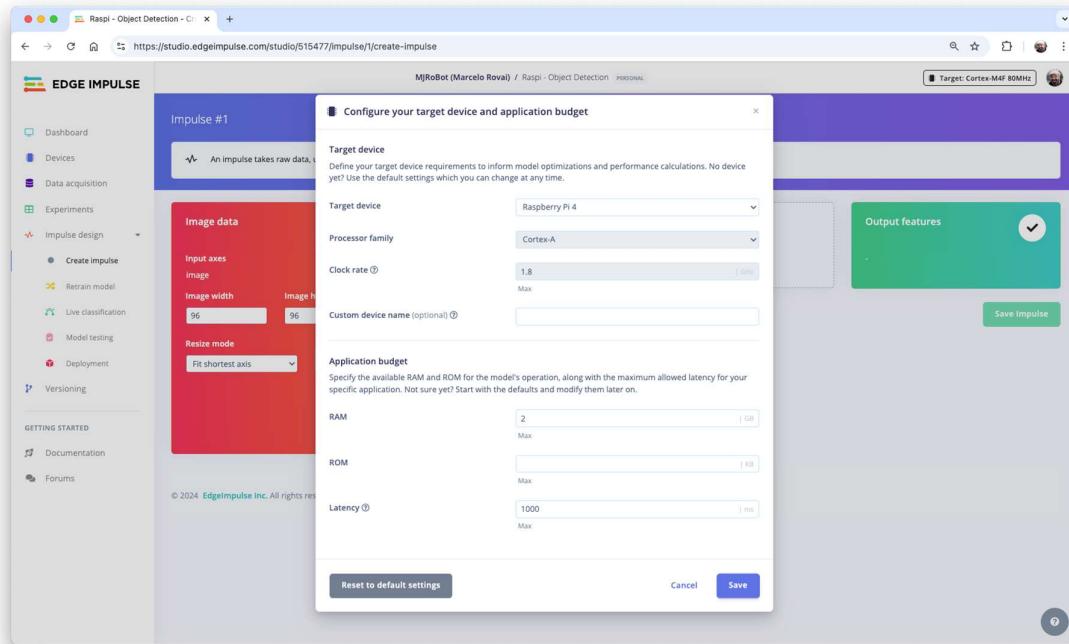
Notare che le etichette saranno archiviate nei file di etichette 0 e 1, che sono equivalenti a `box` e `wheel`.



### **Impulse Design**

La prima cosa da definire quando entriamo nella fase `Create impulse` è descrivere il dispositivo target per la distribuzione. Apparirà una finestra pop-up. Selezioneremo Raspberry 4, un dispositivo intermedio tra Raspi-Zero e Raspi-5.

Questa scelta non interferirà con l'addestramento; ci darà solo un'idea della latenza del modello su quel target specifico.

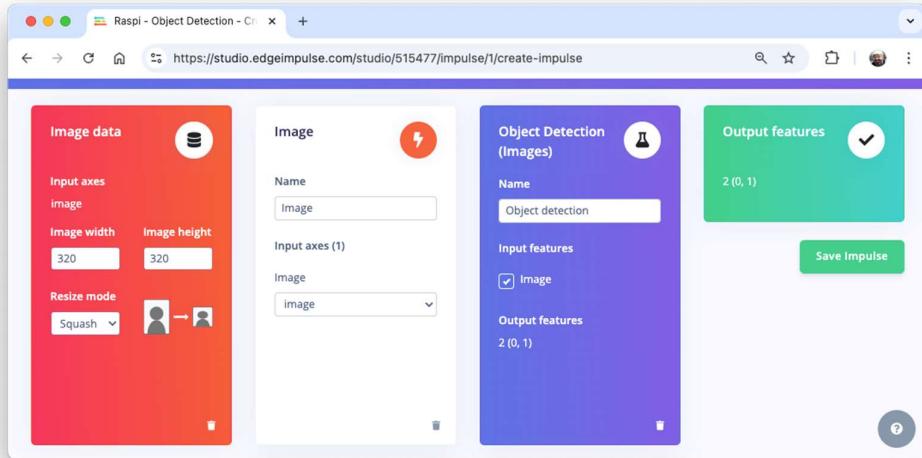


In questa fase, si deve definire come:

- Il **pre-processing** consiste nel ridimensionare le singole immagini. Nel nostro caso, le immagini sono state pre-elaborate su Roboflow, a 320x320, quindi teniamole. La modifica delle dimensioni non avrà importanza qui perché le immagini sono già quadrate. Se si carica un'immagine rettangolare, la si deve schiacciare (forma

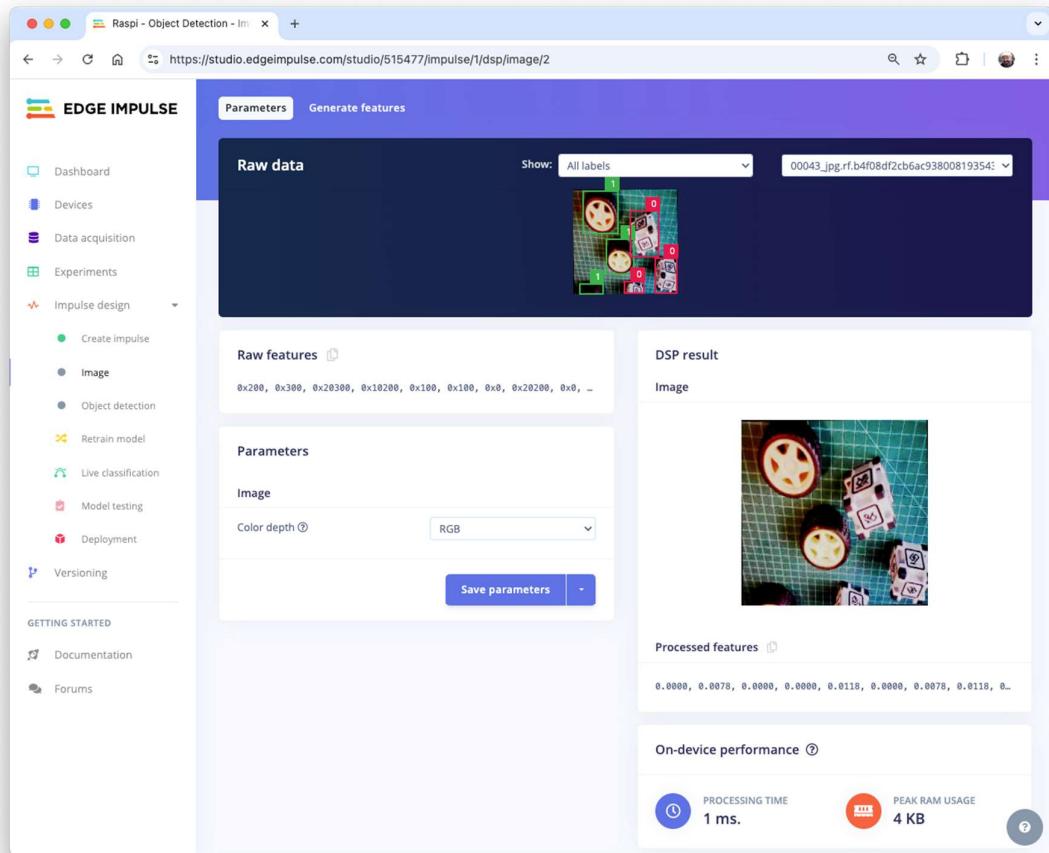
quadrata, senza ritagliarla). In seguito, si potrebbe definire se le immagini vengono convertite da RGB a scala di grigi o meno.

- **Design a Model**, in questo caso, “Object Detection”.

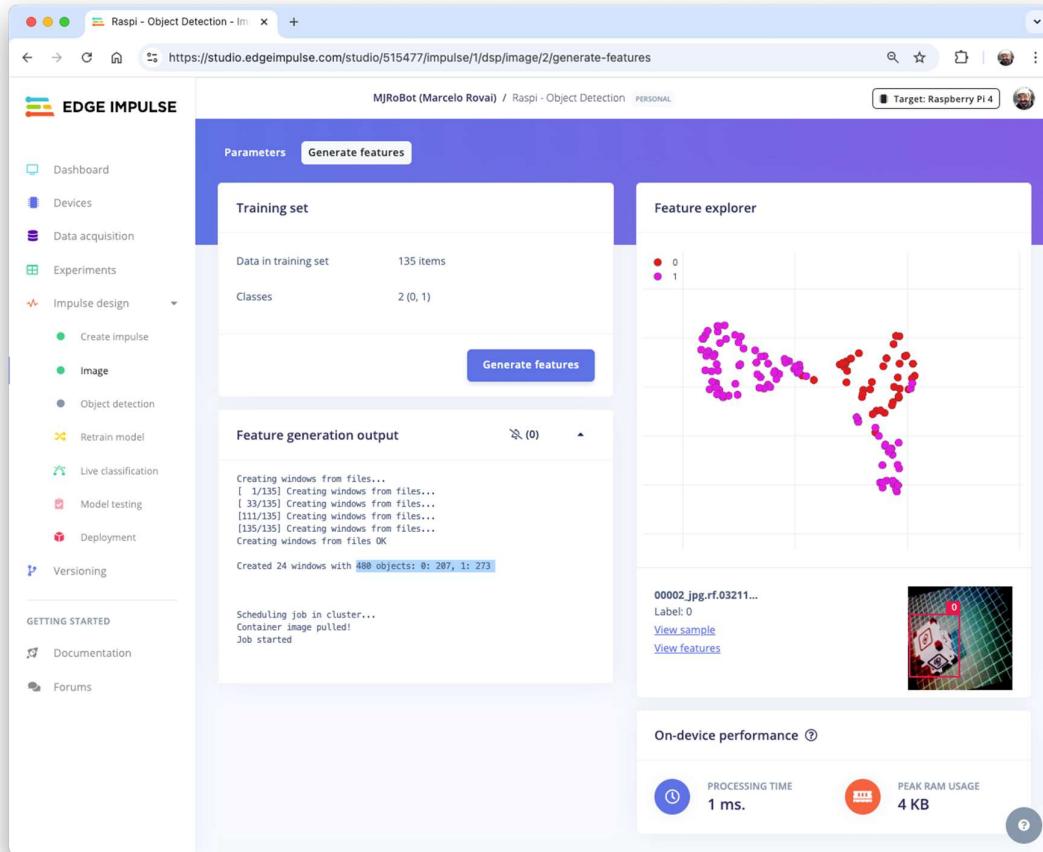


### Pre-elaborazione di tutti i dataset

Nella sezione **Image**, selezionare **Color depth** come **RGB** e premere **Save parameters**.



Lo Studio passa automaticamente alla sezione successiva, **Generate features**, dove tutti i campioni verranno pre-elaborati, ottenendo 480 oggetti: 207 box e 273 ruote.



L'esploratore di feature mostra che tutti i campioni evidenziano una buona separazione dopo la generazione delle feature.

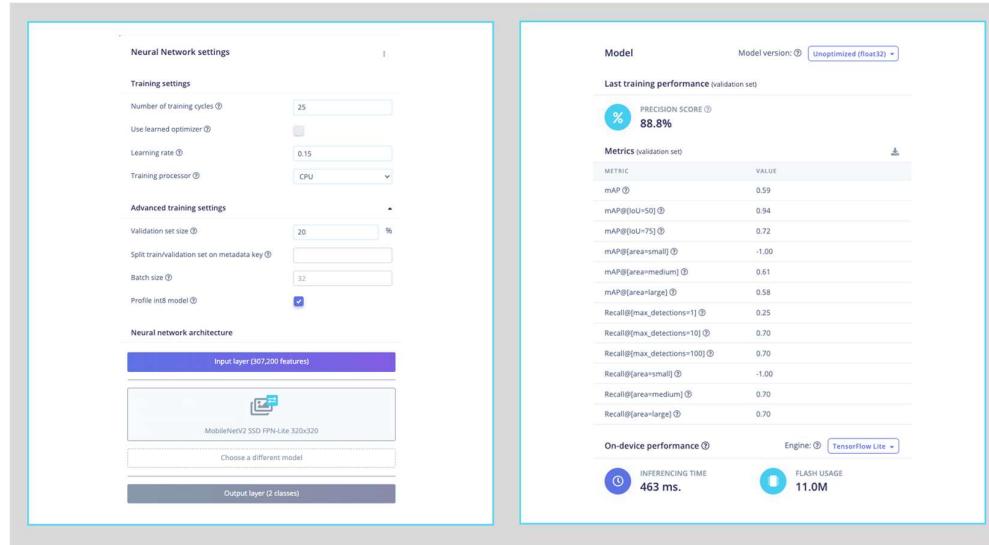
### **Progettazione, Addestramento e Test del Modello**

Per l'addestramento, dovremmo selezionare un modello pre-addestrato. Usiamo **MobileNetV2 SSD FPN-Lite (solo 320x320)**. È un modello di rilevamento oggetti pre-addestrato progettato per individuare fino a 10 oggetti all'interno di un'immagine, generando un riquadro di delimitazione per ogni oggetto rilevato. Il modello è di circa 3,7 MB. Supporta un input RGB a 320x320px.

Per quanto riguarda gli iperparametri di training, il modello verrà addestrato con:

- Epochs: 25
- Batch size: 32
- Learning Rate: 0.15.

Per la convalida durante l'addestramento, il 20% del set di dati (*validation\_dataset*) verrà risparmiato.



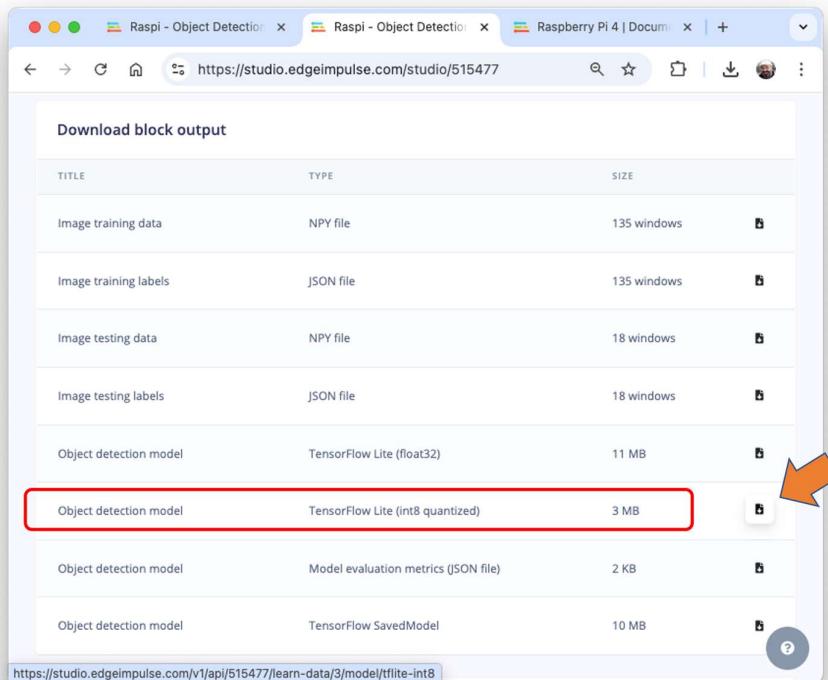
Di conseguenza, il modello termina con un punteggio di precisione complessivo (basato su COCO mAP) dell'88,8%, superiore al risultato ottenuto utilizzando i dati di test (83,3%).

### Distribuzione del modello

Abbiamo due modi per distribuire il modello:

- **Modello TFLite**, che consente di distribuire il modello addestrato come `.tflite` affinché Raspi lo esegua tramite Python.
- **Linux (AARCH64)**, un binario per Linux (AARCH64), implementa il protocollo Edge Impulse Linux, che consente di eseguire i modelli su qualsiasi scheda di sviluppo basata su Linux, ad esempio con SDK per Python. Consulta la documentazione per maggiori informazioni e le [istruzioni di configurazione](#).

Distribuiamo il **modello TFLite**. Nella scheda `Dashboard`, si va su Transfer learning model (int8 quantized) e si clicca sull'icona di download:



Trasferire il modello dal computer alla cartella Raspi `./models` e catturare o ottenere alcune immagini per l'inferenza e salvale nella cartella `./images`.

## Inferenza e Post-Elaborazione

L'inferenza può essere fatta come discusso nella *Panoramica dei Modelli di Rilevamento degli Oggetti Pre-Addestrati*. Iniziamo un nuovo [notebook](#) per seguire tutti i passaggi per rilevare cubi e ruote su un'immagine.

Importare le librerie necessarie:

```
import time
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image
import tensorflow.lite_runtime.interpreter as tflite
```

Definire il percorso del modello e delle etichette:

```
model_path = "./models/ei-raspi-object-detection-SSD-MobileNetv2-320x0320-\
int8.tflite"
labels = ['box', 'wheel']
```

Ricordare che il modello restituirà l'ID della classe come valori (0 e 1), seguendo un ordine alfabetico in base ai nomi delle classi.

Caricare il modello, allocare i tensori e ottenere i dettagli dei tensori di input e output:

```
Load the TFLite model
interpreter = tflite.Interpreter(model_path=model_path)
interpreter.allocate_tensors()

Get input and output tensors
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

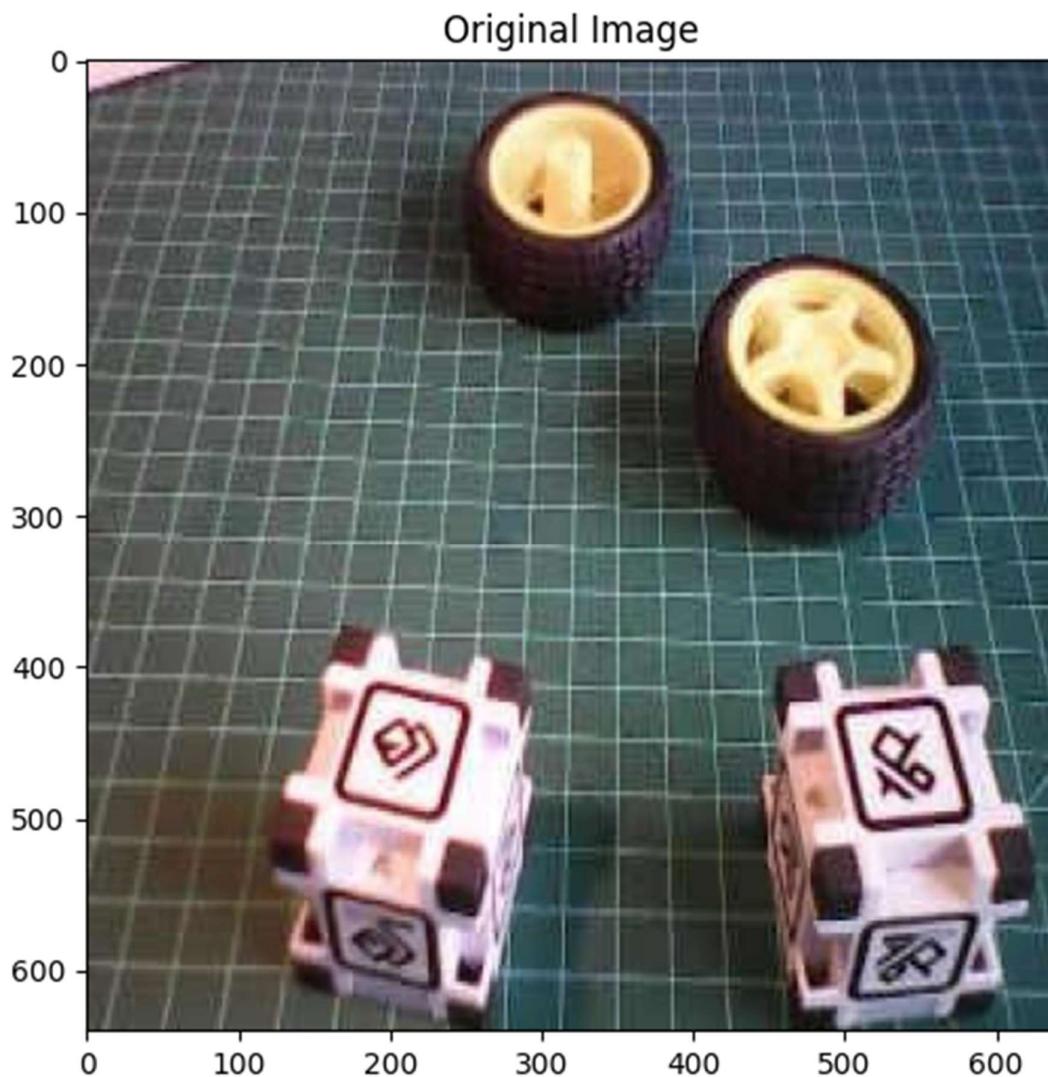
Una differenza fondamentale da notare è che il `dtype` dei dettagli di input del modello è ora `int8`, il che significa che i valori di input vanno da -128 a +127, mentre ogni pixel della nostra immagine raw va da 0 a 256. Ciò significa che dovremmo pre-elaborare l'immagine per farla corrispondere. Possiamo controllare qui:

```
input_dtype = input_details[0]['dtype']
input_dtype
numpy.int8
```

Quindi, apriamo l'immagine e mostriamola:

```
Load the image
img_path = "./images/box_2_wheel_2.jpg"
orig_img = Image.open(img_path)

Display the image
plt.figure(figsize=(6, 6))
plt.imshow(orig_img)
plt.title("Original Image")
plt.show()
```



Ed eseguiamo la pre-elaborazione:

```
scale, zero_point = input_details[0]['quantization']
img = orig_img.resize((input_details[0]['shape'][1],
 input_details[0]['shape'][2]))
img_array = np.array(img, dtype=np.float32) / 255.0
img_array = (img_array / scale + zero_point).clip(-128, 127).astype(np.int8)
input_data = np.expand_dims(img_array, axis=0)
```

Controllando i dati di input, possiamo verificare che il tensore di input è compatibile con quanto previsto dal modello:

```
input_data.shape, input_data.dtype
((1, 320, 320, 3), dtype('int8'))
```

Adesso è il momento di effettuare l'inferenza. Calcoliamo anche la latenza del modello:

```
Inference on Raspi-Zero
start_time = time.time()
interpreter.set_tensor(input_details[0]['index'], input_data)
interpreter.invoke()
end_time = time.time()
inference_time = (end_time - start_time) * 1000 # Convert to milliseconds
print ("Inference time: {:.1f}ms".format(inference_time))
```

Il modello impiegherà circa 600 ms per eseguire l'inferenza nel Raspi-Zero, che è circa 5 volte più lungo di un Raspi-5.

Ora possiamo ottenere le classi di output degli oggetti rilevati, le coordinate dei suoi bounding box e le probabilità.

```
boxes = interpreter.get_tensor(output_details[1]['index'])[0]
classes = interpreter.get_tensor(output_details[3]['index'])[0]
```

```

scores = interpreter.get_tensor(output_details[0]['index'])[0]
num_detections = int(interpreter.get_tensor(output_details[2]['index'])[0])
for i in range(num_detections):
 if scores[i] > 0.5: # Confidence threshold
 print(f"Object {i}:")
 print(f" Bounding Box: {boxes[i]}")
 print(f" Confidence: {scores[i]}")
 print(f" Class: {classes[i]}")

Object 0:
 Bounding Box: [0.01461247 0.38439587 0.2793928 0.62159896]
 Confidence: 0.86328125
 Class: 1.0
Object 1:
 Bounding Box: [0.19234724 0.6176628 0.5012042 0.888332]
 Confidence: 0.86328125
 Class: 1.0
Object 2:
 Bounding Box: [0.5792029 0.19102246 0.9971932 0.47538966]
 Confidence: 0.7734375
 Class: 0.0
Object 3:
 Bounding Box: [0.5792029 0.68904555 0.9971932 0.97973716]
 Confidence: 0.6484375
 Class: 0.0

```

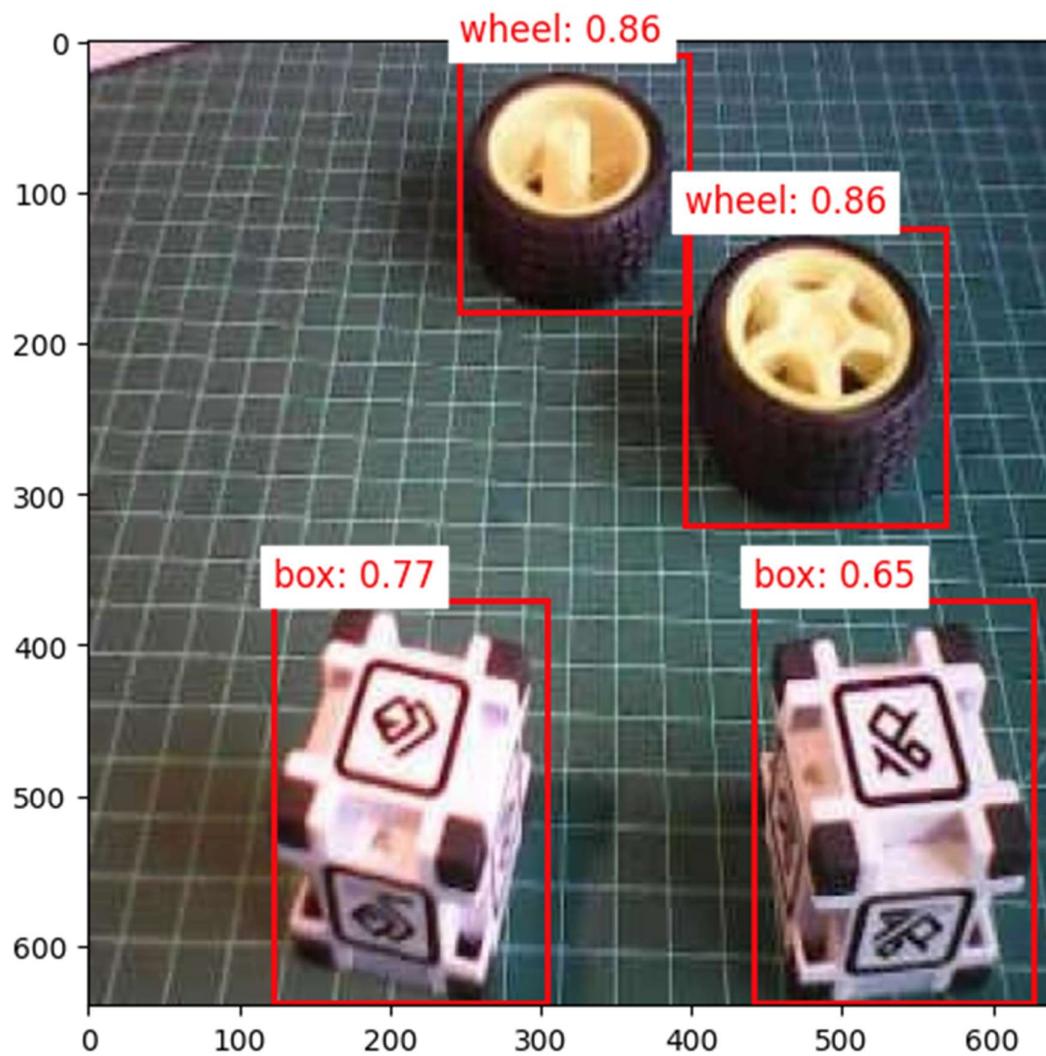
Dai risultati, possiamo vedere che sono stati rilevati 4 oggetti: due con ID classe 0 (box) e due con ID classe 1 (wheel), cosa è corretto!

Visualizziamo il risultato per un `threshold` [soglia] di 0,5

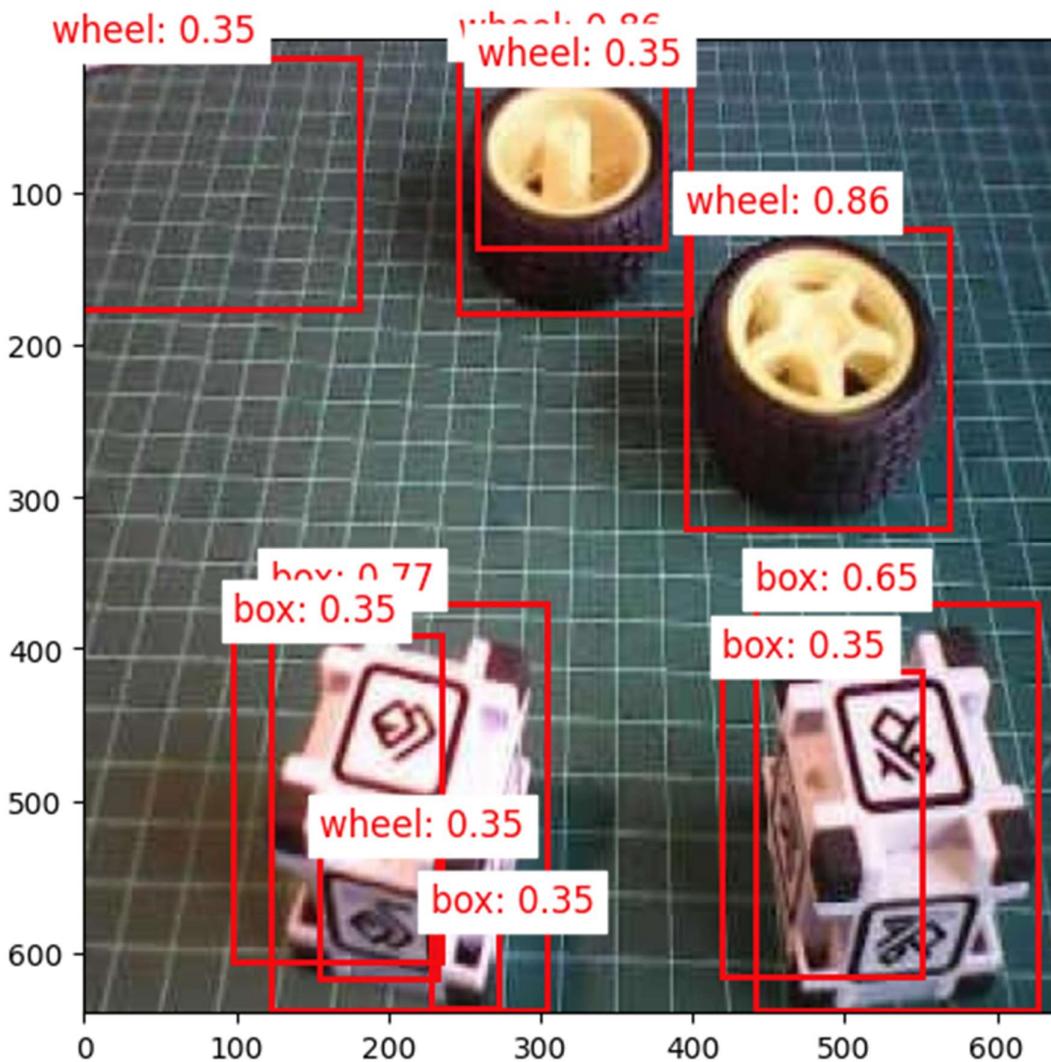
```

threshold = 0.5
plt.figure(figsize=(6,6))
plt.imshow(orig_img)
for i in range(num_detections):
 if scores[i] > threshold:
 ymin, xmin, ymax, xmax = boxes[i]
 (left, right, top, bottom) = (xmin * orig_img.width,
 xmax * orig_img.width,
 ymin * orig_img.height,
 ymax * orig_img.height)
 rect = plt.Rectangle((left, top), right-left, bottom-top,
 fill=False, color='red', linewidth=2)
 plt.gca().add_patch(rect)
 class_id = int(classes[i])
 class_name = labels[class_id]
 plt.text(left, top-10, f'{class_name}: {scores[i]:.2f}',
 color='red', fontsize=12, backgroundcolor='white')

```



Ma cosa succede se riduciamo la soglia a 0.3, ad esempio?



Cominciamo a vedere falsi positivi e **rilevazioni multiple**, in cui il modello rileva lo stesso oggetto più volte con diversi livelli di confidenza e bounding box leggermente diversi. Di solito, a volte, dobbiamo regolare la soglia su valori più piccoli per catturare tutti gli oggetti, evitando falsi negativi, che porterebbero a rilevazioni multiple.

Per migliorare i risultati di rilevamento, dovremmo implementare **Non-Maximum Suppression (NMS)**, che aiuta a eliminare le bounding box sovrapposte e mantiene solo il rilevamento più affidabile.

Per questo, creiamo una funzione generale denominata `non_max_suppression()`, con il ruolo di perfezionare i risultati di rilevamento degli oggetti eliminando le bounding box ridondanti e sovrapposte. Ciò è possibile selezionando in modo iterativo la rilevazione con il punteggio di confidenza più elevato e rimuovendo altre rilevazioni significativamente sovrapposte in base a una soglia di “Intersection over Union (IoU)” [intersezione su unione].

```
def non_max_suppression(boxes, scores, threshold):
 # Convert to corner coordinates
 x1 = boxes[:, 0]
 y1 = boxes[:, 1]
 x2 = boxes[:, 2]
 y2 = boxes[:, 3]

 areas = (x2 - x1 + 1) * (y2 - y1 + 1)
 order = scores.argsort() [::-1]

 keep = []
 while order.size > 0:
 i = order[0]
```

```

 keep.append(i)
 xx1 = np.maximum(x1[i], x1[order[1:]])
 yy1 = np.maximum(y1[i], y1[order[1:]])
 xx2 = np.minimum(x2[i], x2[order[1:]])
 yy2 = np.minimum(y2[i], y2[order[1:]])

 w = np.maximum(0.0, xx2 - xx1 + 1)
 h = np.maximum(0.0, yy2 - yy1 + 1)
 inter = w * h
 ovr = inter / (areas[i] + areas[order[1:]] - inter)

 inds = np.where.ovr <= threshold)[0]
 order = order[inds + 1]

 return keep

```

Come funziona:

1. Sorting: Inizia ordinando tutte le rilevazioni in base ai loro punteggi di confidenza, dal più alto al più basso.
2. Selection: Seleziona la casella con il punteggio più alto e la aggiunge all'elenco finale delle rilevazioni.
3. Comparison: Questa casella selezionata viene confrontata con tutte le restanti caselle con punteggio più basso.
4. Elimination: Qualsiasi casella che si sovrapponga in modo significativo (oltre la soglia IoU) alla casella selezionata viene eliminata.
5. Iteration: Questo processo si ripete con la casella con il punteggio più alto successivo finché tutte le caselle non vengono elaborate.

Ora, possiamo definire una funzione di visualizzazione più precisa che prenderà in considerazione una soglia IoU, rilevando solo gli oggetti selezionati dalla funzione

`non_max_suppression`:

```

def visualize_detections(image, boxes, classes, scores,
 labels, threshhold, iou_threshold):
 if isinstance(image, Image.Image):
 image_np = np.array(image)
 else:
 image_np = image

 height, width = image_np.shape[:2]

 # Convert normalized coordinates to pixel coordinates
 boxes_pixel = boxes * np.array([height, width, height, width])

 # Apply NMS
 keep = non_max_suppression(boxes_pixel, scores, iou_threshold)

 # Set the figure size to 12x8 inches
 fig, ax = plt.subplots(1, figsize=(12, 8))

 ax.imshow(image_np)

 for i in keep:
 if scores[i] > threshhold:
 ymin, xmin, ymax, xmax = boxes[i]
 rect = patches.Rectangle((xmin * width, ymin * height),
 (xmax - xmin) * width,
 (ymax - ymin) * height,
 linewidth=2, edgecolor='r', facecolor='none')
 ax.add_patch(rect)
 class_name = labels[int(classes[i])]
 ax.text(xmin * width, ymin * height - 10,
 f'{class_name}: {scores[i]:.2f}', color='red',
 fontsize=12, backgroundcolor='white')

 plt.show()

```

Ora possiamo creare una funzione che chiamerà le altre, eseguendo l'inferenza su qualsiasi immagine:

```

def detect_objects(img_path, conf=0.5, iou=0.5):
 orig_img = Image.open(img_path)
 scale, zero_point = input_details[0]['quantization']
 img = orig_img.resize((input_details[0]['shape'][1],
 input_details[0]['shape'][2]))
 img_array = np.array(img, dtype=np.float32) / 255.0
 img_array = (img_array / scale + zero_point).clip(-128, 127).\
 astype(np.int8)
 input_data = np.expand_dims(img_array, axis=0)

 # Inference on Raspi-Zero
 start_time = time.time()
 interpreter.set_tensor(input_details[0]['index'], input_data)
 interpreter.invoke()
 end_time = time.time()
 inference_time = (end_time - start_time) * 1000 # Convert to ms
 print ("Inference time: {:.1f}ms".format(inference_time))

 # Extract the outputs
 boxes = interpreter.get_tensor(output_details[1]['index'])[0]
 classes = interpreter.get_tensor(output_details[3]['index'])[0]
 scores = interpreter.get_tensor(output_details[0]['index'])[0]
 num_detections = int(interpreter.get_tensor(output_details[2]['index'])[0])

 visualize_detections(orig_img, boxes, classes, scores, labels,
 threshold=conf,
 iou_threshold=iou)

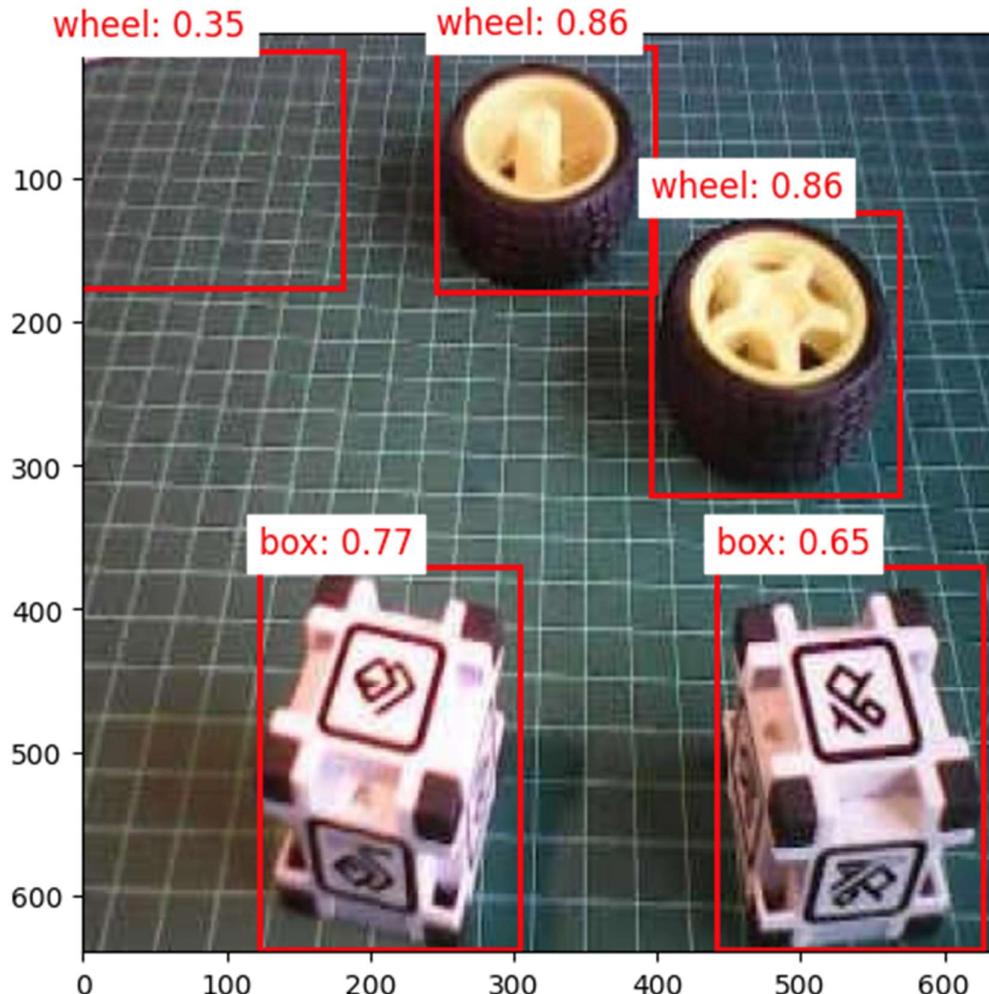
```

Ora, eseguendo il codice, ottenendo di nuovo la stessa immagine con una soglia di confidenza di 0.3, ma con un piccolo IoU:

```

img_path = "./images/box_2_wheel_2.jpg"
detect_objects(img_path, conf=0.3,iou=0.05)

```



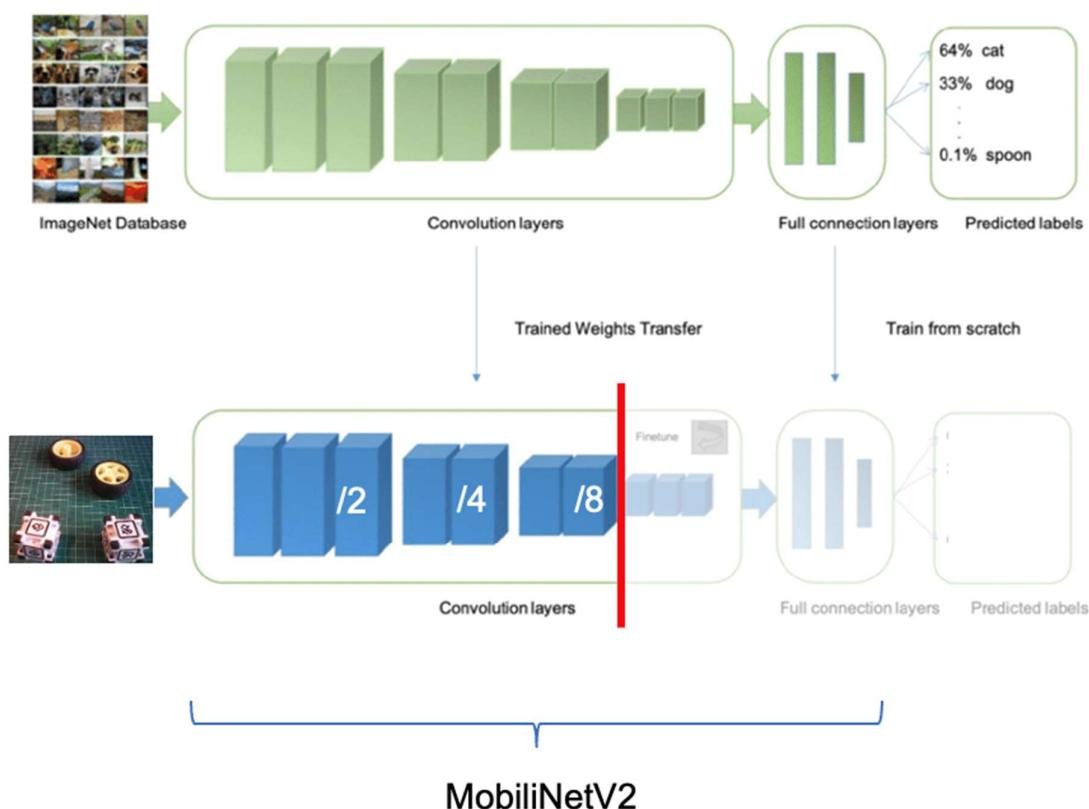
## Training di un modello FOMO su Edge Impulse Studio

L'inferenza con il modello SSD MobileNet ha funzionato bene, ma la latenza era significativamente alta. L'inferenza variava da 0.5 a 1.3 secondi su un Raspi-Zero, il che significa circa o meno di 1 FPS (1 frame al secondo). Un'alternativa per accelerare il processo è usare FOMO (Faster Objects, More Objects).

Questo nuovo algoritmo di apprendimento automatico consente di contare più oggetti e di trovare la loro posizione in un'immagine in tempo reale utilizzando fino a 30 volte meno potenza di elaborazione e memoria rispetto a MobileNet SSD o YOLO. Il motivo principale per cui ciò è possibile è che mentre altri modelli calcolano le dimensioni dell'oggetto disegnando un quadrato attorno ad esso (bounding box), FOMO ignora le dimensioni dell'immagine, fornendo solo le informazioni sulla posizione dell'oggetto nell'immagine tramite le sue coordinate del centroide.

### Come funziona FOMO?

In una tipica pipeline di rilevamento degli oggetti, la prima fase consiste nell'estrare le feature dall'immagine di input. **FOMO sfrutta MobileNetV2 per eseguire questa attività.** MobileNetV2 elabora l'immagine di input per produrre una feature map che cattura le caratteristiche essenziali, come texture, forme e bordi degli oggetti, in modo computazionalmente efficiente.

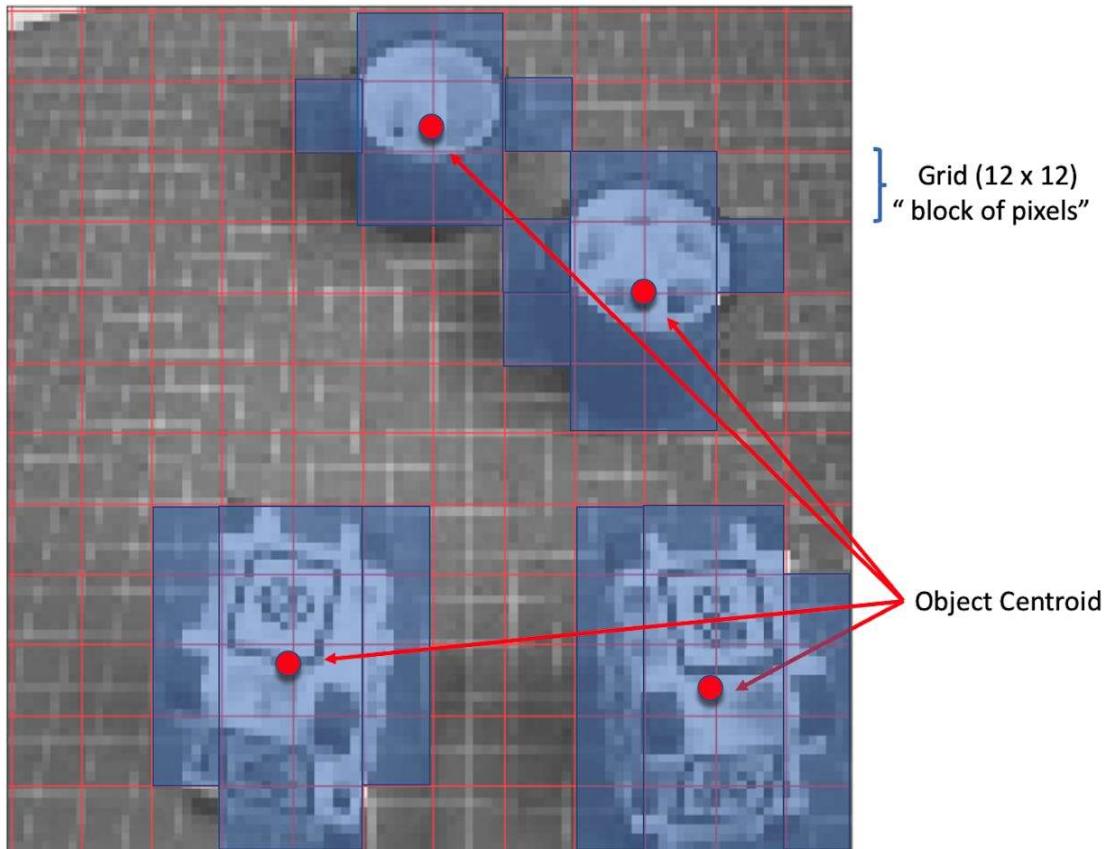


Una volta estratte queste feature, l'architettura più semplice di FOMO, focalizzata sul rilevamento del punto centrale, interpreta la feature map per determinare dove si trovano gli oggetti nell'immagine. L'output è una griglia di celle, in cui ogni cella rappresenta se è stato rilevato o meno un centro dell'oggetto. Il modello restituisce uno o più punteggi di confidenza per ogni cella, indicando la probabilità che un oggetto sia presente.

Vediamo come funziona su un'immagine.

FOMO divide l'immagine in blocchi di pixel usando un fattore di 8. Per l'input di 96x96, la griglia è 12x12 ( $96/8=12$ ). Per un 160x160, la griglia sarà 20x20 e così via. Successivamente,

FOMO eseguirà un classificatore attraverso ogni blocco di pixel per calcolare la probabilità che ci sia un box o una ruota in ognuno di essi e, successivamente, determinerà le regioni che hanno la più alta probabilità di contenere l'oggetto (se un blocco di pixel non ha oggetti, verrà classificato come *background*). Dalla sovrapposizione della regione finale, FOMO fornisce le coordinate (relative alle dimensioni dell'immagine) del centroide di questa regione.

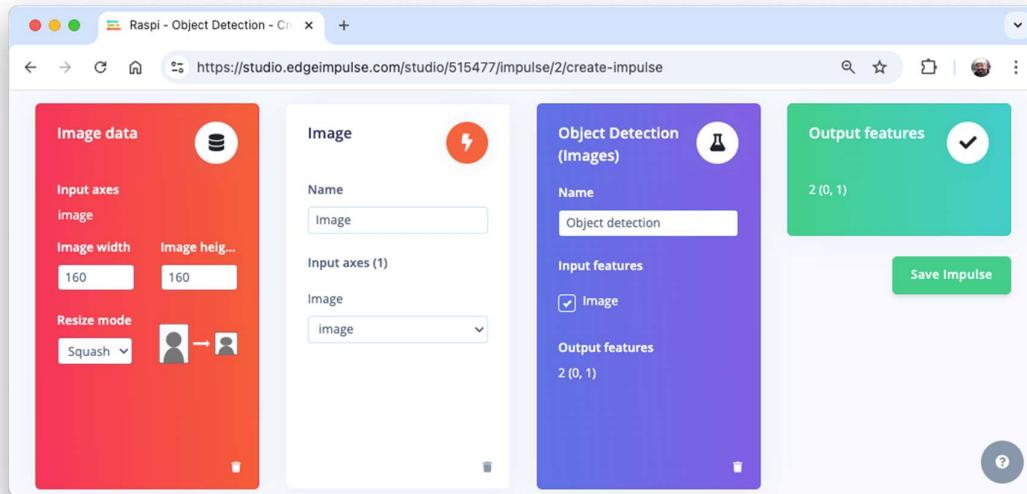


#### Compromesso Tra Velocità e Precisione:

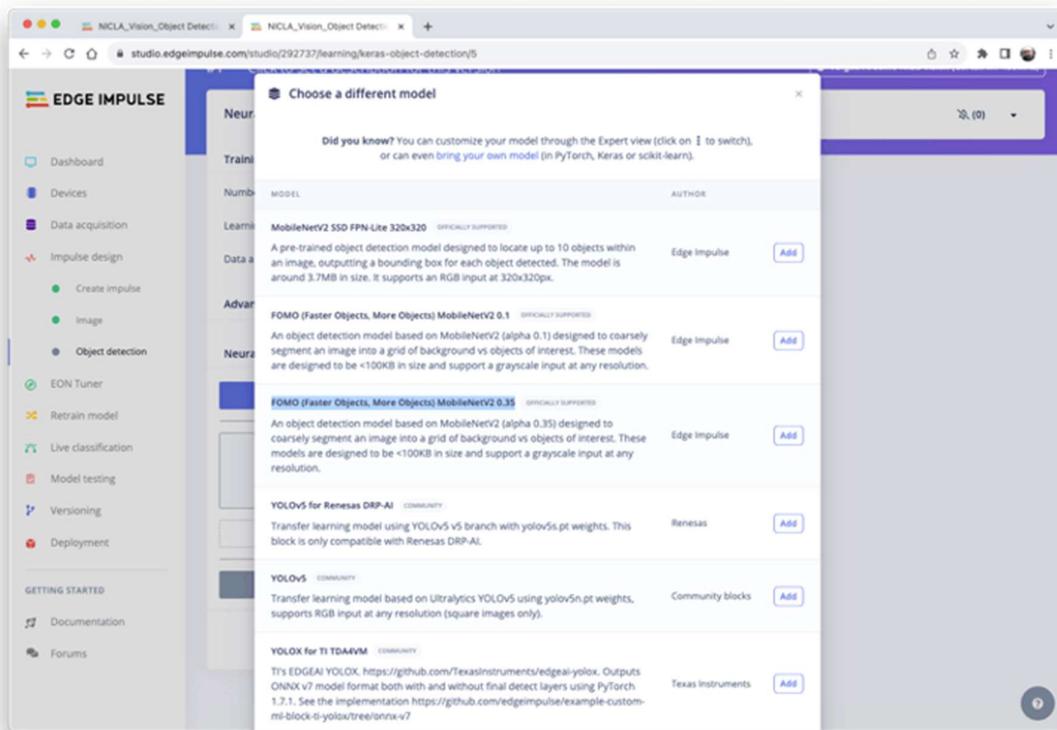
- **Risoluzione della Griglia:** FOMO utilizza una griglia di risoluzione fissa, il che significa che ogni cella può rilevare se un oggetto è presente in quella parte dell'immagine. Sebbene non fornisca un'elevata precisione di localizzazione, fa un compromesso essendo veloce e computazionalmente leggero, il che è fondamentale per i dispositivi edge.
- **Rilevamento Multi-Oggetto:** Poiché ogni cella è indipendente, FOMO può rilevare più oggetti contemporaneamente in un'immagine identificando più centri.

#### ***Impulse Design, nuovo Training e Test***

Tornare a Edge Impulse Studio e nella scheda **Experiments**, creare un altro impulso. Ora, le immagini di input dovrebbero essere 160x160 (questa è la dimensione di input prevista per MobilenetV2).



Nella scheda `Image`, generare le feature e spostarsi nella scheda `Object detection`. Dovremo selezionare un modello pre-addestrato per l'addestramento. Usiamo **FOMO (Faster Objects, More Objects) MobileNetV2 0.35**.

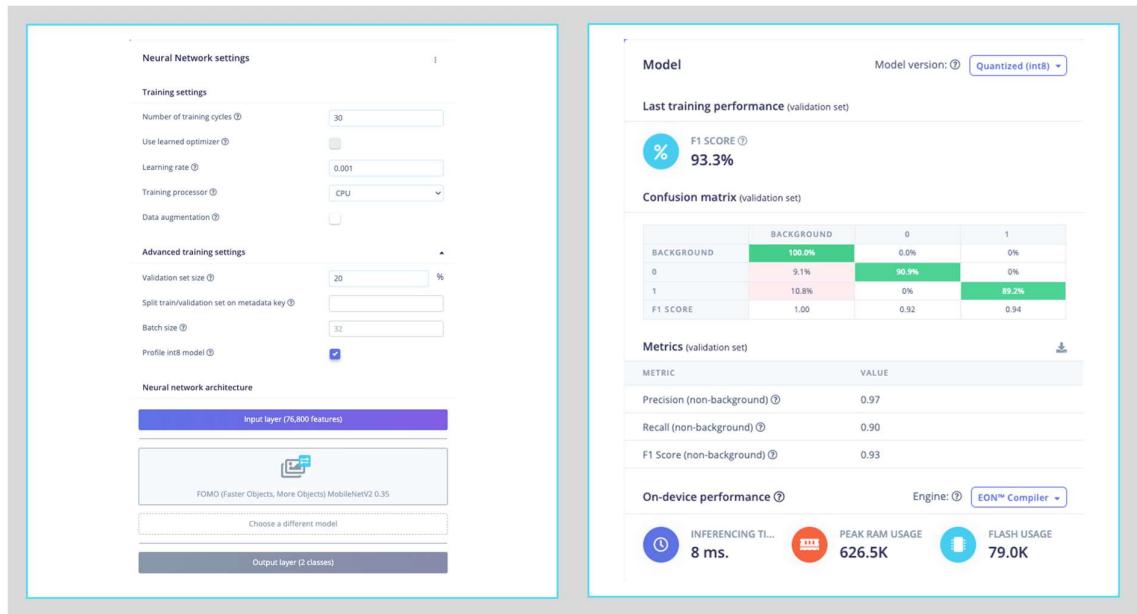


Per quanto riguarda gli iperparametri di training, il modello verrà addestrato con:

- Epochs: 30
- Batch size: 32
- Learning Rate: 0.001.

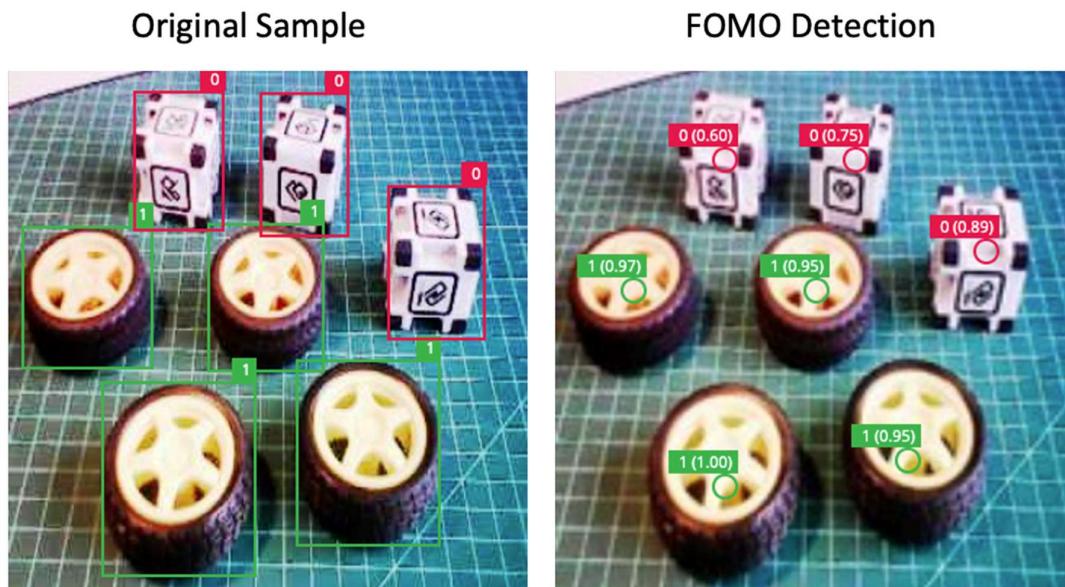
Per la convalida durante l'addestramento, il 20% del set di dati (`validation_dataset`) verrà risparmiato. Non applicheremo il “Data Augmentation” per il restante 80% (`train_dataset`) perché il dataset è già stato aumentato durante la fase di etichettatura su Roboflow.

Di conseguenza, il modello termina con un punteggio F1 complessivo del 93,3% con una latenza impressionante di 8 ms (Raspi-4), circa 60 volte inferiore a quella ottenuta con SSD MobileNetV2.



Notare che FOMO ha aggiunto automaticamente una terza etichetta di background ai due box (0) e ruote (1) definite in precedenza.

Nella scheda **Model testing**, possiamo vedere che l'accuratezza era del 94%. Ecco uno dei risultati del campione di test:



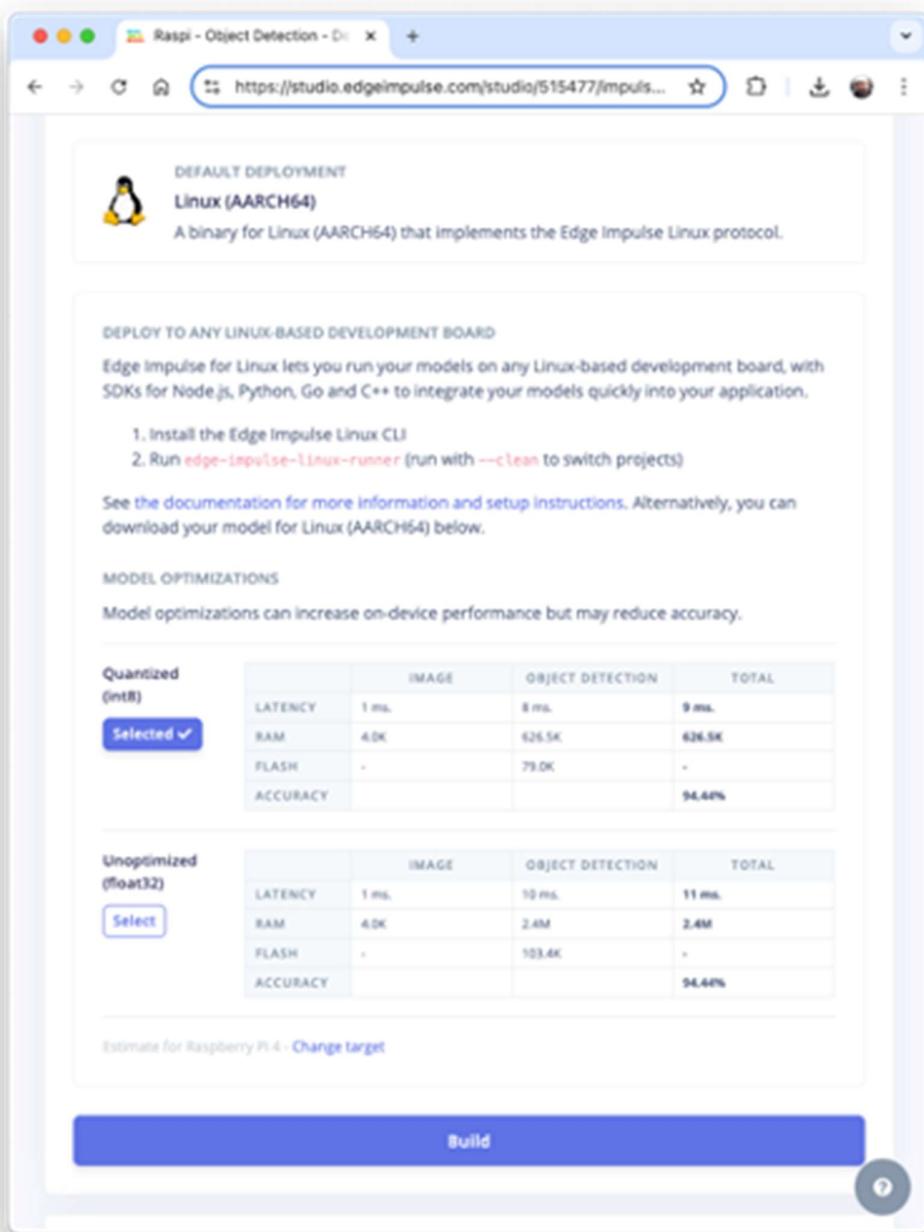
Nelle attività di rilevamento di oggetti, l'accuratezza non è in genere la **metrica di valutazione primaria**. Il rilevamento di oggetti comporta la classificazione degli oggetti e la definizione di riquadri di delimitazione attorno a essi, il che lo rende un problema più complesso della semplice classificazione. Il problema è che non abbiamo il riquadro di delimitazione, solo i centroidi. In breve, usare l'accuratezza come metrica potrebbe essere fuorviante e potrebbe non fornire una comprensione completa delle prestazioni del modello.

### Distribuzione del modello

Come abbiamo fatto nella sezione precedente, possiamo distribuire il modello addestrato come TFLite o Linux (AARCH64). Ora facciamolo come **Linux (AARCH64)**, un binario che implementa il protocollo [Edge Impulse Linux](#).

Edge Impulse per i modelli Linux viene fornito in formato `.eim`. Questo [eseguibile](#) contiene il nostro “impulso completo” creato in Edge Impulse Studio. L’impulso è costituito dai blocchi di elaborazione del segnale e da qualsiasi blocco di apprendimento e anomalia che abbiamo aggiunto e addestrato. È compilato con ottimizzazioni per il nostro processore o GPU (ad esempio, istruzioni NEON su core ARM), più un semplice layer IPC (su un socket Unix).

Nella scheda Deploy, selezionare l’opzione `Linux (AARCH64)`, il modello `int8` e premere `Build`.



Il modello verrà scaricato automaticamente sul computer.

Sul nostro Raspi, creiamo una nuova area di lavoro:

```
cd ~
cd Documents
mkdir EI_Linux
```

```
cd EI_Linux
mkdir models
mkdir images
```

Rinominare il modello per facilitarne l'identificazione:

Ad esempio, `raspi-object-detection-linux-aarch64-FOMO-int8.eim` e trasferirlo nella nuova cartella Raspi `./models` e catturare o ottenere alcune immagini per l'inferenza e salvarle nella cartella `./images`.

### **Inferenza e Post-Elaborazione**

L'inferenza verrà effettuata utilizzando [Linux Python SDK](#). Questa libreria consente di eseguire modelli di apprendimento automatico e raccogliere dati dei sensori su macchine [Linux](#) utilizzando Python. L'SDK è open source e ospitato su GitHub: [edgeimpulse/linux-sdk-python](#).

Impostiamo un ambiente virtuale per lavorare con Linux Python SDK

```
python3 -m venv ~/eilinx
source ~/eilinx/bin/activate
```

E installare tutte le librerie necessarie:

```
sudo apt-get update
sudo apt-get install libatlas-base-dev libportaudio0 libportaudio2
sudo apt-get install libportaudiocpp0 portaudio19-dev

pip3 install edge_impulse_linux -i https://pypi.python.org/simple
pip3 install Pillow matplotlib pyaudio opencv-contrib-python

sudo apt-get install portaudio19-dev
pip3 install pyaudio
pip3 install opencv-contrib-python
```

Consentire al modello di essere eseguibile.

```
chmod +x raspi-object-detection-linux-aarch64-FOMO-int8.eim
```

Installare Jupiter Notebook nel nuovo ambiente

```
pip3 install jupyter
```

Eseguire un notebook in locale (su Raspi-4 o 5 con desktop)

```
jupyter notebook
```

o sul browser del computer:

```
jupyter notebook --ip=192.168.4.210 --no-browser
```

Avviamo un nuovo [notebook](#) seguendo tutti i passaggi per rilevare cubi e ruote su un'immagine utilizzando il modello FOMO e l'SDK Python di Edge Impulse Linux.

Importare le librerie necessarie:

```
import sys, time
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image
import cv2
from edge_impulse_linux.image import ImageImpulseRunner
```

Definire il percorso del modello e delle etichette:

```
model_file = "raspi-object-detection-linux-aarch64-int8.eim"
model_path = "models/" + model_file # Trained ML model from Edge Impulse
labels = ['box', 'wheel']
```

Ricordare che il modello restituirà l'ID della classe come valori (0 e 1), seguendo un ordine alfabetico in base ai nomi delle classi.

Caricare e inizializzare il modello:

```
Load the model file
runner = ImageImpulseRunner(model_path)

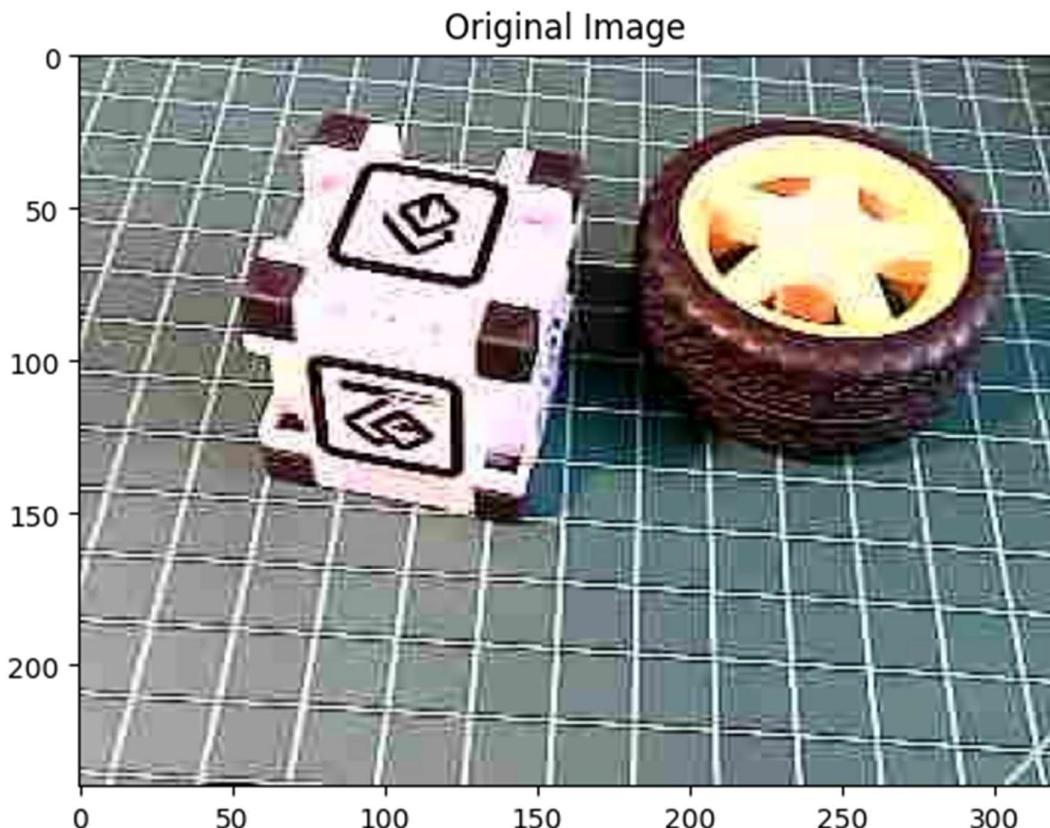
Initialize model
model_info = runner.init()
```

Il `model_info` conterrà informazioni critiche sul modello. Tuttavia, a differenza dell'interprete TFLite, la libreria EI Linux Python SDK preparerà ora il modello per l'inferenza.

Quindi, apriamo l'immagine e mostriamola (ora, per compatibilità, useremo OpenCV, la libreria CV utilizzata internamente da EI. OpenCV legge l'immagine come BGR, quindi dovremo convertirla in RGB):

```
Load the image
img_path = "./images/1_box_1_wheel.jpg"
orig_img = cv2.imread(img_path)
img_rgb = cv2.cvtColor(orig_img, cv2.COLOR_BGR2RGB)

Display the image
plt.imshow(img_rgb)
plt.title("Original Image")
plt.show()
```



Ora otterremo le feature e l'immagine preelaborata (ritagliata) utilizzando `runner`:

```
features, cropped = runner.get_features_from_image_auto_studio_settings(img_rgb)
```

Ed eseguiamo l'inferenza. Calcoliamo anche la latenza del modello:

```
res = runner.classify(features)
```

Otteniamo le classi di output degli oggetti rilevati, i loro centroidi dei bounding box e le probabilità.

```
print('Found %d bounding boxes (%d ms.)' % (
 len(res["result"]["bounding_boxes"]),
 res['timing']['dsp'] + res['timing']['classification']))
for bb in res["result"]["bounding_boxes"]:
 print('\t%.2f: x=%d y=%d w=%d h=%d' % (
 bb['label'], bb['value'], bb['x'],
 bb['y'], bb['width'], bb['height']))
Found 2 bounding boxes (29 ms.)
 1 (0.91): x=112 y=40 w=16 h=16
 0 (0.75): x=48 y=56 w=8 h=8
```

I risultati mostrano che sono stati rilevati due oggetti: uno con ID classe 0 (`box`) e uno con ID classe 1 (`wheel`), il che è corretto!

Visualizziamo il risultato (il `threshold` [soglia] è 0.5, il valore predefinito impostato durante il test del modello su Edge Impulse Studio).

```
print('\tFound %d bounding boxes (latency: %d ms.)' % (
 len(res["result"]["bounding_boxes"])),
```

```

res['timing']['dsp'] + res['timing']['classification']))
plt.figure(figsize=(5,5))
plt.imshow(cropped)

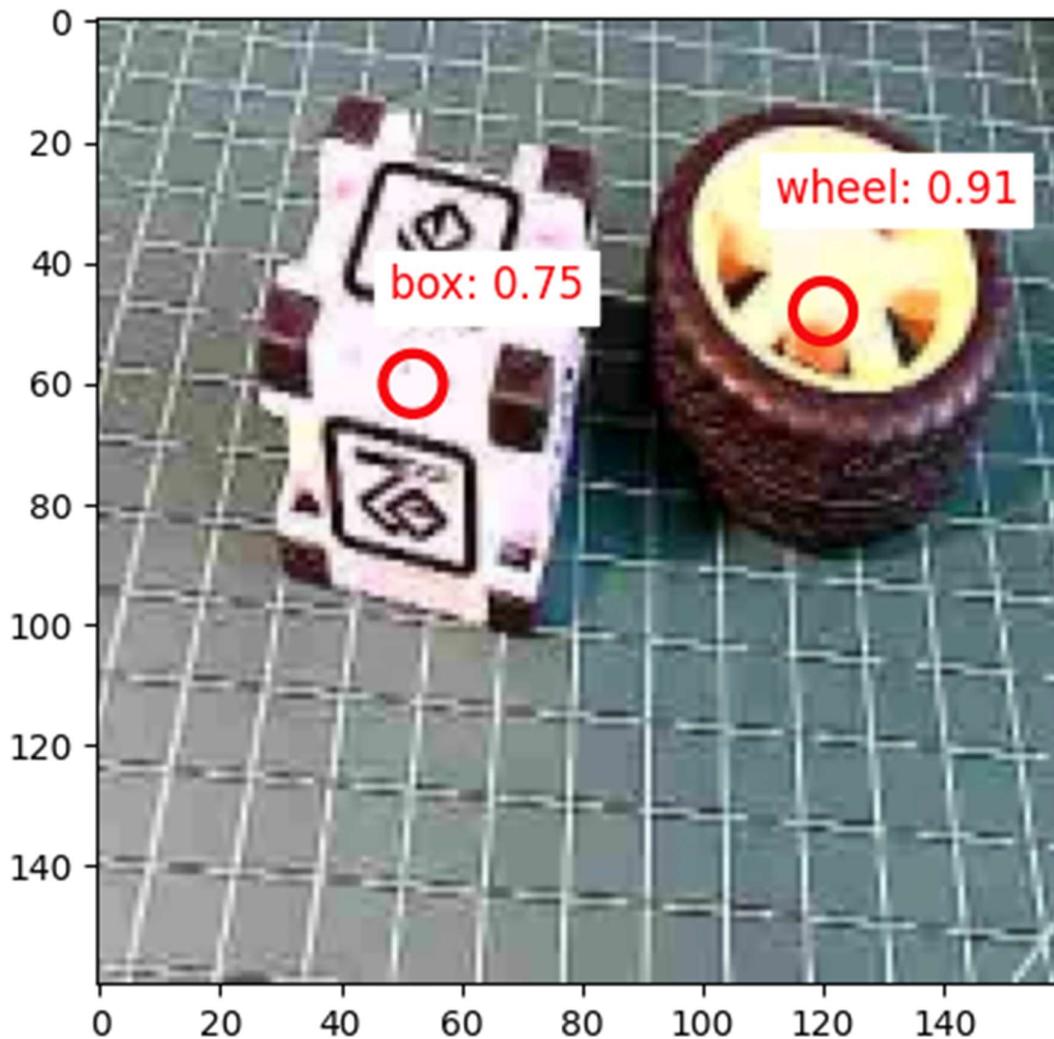
Go through each of the returned bounding boxes
bboxes = res['result']['bounding_boxes']
for bbox in bboxes:

 # Get the corners of the bounding box
 left = bbox['x']
 top = bbox['y']
 width = bbox['width']
 height = bbox['height']

 # Draw a circle centered on the detection
 circ = plt.Circle((left+width//2, top+height//2), 5,
 fill=False, color='red', linewidth=3)
 plt.gca().add_patch(circ)
 class_id = int(bbox['label'])
 class_name = labels[class_id]
 plt.text(left, top-10, f'{class_name}: {bbox["value"]:.2f}',
 color='red', fontsize=12, backgroundcolor='white')
plt.show()

```

Found 2 bounding boxes (latency: 29 ms)



## Esplorazione di un Modello YOLO tramite Ultralytics

Per questo laboratorio, esploreremo YOLOv8. [Ultralytics YOLOv8](#) è una versione dell'acclamato modello di rilevamento di oggetti in tempo reale e segmentazione delle

immagini, YOLO. YOLOv8 è basato su progressi all'avanguardia nel deep learning e nella visione artificiale, offrendo prestazioni senza pari in termini di velocità e precisione. Il suo design semplificato lo rende adatto a varie applicazioni e facilmente adattabile a diverse piattaforme hardware, dai dispositivi edge alle API cloud.

### **A proposito del modello YOLO**

Il modello YOLO (You Only Look Once) è un algoritmo di rilevamento di oggetti altamente efficiente e ampiamente utilizzato, noto per le sue capacità di elaborazione in tempo reale. A differenza dei tradizionali sistemi di rilevamento di oggetti che riutilizzano classificatori o localizzatori per eseguire il rilevamento, YOLO inquadra il problema del rilevamento come un singolo compito di regressione. Questo approccio innovativo consente a YOLO di prevedere simultaneamente più bounding box e le relative probabilità di classe da immagini complete in un'unica valutazione, aumentando notevolmente la sua velocità.

### **Caratteristiche Principali:**

#### **1. Architettura “ingle Network”:**

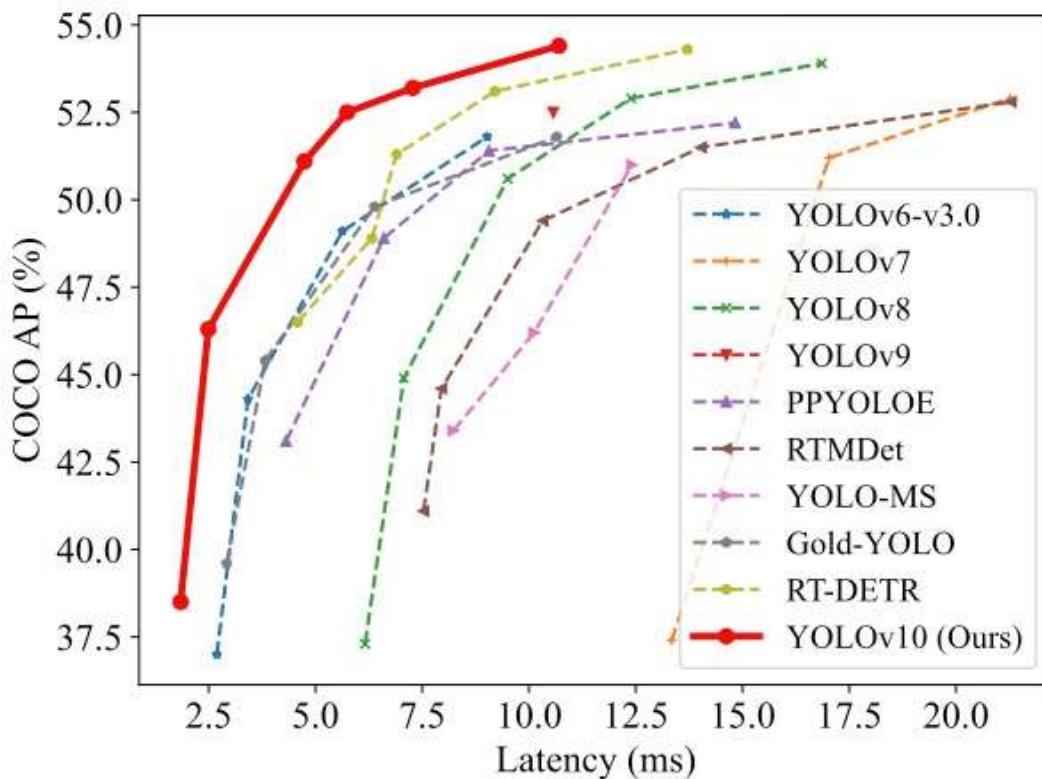
- YOLO impiega una singola rete neurale per elaborare l'intera immagine. Questa rete divide l'immagine in una griglia e, per ogni cella della griglia, prevede direttamente i bounding box e le relative probabilità di classe. Questa formazione end-to-end migliora la velocità e semplifica l'architettura del modello.

#### **2. Elaborazione in Tempo Reale:**

- Una delle caratteristiche distintive di YOLO è la sua capacità di eseguire il rilevamento di oggetti in tempo reale. A seconda della versione e dell'hardware, YOLO può elaborare immagini con frame al secondo (FPS) elevati. Ciò lo rende ideale per applicazioni che richiedono un rilevamento di oggetti rapido e accurato, come videosorveglianza, guida autonoma e analisi di eventi sportivi in diretta.

#### **3. Evoluzione delle Versioni:**

- Nel corso degli anni, YOLO ha subito miglioramenti significativi, da YOLOv1 all'ultimo YOLOv10. Ogni iterazione ha introdotto miglioramenti in termini di accuratezza, velocità ed efficienza. YOLOv8, ad esempio, incorpora progressi nell'architettura di rete, metodologie di training migliorate e un supporto migliore per vari hardware, garantendo prestazioni più robuste.
- Sebbene YOLOv10 sia il membro più recente della famiglia con prestazioni incoraggianti in base alla sua documentazione, è stato appena rilasciato (maggio 2024) e non è completamente integrato con la libreria Ultralightcs. Al contrario, l'analisi della curva di precisione-richiamo suggerisce che YOLOv8 generalmente supera YOLOv9, catturando una percentuale maggiore di veri positivi e riducendo al minimo i falsi positivi in modo più efficace (per maggiori dettagli, vedere questo [articolo](#)). Quindi, questo laboratorio si basa su YOLOv8n.



#### 4. Precisione ed Efficienza:

- Mentre le prime versioni di YOLO barattavano un po' di precisione per la velocità, le versioni recenti hanno fatto notevoli progressi nell'equilibrare entrambi. I modelli più recenti sono più veloci e precisi, rilevano piccoli oggetti (come le api) e funzionano bene su set di dati complessi.

#### 5. Ampia Gamma di Applicazioni:

- La versatilità di YOLO ha portato alla sua adozione in numerosi campi. Viene utilizzato nei sistemi di monitoraggio del traffico per rilevare e contare i veicoli, nelle applicazioni di sicurezza per identificare potenziali minacce e nella tecnologia agricola per monitorare raccolti e bestiame. La sua applicazione si estende a qualsiasi dominio che richieda un rilevamento efficiente e accurato degli oggetti.

#### 6. Comunità e Sviluppo:

- YOLO continua a evolversi ed è supportato da una solida comunità di sviluppatori e ricercatori (essendo YOLOv8 molto forte). Le implementazioni open source e l'ampia documentazione lo hanno reso accessibile per la personalizzazione e l'integrazione in vari progetti. Framework di deep learning popolari come Darknet, TensorFlow e PyTorch supportano YOLO, ampliandone ulteriormente l'applicabilità.
- Ultralytics YOLOv8** non solo può **Detect** [rilevare] (il nostro caso qui) ma anche **Segmentare** e mettere in posa con **Pose** modelli pre-addestrati sul set di dati **COCO** e YOLOv8 **Classifica** modelli pre-addestrati sul set di dati **ImageNet**. La modalità **Track** è disponibile per tutti i modelli Detect, Segment e Pose.



Attività supportate da Ultralytics YOLO

### Installazione

Sul nostro Raspi, disattiviamo l'ambiente corrente per creare una nuova area di lavoro:

```
deactivate
cd ~
cd Documents/
mkdir YOLO
cd YOLO
mkdir models
mkdir images
```

### Impostiamo un Virtual Environment per lavorare con Ultralytics YOLOv8

```
python3 -m venv ~/yolo
source ~/yolo/bin/activate
```

E installiamo i pacchetti Ultralytics per l'inferenza locale sul Raspi

1. Aggiorniamo l'elenco dei pacchetti, installiamo pip ed eseguiamo l'aggiornamento all'ultima versione:

```
sudo apt update
sudo apt install python3-pip -y
pip install -U pip
```

2. Installiamo il pacchetto pip `ultralytics` con le dipendenze opzionali:

```
pip install ultralytics [export]
```

3. Riavviamo il dispositivo:

```
sudo reboot
```

### Test di YOLO

Dopo l'avvio di Raspi-Zero, attiviamo l'ambiente `yolo`, andiamo alla directory di lavoro,

```
source ~/yolo/bin/activate
cd /Documents/YOLO
```

ed eseguiamo l'inferenza su un'immagine che verrà scaricata dal sito web di Ultralytics, che utilizza il modello YOLOV8n (il più piccolo della famiglia) nel Terminal (CLI):

```
yolo predict model='yolov8n' source='https://ultralytics.com/images/bus.jpg'
```

La famiglia di modelli YOLO è pre-addestrata con il dataset COCO.

Il risultato dell'inferenza apparirà nel terminale. Nell'immagine (bus.jpg), sono state rilevate 4 persone, 1 autobus e 1 segnale di stop:

```
marcelo_rovai@raspi-zero:~/Documents/YOLO$ cd -
marcelo_rovai@raspi-zero:~$ source ./yolo/bin/activate
(yolo) marcelo_rovai@raspi-zero:~$ cd Documents/YOLO/
(yolo) marcelo_rovai@raspi-zero:~/Documents/YOLO$ yolo predict model='yolov8n' source='https://ultralytics.com/images/bus.jpg'
Downloading https://github.com/ultralytics/assets/releases/download/v8.2.0/yolov8n.pt to 'yolov8n.pt'...
100%|██████████| 6.25M/6.25M [00:01<00:00, 4.82MB/s]
Ultralytics YOLOv8.2.90 ✅ Python-3.11.2 torch-2.4.1 CPU (Cortex-A53)
YOLOv8n summary (fused): 168 layers, 3,151,904 parameters, 0 gradients, 8.7 GFLOPS

Downloading https://ultralytics.com/images/bus.jpg to 'bus.jpg'...
100%|██████████| 134k/134k [00:00<00:00, 2.92MB/s]
image 1/1 /home/mjrovai/Documents/YOLO/bus.jpg: 640x480 4 persons, 1 bus, 1 stop sign, 17946.4ms
Speed: 1985.4ms preprocess, 17946.4ms inference, 825.9ms postprocess per image at shape (1, 3, 640, 480)
Results saved to runs/detect/predict
💡 Learn more at https://docs.ultralytics.com/modes/predict
```

Inoltre, abbiamo ricevuto un messaggio che indica Results saved to runs/detect/predict.

Ispezionando quella directory, possiamo vedere una nuova immagine salvata (bus.jpg).

Scarichiamola dal Raspi-Zero sul desktop per l'ispezione:



Quindi, Ultrayitics YOLO è installato correttamente sul nostro Raspi. Ma, su Raspi-Zero, un problema è l'elevata latenza per questa inferenza, circa 18 secondi, anche con il modello più in miniatura della famiglia (YOLOv8n).

### **Esportazione del Modello in formato NCNN**

L'implementazione di modelli di visione artificiale su dispositivi edge con potenza di calcolo limitata, come Raspi-Zero, può causare problemi di latenza. Un'alternativa è quella di utilizzare un formato ottimizzato per prestazioni ottimali. Ciò garantisce che anche i dispositivi con potenza di elaborazione limitata possano gestire bene attività di visione artificiale avanzate.

Di tutti i formati di esportazione del modello supportati da Ultralytics, **NCNN** è un framework di elaborazione inferenziale di reti neurali ad alte prestazioni ottimizzato per piattaforme mobili. Fin dall'inizio della progettazione, NCNN è stato profondamente attento all'implementazione e all'uso su telefoni cellulari e non aveva dipendenze di terze parti. È multipiattaforma e funziona più velocemente di tutti i framework open source noti (come TFLite).

NCNN offre le migliori prestazioni di inferenza quando si lavora con dispositivi Raspberry Pi. NCNN è altamente ottimizzato per piattaforme mobili embedded (come l'architettura ARM). Quindi, convertiamo il nostro modello e rieseguiamo l'inferenza:

1. Esportiamo un modello PyTorch YOLOv8n in formato NCNN, creando:  
'yolov8n\_ncnn\_model'

```
yolo export model=yolov8n.pt format=ncnn
```

2. Eseguiamo l'inferenza con il modello esportato (ora la sorgente potrebbe essere l'immagine bus.jpg scaricata dal sito Web nella directory corrente nell'ultima inferenza):

```
yolo predict model='./yolov8n_ncnn_model' source='bus.jpg'
```

La prima inferenza, quando il modello viene caricato, di solito ha una latenza elevata (circa 17 s), ma dalla seconda, è possibile notare che l'inferenza scende a circa 2 s.

### **Esplorare YOLO con Python**

Per iniziare, chiamiamo l'interprete Python in modo da poter esplorare il funzionamento del modello YOLO, riga per riga:

```
python3
```

Ora, dovremmo chiamare la libreria YOLO da Ultralitics e caricare il modello:

```
from ultralytics import YOLO
model = YOLO('yolov8n_ncnn_model')
```

Quindi, eseguire l'inferenza su un'immagine (usiamo di nuovo bus.jpg):

```
img = 'bus.jpg'
result = model.predict(img, save=True, imgsz=640, conf=0.5, iou=0.3)
```

```
>>> from ultralytics import YOLO
>>> model = YOLO('yolov8n_ncnn_model')
WARNING ▲Unable to automatically guess model task, assuming 'task=detect'. Explicitly define task for your model, i.e. 'task=detect', 'segment', 'classify', 'pose' or 'obb'.
>>> img = 'bus.jpg'
>>> result = model.predict(img, save=True, imgsz=640, conf=0.5, iou=0.3)
Loading yolov8n_ncnn_model for NCNN inference...

image 1/1 /home/mjrovai/Documents/YOLO/bus.jpg: 640x640 3 persons, 1 bus, 2362.9ms
Speed: 631.0ms preprocess, 2362.9ms inference, 849.7ms postprocess per image at shape (1, 3, 640, 640)
Results saved to runs/detect/predict6
>>>
```

Possiamo verificare che il risultato è quasi identico a quello che otteniamo eseguendo l'inferenza a livello di terminale (CLI), tranne per il fatto che la fermata dell'autobus non è stata rilevata con il modello NCNN ridotto. Notare che la latenza è stata ridotta.

Analizziamo il contenuto di “result”.

Ad esempio, possiamo vedere `result[0].boxes.data`, che mostra il risultato principale dell’inferenza, che è un profilo tensoriale (4, 6). Ogni riga è uno degli oggetti rilevati, ovvero le prime 4 colonne, le coordinate delle bounding box, la quinta, la confidenza e la sesta, la classe (in questo caso, 0: person e 5: bus):

```
marcelo_rovai — mjrovai@raspi-zero: ~/Documents/YOLO — ssh mjrovai@192.168.4.210 — 85x6
>>> result[0].boxes.data
tensor([[6.7045e+02, 3.8056e+02, 8.0992e+02, 8.7965e+02, 8.9021e-01, 0.0000e+00],
 [2.2168e+02, 4.0742e+02, 3.4378e+02, 8.5619e+02, 8.8328e-01, 0.0000e+00],
 [5.0682e+01, 3.9730e+02, 2.4440e+02, 9.0538e+02, 8.7844e-01, 0.0000e+00],
 [3.1489e+01, 2.3074e+02, 8.0141e+02, 7.7578e+02, 8.4337e-01, 5.0000e+00]])
>>>
```

Possiamo accedere a diversi risultati di inferenza separatamente, come il tempo di inferenza, e stamparli in un formato migliore:

```
inference_time = int(result[0].speed['inference'])
print(f"Inference Time: {inference_time} ms")
```

Ottimamente possiamo avere il numero totale di oggetti rilevati:

```
print(f'Number of objects: {len(result[0].boxes.cls)})'
```

```
marcelo_rovai — mjrovai@raspi-zero: ~/Documents/YOLO — ssh mjrovai@192.168.4.210 — 64x6
>>> inference_time = int(result[0].speed['inference'])
>>> print(f"Inference Time: {inference_time} ms")
Inference Time: 2362 ms
>>> print(f'Number of objects: {len(result[0].boxes.cls)})')
Number of objects: 4
>>>
```

Con Python, possiamo creare un output dettagliato che soddisfi le nostre esigenze (vedere [Model Prediction with Ultralytics YOLO](#) per maggiori dettagli). Eseguiamo uno script Python invece di inserirlo manualmente riga per riga nell’interprete, come mostrato di seguito.

Usiamo `nano` come editor di testo. Per prima cosa, dovremmo creare uno script Python vuoto denominato, ad esempio, `yolov8_tests.py`:

```
nano yolov8_tests.py
```

Si inseriscono le righe di codice:

```
from ultralytics import YOLO

Load the YOLOv8 model
model = YOLO('yolov8n_ncnn_model')

Run inference
img = 'bus.jpg'
result = model.predict(img, save=False, imgsz=640, conf=0.5, iou=0.3)

Print the results
inference_time = int(result[0].speed['inference'])
print(f"Inference Time: {inference_time} ms")
print(f"Number of objects: {len(result[0].boxes.cls)})")
```

The screenshot shows a terminal window titled "marcelo\_rovai — mjrovai@raspi-zero: ~/Documents/YOLO — ssh mjrovai@192.168.4.210 — 70x19". The window contains the following Python code:

```
GNU nano 7.2 yolov8 tests.py *
from ultralytics import YOLO

Load the YOLOv8 model
model = YOLO('yolov8n_ncnn_model')

Run inference
img = 'bus.jpg'
result = model.predict(img, save=False, imgsz=640, conf=0.5, iou=0.3)

print the results
inference_time = int(result[0].speed['inference'])
print(f"Inference Time: {inference_time} ms")
print(f"Number of objects: {len(result[0].boxes.cls)}")
```

At the bottom of the terminal, there is a menu bar with keyboard shortcuts:

- ^G Help
- ^O Write Out
- ^W Where Is
- ^K Cut
- ^T Execute
- ^X Exit
- ^R Read File
- ^\\ Replace
- ^U Paste
- ^J Justify

E si inseriscono i comandi: [CTRL+O] + [INVIO] + [CTRL+X] per salvare lo script Python.  
Eseguire lo script:

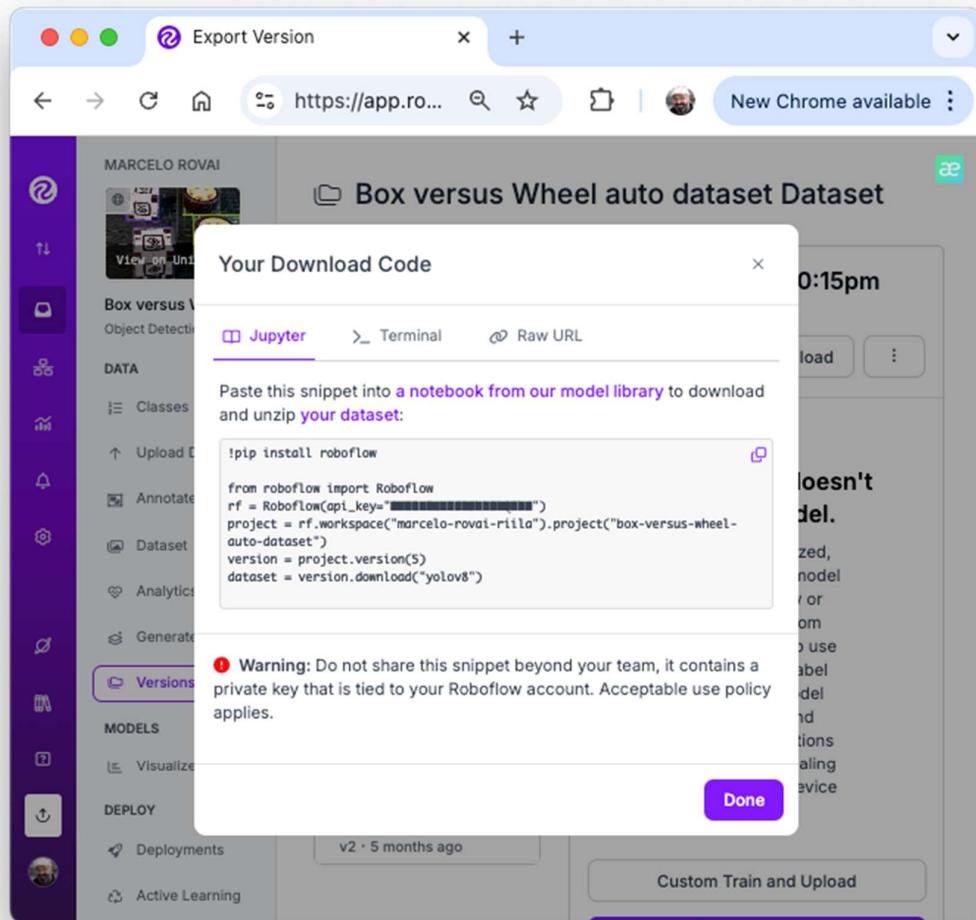
```
python yolov8_tests.py
```

Il risultato è lo stesso dell'esecuzione dell'inferenza a livello di terminale (CLI) e con l'interprete Python nativo.

Chiamare la libreria YOLO e caricare il modello per l'inferenza per la prima volta richiede molto tempo, ma le inferenze successive saranno molto più veloci. Ad esempio, la prima singola inferenza può richiedere diversi secondi, ma in seguito il tempo di inferenza dovrebbe essere ridotto a meno di 1 secondo.

#### ***Addestramento di YOLOv8 su un Dataset Personalizzato***

Torniamo al nostro set di dati “Boxe versus Wheel”, etichettato su [Roboflow](#). Nell'opzione `Download Dataset`, invece dell'opzione `Download a zip to computer` eseguita per l'addestramento su Edge Impulse Studio, opteremo per `Show download code`. Questa opzione aprirà una finestra pop-up con un frammento di codice che dovrebbe essere incollato nel nostro notebook di training.



Per l'addestramento, adattiamo uno degli esempi pubblici disponibili da Ultralytics ed eseguiamolo su Google Colab. Di seguito, si può trovare il mio da adattare al progetto:

- YOLOv8 Box versus Wheel Dataset Training [Open In Colab]

#### Punti critici sul Notebook:

1. Eseguirlo con GPU (Nvidia T4 è gratuito)
2. Installare Ultralytics tramite PIP.

```

9s [3] 1 # Pip install method (recommended)
 2
 3 !pip install ultralytics
 4
 5 from IPython import display
 6 display.clear_output()
 7
 8 import ultralytics
 9 ultralytics.checks()

→ Ultralytics YOLOv8.2.91 🎨 Python-3.10.12 torch-2.4.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Setup complete ✓ (2 CPUs, 12.7 GB RAM, 32.8/112.6 GB disk)

```

3. Ora, si può importare YOLO e caricare il dataset sul CoLab, incollando il codice di download che riceviamo da Roboflow. Notare che il dataset verrà montato in /content/datasets/:

```

File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Dataset
1 !mkdir {HOME}/datasets
2 %cd {HOME}/datasets
3
4 !pip install roboflow --quiet
5
6 from roboflow import Roboflow
7 rf = Roboflow(api_key="8NE0n2tX76Ejb9h0Tb4")
8 project = rf.workspace("marcelo-rovai-riila").project("box-versus-wheel-auto-dataset")
9 version = project.version(5)
10 dataset = version.download("yolov8")
11
/content/datasets
79.9/79.9 KB 5.5 MB/s eta 0:00:00
66.8/66.8 KB 5.3 MB/s eta 0:00:00
54.5/54.5 KB 5.3 MB/s eta 0:00:00
loading Roboflow workspace...
loading Roboflow project...
Dependency ultralytics==8.0.196 is required but found version=8.2.91, to fix: 'pip install ultralytics==8.0.196'
Downloading Dataset Version Zip in Box-versus-Wheel-auto-dataset-5 to yolov8: 100%|██████████| 4286/4286 [00:00<00:00, 16174.48it/s]
Extracting Dataset Version Zip to Box-versus-Wheel-auto-dataset-5 in yolov8: 100%|██████████| 318/318 [00:00<00:00, 8613.15it/s]
6s completed at 10:48AM

```

4. È essenziale verificare e modificare il file `data.yaml` con il path corretto per le immagini (copiare il percorso su ogni cartella `images`).

```

names:
- box
- wheel
nc: 2
roboflow:
 license: CC BY 4.0
 project: box-versus-wheel-auto-dataset
 url: https://universe.roboflow.com/marcelo-rovai-riila/box-versus-wheel-auto-dataset/da
taset/5
 version: 5
 workspace: marcelo-rovai-riila
test: /content/datasets/Box-versus-Wheel-auto-dataset-5/test/images
train: /content/datasets/Box-versus-Wheel-auto-dataset-5/train/images
val: /content/datasets/Box-versus-Wheel-auto-dataset-5/valid/images

```

5. Definire i principali iperparametri da modificare rispetto ai valori di default, ad esempio:

```

MODEL = 'yolov8n.pt'
IMG_SIZE = 640
EPOCHS = 25 # For a final project, you should consider at least 100 epochs

```

6. Eseguire il training (utilizzando la CLI):

```

!yolo task=detect mode=train model={MODEL} data={dataset.location}/data.yaml epoch
s={EPOCHS} imgsz={IMG_SIZE} plots=True

```

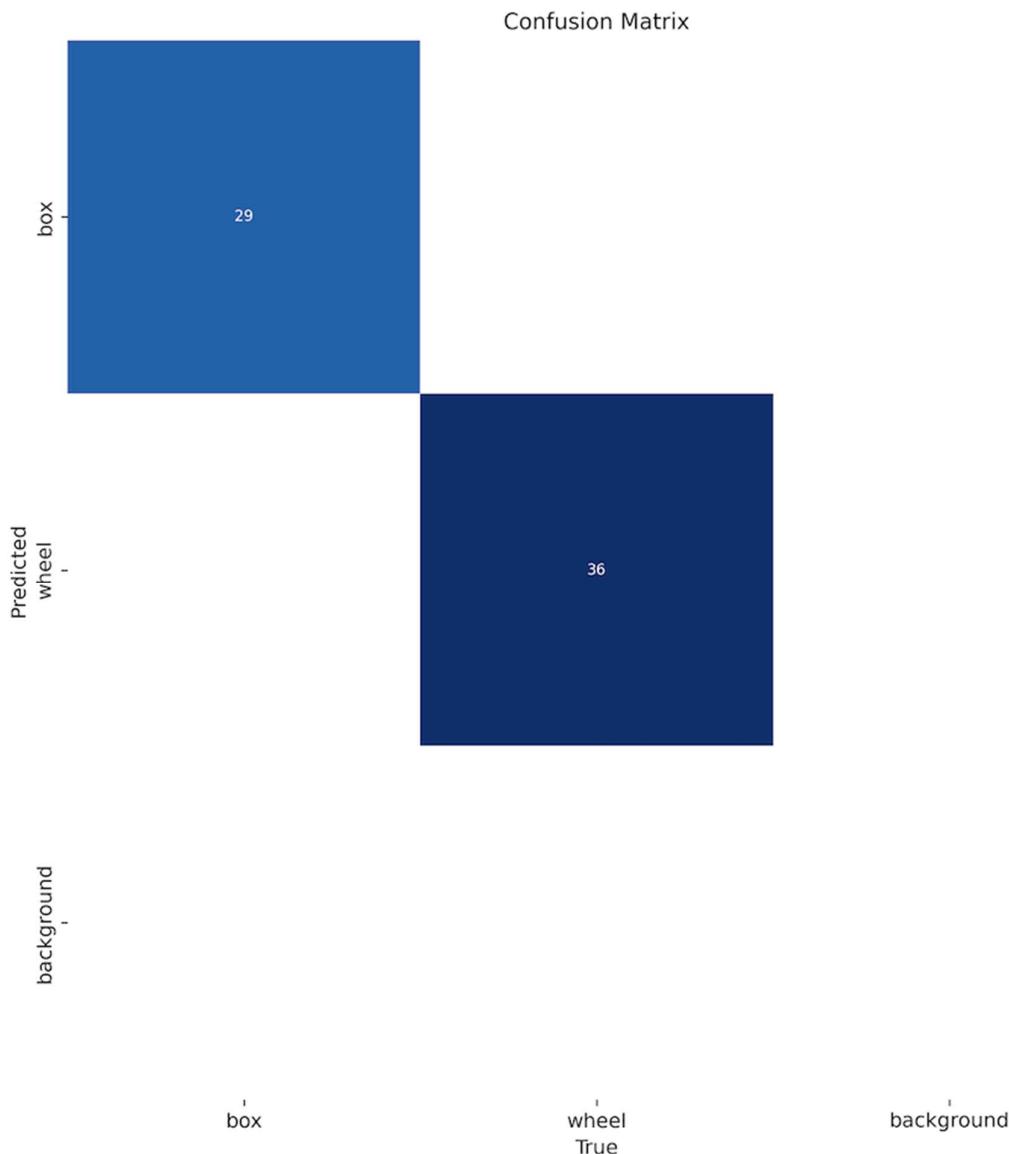
```

25 epochs completed in 0.026 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 6.2MB
Optimizer stripped from runs/detect/train/weights/best.pt, 6.2MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.2.91 🦄 Python 3.10.12 torch-2.4.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3,006,038 parameters, 0 gradients, 8.1 GFLOPs
 Class Images Instances Box(P) R mAP50 mAP50-95: 100% 1/1 [00:00<00:00, 7.61it/s]
 all 12 65 0.997 1 0.995 0.899
 box 11 29 0.999 1 0.995 0.903
 wheel 11 36 0.995 1 0.995 0.896
Speed: 0.2ms preprocess, 2.6ms inference, 0.0ms loss, 3.2ms postprocess per image
image-20240910111319804

```

L'addestramento del modello ha richiesto alcuni minuti e ha prodotto risultati eccellenti (mAP50 pari a 0,995). Al termine del training, tutti i risultati vengono salvati nella cartella elencata, ad esempio: `/runs/detect/train/`. Lì si trova, ad esempio, la matrice di confusione.



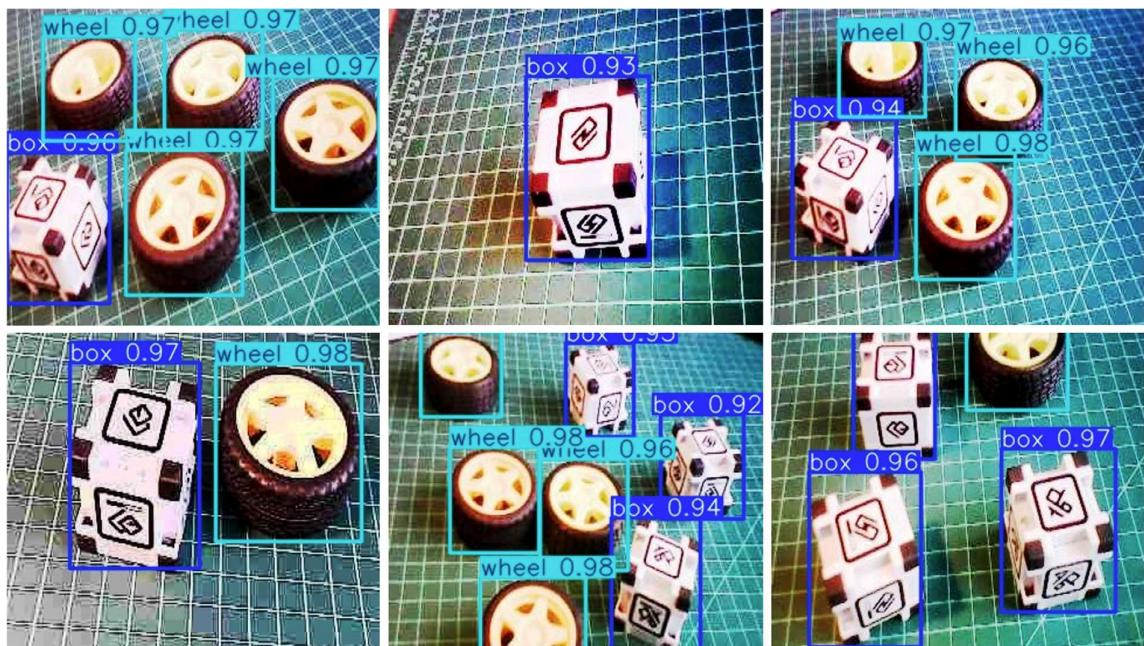
7. Notare che il modello addestrato (`best.pt`) viene salvato nella cartella `/runs/detect/train/weights/`. Ora, si deve convalidare il modello addestrato con `valid/images`.

```
!yolo task=detect mode=val model={HOME}/runs/detect/train/weights/best.pt data={dataset.location}/data.yaml
I risultati erano simili al training.
```

8. Ora, si deve eseguire l'inferenza sulle immagini lasciate a parte per il test

```
!yolo task=detect mode=predict model={HOME}/runs/detect/train/weights/best.pt conf=0.25 source={dataset.location}/test/images save=True
```

I risultati dell'inferenza vengono salvati nella cartella `runs/detect/predict`. Vediamone alcuni:



9. Si consiglia di esportare i risultati di train, validation e test per un Drive su Google. Per farlo, si deve montare l'unità.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

e copiare il contenuto della cartella `/runs` in una cartella che si deve creare nel Drive, ad esempio:

```
!scp -r /content/runs '/content/gdrive/MyDrive/10_UNIFEI/Box_vs_Wheel_Project'
```

### **Inferenza col modello addestrato, usando Raspi**

Scaricare il modello addestrato `/runs/detect/train/weights/best.pt` nel computer.

Utilizzando l'FTP FileZilla, trasferiamo `best.pt` nella cartella dei modelli Raspi (prima del trasferimento, si può cambiare il nome del modello, ad esempio, `box_wheel_320_yolo.pt`).

Con FileZilla FTP, trasferiamo alcune immagini dal set di dati di prova a `.\YOLO\images`:

Torniamo alla cartella YOLO e utilizziamo l'interprete Python:

```
cd ..
python
```

Come prima, importeremo la libreria YOLO e definiremo il nostro modello convertito per rilevare le api:

```
from ultralytics import YOLO
model = YOLO('./models/box_wheel_320_yolo.pt')
```

Ora, definiamo un'immagine e chiamiamo l'inferenza (stavolta salveremo il risultato dell'immagine per la verifica esterna):

```
img = './images/1_box_1_wheel.jpg'
result = model.predict(img, save=True, imgsz=320, conf=0.5, iou=0.3)
```

Ripetiamo per diverse immagini. Il risultato dell'inferenza viene salvato sulla variabile `result` e l'immagine elaborata su `runs/detect/predict8`

```

marcelo_rovai — mjrovai@raspi-zero: ~/Documents/YOLO — ssh mjrovai@192.168.4.210 — 106x26
>>> model = YOLO('./models/box_wheel_320_yolo.pt')
>>> img = './images/1_box_1_wheel.jpg'
>>> result = model.predict(img, save=True, imgsz=320, conf=0.5, iou=0.3)

image 1/1 /home/mjrovai/Documents/YOLO/images/1_box_1_wheel.jpg: 256x320 1 box, 1 wheel, 2390.8ms
Speed: 164.3ms preprocess, 2390.8ms inference, 75.9ms postprocess per image at shape (1, 3, 256, 320)
Results saved to runs/detect/predict8
>>> img = './images/box_2_wheel_1.jpg'
>>> result = model.predict(img, save=True, imgsz=320, conf=0.5, iou=0.3)

image 1/1 /home/mjrovai/Documents/YOLO/images/box_2_wheel_1.jpg: 256x320 2 boxes, 1 wheel, 1620.4ms
Speed: 292.5ms preprocess, 1620.4ms inference, 49.2ms postprocess per image at shape (1, 3, 256, 320)
Results saved to runs/detect/predict8
>>> img = './images/2_wheel.jpg'
>>> result = model.predict(img, save=True, imgsz=320, conf=0.5, iou=0.3)

image 1/1 /home/mjrovai/Documents/YOLO/images/2_wheel.jpg: 256x320 2 wheels, 1010.3ms
Speed: 6.9ms preprocess, 1010.3ms inference, 7.7ms postprocess per image at shape (1, 3, 256, 320)
Results saved to runs/detect/predict8
>>> img = './images/1_wheel.jpg'
>>> result = model.predict(img, save=True, imgsz=320, conf=0.5, iou=0.3)

image 1/1 /home/mjrovai/Documents/YOLO/images/1_wheel.jpg: 256x320 1 wheel, 1085.1ms
Speed: 11.7ms preprocess, 1085.1ms inference, 7.4ms postprocess per image at shape (1, 3, 256, 320)
Results saved to runs/detect/predict8
>>>

```

Con FileZilla FTP, possiamo inviare il risultato dell'inferenza al Desktop per la verifica:



Possiamo vedere che il risultato dell'inferenza è eccellente! Il modello è stato addestrato in base al modello base più piccolo della famiglia YOLOv8 (YOLOv8n). Il problema è la latenza, circa 1 secondo (o 1 FPS su Raspi-Zero). Naturalmente, possiamo ridurre questa latenza e convertire il modello in TFLite o NCNN.

## Rilevamento di Oggetti su un Live Streaming

Tutti i modelli esplorati in questo laboratorio possono rilevare oggetti in tempo reale utilizzando una telecamera. L'immagine catturata dovrebbe essere l'input per il modello addestrato e convertito. Per Raspi-4 o 5 con un desktop, OpenCV può catturare i frame e visualizzare il risultato dell'inferenza.

Tuttavia, è anche possibile creare un live streaming con una webcam per rilevare oggetti in tempo reale. Ad esempio, iniziamo con lo script sviluppato per l'app Image Classification e adattiamolo per un'*Applicazione Web di Rilevamento di Oggetti in Tempo Reale Utilizzando TensorFlow Lite e Flask*.

Questa versione dell'app funzionerà per tutti i modelli TFLite. Verificare che il modello sia nella cartella corretta, ad esempio:

```
model_path = "./models/ssd-mobilenet-v1-tflite-default-v1.tflite"
```

Scaricare lo script Python `object_detection_app.py` da [GitHub](#).

E sul terminale, si esegue:

```
python3 object_detection_app.py
```

E accedere all'interfaccia web:

- Sul Raspberry Pi stesso (se si ha una GUI): si apre un browser web e si va [su `http://localhost:5000`](http://localhost:5000)
- Da un altro dispositivo sulla stessa rete: aprire un browser web e andare su [http://<raspberry\\_pi\\_ip>:5000](http://<raspberry_pi_ip>:5000) (Sostituire <raspberry\_pi\_ip> con l'indirizzo IP del Raspberry Pi). Per esempio: <http://192.168.4.210:5000/>

Ecco alcuni screenshot dell'app in esecuzione su un desktop esterno



Diamo un'occhiata a una descrizione tecnica dei moduli chiave utilizzati nell'applicazione di rilevamento degli oggetti:

#### 1. **TensorFlow Lite (tflite\_runtime):**

- Scopo: Inferenza efficiente di modelli di machine learning su dispositivi edge.
- Perché: TFLite offre dimensioni ridotte del modello e prestazioni ottimizzate rispetto a TensorFlow completo, il che è fondamentale per dispositivi con risorse limitate come Raspberry Pi. Supporta l'accelerazione hardware e la quantizzazione, migliorando ulteriormente l'efficienza.

- Funzioni chiave: `Interpreter` per caricare ed eseguire il modello, `get_input_details()` e `get_output_details()` per l'interfaccia col modello.

## 2. **Flask:**

- Scopo: Framework web leggero per la creazione del server backend.
- Perché: La semplicità e la flessibilità di Flask lo rendono ideale per lo sviluppo e la distribuzione rapidi di applicazioni web. Richiede meno risorse rispetto a framework più grandi, adatto a dispositivi edge.
- Componenti chiave: decoratori di route per definire endpoint API, oggetti `Response` per lo streaming video, `render_template_string` per servire HTML dinamico.

## 3. **Picamera2:**

- Scopo: Interfaccia con il modulo fotocamera Raspberry Pi.
- Perché: Picamera2 è la libreria più recente per il controllo delle fotocamere Raspberry Pi, che offre prestazioni e funzionalità migliorate rispetto alla libreria Picamera originale.
- Funzioni chiave: `create_preview_configuration()` per impostare la fotocamera, `capture_file()` per catturare i fotogrammi.

## 4. **PIL (Python Imaging Library):**

- Scopo: Elaborazione e manipolazione delle immagini.
- Perché: PIL fornisce un'ampia gamma di funzionalità di elaborazione delle immagini. Viene utilizzato qui per ridimensionare le immagini, disegnare riquadri di delimitazione e convertire tra formati di immagine.
- Classi chiave: `Image` per caricare e manipolare immagini, `ImageDraw` per disegnare forme e testo sulle immagini.

## 5. **NumPy:**

- Scopo: Operazioni array efficienti e calcolo numerico.
- Perché: le operazioni su array di NumPy sono molto più veloci delle liste Python pure, il che è fondamentale per elaborare in modo efficiente i dati delle immagini e gli input/output del modello.
- Funzioni chiave: `array()` per creare array, `expand_dims()` per aggiungere dimensioni agli array.

## 6. **Threading:**

- Scopo: Esecuzione simultanea di attività.
- Perché: Il threading consente l'acquisizione simultanea di frame, il rilevamento di oggetti e il funzionamento del server Web, cruciali per il mantenimento delle prestazioni in tempo reale.
- Componenti chiave: La classe `Thread` crea thread di esecuzione separati e `Lock` viene utilizzato per la sincronizzazione dei thread.

## 7. **io.BytesIO:**

- Scopo: Stream binari in memoria.
- Perché: Consente una gestione efficiente dei dati delle immagini in memoria senza bisogno di file temporanei, migliorando la velocità e riducendo le operazioni di I/O.

## 8. **time:**

- Scopo: Funzioni correlate al tempo.
- Perché: Utilizzato per aggiungere ritardi (`time.sleep()`) per controllare la frequenza dei fotogrammi e per le misure delle prestazioni.

## 9. **jQuery (client-side):**

- Scopo: Manipolazione DOM semplificata e richieste AJAX.
- Perché: Semplifica l'aggiornamento dinamico dell'interfaccia web e la comunicazione con il server senza ricaricamenti di pagina.
- Funzioni chiave: `.get()` e `.post()` per richieste AJAX, metodi di manipolazione DOM per l'aggiornamento dell'interfaccia utente.

Per quanto riguarda l'architettura del sistema dell'app principale:

1. **Main Thread:** Esegue il server Flask, gestisce le richieste HTTP e serve l'interfaccia web.
2. **Camera Thread:** Cattura continuamente i frame dalla fotocamera.
3. **Detection Thread:** Elabora i frame tramite il modello TFLite per il rilevamento degli oggetti.
4. **Frame Buffer:** Spazio di memoria condiviso (protetto da blocchi) che memorizza i frame più recenti e i risultati del rilevamento.

E il flusso di dati dell'app, possiamo descriverlo in breve:

1. La fotocamera cattura il frame → Frame Buffer
2. Il thread di rilevamento legge dal Frame Buffer → Elabora tramite il modello TFLite → Aggiorna i risultati del rilevamento nel Frame Buffer
3. Flask indirizza l'accesso al Frame Buffer per fornire i risultati più recenti del frame e del rilevamento
4. Il client Web riceve gli aggiornamenti tramite AJAX e aggiorna l'interfaccia utente

Questa architettura consente un rilevamento efficiente degli oggetti in tempo reale, mantenendo al contempo un'interfaccia Web reattiva in esecuzione su un dispositivo edge con risorse limitate come un Raspberry Pi. Il threading e le librerie efficienti come TFLite e PIL consentono al sistema di elaborare i frame video in tempo reale, mentre Flask e jQuery forniscono un modo intuitivo per interagire con essi.

Si può testare l'app con un altro modello pre-elaborato, come EfficientDet, modificando la riga dell'app:

```
model_path = "./models/lite-model_efficientdet_lite0_detection_metadata_1.tflite"
```

Per usare l'app per il modello SSD-MobileNetV2, addestrato su Edge Impulse Studio con il set di dati “Box versus Wheel”, il codice dovrebbe essere adattato anche in base ai dettagli di input, come abbiamo esplorato sul suo [notebook](#).

## Conclusione

Questo laboratorio ha esplorato l'implementazione del rilevamento di oggetti su dispositivi edge come Raspberry Pi, dimostrando la potenza e il potenziale dell'esecuzione di attività avanzate di computer vision su hardware con risorse limitate. Abbiamo trattato diversi aspetti essenziali:

1. **Confronto tra Modelli:** Abbiamo esaminato diversi modelli di rilevamento oggetti, tra cui SSD-MobileNet, EfficientDet, FOMO e YOLO, confrontandone le prestazioni e i compromessi sui dispositivi edge.
2. **Training e Deployment:** Utilizzando un set di dati personalizzato di scatole e ruote (etichettato su Roboflow), abbiamo esaminato il processo di addestramento dei modelli utilizzando Edge Impulse Studio e Ultralytics e la loro distribuzione su Raspberry Pi.
3. **Tecniche di Ottimizzazione:** Per migliorare la velocità di inferenza sui dispositivi edge, abbiamo esplorato vari metodi di ottimizzazione, come la quantizzazione del modello (TFLite int8) e la conversione del formato (ad esempio, in NCNN).
4. **Applicazioni in Tempo Reale:** Il laboratorio ha esemplificato un'applicazione Web di rilevamento oggetti in tempo reale, dimostrando come questi modelli possono essere integrati in sistemi pratici e interattivi.

5. **Considerazioni sulle Prestazioni:** Durante il laboratorio abbiamo discusso l'equilibrio tra accuratezza del modello e velocità di inferenza, un aspetto fondamentale per le applicazioni di IA edge.

La capacità di eseguire il rilevamento di oggetti su dispositivi edge apre numerose possibilità in vari ambiti, dall'agricoltura di precisione, all'automazione industriale e al controllo di qualità, alle applicazioni per la casa intelligente e al monitoraggio ambientale. Elaborando i dati localmente, questi sistemi possono offrire latenza ridotta, privacy migliorata e funzionamento in ambienti con connettività limitata.

Guardando al futuro, le potenziali aree di ulteriore esplorazione includono: -

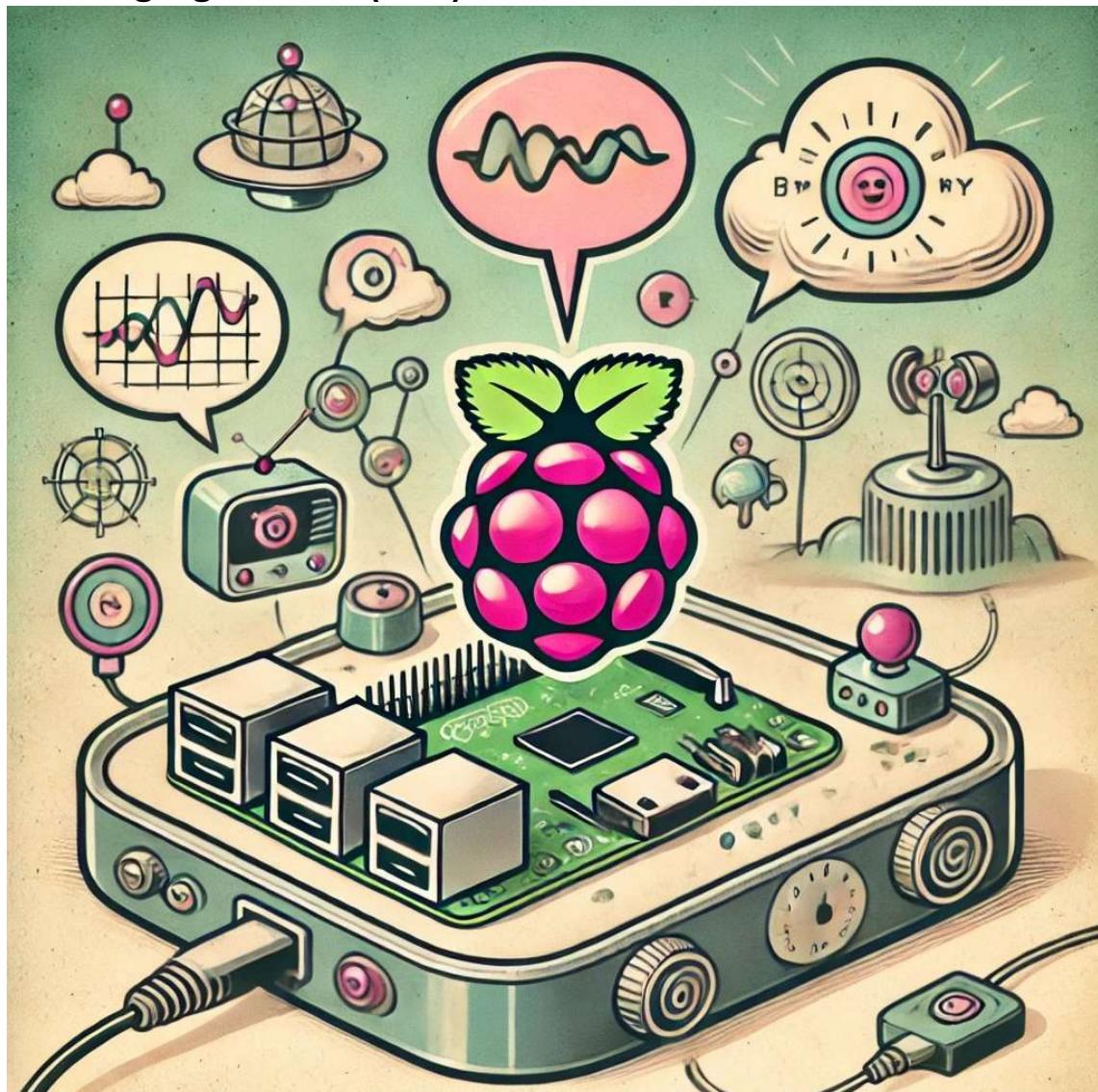
Implementazione di pipeline multi-modello per attività più complesse - Esplorazione di opzioni di accelerazione hardware per Raspberry Pi - Integrazione del rilevamento di oggetti con altri sensori per sistemi AI edge più completi - Sviluppo di soluzioni edge-to-cloud che sfruttano sia l'elaborazione locale che le risorse cloud

Il rilevamento di oggetti su dispositivi edge può creare sistemi intelligenti e reattivi che portano la potenza dell'IA direttamente nel mondo fisico, aprendo nuove frontiere nel modo in cui interagiamo e comprendiamo il nostro ambiente.

## Risorse

- [Dataset \(“Box versus Wheel”\)](#)
- [SSD-MobileNet Notebook on a Raspi](#)
- [EfficientDet Notebook on a Raspi](#)
- [FOMO - EI Linux Notebook on a Raspi](#)
- [YOLOv8 Box versus Wheel Dataset Training on CoLab](#)
- [Edge Impulse Project - SSD MobileNet and FOMO](#)
- [Script Python](#)
- [Modelli](#)

## Small Language Models (SLM)



DALL-E prompt - Un'illustrazione in stile cartoon anni '50 che mostra un Raspberry Pi che esegue un piccolo modello di linguaggio all'edge. Il Raspberry Pi è stilizzato in modo retro-futuristico con bordi arrotondati e cromature, collegato a sensori e dispositivi da cartone animato giocosi. I fumetti fluttuano in giro, rappresentando l'elaborazione del linguaggio, e lo sfondo ha un paesaggio stravagante di dispositivi interconnessi con fili e piccoli gadget, tutti disegnati in uno stile cartoon vintage. La tavolozza dei colori utilizza tenui colori pastello e contorni audaci tipici dei cartoni animati degli anni '50, conferendo un'atmosfera divertente e nostalgica alla scena.

### Panoramica

Nell'area in rapida crescita dell'intelligenza artificiale, l'edge computing offre l'opportunità di decentralizzare le capacità tradizionalmente riservate a server potenti e centralizzati. Questo laboratorio esplora l'integrazione pratica di piccole versioni di modelli linguistici tradizionali di grandi dimensioni (LLM) in un Raspberry Pi 5, trasformando questo dispositivo edge in un hub di IA in grado di elaborare dati in tempo reale e in loco.

Man mano che i modelli linguistici di grandi dimensioni crescono in dimensioni e complessità, gli Small Language Model (SLM) offrono un'alternativa interessante per i dispositivi edge, raggiungendo un equilibrio tra prestazioni ed efficienza delle risorse. Eseguendo questi modelli direttamente su Raspberry Pi, possiamo creare applicazioni reattive e rispettose della privacy che funzionano anche in ambienti con connettività Internet limitata o assente.

Questo laboratorio guiderà attraverso la configurazione, l'ottimizzazione e lo sfruttamento degli SLM su Raspberry Pi. Esploreremo l'installazione e l'utilizzo di [Ollama](#). Questo framework open source ci consente di eseguire LLM localmente sulle nostre macchine (i nostri desktop o dispositivi edge come Raspberry Pi o NVidia Jetson). Ollama è progettato per essere efficiente, scalabile e facile da usare, il che lo rende una buona opzione per distribuire modelli di IA come Microsoft Phi, Google Gemma, Meta Llama e LLaVa (Multimodal). Integreremo alcuni di questi modelli in progetti utilizzando l'ecosistema Python, esplorandone il potenziale in scenari del mondo reale (o almeno indicheremo questa direzione).

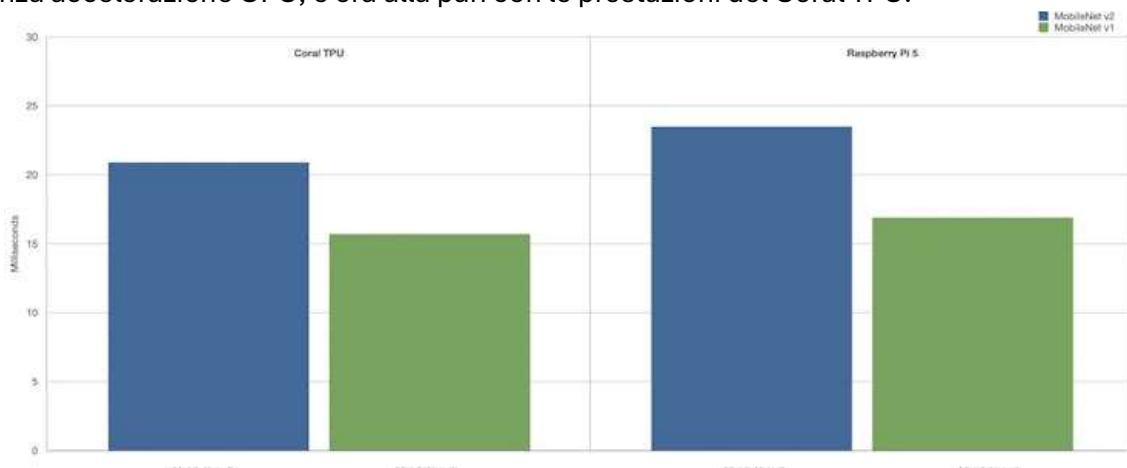


## Setup

Nei laboratori precedenti avremmo potuto usare qualsiasi modello Raspi, ma qui la scelta deve ricadere sul Raspberry Pi 5 (Raspi-5). È una piattaforma robusta che aggiorna sostanzialmente l'ultima versione 4, dotata di Broadcom BCM2712, una CPU Arm Cortex-A76 quad-core a 64 bit da 2,4 GHz con Cryptographic Extension e capacità di caching migliorate. Vanta una GPU VideoCore VII, due uscite HDMI® 4Kp60 con HDR e un decoder HEVC 4Kp60. Le opzioni di memoria includono 4 GB e 8 GB di SDRAM LPDDR4X ad alta velocità, con 8 GB come nostra scelta per eseguire SLM. Presenta inoltre un'archiviazione espandibile tramite uno slot per schede microSD e un'interfaccia PCIe 2.0 per periferiche veloci come gli SSD M.2 (Solid State Drive).

Per applicazioni SSL reali, gli SSD sono un'opzione migliore delle schede SD.

A proposito, come ha spiegato [Alasdair Allan](#), l'inferenza diretta sulla CPU Raspberry Pi 5, senza accelerazione GPU, è ora alla pari con le prestazioni del Coral TPU.

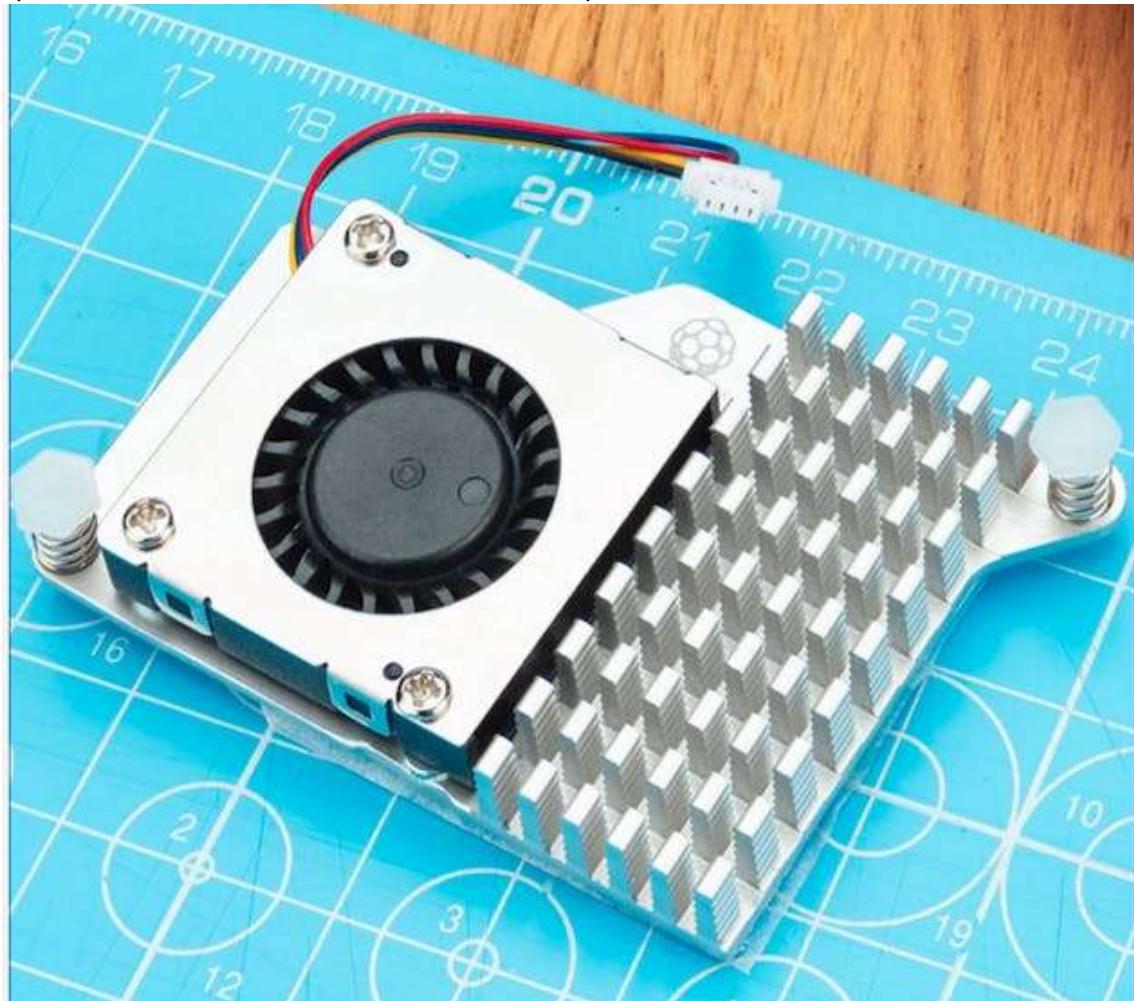


Per maggiori informazioni, consultare l'articolo completo: [Benchmarking TensorFlow e TensorFlow Lite su Raspberry Pi 5](#).

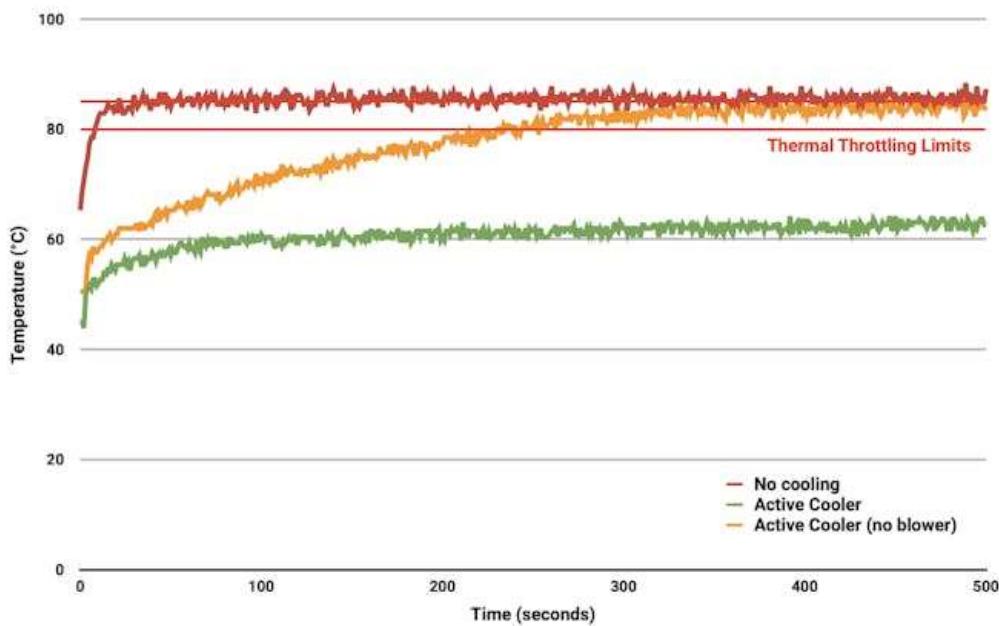
### Raspberry Pi Active Cooler

Suggeriamo di installare un Active Cooler, una soluzione di raffreddamento clip-on dedicata per Raspberry Pi 5 (Raspi-5), per questo laboratorio. Combina un dissipatore di calore in

alluminio con una ventola di raffreddamento a temperatura controllata per mantenere il Raspi-5 in funzione comodamente sotto carichi pesanti, come l'esecuzione di SLM.



L'Active Cooler ha dei pad termici pre-applicati per il trasferimento del calore ed è montato direttamente sulla scheda Raspberry Pi 5 tramite perni a molla. Il firmware Raspberry Pi lo gestisce attivamente: a 60 °C, la ventola del soffiatore viene accesa; a 67,5 °C, la velocità della ventola verrà aumentata; e infine, a 75 °C, la ventola aumenterà fino alla massima velocità. La ventola del soffiatore rallenterà automaticamente quando la temperatura scenderà al di sotto di questi limiti.



Per evitare il surriscaldamento, tutte le schede Raspberry Pi iniziano a limitare il processore quando la temperatura raggiunge gli 80 °C e a limitarlo ulteriormente quando raggiunge la temperatura massima di 85 °C (maggiori dettagli [qui](#)).

## Generative AI (GenAI)

Generative AI è un sistema di IA in grado di creare nuovi contenuti originali su vari supporti come **testo, immagini, audio e video**. Questi sistemi apprendono modelli da dati esistenti e utilizzano tale conoscenza per generare nuovi output che in precedenza non esistevano.

**Large Language Models (LLM), Small Language Models (SLM) e modelli multimodali** possono essere tutti considerati tipi di GenAI quando utilizzati per attività generative.

GenAI fornisce il framework concettuale per la creazione di contenuti basati sull'intelligenza artificiale, con gli LLM che fungono da potenti generatori di testo per uso generale. Gli SLM adattano questa tecnologia all'edge computing, mentre i modelli multimodali estendono le capacità di GenAI a diversi tipi di dati. Insieme, rappresentano uno spettro di tecnologie di intelligenza artificiale generativa, ciascuna con i suoi punti di forza e applicazioni, che guidano collettivamente la creazione e la comprensione di contenuti basati sull'IA.

### Large Language Models (LLM)

I “Large Language Model (LLM)” sono sistemi avanzati di intelligenza artificiale che comprendono, elaborano e generano testi simili a quelli umani. Questi modelli sono caratterizzati dalla loro enorme scala in termini di quantità di dati su cui vengono addestrati e numero di parametri che contengono. Gli aspetti critici degli LLM includono:

1. **Dimensioni:** Gli LLM contengono in genere miliardi di parametri. Ad esempio, GPT-3 ha 175 miliardi di parametri, mentre alcuni modelli più recenti superano un trilione di parametri.
2. **Dati di Addestramento:** Vengono addestrati su grandi quantità di dati di testo, spesso inclusi libri, siti Web e altre fonti diverse, che ammontano a centinaia di gigabyte o persino terabyte di testo.
3. **Architettura:** La maggior parte degli LLM utilizza [architetture basate su trasformatori](#), che consentono loro di elaborare e generare testo prestando attenzione a diverse parti dell'input contemporaneamente.
4. **Capacità:** Gli LLM possono eseguire un'ampia gamma di attività linguistiche senza una messa a punto specifica, tra cui:
  - Generazione di testo

- Traduzione
  - Riepilogo
  - Risposte a domande
  - Generazione di codice
  - Ragionamento logico
5. **Apprendimento a Intervalli:** Spesso possono comprendere ed eseguire nuove attività con esempi o istruzioni minimi.
  6. **Richiede molte Risorse:** A causa delle loro dimensioni, gli LLM richiedono in genere risorse di elaborazione significative per funzionare, spesso necessitando di potenti GPU o TPU.
  7. **Sviluppo Continuo:** Il campo degli LLM è in rapida evoluzione, con nuovi modelli e tecniche che emergono costantemente.
  8. **Considerazioni Etiche:** L'uso degli LLM solleva importanti questioni su pregiudizi, disinformazione e impatto ambientale della formazione di modelli così grandi.
  9. **Applicazioni:** Gli LLM sono utilizzati in vari campi, tra cui la creazione di contenuti, il servizio clienti, l'assistenza alla ricerca e lo sviluppo di software.
  10. **Limitazioni:** Nonostante la loro potenza, gli LLM possono produrre informazioni errate o distorte e non hanno capacità di vera comprensione o ragionamento.

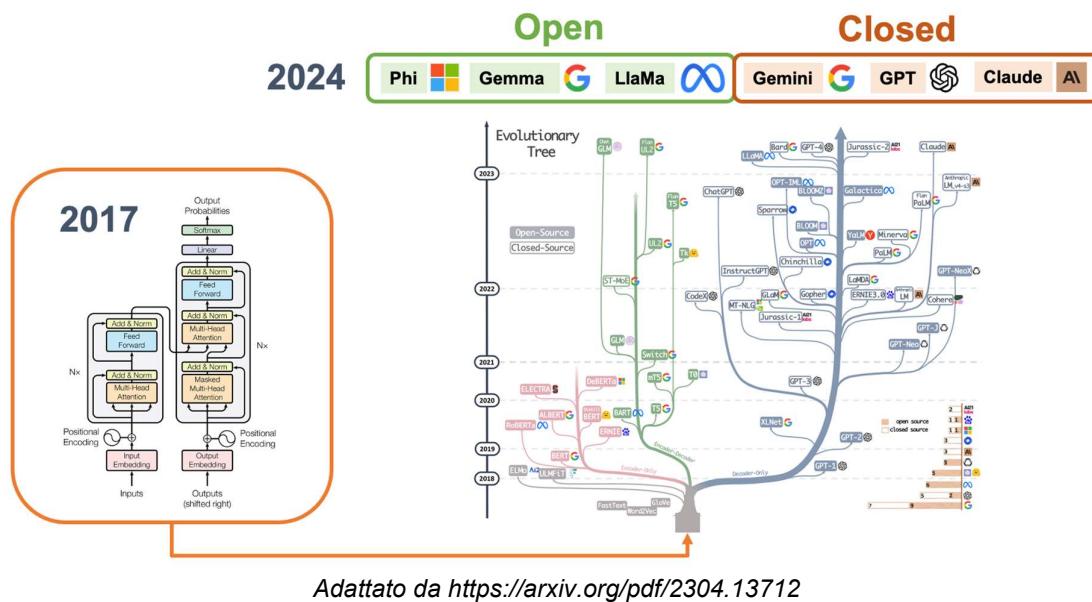
Dobbiamo notare che utilizziamo modelli di grandi dimensioni oltre al testo, chiamandoli *modelli multimediali*. Questi modelli integrano ed elaborano informazioni da più tipi di input contemporaneamente. Sono progettati per comprendere e generare contenuti in varie forme di dati, come testo, immagini, audio e video.

Certamente. Definiamo modelli “open” [aperti] e “closed” [chiusi] nel contesto di modelli di linguaggio e IA:

#### **Modelli Chiusi e Aperti:**

**Modelli Chiusi**, detti anche modelli proprietari, sono modelli di IA il cui funzionamento interno, codice e dati di addestramento non sono divulgati pubblicamente. Esempi: GPT-4 (di OpenAI), Claude (di Anthropic), Gemini (di Google).

**Modelli Aperti**, noti anche come modelli open source, sono modelli di intelligenza artificiale il cui codice sottostante, architettura e spesso dati di addestramento sono disponibili e accessibili al pubblico. Esempi: Gemma (di Google), LLaMA (di Meta) e Phi (di Microsoft). I modelli aperti sono particolarmente rilevanti per l'esecuzione di modelli su dispositivi edge come Raspberry Pi in quanto possono essere più facilmente adattati, ottimizzati e distribuiti in ambienti con risorse limitate. Tuttavia, è fondamentale verificare le loro Licenze. I modelli aperti sono dotati di varie licenze open source che possono influenzare il loro utilizzo in applicazioni commerciali, mentre i modelli chiusi hanno termini di servizio chiari, seppur restrittivi.



### Small Language Models (SLM)

Nel contesto dell'edge computing su dispositivi come Raspberry Pi, gli LLM su larga scala sono in genere troppo grandi e dispendiosi in termini di risorse per essere eseguiti direttamente. Questa limitazione ha spinto lo sviluppo di modelli più piccoli ed efficienti, come gli Small Language Models (SLM).

Gli SLM sono versioni compatte degli LLM progettate per essere eseguite in modo efficiente su dispositivi con risorse limitate come smartphone, dispositivi IoT e computer a scheda singola come Raspberry Pi. Questi modelli sono significativamente più piccoli in termini di dimensioni e requisiti di calcolo rispetto alle loro controparti più grandi, pur mantenendo impressionanti capacità di comprensione e generazione del linguaggio.

Le caratteristiche principali degli SLM includono:

1. **Conteggio ridotto dei parametri:** In genere vanno da poche centinaia di milioni a qualche miliardo di parametri, rispetto ai miliardi a due cifre nei modelli più grandi.
2. **Meno ingombro di memoria:** Richiedono, al massimo, pochi gigabyte di memoria anziché decine o centinaia di gigabyte.
3. **Tempo di inferenza più rapido:** Possono generare risposte in millisecondi o secondi su dispositivi edge.
4. **Efficienza energetica:** Consumano meno energia, rendendoli adatti per dispositivi alimentati a batteria.
5. **Tutela della privacy:** Abilitano l'elaborazione sul dispositivo senza inviare dati ai server cloud.
6. **Funzionalità offline:** Funzionano senza una connessione Internet.

Gli SLM raggiungono le loro dimensioni compatte attraverso varie tecniche come la distillazione della conoscenza, la potatura del modello e la quantizzazione. Sebbene non possano eguagliare le ampie capacità dei modelli più grandi, gli SLM eccellono in compiti e domini specifici, il che li rende ideali per applicazioni mirate su dispositivi edge.

In genere prenderemo in considerazione gli SLM, modelli linguistici con meno di 5 miliardi di parametri quantizzati a 4 bit.

Esempi di SLM includono versioni compresse di modelli come Meta Llama, Microsoft PHI e Google Gemma. Questi modelli consentono un'ampia gamma di attività di elaborazione del linguaggio naturale direttamente sui dispositivi edge, dalla classificazione del testo e analisi del sentimento alle risposte alle domande e alla generazione di testo limitato.

Per maggiori informazioni sugli SLM, il documento, [LLM Pruning and Distillation in Practice: The Minitron Approach](#), fornisce un approccio che applica il pruning e la distillazione per

ottenere SLM da LLM. E, [SMALL LANGUAGE MODELS: SURVEY, MEASUREMENTS, AND INSIGHTS](#), presenta un'indagine e un'analisi completa di Small Language Models (SLM), che sono modelli linguistici con da 100 milioni a 5 miliardi di parametri progettati per dispositivi con risorse limitate.

## Ollama



*logo di ollama*

[Ollama](#) è un framework open source che ci consente di eseguire modelli linguistici (LM), grandi o piccoli, localmente sulle nostre macchine. Ecco alcuni punti critici su Ollama:

1. **Esecuzione del Modello Locale:** Ollama consente di eseguire LM su personal computer o dispositivi edge come Raspi-5, eliminando la necessità di chiamate API basate su cloud.
2. **Facilità d'Uso:** Fornisce una semplice interfaccia a riga di comando per scaricare, eseguire e gestire diversi modelli linguistici.
3. **Varietà di Modelli:** Ollama supporta vari LLM, tra cui Phi, Gemma, Llama, Mistral e altri modelli open source.
4. **Personalizzazione:** Gli utenti possono creare e condividere modelli personalizzati su misura per esigenze o domini specifici.
5. **Leggero:** Progettato per essere efficiente e funzionare su hardware di livello consumer.
6. **Integrazione API:** Offre un'API che consente l'integrazione con altre applicazioni e servizi.
7. **Incentrato sulla Privacy:** Eseguendo i modelli localmente, affronta i problemi di privacy associati all'invio di dati a server esterni.
8. **Multipiattaforma:** Disponibile per sistemi macOS, Windows e Linux (il nostro caso, qui).
9. **Sviluppo Attivo:** Aggiornato regolarmente con nuove funzionalità e supporto per i modelli.
10. **Guidato dalla Community:** Trae vantaggio dai contributi della community e dalla condivisione dei modelli.

Per saperne di più su cosa sia Ollama e come funziona sotto internamente, si può guardare questo breve video di [Matt Williams](#), uno dei fondatori di Ollama:

<https://www.youtube.com/embed/90ozfdsQOKo>

Matt ha un corso completamente gratuito su Ollama che consigliamo:

[https://youtu.be/9KEUFe4KQAI?si=D\\_-q3CMbHiT-twuy](https://youtu.be/9KEUFe4KQAI?si=D_-q3CMbHiT-twuy)

### *Installazione di Ollama*

Impostiamo e attiviamo un ambiente virtuale per lavorare con Ollama:

```
python3 -m venv ~/ollama
source ~/ollama/bin/activate
```

Ed eseguiamo il comando per installare Ollama:

```
curl -fsSL https://ollama.com/install.sh | sh
```

Di conseguenza, un'API verrà eseguita in background su 127.0.0.1:11434. D'ora in poi, possiamo eseguire Ollama tramite il terminale. Per iniziare, verifichiamo la versione di Ollama, che ci dirà anche se è installata correttamente:

```
ollama -v
```

```
mjrovai@raspi-5:~$ python3 -m venv ~/ollama
[mjrovai@raspi-5:~$ source ~/ollama/bin/activate
(ollama) mjrovai@raspi-5:~$ curl -fsSL https://ollama.com/install.sh | sh
>>> Installing ollama to /usr/local
>>> Downloading Linux arm64 bundle
#####
#100.0%#
#####
#100.0%#
>>> Creating ollama user...
>>> Adding ollama user to render group...
>>> Adding ollama user to video group...
>>> Adding current user to ollama group...
>>> Creating ollama systemd service...
>>> Enabling and starting ollama service...
Created symlink /etc/systemd/system/default.target.wants/ollama.service → /etc/systemd/system/ollama.service.
>>> The Ollama API is now available at 127.0.0.1:11434.
>>> Install complete. Run "ollama" from the command line.
WARNING: No NVIDIA/AMD GPU detected. Ollama will run in CPU-only mode.
(ollama) mjrovai@raspi-5:~$ ollama -v
ollama version is 0.3.11
(ollama) mjrovai@raspi-5:~$
```

Nella [pagina della Libreria Ollama](#), possiamo trovare i modelli supportati da Ollama. Ad esempio, filtrando per [Most popular](#), possiamo vedere Meta Llama, Google Gemma, Microsoft Phi, LLaVa, ecc.

### **Meta Llama 3.2 1B/3B**

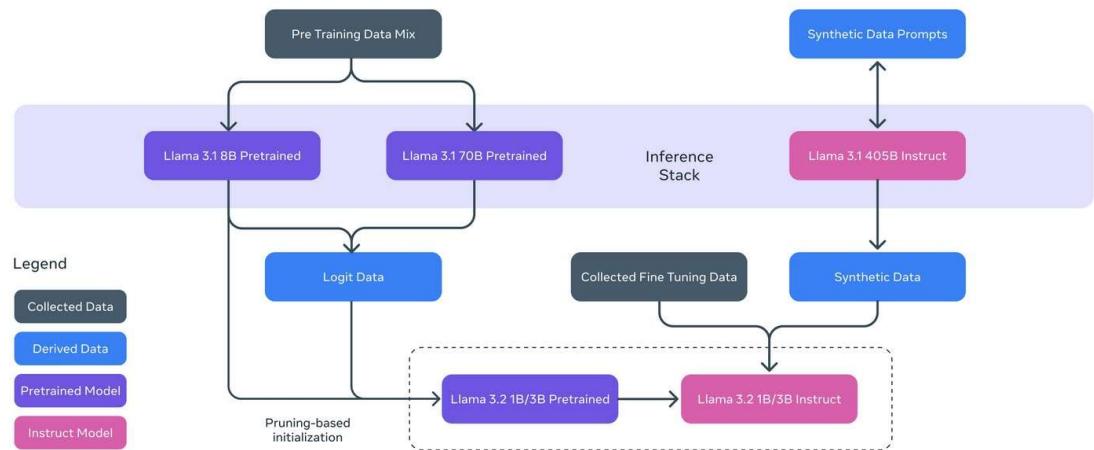


Installiamo ed eseguiamo il nostro primo piccolo modello linguistico, [Llama 3.2 1B \(e 3B\)](#). La serie Meta Llama 3.2 comprende un set di modelli linguistici generativi multilingue disponibili in dimensioni di parametri pari a 1 miliardo e 3 miliardi. Questi modelli sono progettati per elaborare input di testo e generare output di testo. Le varianti sintonizzate sulle istruzioni all'interno di questa raccolta sono specificamente ottimizzate per applicazioni conversazionali multilingue, tra cui attività che comportano il recupero e la sintesi delle

informazioni con un approccio agentico. Rispetto a molti modelli di chat open source e proprietari esistenti, i modelli sintonizzati sulle istruzioni Llama 3.2 dimostrano prestazioni superiori su benchmark di settore ampiamente utilizzati.

I modelli 1B e 3B sono stati potati da Llama 8B, quindi i logit dai modelli 8B e 70B sono stati utilizzati come target a livello di token (distillazione a livello di token). La distillazione della conoscenza è stata utilizzata per recuperare le prestazioni (sono stati addestrati con 9 trilioni di token). Il modello 1B ha 1,24B, quantizzati a intero (Q8\_0), e i parametri 3B, 3,12B, con una quantizzazione Q4\_0, che termina con una dimensione di 1,3 GB e 2 GB, rispettivamente. La sua finestra di contesto è di 131.072 token.

### 1B & 3B Pruning & Distillation



### Installare ed eseguire il Model

```
ollama run llama3.2:1b
```

Eseguendo il modello col comando precedente, dovremmo avere il prompt di Ollama disponibile per inserire una domanda e iniziare a chattare con il modello LLM; ad esempio,

```
>>> What is the capital of France?
```

Quasi immediatamente, otteniamo la risposta corretta:

```
The capital of France is Paris.
```

Utilizzando l'opzione `--verbose` quando si richiama il modello, verranno generate diverse statistiche sulle sue prestazioni (il modello eseguirà il polling solo la prima volta che eseguiamo il comando).

```
(ollama) mjrovai@raspi-5:~$ ollama run llama3.2:1b --verbose
pulling manifest
pulling 74701a8c35f6... 100% [██████████] 1.3 GB
pulling 966de95ca8a6... 100% [██████████] 1.4 KB
pulling fcc5a6bec9da... 100% [██████████] 7.7 KB
pulling a70ff7e570d9... 100% [██████████] 6.0 KB
pulling 4f659ale86d7... 100% [██████████] 485 B

verifying sha256 digest
writing manifest
success
>>> What is the capital of France?
The capital of France is Paris.

total duration: 2.620170326s
load duration: 39.947908ms
prompt eval count: 32 token(s)
prompt eval duration: 1.644773s
prompt eval rate: 19.46 tokens/s
eval count: 8 token(s)
eval duration: 889.941ms
eval rate: 8.99 tokens/s
```

Ogni metrica fornisce informazioni su come il modello elabora gli input e genera gli output. Ecco una ripartizione del significato di ogni metrica:

- **Total Duration (2.620170326s):** Questo è il tempo completo impiegato dall'inizio del comando al completamento della risposta. Comprende il caricamento del modello, l'elaborazione del prompt di input e la generazione della risposta.
- **Load Duration (39.947908ms):** Questa durata indica il tempo necessario per caricare il modello o i componenti necessari nella memoria. Se questo valore è minimo, può suggerire che il modello è stato precaricato o che è stata richiesta solo una configurazione minima.
- **Prompt Eval Count (32 tokens):** Il numero di token nel prompt di input. In NLP, i token sono in genere parole o sotto-parole, quindi questo conteggio include tutti i token che il modello ha valutato per comprendere e rispondere alla query.
- **Prompt Eval Duration (1.644773s):** Misura il tempo impiegato dal modello per valutare o elaborare il prompt di input. Rappresenta la maggior parte della durata totale, il che implica che comprendere la query e preparare una risposta è la parte più dispendiosa in termini di tempo del processo.
- **Prompt Eval Rate (19.46 tokens/s):** Questa frequenza indica la rapidità con cui il modello elabora i token dal prompt di input. Riflette la velocità del modello in termini di comprensione del linguaggio naturale.
- **Eval Count (8 token(s)):** Questo è il numero di token nella risposta del modello, che in questo caso era "The capital of France is Paris."
- **Eval Duration (889.941ms):** Questo è il tempo impiegato per generare l'output in base all'input valutato. È molto più breve della valutazione del prompt, il che suggerisce che generare la risposta è meno complesso o computazionalmente intensivo rispetto alla comprensione del prompt.

- **Eval Rate (8.99 tokens/s):** Simile alla frequenza di valutazione del prompt, indica la velocità con cui il modello genera token di output. È una metrica fondamentale per comprendere l'efficienza del modello nella generazione di output.

Questa ripartizione dettagliata può aiutare a comprendere le richieste computazionali e le caratteristiche delle prestazioni dell'esecuzione di SLM come Llama su dispositivi edge come Raspberry Pi 5. Mostra che mentre la valutazione del prompt richiede più tempo, la generazione effettiva delle risposte è relativamente più rapida. Questa analisi è fondamentale per ottimizzare le prestazioni e diagnosticare potenziali colli di bottiglia nelle applicazioni in tempo reale.

Caricando ed eseguendo il modello 3B, possiamo vedere la differenza nelle prestazioni per lo stesso prompt;

```
marcelo_rovai — mjrovai@raspi-5: ~ — ssh mjrovai@192.168.4.209 — 74x12
(ollama) mjrovai@raspi-5: ~ $ ollama run llama3.2:3b --verbose
>>> What is the capital of France?
The capital of France is Paris.

total duration: 1.808927736s
load duration: 39.854862ms
prompt eval count: 32 token(s)
prompt eval duration: 221.506ms
prompt eval rate: 144.47 tokens/s
eval count: 8 token(s)
eval duration: 1.506376s
eval rate: 5.31 tokens/s
```

Il tasso di valutazione è inferiore, 5,3 token/s rispetto ai 9 token/s del modello più piccolo.

Quando si chiede

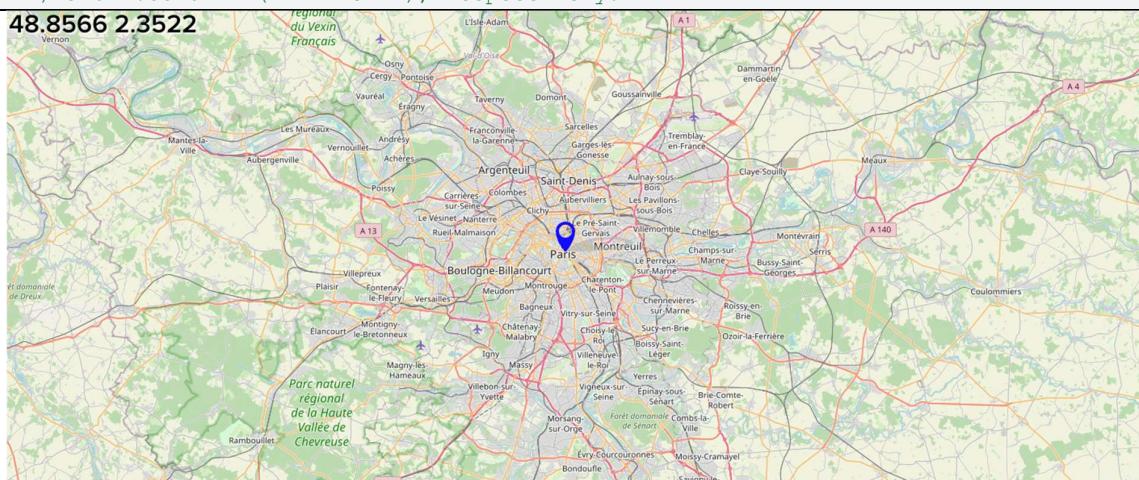
>>> What is the distance between Paris and Santiago, Chile?

Il modello 1B ha risposto 9,841 kilometers (6,093 miles), il che è impreciso, e il modello 3B ha risposto 7,300 miles (11,700 km), il che è vicino alla distanza corretta (11,642 km).

Chiediamo le coordinate di Parigi:

>>> what is the latitude and longitude of Paris?

The latitude and longitude of Paris are 48.8567° N (48°55' 42" N) and 2.3510° E (2°22' 8" E), respectively.

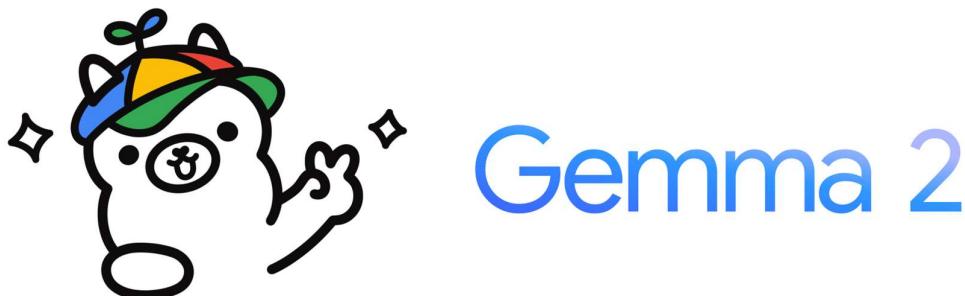


Sia i modelli 1B che 3B hanno dato risposte corrette.

### Google Gemma 2 2B

Installiamo **Gemma 2**, un modello efficiente e ad alte prestazioni disponibile in tre dimensioni: 2B, 9B e 27B. Installeremo **Gemma 2B**, un modello leggero addestrato con 2 trilioni di token che produce risultati sproporzionati imparando da modelli più grandi tramite

distillazione. Il modello ha 2,6 miliardi di parametri e una quantizzazione Q4\_0, che termina con una dimensione di 1,6 GB. La sua finestra di contesto è di 8.192 token.



### Installare ed eseguire il Model

```
ollama run gemma2:2b --verbose
```

Eseguendo il modello col comando precedente, dovremmo avere il prompt di Ollama disponibile per inserire una domanda e iniziare a chattare con il modello LLM; ad esempio,

```
>>> What is the capital of France?
```

Quasi immediatamente, otteniamo la risposta corretta:

```
The capital of France is **Paris**. 🏰
```

E le statistiche.

```
(ollama) mjrovai@raspi-5:~$ ollama run gemma2:2b --verbose
>>> What is the capital of France?
The capital of France is **Paris**. 🏰

total duration: 4.373339337s
load duration: 48.129697ms
prompt eval count: 16 token(s)
prompt eval duration: 1.968114s
prompt eval rate: 8.13 tokens/s
eval count: 13 token(s)
eval duration: 2.313284s
eval rate: 5.62 tokens/s
```

Possiamo vedere che Gemma 2:2B ha più o meno le stesse prestazioni di Llama 3.2:3B, ma con meno parametri.

### Altri esempi:

```
>>> What is the distance between Paris and Santiago, Chile?
```

The distance between Paris, France and Santiago, Chile is approximately \*\*7,000 miles (11,267 kilometers)\*\*.

Keep in mind that this is a straight-line distance, and actual travel distance can vary depending on the chosen routes and any stops along the way. 🛎️

Inoltre, una buona risposta ma meno precisa di Llama3.2:3B.

```
>>> what is the latitude and longitude of Paris?
```

You got it! Here are the latitudes and longitudes of Paris, France:

```
* **Latitude:** 48.8566° N (north)
* **Longitude:** 2.3522° E (east)
```

Let me know if you'd like to explore more about Paris or its location! 🇫🇷

Una risposta buona e precisa (un po' più prolissa delle risposte del lama).

### Microsoft Phi3.5 3.8B

Prendiamo un modello più grande (ma comunque “tiny”), il **PHI3.5**, un modello aperto all'avanguardia leggero da 3,8B di Microsoft. Il modello appartiene alla famiglia di modelli Phi-3 e supporta la lunghezza del contesto `token 128K` e le lingue: arabo, cinese, ceco, danese, olandese, inglese, finlandese, francese, tedesco, ebraico, ungherese, italiano, giapponese, coreano, norvegese, polacco, portoghese, russo, spagnolo, svedese, tailandese, turco e ucraino.

La dimensione del modello, in termini di byte, dipenderà dal formato di quantizzazione specifico utilizzato. La dimensione può variare dalla quantizzazione a 2 bit (`q2_k`) di 1,4 GB (prestazioni più elevate/qualità inferiore) alla quantizzazione a 16 bit (fp-16) di 7,6 GB (prestazioni più basse/qualità più elevata).

Eseguiamo la quantizzazione a 4 bit (`q4_0`), che richiederà 2,2 GB di RAM, con un compromesso intermedio per quanto riguarda la qualità dell'output e le prestazioni.

```
ollama run phi3.5:3.8b --verbose
```

Si può usare `run o pull` per scaricare il modello. Ciò che accade è che Ollama tiene nota dei modelli estratti e, una volta che PHI3 non esiste, prima di eseguirlo, Ollama lo estraе.

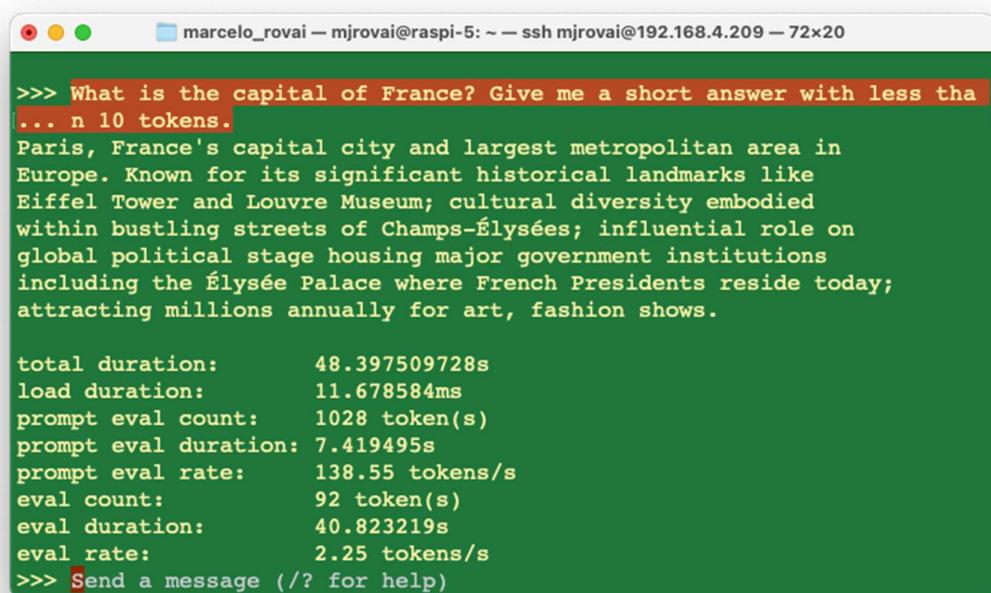
Immettiamo lo stesso prompt usato prima:

```
>>> What is the capital of France?
```

```
The capital of France is Paris. It's extradites significant historical, cultural, and political importance to the country as well as being a major European city known for its art, fashion, gastronomy, and culture. Its influence extends beyond national borders, with millions of tourists visiting each year from around the globe. The Seine River flows through Paris before it reaches the broader English Channel at Le Havre. Moreover, France is one of Europe's leading economies with its capital playing a key role
```

```
...
```

La risposta è stata molto “verbosa”, specifichiamo un prompt migliore:



A screenshot of a terminal window titled "marcelo\_rovai — mjrovai@raspi-5: ~ — ssh mjrovai@192.168.4.209 — 72x20". The terminal shows the following interaction:

```
>>> What is the capital of France? Give me a short answer with less than 10 tokens.
Paris, France's capital city and largest metropolitan area in Europe. Known for its significant historical landmarks like Eiffel Tower and Louvre Museum; cultural diversity embodied within bustling streets of Champs-Élysées; influential role on global political stage housing major government institutions including the Élysée Palace where French Presidents reside today; attracting millions annually for art, fashion shows.

total duration: 48.397509728s
load duration: 11.678584ms
prompt eval count: 1028 token(s)
prompt eval duration: 7.419495s
prompt eval rate: 138.55 tokens/s
eval count: 92 token(s)
eval duration: 40.823219s
eval rate: 2.25 tokens/s
>>> Send a message (/? for help)
```

In questo caso, la risposta è stata comunque più lunga di quanto ci aspettassimo, con una velocità di valutazione di 2,25 token/s, più del doppio di quella di Gemma e Llama.

Scegliere il prompt più appropriato è una delle competenze più importanti da utilizzare con gli LLM, indipendentemente dalle dimensioni.

Quando abbiamo posto le stesse domande su distanza e latitudine/longitudine, non abbiamo ottenuto una buona risposta per una distanza di 13,507 kilometers (8,429 miles), ma andava bene per le coordinate. Di nuovo, avrebbe potuto essere meno verbosa (più di 200 token per ogni risposta).

Possiamo usare qualsiasi modello come assistente poiché la loro velocità è relativamente decente, ma al 24 settembre (2023), Llama2:3B è una scelta migliore. Si dovrebbero provare altri modelli, a seconda delle esigenze.  [Open LLM Leaderboard](#) può dare un'idea sui migliori modelli in termini di dimensioni, benchmark, licenza, ecc.

Il miglior modello da usare è quello adatto alle specifiche necessità. Inoltre, si tenga presente che questo campo si evolve con nuovi modelli ogni giorno,

### **Modelli Multimodali**

I modelli multimodali sono sistemi di intelligenza artificiale (IA) in grado di elaborare e comprendere informazioni provenienti da più fonti, come immagini, testo, audio e video. Nel nostro contesto, gli LLM multimodali possono elaborare vari input, tra cui testo, immagini e audio, come prompt e convertire tali prompt in vari output, non solo il tipo di sorgente.

Qui lavoreremo con [LLaVA-Phi-3](#), un modello LLaVA ottimizzato da Phi 3 Mini 4k. Ha solidi benchmark di prestazioni che sono alla pari con il modello originale [LLaVA](#) (Large Language and Vision Assistant).

LLaVA-Phi-3 è un modello multimodale di grandi dimensioni addestrato end-to-end progettato per comprendere e generare contenuti in base a input visivi (immagini) e istruzioni testuali. Combina le capacità di un codificatore visivo e di un modello linguistico per elaborare e rispondere a input multimodali.

Installiamo il modello:

```
ollama run llava-phi3:3.8b --verbose
```

Cominciamo con un input di testo:

```
>>> You are a helpful AI assistant. What is the capital of France?
```

```
As an AI language model, I can tell you that the capital of France
is Paris. It's not only the largest city in the country but also
serves as its political and administrative center. Paris is known
for its iconic landmarks such as the Eiffel Tower, Notre-Dame
Cathedral, and the Louvre Museum. The city has a rich history,
beautiful architecture, and is widely considered to be one of the
most romantic cities in the world.
```

La risposta ha richiesto circa 30 secondi, con una velocità di valutazione di 3,93 token/s!

Niente male!

Ma vediamo di immettere un'immagine come input. Per questo, creiamo una directory per lavorare:

```
cd Documents/
mkdir OLLAMA
cd OLLAMA
```

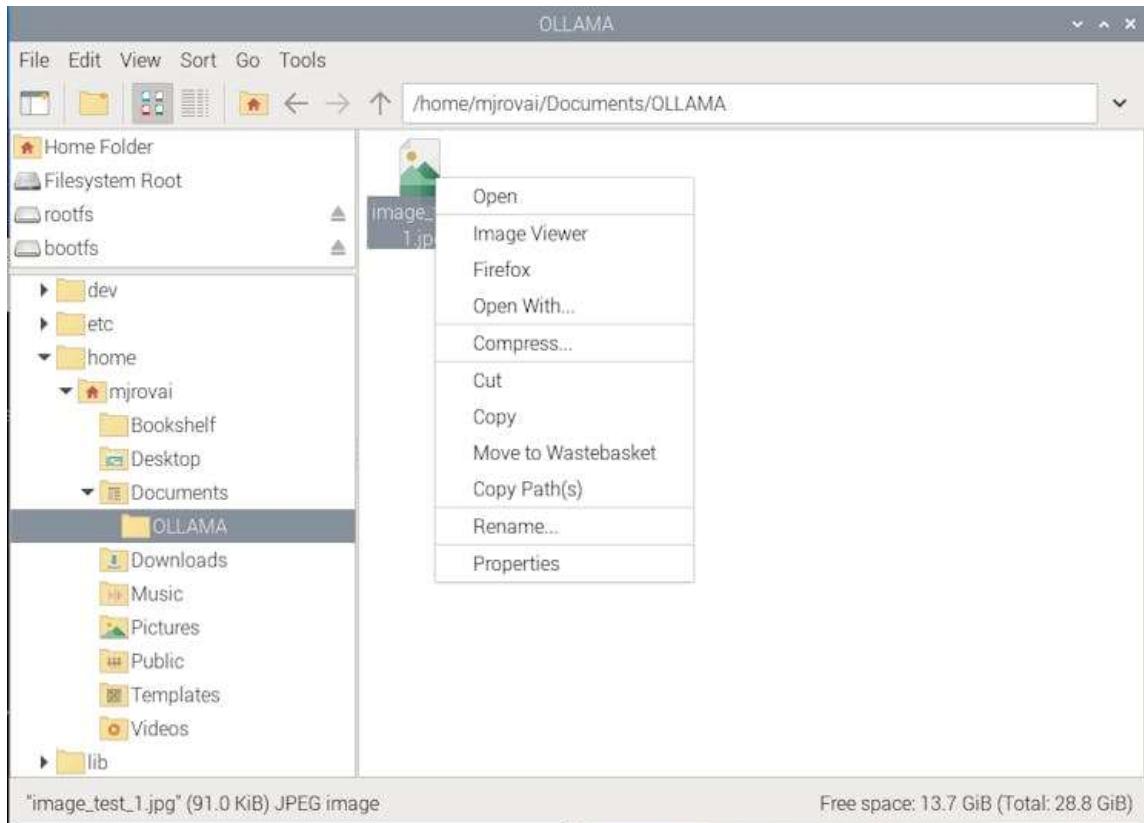
Scarichiamo un'immagine 640x320 da Internet, per esempio (Wikipedia: [Paris, France](#)):



Utilizzando FileZilla, ad esempio, carichiamo l'immagine nella cartella OLLAMA sul Raspi-5 e chiamiamola `image_test_1.jpg`. Dovremmo avere l'intero path dell'immagine (possiamo usare `pwd` per ottenerlo).

```
/home/mjrovai/Documents/OLLAMA/image_test_1.jpg
```

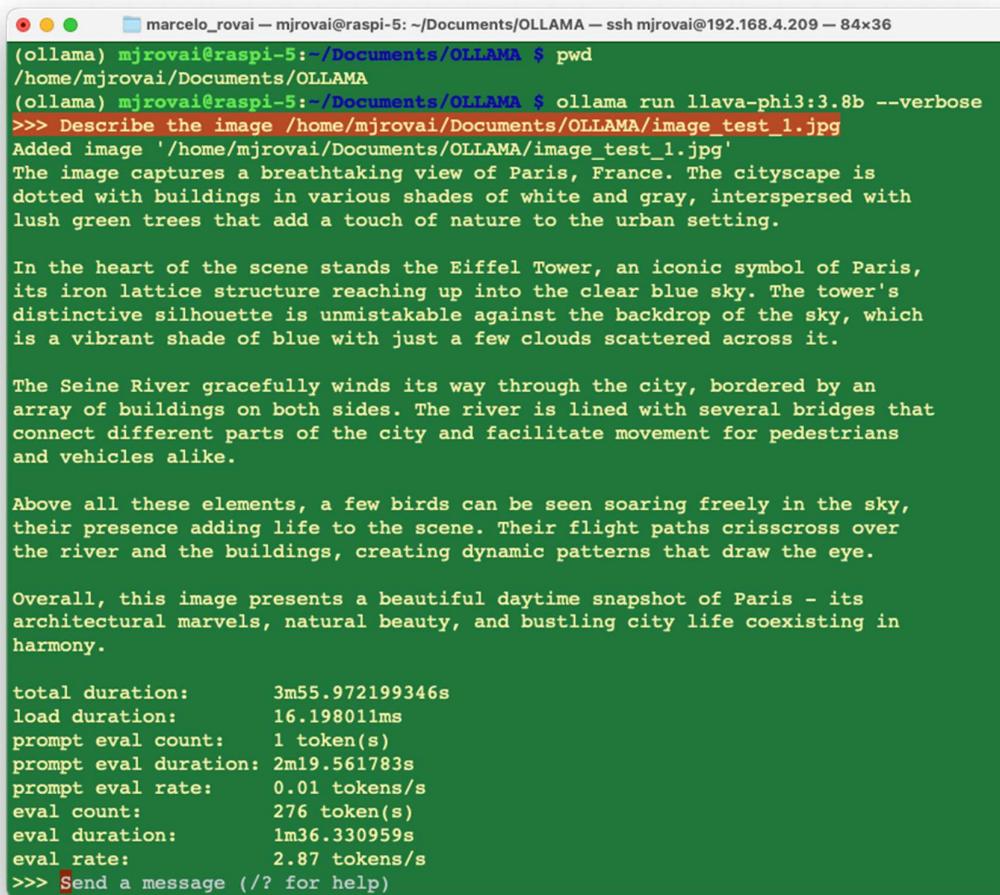
Se si usa un desktop, si può copiare il path dell'immagine cliccando sull'immagine con il tasto destro del mouse.



Immettiamo questo prompt:

```
>>> Describe the image /home/mjrovai/Documents/OLLAMA/image_test_1.jpg
```

Il risultato è stato ottimo, ma la latenza complessiva è stata significativa: quasi 4 minuti per eseguire l'inferenza.



```

marcelo_rovai — mjrovai@raspi-5: ~/Documents/OLLAMA — ssh mjrovai@192.168.4.209 — 84x36
(ollama) mjrovai@raspi-5:~/Documents/OLLAMA $ pwd
/home/mjrovai/Documents/OLLAMA
(ollama) mjrovai@raspi-5:~/Documents/OLLAMA $ ollama run llava-phi3:3.8b --verbose
>>> Describe the image /home/mjrovai/Documents/OLLAMA/image_test_1.jpg
Added image '/home/mjrovai/Documents/OLLAMA/image_test_1.jpg'
The image captures a breathtaking view of Paris, France. The cityscape is dotted with buildings in various shades of white and gray, interspersed with lush green trees that add a touch of nature to the urban setting.

In the heart of the scene stands the Eiffel Tower, an iconic symbol of Paris, its iron lattice structure reaching up into the clear blue sky. The tower's distinctive silhouette is unmistakable against the backdrop of the sky, which is a vibrant shade of blue with just a few clouds scattered across it.

The Seine River gracefully winds its way through the city, bordered by an array of buildings on both sides. The river is lined with several bridges that connect different parts of the city and facilitate movement for pedestrians and vehicles alike.

Above all these elements, a few birds can be seen soaring freely in the sky, their presence adding life to the scene. Their flight paths crisscross over the river and the buildings, creating dynamic patterns that draw the eye.

Overall, this image presents a beautiful daytime snapshot of Paris - its architectural marvels, natural beauty, and bustling city life coexisting in harmony.

total duration: 3m55.972199346s
load duration: 16.198011ms
prompt eval count: 1 token(s)
prompt eval duration: 2m19.561783s
prompt eval rate: 0.01 tokens/s
eval count: 276 token(s)
eval duration: 1m36.330959s
eval rate: 2.87 tokens/s
>>> Send a message (/? for help)

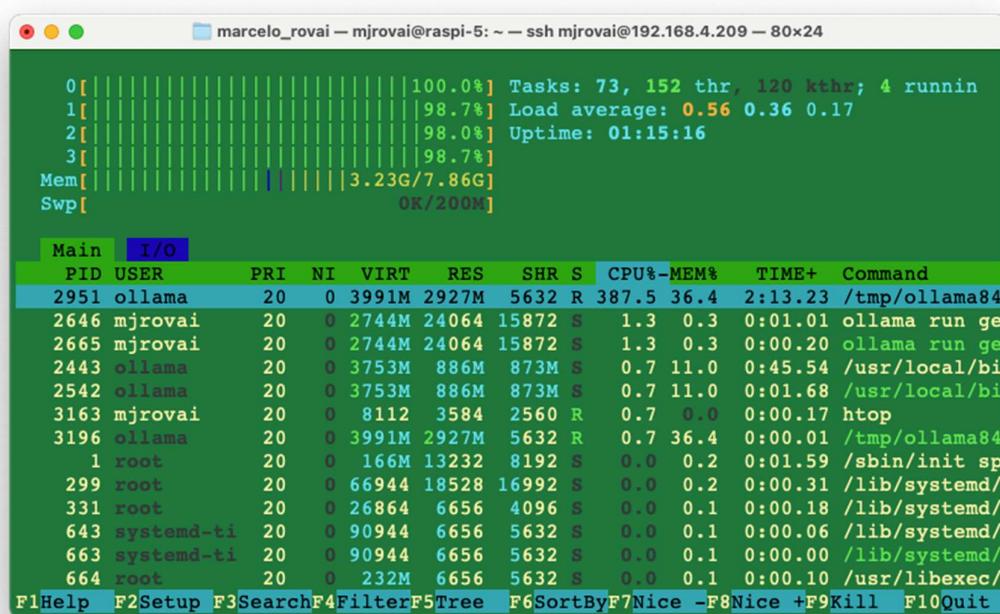
```

### Ispezione delle risorse locali

Utilizzando htop, possiamo monitorare le risorse in esecuzione sul nostro dispositivo.

[htop](#)

Durante il periodo in cui il modello è in esecuzione, possiamo ispezionare le risorse:



PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%-MEM%	TIME+	Command
2951	ollama	20	0	3991M	2927M	5632	R	387.5 36.4	2:13.23	/tmp/ollama84
2646	mjrovai	20	0	2744M	24064	15872	S	1.3 0.3	0:01.01	ollama run ge
2665	mjrovai	20	0	2744M	24064	15872	S	1.3 0.3	0:00.20	ollama run ge
2443	ollama	20	0	3753M	886M	873M	S	0.7 11.0	0:45.54	/usr/local/bi
2542	ollama	20	0	3753M	886M	873M	S	0.7 11.0	0:01.68	/usr/local/bi
3163	mjrovai	20	0	8112	3584	2560	R	0.7 0.0	0:00.17	htop
3196	ollama	20	0	3991M	2927M	5632	R	0.7 36.4	0:00.01	/tmp/ollama84
1	root	20	0	166M	13232	8192	S	0.0 0.2	0:01.59	/sbin/init sp
299	root	20	0	66944	18528	16992	S	0.0 0.2	0:00.31	/lib/systemd/
331	root	20	0	26864	6656	4096	S	0.0 0.1	0:00.18	/lib/systemd/
643	systemd-ti	20	0	90944	6656	5632	S	0.0 0.1	0:00.06	/lib/systemd/
663	systemd-ti	20	0	90944	6656	5632	S	0.0 0.1	0:00.00	/lib/systemd/
664	root	20	0	232M	6656	5632	S	0.0 0.1	0:00.10	/usr/libexec/

Tutte e quattro le CPU funzionano a quasi il 100% della loro capacità e la memoria utilizzata con il modello caricato è di 3.24GB. Uscendo da Ollama, la memoria scende a circa 377 MB (senza desktop).

È anche essenziale monitorare la temperatura. Quando si esegue Raspberry con un desktop, è possibile visualizzare la temperatura sulla barra delle applicazioni:



Se si è “headless”, la temperatura può essere monitorata col comando:

```
vcgencmd measure_temp
```

Se non si fa nulla, la temperatura è di circa 50°C per le CPU in esecuzione all’1%. Durante l’inferenza, con le CPU al 100%, la temperatura può salire fino a quasi 70°C. Questo è OK e significa che il dissipatore attivo funziona, mantenendo la temperatura al di sotto di 80°C / 85°C (il suo limite).

## Libreria Python Ollama

Finora, abbiamo esplorato la capacità di chat degli SLM utilizzando la riga di comando su un terminale. Tuttavia, vogliamo integrare quei modelli nei nostri progetti, quindi Python sembra essere la strada giusta. La buona notizia è che Ollama ha una libreria del genere.

La [libreria Python Ollama](#) semplifica l’interazione con modelli LLM avanzati, consentendo risposte e capacità più sofisticate, oltre a fornire il modo più semplice per integrare progetti Python 3.8+ con [Ollama](#).

Per una migliore comprensione di come creare app usando Ollama con Python, possiamo seguire i [video di Matt Williams](#), come quello qui sotto:

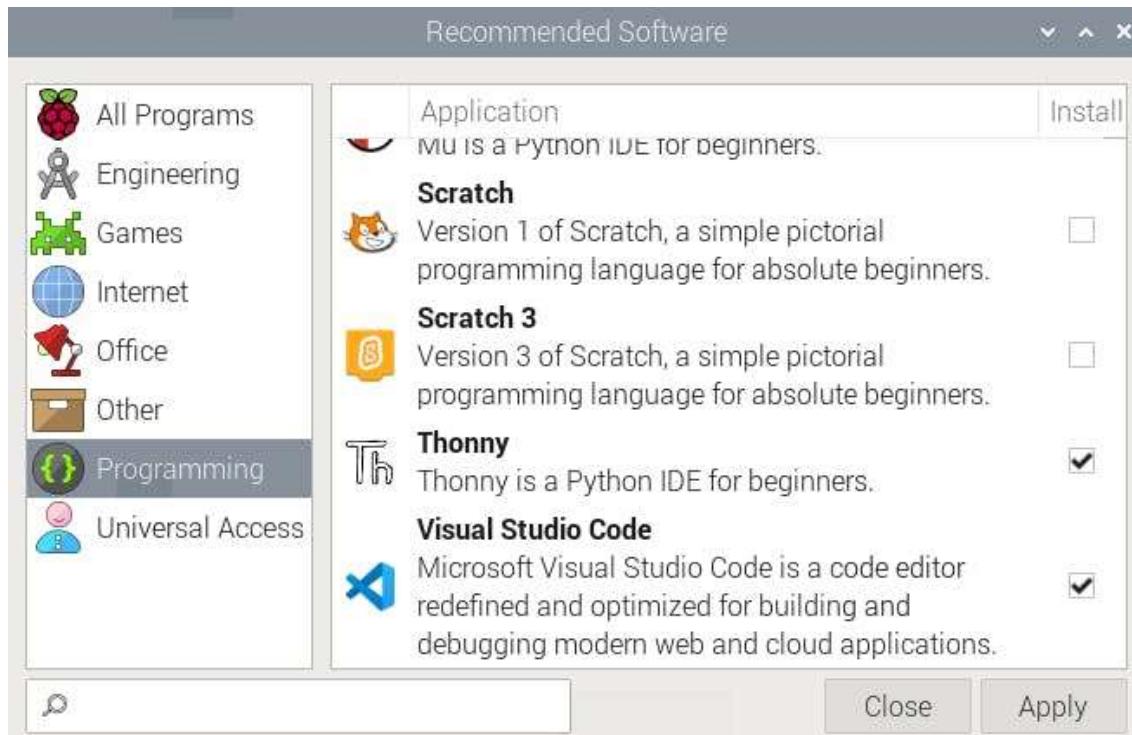
[https://www.youtube.com/embed/\\_4K20tOsXK8](https://www.youtube.com/embed/_4K20tOsXK8)

### Installazione:

Nel terminale, eseguire il comando:

```
pip install ollama
```

Avremo bisogno di un editor di testo o di un IDE per creare uno script Python. Se si esegue Raspberry OS su un desktop, diverse opzioni, come Thonny e Geany, sono già state installate di default (accessibili tramite [Menu] [Programming]). Si possono scaricare altri IDE, come Visual Studio Code, da [Menu] [Recommended Software]. Quando si apre la finestra, si va su [Programming], e si seleziona l’opzione che si preferisce e si preme [Apply].



Se si preferisce usare Jupyter Notebook per lo sviluppo:

```
pip install jupyter
jupyter notebook --generate-config
```

Per eseguire Jupyter Notebook, si lancia il comando (cambiare l'indirizzo IP per il proprio):

```
jupyter notebook --ip=192.168.4.209 --no-browser
```

Sul terminale, si può vedere l'indirizzo URL locale per aprire il notebook:

```
(ollama) mjr ovai@raspi-5: ~ - ssh mjr ovai@192.168.4.209 - 130x31
[ollama] mjr ovai@raspi-5: ~ $ jupyter notebook --ip=192.168.4.209 --no-browser
[I 2024-09-25 15:25:03.768 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2024-09-25 15:25:03.772 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2024-09-25 15:25:03.776 ServerApp] jupyterlab | extension was successfully linked.
[I 2024-09-25 15:25:03.780 ServerApp] notebook | extension was successfully linked.
[I 2024-09-25 15:25:04.022 ServerApp] notebook_shim | extension was successfully linked.
[I 2024-09-25 15:25:04.032 ServerApp] notebook_shim | extension was successfully loaded.
[I 2024-09-25 15:25:04.038 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2024-09-25 15:25:04.039 LabApp] JupyterLab extension loaded from /home/mjr ovai/ollama/lib/python3.11/site-packages/jupyterlab
[I 2024-09-25 15:25:04.039 LabApp] JupyterLab application directory is /home/mjr ovai/ollama/share/jupyter/lab
[I 2024-09-25 15:25:04.039 LabApp] Extension Manager is 'pypi'.
[I 2024-09-25 15:25:04.082 ServerApp] jupyterlab | extension was successfully loaded.
[I 2024-09-25 15:25:04.085 ServerApp] notebook | extension was successfully loaded.
[I 2024-09-25 15:25:04.085 ServerApp] Serving notebooks from local directory: /home/mjr ovai
[I 2024-09-25 15:25:04.088 ServerApp] Jupyter Server 2.14.2 is running at:
[I 2024-09-25 15:25:04.088 ServerApp] http://192.168.4.209:8888/tree?token=79a989d699951f61d357cd5aa1146d350eaf3ed1471a422
[I 2024-09-25 15:25:04.088 ServerApp] http://127.0.0.1:8888/tree?token=79a989d699951f61d357cd5aa1146d350eaf3ed1471a422
[I 2024-09-25 15:25:04.088 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 2024-09-25 15:25:04.087 ServerApp]

To access the server, open this file in a browser:
file:///home/mjr ovai/.local/share/jupyter/runtime/jpserver-10313-open.html
Or copy and paste one of these URLs:
http://192.168.4.209:8888/tree?token=79a989d699951f61d357cd5aa1146d350eaf3ed1471a422
http://127.0.0.1:8888/tree?token=79a989d699951f61d357cd5aa1146d350eaf3ed1471a422
[I 2024-09-25 15:25:04.098 ServerApp] Skipped non-installed server(s): bash-language-server, dockerfile-language-server-nodejs, javascript-typescript-langsServer, jedi-language-server, julia-language-server, pyright, python-language-server, python-lsp-server, r-language-server, sql-language-server, texlab, typescript-language-server, unified-language-server, vscode-css-languageserver-bin, vscode-html-languageserver-bin, vscode-json-languageserver-bin, yaml-language-server
```

Possiamo accedervi da un altro computer inserendo l'indirizzo IP del Raspberry Pi e il token fornito in un browser web (dovremmo copiarlo dal terminale).

Nella nostra directory di lavoro nel Raspi, creeremo un nuovo notebook Python 3.

Entriamo con uno script molto semplice per verificare i modelli installati:

```
import ollama
ollama.list()
```

Tutti i modelli verranno stampati come un dizionario, ad esempio:

```
{'name': 'gemma2:2b',
'model': 'gemma2:2b',
```

```
'modified_at': '2024-09-24T19:30:40.053898094+01:00',
'size': 1629518495,
'digest': '8ccf136fdd5298f3ffe2d69862750ea7fb56555fa4d5b18c04e3fa4d82ee09d7',
'details': {'parent_model': '',
'format': 'gguf',
'family': 'gemma2',
'families': ['gemma2'],
'parameter_size': '2.6B',
'quantization_level': 'Q4_0'}}]
```

Ripetiamo una delle domande che abbiamo fatto prima, ma ora usando `ollama.generate()` dalla libreria Python Ollama. Questa API genererà una risposta per il prompt specificato con il modello fornito. Questo è un endpoint di streaming, quindi ci saranno una serie di risposte. L'oggetto di risposta finale includerà statistiche e dati aggiuntivi dalla richiesta.

```
MODEL = 'gemma2:2b'
PROMPT = 'What is the capital of France?'

res = ollama.generate(model=MODEL, prompt=PROMPT)
print(res)
```

Nel caso in cui si stia eseguendo il codice come script Python, lo si deve salvare, ad esempio, `test_ollama.py`. Si può usare l'IDE per eseguirlo o farlo direttamente sul terminale. Da ricordare, inoltre, che si deve sempre chiamare il modello e definirlo quando si esegue uno script stand-alone.

```
python test_ollama.py
```

Di conseguenza, avremo la risposta del modello in un formato JSON:

```
{'model': 'gemma2:2b', 'created_at': '2024-09-25T14:43:31.869633807Z',
'response': 'The capital of France is **Paris**. FR \n', 'done': True,
'done_reason': 'stop', 'context': [106, 1645, 108, 1841, 603, 573, 6037, 576,
6081, 235336, 107, 108, 106, 2516, 108, 651, 6037, 576, 6081, 603, 5231, 29437,
168428, 235248, 244304, 241035, 235248, 108], 'total_duration': 24259469458,
'load_duration': 19830013859, 'prompt_eval_count': 16, 'prompt_eval_duration':
1908757000, 'eval_count': 14, 'eval_duration': 2475410000}
```

Come possiamo vedere, vengono generate diverse informazioni, come:

- **response:** il testo di output principale generato dal modello in risposta al nostro prompt.
  - The capital of France is \*\*Paris\*\*. FR
- **context:** gli ID token che rappresentano l'input e il contesto utilizzati dal modello. I token sono rappresentazioni numeriche del testo utilizzate per l'elaborazione dal modello linguistico.
  - [106, 1645, 108, 1841, 603, 573, 6037, 576, 6081, 235336, 107, 108, 106,
2516, 108, 651, 6037, 576, 6081, 603, 5231, 29437, 168428, 235248, 244304,
241035, 235248, 108]

Le Metriche delle Prestazioni:

- **total\_duration:** Tempo totale impiegato per l'operazione in nanosecondi. In questo caso, circa 24,26 secondi.
- **load\_duration:** Tempo impiegato per caricare il modello o i componenti in nanosecondi. Circa 19,83 secondi.
- **prompt\_eval\_duration:** Tempo impiegato per valutare il prompt in nanosecondi. Circa 16 nanosecondi.
- **eval\_count:** Numero di token valutati durante la generazione. Qui, 14 token.
- **eval\_duration:** Tempo impiegato dal modello per generare la risposta in nanosecondi. Circa 2,5 secondi.

Ma ciò che vogliamo è la semplice 'response' e, forse per l'analisi, la durata totale dell'inferenza, quindi modifichiamo il codice per estrarre dal dizionario:

```
print(f"\n{res['response']}")
print(f"\n [INFO] Total Duration: {(res['total_duration']/1e9):.2f} seconds")
```

Ora, abbiamo:

```
The capital of France is **Paris**. FR
```

```
[INFO] Total Duration: 24.26 seconds
```

### Utilizzo di Ollama.chat()

Un altro modo per ottenere la nostra risposta è utilizzare `ollama.chat()`, che genera il messaggio successivo in una chat con un modello fornito. Questo è un endpoint di streaming, quindi si verificheranno una serie di risposte. Lo streaming può essere disabilitato utilizzando `"stream": false`. L'oggetto di risposta finale includerà anche statistiche e dati aggiuntivi dalla richiesta.

```
PROMPT_1 = 'What is the capital of France?'
```

```
response = ollama.chat(model=MODEL, messages=[{'role': 'user', 'content': PROMPT_1},])
resp_1 = response['message']['content']
print(f"\n{resp_1}")
print(f"\n [INFO] Total Duration: {(res['total_duration']/1e9):.2f} seconds")
```

La risposta è la stessa di prima.

Una considerazione importante è che usando `ollama.generate()`, la risposta è “chiara” dalla “memoria” del modello dopo la fine dell’inferenza (usata solo una volta), ma se vogliamo mantenere una conversazione, dobbiamo usare `ollama.chat()`. Vediamolo in azione:

```
PROMPT_1 = 'What is the capital of France?'
response = ollama.chat(model=MODEL, messages=[{'role': 'user', 'content': PROMPT_1},])
resp_1 = response['message']['content']
print(f"\n{resp_1}")
print(f"\n [INFO] Total Duration: {(response['total_duration']/1e9):.2f} seconds")
```

```
PROMPT_2 = 'and of Italy?'
response = ollama.chat(model=MODEL, messages=[{'role': 'user', 'content': PROMPT_1}, {'role': 'assistant', 'content': resp_1}, {'role': 'user', 'content': PROMPT_2},])
resp_2 = response['message']['content']
print(f"\n{resp_2}")
print(f"\n [INFO] Total Duration: {(response['total_duration']/1e9):.2f} seconds")
```

Nel codice sopra, stiamo eseguendo due query e il secondo prompt considera il risultato del primo.

Ecco come ha risposto il modello:

```
The capital of France is **Paris**. FR
```

```
[INFO] Total Duration: 2.82 seconds
```

```
The capital of Italy is **Rome**. IT
```

```
[INFO] Total Duration: 4.46 seconds
```

### Ottenere una descrizione dell’immagine:

Allo stesso modo in cui abbiamo utilizzato il modello `LlaVa-PHI-3` con la riga di comando per analizzare un’immagine, lo stesso può essere fatto qui con Python. Usiamo la stessa immagine di Parigi, ma ora con `ollama.generate()`:

```
MODEL = 'llava-phi3:3.8b'
PROMPT = "Describe this picture"

with open('image_test_1.jpg', 'rb') as image_file:
 img = image_file.read()

response = ollama.generate(
 model=MODEL,
 prompt=PROMPT,
 images= [img]
)
print(f"\n{response['response']}")
print(f"\n [INFO] Total Duration: {(res['total_duration']/1e9):.2f} seconds")
```

Ecco il risultato:

This image captures the iconic cityscape of Paris, France. The vantage point is high, providing a panoramic view of the Seine River that meanders through the heart of the city. Several bridges arch gracefully over the river, connecting different parts of the city. The Eiffel Tower, an iron lattice structure with a pointed top and two antennas on its summit, stands tall in the background, piercing the sky. It is painted in a light gray color, contrasting against the blue sky speckled with white clouds.

The buildings that line the river are predominantly white or beige, their uniform color palette broken occasionally by red roofs peeking through. The Seine River itself appears calm and wide, reflecting the city's architectural beauty in its surface. On either side of the river, trees add a touch of green to the urban landscape.

The image is taken from an elevated perspective, looking down on the city. This viewpoint allows for a comprehensive view of Paris's beautiful architecture and layout. The relative positions of the buildings, bridges, and other structures create a harmonious composition that showcases the city's charm.

In summary, this image presents a serene day in Paris, with its architectural marvels - from the Eiffel Tower to the river-side buildings - all bathed in soft colors under a clear sky.

[INFO] Total Duration: 256.45 seconds

Il modello ha impiegato circa 4 minuti (256,45 s) per restituire una descrizione dettagliata dell'immagine.

Nel notebook [10-Ollama\\_Python\\_Library](#), è possibile trovare gli esperimenti con la libreria Python Ollama.

### **Chiamata di Funzione**

Finora, possiamo osservare che utilizzando la risposta del modello in una variabile, possiamo incorporarla efficacemente in progetti reali. Tuttavia, sorge un problema importante quando il modello fornisce risposte diverse allo stesso input. Ad esempio, supponiamo di aver bisogno solo del nome della capitale di un paese e delle sue coordinate come risposta del modello negli esempi precedenti, senza ulteriori informazioni, anche quando si utilizzano modelli dettagliati come Microsoft Phi. Per garantire risposte coerenti, possiamo utilizzare la "chiamata di funzione Ollama", che è completamente compatibile con l'API OpenAI.

### **Ma cos'è esattamente la “chiamata di funzione”?**

Nell'intelligenza artificiale moderna, la chiamata di funzione con Large Language Models (LLM) consente a questi modelli di eseguire azioni che vanno oltre la generazione di testo. Integrandosi con funzioni o API esterne, gli LLM possono accedere a dati in tempo reale, automatizzare attività e interagire con vari sistemi.

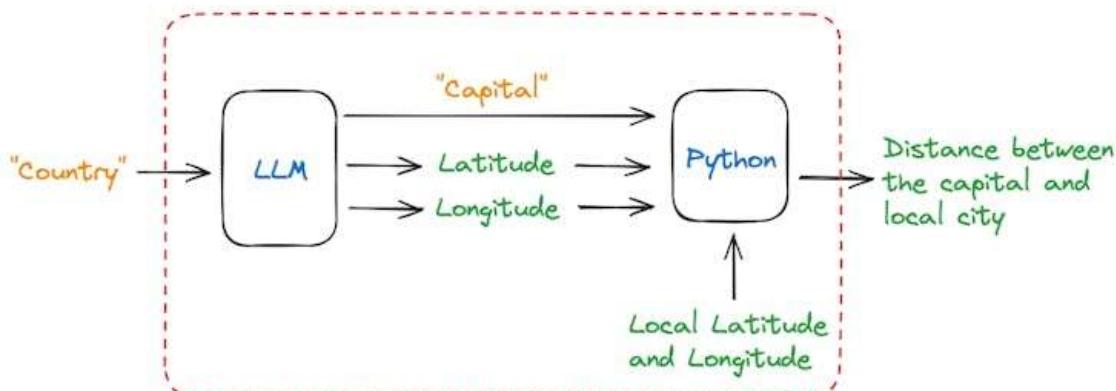
Ad esempio, invece di rispondere semplicemente a una query sul meteo, un LLM può chiamare un'API meteo per recuperare le condizioni attuali e fornire informazioni accurate e aggiornate. Questa capacità migliora la pertinenza e l'accuratezza delle risposte del modello e lo rende uno strumento potente per guidare flussi di lavoro e automatizzare i processi, trasformandolo in un partecipante attivo nelle applicazioni del mondo reale.

Per maggiori dettagli sulla “Function Calling” [chiamata di funzione](#), c’è questo video realizzato da [Marvin Prison](#):

<https://www.youtube.com/embed/eHfMCtsb1o>

### **Creiamo un progetto.**

Vogliamo creare un'app in cui l'utente inserisce il nome di un Paese e ottiene, come output, la distanza in km dalla capitale di tale Paese e la posizione dell'app (per semplicità, useremo Santiago del Cile come posizione dell'app).



Una volta che l'utente inserisce il nome di un paese, il modello restituirà il nome della sua capitale (come stringa) e la latitudine e la longitudine di tale città (in float). Utilizzando queste coordinate, possiamo usare una semplice libreria Python ([haversine](#)) per calcolare la distanza tra quei 2 punti.

L'idea di questo progetto è dimostrare una combinazione di interazione del modello linguistico, gestione dei dati strutturati con Pydantic e calcoli geospaziali utilizzando la formula di Haversine (informatica tradizionale).

Per prima cosa, installiamo alcune librerie. Oltre a [Haversine](#), la principale è la [libreria Python OpenAI](#), che fornisce un comodo accesso all'API REST OpenAI da qualsiasi applicazione Python 3.7+. L'altra è [Pydantic](#) (e [instructor](#)), una libreria robusta di convalida dei dati e gestione delle impostazioni progettata da Python per migliorare la robustezza e l'affidabilità della nostra base di codice. In breve, [Pydantic](#) ci aiuterà a garantire che la risposta del nostro modello sia sempre coerente.

```
pip install haversine
pip install openai
pip install pydantic
pip install instructor
```

Ora, dovremmo creare uno script Python progettato per interagire con il nostro modello (LLM) per determinare le coordinate della capitale di un paese e calcolare la distanza da Santiago del Cile a quella capitale.

Diamo un'occhiata al codice:

### **1. Importazione delle Librerie**

```
import sys
from haversine import haversine
from openai import OpenAI
from pydantic import BaseModel, Field
import instructor
```

- **sys**: Fornisce accesso a parametri e funzioni specifici del sistema. Viene utilizzato per ottenere argomenti dalla riga di comando.
- **haversine**: Una funzione della libreria haversine che calcola la distanza tra due punti geografici utilizzando la formula Haversine.
- **openAI**: Un modulo per interagire con l'API OpenAI (anche se viene utilizzato insieme a una configurazione locale, Ollama). Qui tutto è offline.
- **pydantic**: Fornisce la convalida dei dati e la gestione delle impostazioni utilizzando annotazioni di tipo Python. Viene utilizzato per definire la struttura dei dati di risposta previsti.
- **instructor**: Un modulo viene utilizzato per applicare patch al client OpenAI per funzionare in una modalità specifica (probabilmente correlata alla gestione dei dati strutturati).

## 2. Definizione di Input e Modello

```
country = sys.argv[1] # Get the country from command-line arguments
MODEL = 'phi3.5:3.8b' # The name of the model to be used
mylat = -33.33 # Latitude of Santiago de Chile
mylon = -70.51 # Longitude of Santiago de Chile
```

- **country:** In uno script Python, è possibile ottenere il nome del paese dagli argomenti della riga di comando. In un notebook Jupyter, possiamo immettere il suo nome, ad esempio,
  - `country = "France"`
- **MODEL:** Specifica il modello utilizzato, che è, in questo esempio, phi3.5.
- **mylat e mylon:** Coordinate di Santiago del Cile, utilizzate come punto di partenza per il calcolo della distanza.

## 3. Definizione della Struttura dei Dati di Risposta

```
class CityCoord(BaseModel):
 city: str = Field(..., description="Name of the city")
 lat: float = Field(..., description="Decimal Latitude of the city")
 lon: float = Field(..., description="Decimal Longitude of the city")
```

- **CityCoord:** Un modello Pydantic che definisce la struttura prevista della risposta dal LLM. Si aspetta tre campi: city (nome della città), lat (latitudine) e lon (longitudine).

## 4. Impostazione del Client OpenAI

```
client = instructor.patch(
 OpenAI(
 base_url="http://localhost:11434/v1", # Local API base URL (Ollama)
 api_key="ollama", # API key (not used)
),
 mode=instructor.Mode.JSON, # Mode for structured JSON output
)
```

- **OpenAI:** Questa configurazione inizializza un client OpenAI con un URL di base locale e una chiave API (ollama). Utilizza un server locale.
- **instructor.patch:** Applica patch al client OpenAI per funzionare in modalità JSON, abilitando un output strutturato che corrisponde al modello Pydantic.

## 5. Generazione della Risposta

```
resp = client.chat.completions.create(
 model=MODEL,
 messages=[
 {
 "role": "user",
 "content": f"return the decimal latitude and decimal longitude \
of the capital of the {country}."
 }
],
 response_model=CityCoord,
 max_retries=10
)
```

- **client.chat.completions.create:** Chiama l'LLM per generare una risposta.
- **model:** Specifica il modello da utilizzare (llava-phi3).
- **messages:** Contiene il prompt per l'LLM, che chiede latitudine e longitudine della capitale del paese specificato.
- **response\_model:** Indica che la risposta deve essere conforme al modello CityCoord.
- **max\_retries:** Numero massimo di tentativi di ripetizione se la richiesta fallisce.

## 6. Calcolo della Distanza

```
distance = haversine((mylat, mylon), (resp.lat, resp.lon), unit='km')
print(f"Santiago de Chile is about {int(round(distance, -1))} km \
kilometers away from {resp.city}.")
```

- **haversine**: Calcola la distanza tra Santiago del Cile e la capitale restituita dall'LLM utilizzando le rispettive coordinate.
- **(mylat, mylon)**: Coordinate di Santiago del Cile.
- **resp.city**: Nome della capitale del paese
- **(resp.lat, resp.lon)**: Le coordinate della capitale sono fornite dalla risposta dell'LLM.
- **unit='km'**: Specifica che la distanza deve essere calcolata in chilometri.
- **print**: Restituisce la distanza, arrotondata ai 10 chilometri più vicini, con migliaia di separatori per una migliore leggibilità.

### Esecuzione del codice

Se inseriamo paesi diversi, ad esempio Francia, Colombia e Stati Uniti, possiamo notare che riceviamo sempre le stesse informazioni strutturate:

Santiago de Chile is about 8,060 kilometers away from Washington, D.C..

Santiago de Chile is about 4,250 kilometers away from Bogotá.

Santiago de Chile is about 11,630 kilometers away from Paris.

Eseguendo il codice come script, il risultato verrà stampato sul terminale:

```
mjrovai@rpi-5:~/Documents/OLLAMA
File Edit Tabs Help
mjrovai@rpi-5:~/Documents/OLLAMA $
mjrovai@rpi-5:~/Documents/OLLAMA $ python calc_distance.py "United States"
Santiago de Chile is about 8,060 kilometers away from Washington, D.C..
mjrovai@rpi-5:~/Documents/OLLAMA $ python calc_distance.py "Colombia"
Santiago de Chile is about 4,250 kilometers away from Bogotá.
mjrovai@rpi-5:~/Documents/OLLAMA $ python calc_distance.py "France"
Santiago de Chile is about 11,630 kilometers away from Paris.
mjrovai@rpi-5:~/Documents/OLLAMA $
```

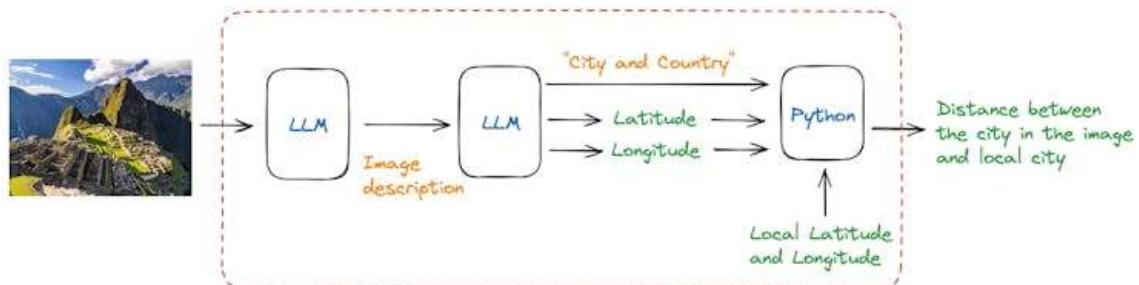
E i calcoli sono piuttosto buoni!



Nel notebook [20-Ollama\\_Function\\_Calling](#), è possibile trovare esperimenti con tutti i modelli installati.

### Aggiunta di immagini

Ora è il momento raccogliere il tutto! Modifichiamo lo script in modo che invece di immettere il nome del paese (come testo), l'utente immetta un'immagine e l'applicazione (basata su SLM) restituisca la città nell'immagine e la sua posizione geografica. Con quei dati, possiamo calcolare la distanza come prima.



Per semplicità, implementeremo questo nuovo codice in due passaggi. Innanzitutto, LLM analizzerà l'immagine e creerà una descrizione (testo). Questo testo verrà passato a un'altra istanza, dove il modello estrarrà le informazioni necessarie per il passaggio.

Inizieremo ad importare le librerie

```
import sys
import time
from haversine import haversine
import ollama
from openai import OpenAI
from pydantic import BaseModel, Field
import instructor
```

Possiamo vedere l'immagine se si esegue il codice su Jupyter Notebook. Per questo dobbiamo anche importare:

```
import matplotlib.pyplot as plt
from PIL import Image
```

Tali librerie non sono necessarie se eseguiamo il codice come uno script.

Ora, definiamo il modello e le coordinate locali:

```
MODEL = 'llava-phi3:3.8b'
mylat = -33.33
mylon = -70.51
```

Possiamo effettuare il download di una nuova immagine, ad esempio Machu Picchu da Wikipedia. Sul Notebook possiamo vederla:

```
Load the image
img_path = "image_test_3.jpg"
img = Image.open(img_path)

Display the image
plt.figure(figsize=(8, 8))
plt.imshow(img)
plt.axis('off')
plt.title("Image")
plt.show()
```



Ora, definiamo una funzione che riceverà l'immagine e che return the decimal latitude and decimal longitude of the city in the image, its name, and what country it is located

[restituirà la latitudine decimale e la longitudine decimale della città nell'immagine, il suo nome e il paese in cui si trova].

```
def image_description(img_path):
 with open(img_path, 'rb') as file:
 response = ollama.chat(
 model=MODEL,
 messages=[
 {
 'role': 'user',
 'content': '''return the decimal latitude and decimal longitude
 of the city in the image, its name, and
 what country it is located''',
 'images': [file.read()],
 },
],
 options = {
 'temperature': 0,
 }
)

 #print(response['message']['content'])
 return response['message']['content']
```

Possiamo stampare l'intera risposta per debug.

La descrizione dell'immagine generata per la funzione verrà passata di nuovo come prompt per il modello.

```
start_time = time.perf_counter() # Start timing

class CityCoord(BaseModel):
 city: str = Field(..., description="Name of the city in the image")
 country: str = Field(..., description="""Name of the country where
 the city in the image is located
 """)
 lat: float = Field(..., description="""Decimal Latitude of the city in
 the image""")
 lon: float = Field(..., description="""Decimal Longitude of the city in
 the image""")

enables `response_model` in create call
client = instructor.patch(
 OpenAI(
 base_url="http://localhost:11434/v1",
 api_key="ollama"
),
 mode=instructor.Mode.JSON,
)

image_description = image_description(img_path)
Send this description to the model
resp = client.chat.completions.create(
 model=MODEL,
 messages=[
 {
 "role": "user",
 "content": image_description,
 }
],
 response_model=CityCoord,
 max_retries=10,
 temperature=0,
)
```

Se stampiamo la descrizione dell'immagine, otterremo:

The image shows the ancient city of Machu Picchu, located in Peru. The city is perched on a steep hillside and consists of various structures made of stone. It is surrounded by lush greenery and towering mountains. The sky above is blue with scattered clouds.

```
Machu Picchu's latitude is approximately 13.5086° S, and its longitude is around
72.5494° W.
```

E la seconda risposta del modello (resp) sarà:

```
CityCoord(city='Machu Picchu', country='Peru', lat=-13.5086, lon=-72.5494)
```

Ora possiamo effettuare un “Post-Processing”, calcolando la distanza e preparando la risposta finale:

```
distance = haversine((mylat, mylon), (resp.lat, resp.lon), unit='km')

print(f"\n The image shows {resp.city}, with lat:{round(resp.lat, 2)} and \
 long: {round(resp.lon, 2)}, located in {resp.country} and about \
 {int(round(distance, -1))} kilometers away from \
 Santiago, Chile.\n")

end_time = time.perf_counter() # End timing
elapsed_time = end_time - start_time # Calculate elapsed time
print(f" [INFO] ==> The code (running {MODEL}), took {elapsed_time:.1f} \
 seconds to execute.\n")
```

E otterremo:

```
The image shows Machu Picchu, with lat:-13.16 and long: -72.54, located in Peru
and about 2,250 kilometers away from Santiago, Chile.
```

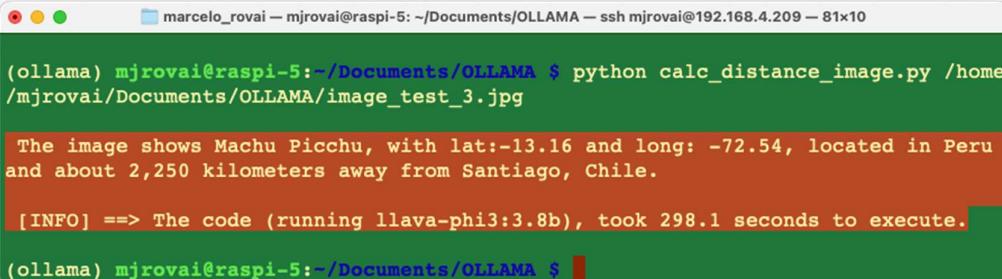
```
[INFO] ==> The code (running llava-phi3:3.8b), took 491.3 seconds to execute.
```

Nel notebook [30-Function\\_Calling\\_with\\_images](#) è possibile trovare gli esperimenti con più immagini.

Scarichiamo ora lo script `calc_distance_image.py` da **Errore. Riferimento a collegamento ipertestuale non valido.** ed eseguiamolo sul terminale con il comando:

```
python calc_distance_image.py /home/mjrovai/Documents/OLLAMA/image_test_3.jpg
```

Immettere la patch completa dell'immagine di Machu Picchu come argomento. Otterremo lo stesso risultato precedente.



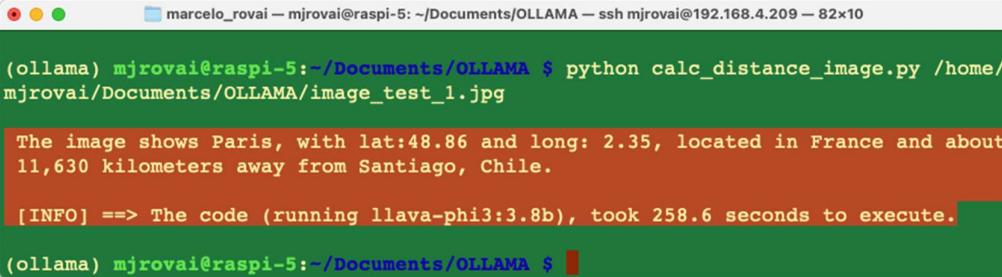
```
(ollama) mjrovai@raspi-5:~/Documents/OLLAMA $ python calc_distance_image.py /home/mjrovai/Documents/OLLAMA/image_test_3.jpg

The image shows Machu Picchu, with lat:-13.16 and long: -72.54, located in Peru
and about 2,250 kilometers away from Santiago, Chile.

[INFO] ==> The code (running llava-phi3:3.8b), took 298.1 seconds to execute.

(ollama) mjrovai@raspi-5:~/Documents/OLLAMA $
```

Per quanto riguarda Parigi?



```
(ollama) mjrovai@raspi-5:~/Documents/OLLAMA $ python calc_distance_image.py /home/mjrovai/Documents/OLLAMA/image_test_1.jpg

The image shows Paris, with lat:48.86 and long: 2.35, located in France and about
11,630 kilometers away from Santiago, Chile.

[INFO] ==> The code (running llava-phi3:3.8b), took 258.6 seconds to execute.

(ollama) mjrovai@raspi-5:~/Documents/OLLAMA $
```

Naturalmente, ci sono molti modi per ottimizzare il codice utilizzato qui. Tuttavia, l’idea è di esplorare il notevole potenziale della *function calling* con SLM nell’edge, consentendo a tali modelli di integrarsi con funzioni o API esterne. Andando oltre la generazione di testo, gli SLM possono accedere a dati in tempo reale, automatizzare attività e interagire con vari sistemi.

## SLM: Tecniche di Ottimizzazione

I Large Language Model (LLM) hanno rivoluzionato l'elaborazione del linguaggio naturale, ma la loro distribuzione e ottimizzazione presentano sfide uniche. Un problema significativo è la tendenza degli LLM (e più in particolare degli SLM) a generare informazioni che sembrano plausibili ma di fatto errate, un fenomeno noto come **allucinazione**. Ciò si verifica quando i modelli producono contenuti che sembrano coerenti ma non sono basati sulla verità o sui fatti del mondo reale.

Altre sfide includono le immense risorse computazionali richieste per l'addestramento e l'esecuzione di questi modelli, la difficoltà nel mantenere aggiornate le conoscenze all'interno del modello e la necessità di adattamenti specifici per dominio. Problemi di privacy sorgono anche quando si gestiscono dati sensibili durante l'addestramento o l'inferenza. Inoltre, garantire prestazioni coerenti in diverse attività e mantenere un uso etico di questi potenti strumenti presentano sfide continue. Affrontare questi problemi è fondamentale per l'implementazione efficace e responsabile di LLM in applicazioni del mondo reale.

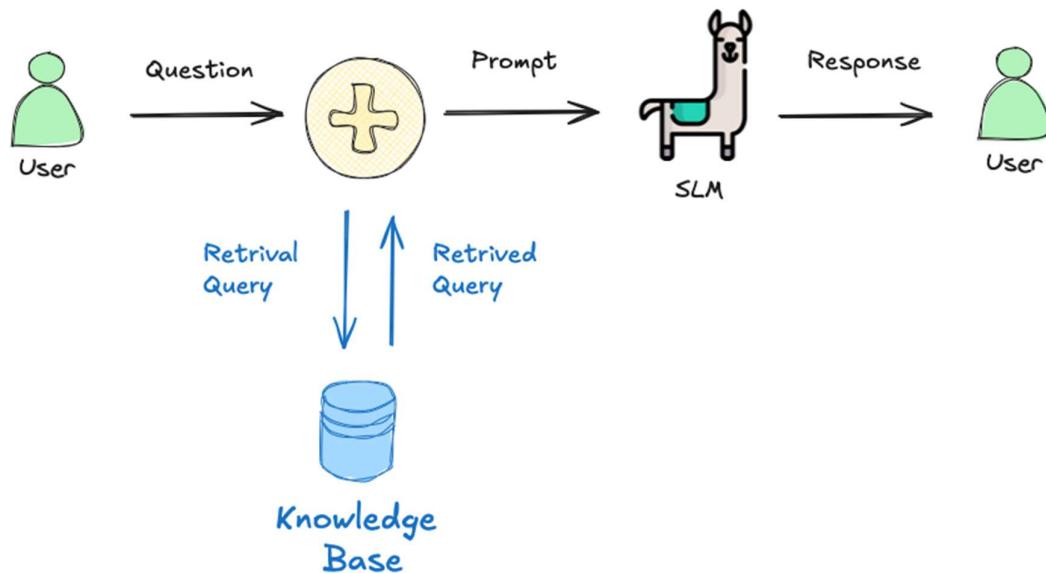
Le tecniche fondamentali per migliorare le prestazioni e l'efficienza di LLM (e SLM) sono Fine-tuning, Prompt engineering e Retrieval-Augmented Generation (RAG).

- **Fine-tuning**, sebbene richieda più risorse, offre un modo per specializzare LLM per domini o attività particolari. Questo processo comporta un ulteriore addestramento del modello su set di dati attentamente curati, consentendogli di adattare la sua vasta conoscenza generale ad applicazioni specifiche. Il Fine-tuning può portare a miglioramenti sostanziali nelle prestazioni, specialmente in campi specializzati o per casi d'uso unici.
- **Prompt engineering** è in prima linea nell'ottimizzazione LLM. Creando attentamente prompt di input, possiamo guidare i modelli per produrre output più accurati e pertinenti. Questa tecnica comporta la strutturazione di query che sfruttano le conoscenze e le capacità pre-addestrate del modello, spesso incorporando esempi o istruzioni specifiche per modellare la risposta desiderata.
- **Retrieval-Augmented Generation (RAG)** rappresenta un altro approccio potente per migliorare le prestazioni LLM. Questo metodo combina la vasta conoscenza incorporata nei modelli pre-addestrati con la capacità di accedere e incorporare informazioni esterne aggiornate. Recuperando dati pertinenti per integrare il processo decisionale del modello, RAG può migliorare significativamente l'accuratezza e ridurre la probabilità di generare informazioni obsolete o false.

Per le applicazioni edge, è più utile concentrarsi su tecniche come RAG che possono migliorare le prestazioni del modello senza dover effettuare un "fine-tuning" sul dispositivo. Esploriamolo.

## Implementazione del RAG

In un'interazione di base tra un utente e un modello linguistico, l'utente pone una domanda, che viene inviata come prompt al modello. Il modello genera una risposta basata esclusivamente sulla sua conoscenza pre-addestrata. In un processo RAG, c'è un passaggio aggiuntivo tra la domanda dell'utente e la risposta del modello. La domanda dell'utente innesca un processo di recupero da una "knowledge base".



### **Un semplice progetto RAG**

Ecco i passaggi per implementare una Retrieval Augmented Generation (RAG) di base:

- **Determinare il tipo di documenti che si utilizzeranno:** I tipi migliori sono documenti da cui possiamo ottenere testo pulito e non oscurato. I PDF possono essere problematici perché sono progettati per la stampa, non per estrarre testo sensato. Per lavorare con i PDF, dovremmo ottenere il documento di origine o utilizzare strumenti per gestirli.
- **Suddividere il testo in blocchi:** Non possiamo archiviare il testo come un unico lungo flusso a causa delle limitazioni delle dimensioni del contesto e della potenziale confusione. La suddivisione in blocchi comporta la suddivisione del testo in parti più piccole. Il testo in blocchi ha molti modi, come conteggio dei caratteri, token, parole, paragrafi o sezioni. È anche possibile sovrapporre i blocchi.
- **Crea embedding:** Gli “embedding” [incorporamenti] sono rappresentazioni numeriche del testo che catturano il significato semantico. Creiamo incorporamenti passando ogni blocco di testo attraverso un particolare modello di embedding. Il modello genera un vettore, la cui lunghezza dipende dal modello di embedding utilizzato. Dovremmo estrarre uno (o più) **modelli di embedding** da Ollama, per eseguire questa attività. Ecco alcuni esempi di modelli di embedding disponibili su Ollama.

Modello	Dimensione del parametro	Dimensione dell'Embedding
mxbai-embed-large	334M	1024
nomic-embed-text	137M	768
all-minilm	23M	384

In genere, dimensioni dell'embedding maggiori catturano informazioni più sfumate sull'input. Tuttavia, richiedono anche più risorse per l'elaborazione e un numero maggiore di parametri dovrebbe aumentare la latenza (ma anche la qualità della risposta).

- **Memorizzare i blocchi e gli embedding in un database vettoriale:** Avremo bisogno di un modo per trovare in modo efficiente i blocchi di testo più rilevanti per un dato prompt, ed è qui che entra in gioco un database vettoriale. Useremo **Chromadb**, un database vettoriale open source nativo IA, che semplifica la creazione di RAG creando conoscenze, fatti e competenze collegabili per LLM. Vengono memorizzati sia embedding che il testo sorgente per ogni blocco.

- Creare il prompt:** Quando abbiamo una domanda, creiamo un embedding e interroghiamo il database vettoriale per i blocchi più simili. Poi, selezioniamo i primi risultati e includiamo il loro testo nel prompt.

L'obiettivo di RAG è fornire al modello le informazioni più rilevanti dai nostri documenti, consentendogli di generare risposte più accurate e informative. Quindi, implementiamo un semplice esempio di un SLM che incorpora un set particolare di fatti sulle api (“Bee Facts”). All'interno dell'ambiente `ollama`, si inserisce il comando nel terminale per l'installazione di Chromadb:

```
pip install ollama chromadb
```

Tiriamo fuori un modello di embedding intermedio, `nomic-embed-text`

```
ollama pull nomic-embed-text
```

E creiamo una directory di lavoro:

```
cd Documents/OLLAMA/
mkdir RAG-simple-bee
cd RAG-simple-bee/
```

Creiamo un nuovo notebook Jupyter, `40-RAG-simple-bee` per qualche esplorazione:

Importare le librerie necessarie:

```
import ollama
import chromadb
import time
```

E definiamo i modelli aor:

```
EMB_MODEL = "nomic-embed-text"
MODEL = 'llama3.2:3B'
```

Inizialmente, dovrebbe essere creata una “knowledge base” sui fatti sulle api. Ciò comporta la raccolta di documenti rilevanti e la loro conversione in embedding vettoriali. Questi embedding vengono poi archiviati in un database vettoriale, consentendo in seguito ricerche di similarità efficienti. Si immette il “document”, una base di “fatti sulle api” come un elenco:

```
documents = [
 "Bee-keeping, also known as apiculture, involves the maintenance of bee \
colonies, typically in hives, by humans.",
 "The most commonly kept species of bees is the European honey bee (Apis \
mellifera).",
 ...
 "There are another 20,000 different bee species in the world.",
 "Brazil alone has more than 300 different bee species, and the \
vast majority, unlike western honey bees, don't sting.",
 "Reports written in 1577 by Hans Staden, mention three native bees \
used by indigenous people in Brazil.",
 "The indigenous people in Brazil used bees for medicine and food purposes",
 "From Hans Staden report: probable species: mandaçaiá (Melipona \
quadrispina), mandaguari (Scaptotrigona postica) and jataí-amarela \
(Tetragonisca angustula)."
]
```

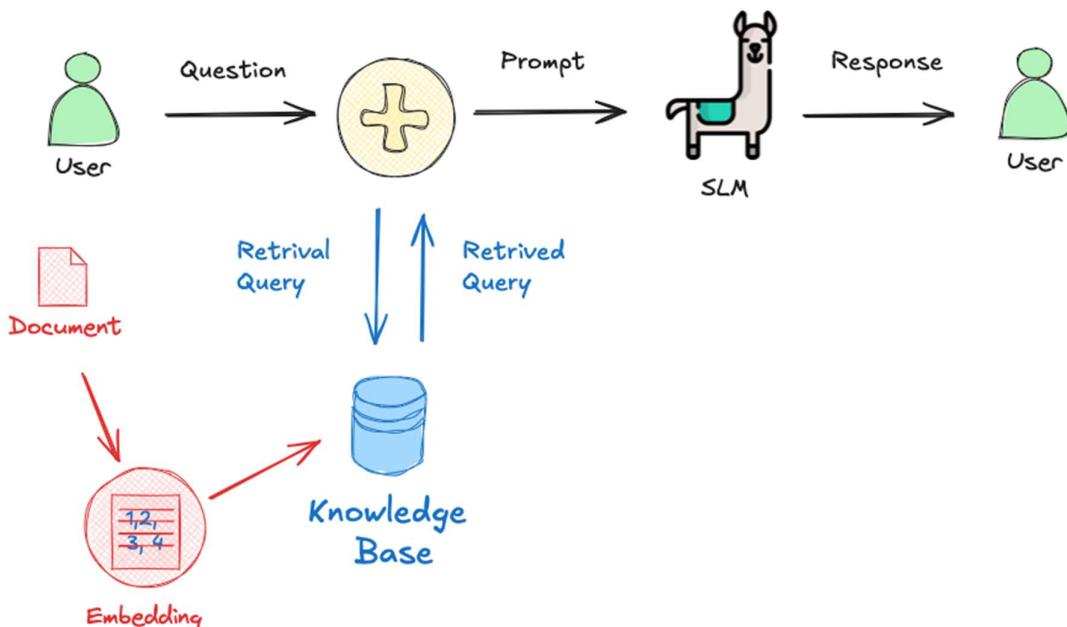
Non abbiamo bisogno di “suddividere” il documento qui perché useremo ogni elemento dell'elenco e un blocco.

Ora creeremo il nostro database di embedding vettoriale `bee_facts` e memorizzeremo il documento in esso:

```
client = chromadb.Client()
collection = client.create_collection(name="bee_facts")

store each document in a vector embedding database
for i, d in enumerate(documents):
 response = ollama.embeddings(model=EMB_MODEL, prompt=d)
 embedding = response["embedding"]
 collection.add(
 ids=[str(i)],
 embeddings=[embedding],
 documents=[d]
)
```

ra che abbiamo creato la nostra “Knowledge Base”, possiamo iniziare a fare query, recuperando dati da essa:



**Query utente:** Il processo inizia quando un utente pone una domanda, come “Quante api ci sono in una colonia? Chi depone le uova e in che quantità? E per quanto riguarda parassiti e malattie comuni?”

```
prompt = "How many bees are in a colony? Who lays eggs and how much? How about common pests and diseases?"
```

**Query Embedding:** La domanda dell’utente viene convertita in un embedding vettoriale utilizzando lo stesso modello di embedding utilizzato per la knowledge base.

```
response = ollama.embeddings(
 prompt=prompt,
 model=EMB_MODEL
)
```

**Recupero di Documenti Pertinenti:** Il sistema esegue una ricerca nella knowledge base utilizzando il “query embedding” per trovare i documenti più pertinenti (in questo caso, i 5 più probabili). Ciò avviene tramite una ricerca di similarità, che confronta la “query embedding” con gli embedding di documenti nel database.

```
results = collection.query(
 query_embeddings=[response["embedding"]],
 n_results=5
)
data = results['documents']
```

**Prompt Augmentation:** Le informazioni rilevanti recuperate vengono combinate con la query utente originale per creare un prompt “aumentato”. Questo prompt ora contiene la domanda dell’utente e i fatti pertinenti dalla knowledge base.

```
prompt=f"Using this data: {data}. Respond to this prompt: {prompt}",
```

**Generazione di Risposte:** Il prompt aumentato viene poi immesso in un modello linguistico, in questo caso il modello `llama3.2:3b`. Il modello utilizza questo contesto arricchito per generare una risposta completa. Parametri come temperatura, top\_k e top\_p vengono impostati per controllare la casualità e la qualità della risposta generata.

```
output = ollama.generate(
 model=MODEL,
 prompt=f"Using this data: {data}. Respond to this prompt: {prompt}",
 options={
 "temperature": 0.0,
 "top_k":10,
 "top_p":0.5
 }
)
```

### **Response Delivery:** Infine, il sistema restituisce all'utente la risposta generata.

```
print(output['response'])
Based on the provided data, here are the answers to your questions:
```

1. How many bees are in a colony?

A typical bee colony can contain between 20,000 and 80,000 bees.

2. Who lays eggs and how much?

The queen bee lays up to 2,000 eggs per day during peak seasons.

3. What about common pests and diseases?

Common pests and diseases that affect bees include varroa mites, hive beetles, and foulbrood.

**Creiamo una funzione che ci aiuti a rispondere a nuove domande:**

```
def rag_beans(prompt, n_results=5, temp=0.0, top_k=10, top_p=0.5):
 start_time = time.perf_counter() # Start timing

 # generate an embedding for the prompt and retrieve the data
 response = ollama.embeddings(
 prompt=prompt,
 model=EMB_MODEL
)

 results = collection.query(
 query_embeddings=[response["embedding"]],
 n_results=n_results
)

 data = results['documents']

 # generate a response combining the prompt and data retrieved
 output = ollama.generate(
 model=MODEL,
 prompt=f"Using this data: {data}. Respond to this prompt: {prompt}",
 options={
 "temperature": temp,
 "top_k": top_k,
 "top_p": top_p
 }
)

 print(output['response'])

 end_time = time.perf_counter() # End timing
 elapsed_time = round((end_time - start_time), 1) # Calculate elapsed time

 print(f"\n [INFO] ==> The code for model: {MODEL}, took {elapsed_time}s \
 to generate the answer.\n")
```

**Ora possiamo creare delle query e chiamare la funzione:**

```
prompt = "Are bees in Brazil?"
rag_beans(prompt)
Yes, bees are found in Brazil. According to the data, Brazil has more than 300
different bee species, and indigenous people in Brazil used bees for medicine and
food purposes. Additionally, reports from 1577 mention three native bees used by
indigenous people in Brazil.
```

[INFO] ==> The code for model: llama3.2:3b, took 22.7s to generate the answer.

**A proposito, se il modello utilizzato supporta più lingue, possiamo utilizzarlo (ad esempio, il portoghese), anche se il set di dati è stato creato in inglese:**

```
prompt = "Existem abelhas no Brazil?"
rag_beans(prompt)
Sim, existem abelhas no Brasil! De acordo com o relato de Hans Staden, há três
espécies de abelhas nativas do Brasil que foram mencionadas: mandaçaia (Melipona
quadriasciata), mandaguari (Scaptotrigona postica) e jataí-amarela (Tetragonisca
angustula). Além disso, o Brasil é conhecido por ter mais de 300 espécies diferentes de a
belhas, a maioria das quais não é agressiva e não põe veneno.
```

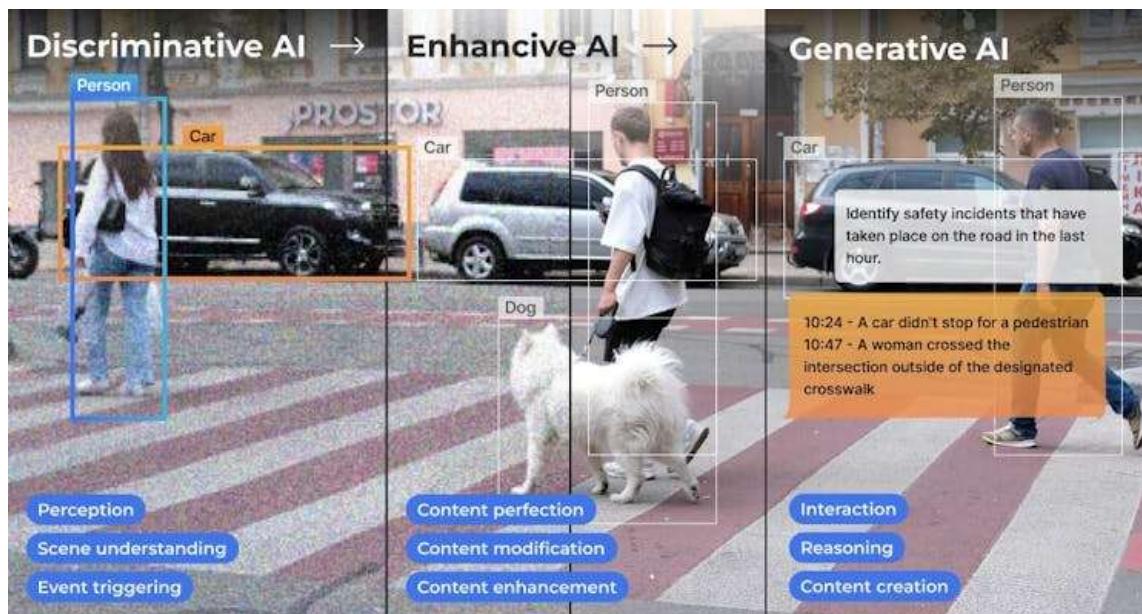
[INFO] ==> The code for model: llama3.2:3b, took 54.6s to generate the answer.

### Andando Oltre

I piccoli modelli LLM testati hanno funzionato bene sull'edge, sia nel testo che con le immagini, ma ovviamente avevano un'elevata latenza per quanto riguarda quest'ultima. Una combinazione di modelli specifici e dedicati può portare a risultati migliori; ad esempio, in casi reali, un modello di rilevamento degli oggetti (come YOLO) può ottenere una descrizione generale e un conteggio degli oggetti su un'immagine che, una volta passati a un LLM, possono aiutare a estrarre informazioni e azioni essenziali.

Secondo Avi Baum, CTO di Hailo,

Nel vasto panorama dell'intelligenza artificiale (IA), uno dei viaggi più intriganti è stata l'evoluzione dell'IA nell'edge. Questo viaggio ci ha portato dalla classica visione artificiale ai regni dell'IA discriminativa, dell'IA potenziata e ora, alla rivoluzionaria frontiera dell'IA generativa. Ogni passo ci ha avvicinato a un futuro in cui i sistemi intelligenti si integrano perfettamente con la nostra vita quotidiana, offrendo un'esperienza immersiva non solo di percezione ma anche di creazione nel palmo della nostra mano.



### Conclusione

Questo laboratorio ha dimostrato come un Raspberry Pi 5 può essere trasformato in un potente hub AI in grado di eseguire “large language model (LLM)” per analisi e approfondimenti dei dati in tempo reale e in loco utilizzando Ollama e Python. La versatilità e la potenza del Raspberry Pi, unite alle capacità di LLM leggeri come Llama 3.2 e LLaVa-Phi-3-mini, lo rendono un'eccellente piattaforma per applicazioni di edge computing.

Il potenziale di esecuzione di LLM sull'edge si estende ben oltre la semplice elaborazione dei dati, come negli esempi di questo laboratorio. Ecco alcuni suggerimenti innovativi per l'utilizzo di questo progetto:

#### 1. Smart Home Automation:

- Integrare gli SLM per interpretare i comandi vocali o analizzare i dati dei sensori per l'automazione domestica intelligente. Ciò potrebbe includere il monitoraggio e il controllo in tempo reale di dispositivi domestici, sistemi di sicurezza e gestione energetica, tutti elaborati localmente senza fare affidamento sui servizi cloud.

#### 2. Raccolta e Analisi dei Dati sul Campo:

- Distribuire gli SLM su Raspberry Pi in configurazioni remote o mobili per la raccolta e l'analisi dei dati in tempo reale. Può essere utilizzato in agricoltura per monitorare la salute delle colture, negli studi ambientali per il monitoraggio della fauna selvatica o nella risposta ai disastri per la consapevolezza della situazione e la gestione delle risorse.

### 3. Strumenti Didattici:

- Creare strumenti didattici interattivi che sfruttano gli SLM per fornire feedback immediato, traduzione linguistica e tutoraggio. Può essere particolarmente utile nelle regioni in via di sviluppo con accesso limitato a tecnologie avanzate e connettività Internet.

### 4. Applicazioni Sanitarie:

- Utilizzare gli SLM per la diagnosi medica e il monitoraggio dei pazienti. Possono fornire analisi in tempo reale dei sintomi e suggerire potenziali trattamenti. Può essere integrato in piattaforme di telemedicina o dispositivi sanitari portatili.

### 5. Intelligence Aziendale Locale:

- Implementare gli SLM in ambienti di vendita al dettaglio o di piccole imprese per analizzare il comportamento dei clienti, gestire l'inventario e ottimizzare le operazioni. La capacità di elaborare i dati localmente garantisce la privacy e riduce la dipendenza dai servizi esterni.

### 6. IoT Industriale:

- Integrare gli SLM nei sistemi IoT industriali per manutenzione predittiva, controllo qualità e ottimizzazione dei processi. Raspberry Pi può fungere da unità di elaborazione dati localizzata, riducendo la latenza e migliorando l'affidabilità dei sistemi automatizzati.

### 7. Veicoli Autonomi:

- Utilizzare gli SLM per elaborare dati sensoriali da veicoli autonomi, consentendo decisioni e navigazione in tempo reale. Questo può essere applicato a droni, robot e auto a guida autonoma per una maggiore autonomia e sicurezza.

### 8. Patrimonio Culturale e Turismo:

- Implementare gli SLM per fornire siti del patrimonio culturale e guide museali interattive e informative. I visitatori possono utilizzare questi sistemi per ottenere informazioni e approfondimenti in tempo reale, migliorando la loro esperienza senza connettività Internet.

### 9. Progetti Artistici e Creativi:

- Utilizzare SLM per analizzare e generare contenuti creativi, come musica, arte e letteratura. Questo può promuovere progetti innovativi nei settori creativi e consentire esperienze interattive uniche in mostre e spettacoli.

### 10. Tecnologie Assistenziali Personalizzate:

- Sviluppare tecnologie assistenziali per persone con disabilità, fornendo supporto personalizzato e adattivo tramite testo in tempo reale, traduzione di lingue e altri strumenti accessibili.

## Risorse

- [10-Ollama\\_Python\\_Library notebook](#)
- [20-Ollama\\_Function\\_Calling notebook](#)
- [30-Function\\_Calling\\_with\\_images notebook](#)
- [40-RAG-simple-bee notebook](#)
- [calc\\_distance\\_image python script](#)

# Lab Condivisi

## KWS Feature Engineering



DALL-E 3 Prompt: Scena di un cartone animato in stile anni '50 ambientata in una sala di ricerca. Due scienziati, uno con una lente di ingrandimento e l'altro che prende appunti, esaminano grandi grafici appesi al muro. Questi grafici raffigurano FFT e curve temporali relative all'analisi dei dati audio. La sala ha un'atmosfera retrò, con tavoli in legno, lampade vintage e classici strumenti di analisi audio.

### Panoramica

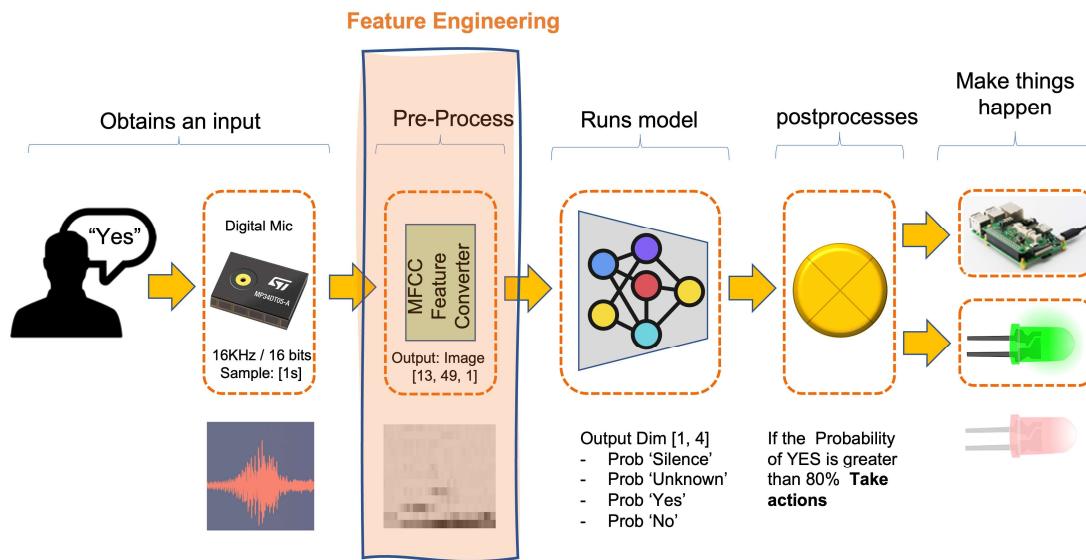
In questo tutorial pratico, l'enfasi è posta sul ruolo critico che la “feature engineering” [ingegneria delle funzionalità] svolge nell’ottimizzazione delle prestazioni dei modelli di machine learning applicati alle attività di classificazione audio, come il riconoscimento vocale. È essenziale essere consapevoli che le prestazioni di qualsiasi modello di apprendimento automatico dipendono in larga misura dalla qualità delle feature utilizzate e ci occuperemo della meccanica “sotto il cofano” dell'estrazione delle feature, concentrando principalmente sui Mel-frequency Cepstral Coefficient (MFCC), una pietra miliare nel campo dell’elaborazione del segnale audio.

I modelli di apprendimento automatico, in particolare gli algoritmi tradizionali, non comprendono le onde audio. Comprendono i numeri disposti in modo significativo, ovvero le feature. Queste feature encapsulano le caratteristiche del segnale audio, rendendo più facile per i modelli distinguere tra suoni diversi.

Questo tutorial si occuperà della generazione di feature specificamente per la classificazione audio. Ciò può essere particolarmente interessante per l'applicazione di machine learning a una varietà di dati audio, sia per il riconoscimento vocale, la categorizzazione musicale, la classificazione degli insetti basata sui suoni del battito delle ali o altre attività di analisi del suono.

## Il KWS

L'applicazione TinyML più comune è Keyword Spotting (KWS), un sottoinsieme del campo più ampio del riconoscimento vocale. Mentre il riconoscimento vocale generale trascrive tutte le parole pronunciate in testo, il Keyword Spotting si concentra sul rilevamento di "parole chiave" o "wake word" specifiche in un flusso audio continuo. Il sistema è addestrato a riconoscere queste parole chiave come frasi o parole predefinite, come yes o no. In breve, KWS è una forma specializzata di riconoscimento vocale con il suo set di sfide e requisiti. Ecco un tipico processo KWS che utilizza MFCC Feature Converter:



## Applicazioni di KWS

- **Assistenti Vocali:** In dispositivi come Alexa di Amazon o Google Home, KWS viene utilizzato per rilevare la wake word ("Alexa" o "Hey Google") per attivare il dispositivo.
- **Comandi Attivati Tramite Voce:** In contesti automobilistici o industriali, KWS può essere utilizzato per avviare comandi specifici come "Avvia motore" o "Spegni luci".
- **Sistemi di Sicurezza:** I sistemi di sicurezza attivati tramite voce possono utilizzare KWS per autenticare gli utenti in base a una passphrase pronunciata.
- **Servizi di Telecomunicazione:** Le linee del servizio clienti possono utilizzare KWS per instradare le chiamate in base a parole chiave pronunciate.

## Differenze dal Riconoscimento Vocale Generale

- **Efficienza Computazionale:** KWS è solitamente progettato per essere meno intensivo dal punto di vista computazionale rispetto al riconoscimento vocale completo, poiché deve riconoscere solo un piccolo set di frasi.
- **Elaborazione in Tempo Reale:** KWS spesso funziona in tempo reale ed è ottimizzato per il rilevamento a bassa latenza delle parole chiave.
- **Vincoli di Risorse:** I modelli KWS sono spesso progettati per essere leggeri, in modo da poter essere eseguiti su dispositivi con risorse computazionali limitate, come microcontrollori o telefoni cellulari.

- **Attività Mirata:** Mentre i modelli di riconoscimento vocale generali sono addestrati per gestire un'ampia gamma di vocabolario e accenti, i modelli KWS sono ottimizzati per riconoscere parole chiave specifiche, spesso in modo accurato in ambienti rumorosi.

## Panoramica sui Segnali Audio

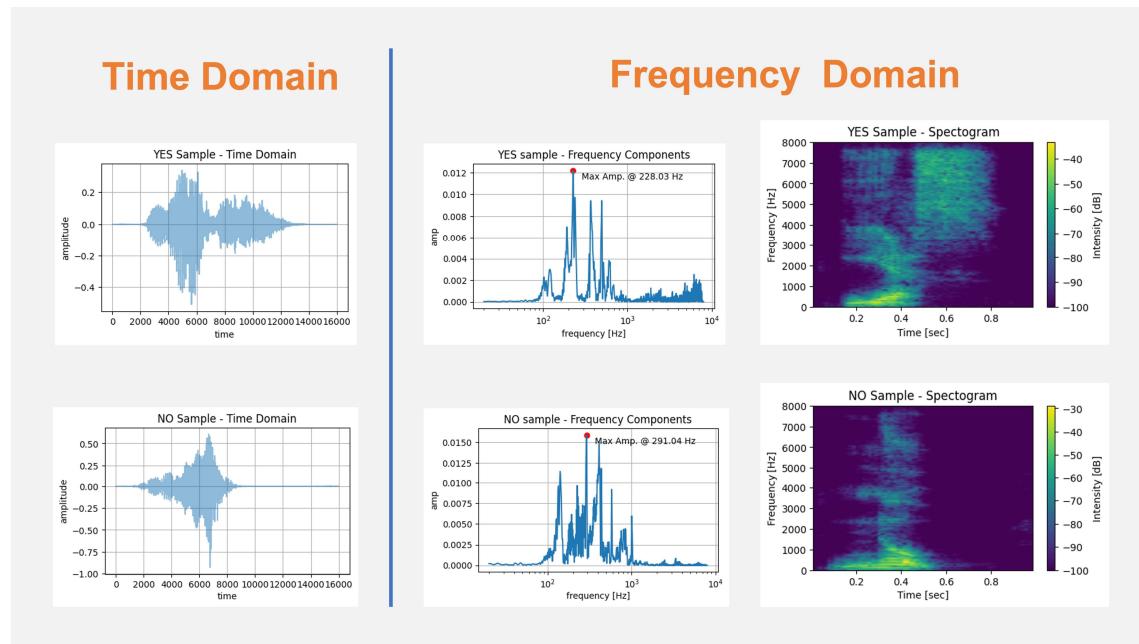
Comprendere le proprietà di base dei segnali audio è fondamentale per un'estrazione efficace delle feature **caratteristiche** e, in ultima analisi, per applicare con successo gli algoritmi di apprendimento automatico nelle attività di classificazione audio. I segnali audio sono forme d'onda complesse che catturano le fluttuazioni della pressione dell'aria nel tempo. Questi segnali possono essere caratterizzati da diversi attributi fondamentali: frequenza di campionamento, frequenza e ampiezza.

- **Frequenza e Aampiezza:** La **Frequenza** si riferisce al numero di oscillazioni che una forma d'onda subisce per unità di tempo e si misura anche in Hz. Nel contesto dei segnali audio, frequenze diverse corrispondono a pitch [toni] diversi. L'**Aampiezza**, d'altra parte, misura la grandezza delle oscillazioni e si correla con l'intensità del suono. Sia la frequenza che l'ampiezza sono feature essenziali che catturano le qualità tonali e ritmiche dei segnali audio.
- **Frequenza di Campionamento:** La **frequenza di campionamento**, spesso indicata in Hertz (Hz), definisce il numero di campioni presi al secondo durante la digitalizzazione di un segnale analogico. Una frequenza di campionamento più elevata consente una rappresentazione digitale più accurata del segnale, ma richiede anche più risorse di elaborazione. Le frequenze di campionamento tipiche includono 44,1 kHz per l'audio di qualità CD e 16 kHz o 8 kHz per le attività di riconoscimento vocale. Comprendere i compromessi nella selezione di una frequenza di campionamento appropriata è essenziale per bilanciare accuratezza ed efficienza computazionale. In generale, con i progetti TinyML, lavoriamo con 16 KHz. Sebbene i toni musicali possano essere uditi a frequenze fino a 20 kHz, la voce raggiunge il massimo a 8 kHz. I sistemi telefonici tradizionali utilizzano una frequenza di campionamento di 8 kHz.

Per una rappresentazione accurata del segnale, la frequenza di campionamento deve essere almeno il doppio della frequenza più alta presente nel segnale.

- **Dominio del Tempo e Dominio della Frequenza:** I segnali audio possono essere analizzati nei domini del tempo e della frequenza. Nel dominio del tempo, un segnale è rappresentato come una forma d'onda in cui l'ampiezza è tracciata in funzione del tempo. Questa rappresentazione aiuta a osservare feature temporali come inizio e durata, ma le caratteristiche tonali del segnale non sono ben evidenziate. Al contrario, una rappresentazione del dominio della frequenza fornisce una vista delle frequenze costituenti del segnale e delle rispettive ampiezze, in genere ottenute tramite una trasformata di Fourier. Questo è prezioso per le attività che richiedono la comprensione del contenuto spettrale del segnale, come l'identificazione di note musicali o fonemi vocali (il nostro caso).

L'immagine seguente mostra le parole **YES** e **NO** con rappresentazioni tipiche nei domini del tempo (audio grezzo) e della frequenza:



### Perché l'Audio Grezzo No?

Sebbene l'utilizzo diretto di dati audio grezzi per attività di apprendimento automatico possa sembrare allettante, questo approccio presenta diverse sfide che lo rendono meno adatto alla creazione di modelli solidi ed efficienti.

L'utilizzo di dati audio grezzi per il Keyword Spotting (KWS), ad esempio, su dispositivi TinyML pone delle sfide dovute alla sua elevata dimensionalità (utilizzando una frequenza di campionamento di 16 kHz), alla complessità computazionale per l'acquisizione di feature temporali, alla suscettibilità al rumore e alla mancanza di feature semanticamente significative, rendendo le tecniche di estrazione di caratteristiche come MFCC una scelta più pratica per applicazioni con risorse limitate.

Ecco alcuni dettagli aggiuntivi sui problemi critici associati all'utilizzo di audio grezzi:

- **Alta Dimensionalità:** I segnali audio, in particolare quelli campionati ad alte velocità, generano grandi quantità di dati. Ad esempio, una clip audio di 1 secondo campionata a 16 kHz avrà 16.000 punti singoli. I dati ad alta dimensionalità aumentano la complessità computazionale, portando a tempi di training più lunghi e costi computazionali più elevati, rendendoli poco pratici per ambienti con risorse limitate. Inoltre, l'ampia gamma dinamica dei segnali audio richiede una quantità significativa di bit per campione, mentre trasmette poche informazioni utili.
- **Dipendenze Temporali:** I segnali audio grezzi hanno strutture temporali che i semplici modelli di apprendimento automatico potrebbero trovare difficili da catturare. Sebbene le reti neurali ricorrenti come **LSTM** possano modellare tali dipendenze, sono computazionalmente intensive e difficili da addestrare su dispositivi di piccole dimensioni.
- **Rumore e Variabilità:** I segnali audio grezzi spesso contengono rumore di fondo e altri elementi non essenziali che influenzano le prestazioni del modello. Inoltre, lo stesso suono può avere caratteristiche diverse in base a vari fattori come la distanza dal microfono, l'orientamento della sorgente sonora e le proprietà acustiche dell'ambiente, aggiungendo complessità ai dati.
- **Mancanza di Significato Semantico:** L'audio grezzo non contiene intrinsecamente feature semanticamente significative per le attività di classificazione. Feature come tono, tempo e feature spettrali, che possono essere cruciali per il riconoscimento vocale, non sono direttamente accessibili dai dati della forma d'onda grezza.

- **Ridondanza del Segnale:** I segnali audio spesso contengono informazioni ridondanti, con alcune porzioni del segnale che contribuiscono poco o per niente al compito da svolgere. Questa ridondanza può rendere l'apprendimento inefficiente e potenzialmente portare a un overfitting.

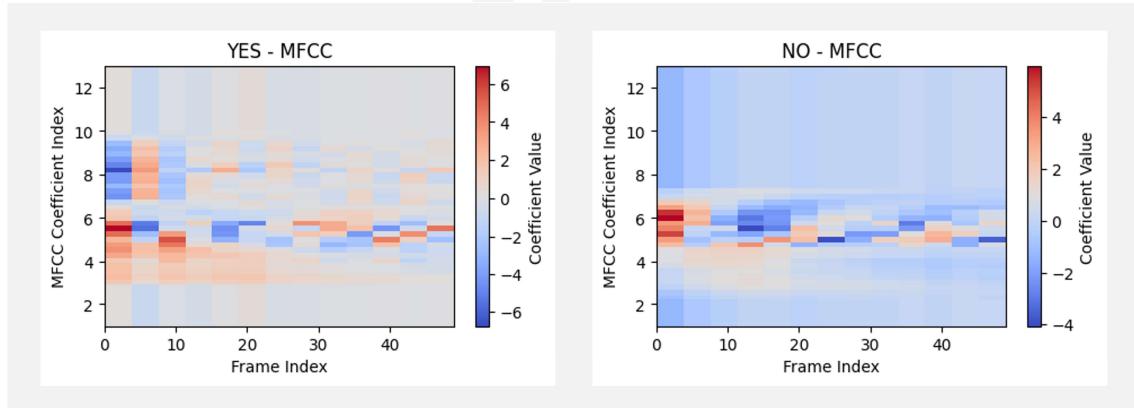
Per queste ragioni, tecniche di estrazione di feature come i Mel-frequency Cepstral Coefficients (MFCCs), le Mel-Frequency Energies (MFEs) e gli spettrogrammi semplici sono comunemente utilizzate per trasformare i dati audio grezzi in un formato più gestibile e informativo. Queste feature catturano le caratteristiche essenziali del segnale audio riducendo dimensionalità e rumore, facilitando un apprendimento automatico più efficace.

## Panoramica sugli MFCC

### Cosa sono gli MFCC?

I **Mel-frequency Cepstral Coefficients (MFCC)** sono un set di feature derivate dal contenuto spettrale di un segnale audio. Si basano sulle percezioni uditive umane e sono comunemente utilizzati per catturare le feature fonetiche di un segnale audio. Gli MFCC vengono calcolati tramite un processo in più fasi che include pre-enfasi, inquadratura, windowing, applicazione della Fast Fourier Transform (FFT) per convertire il segnale nel dominio della frequenza e, infine, applicazione della Discrete Cosine Transform (DCT). Il risultato è una rappresentazione compatta delle caratteristiche spettrali del segnale audio originale.

L'immagine seguente mostra le parole **YES** e **NO** nella loro rappresentazione MFCC:



Questo [video](#) spiega i Mel Frequency Cepstral Coefficients (MFCC) e come calcolarli.

### Perché gli MFCC sono importanti?

Gli MFCC sono fondamentali per diversi motivi, in particolare nel contesto di Keyword Spotting (KWS) e TinyML:

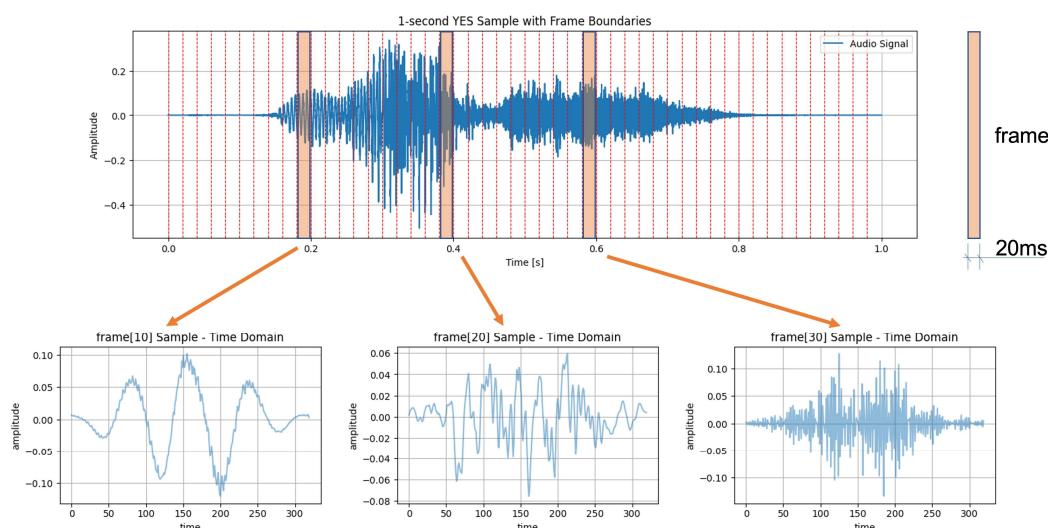
- **Riduzione della Dimensionalità:** Gli MFCC catturano le caratteristiche spettrali essenziali del segnale audio riducendo significativamente la dimensionalità dei dati, rendendoli ideali per applicazioni TinyML con risorse limitate.
- **Robustezza:** Gli MFCC sono meno sensibili al rumore e alle variazioni di tono e ampiezza, offrendo un set di funzionalità più stabile e robusto per le attività di classificazione audio.
- **Modellazione del Sistema Uditivo Umano:** La scala Mel negli MFCC approssima la risposta dell'orecchio umano a diverse frequenze, rendendoli pratici per il riconoscimento vocale in cui è desiderata una percezione simile a quella umana.
- **Efficienza Computazionale:** Il processo di calcolo degli MFCC è efficiente dal punto di vista computazionale, rendendolo adatto per applicazioni in tempo reale su hardware con risorse computazionali limitate.

In sintesi, gli MFCC offrono un equilibrio tra ricchezza di informazioni ed efficienza computazionale, rendendoli popolari per le attività di classificazione audio, in particolare in ambienti limitati come TinyML.

### Calcolo degli MFCC

Il calcolo dei Mel-frequency Cepstral Coefficients (MFCCs) comporta diversi passaggi chiave. Esaminiamoli, che sono particolarmente importanti per le attività di Keyword Spotting (KWS) sui dispositivi TinyML.

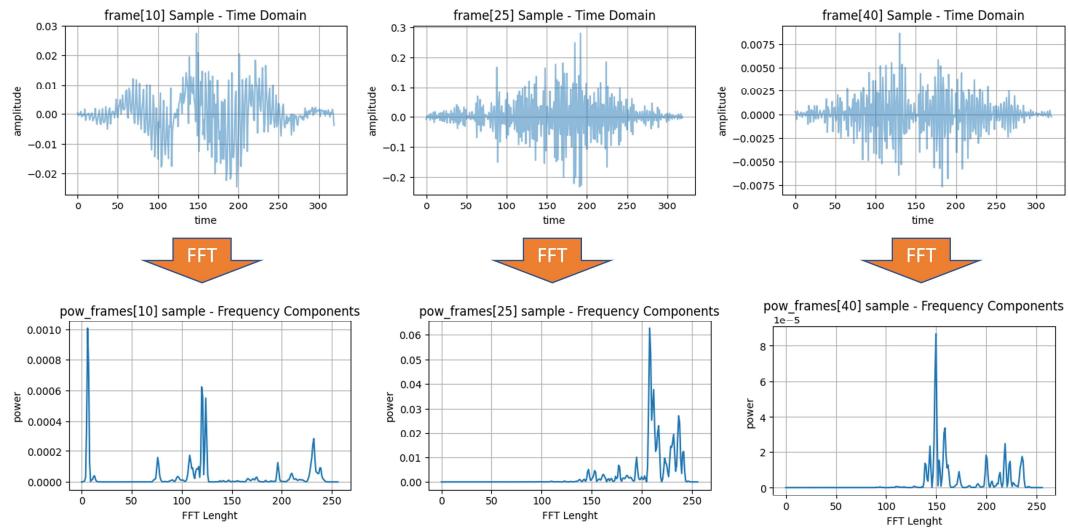
- **Pre-enfasi:** Il primo passaggio è la pre-enfasi, che viene applicata per accentuare le componenti ad alta frequenza del segnale audio e bilanciare lo spettro di frequenza. Ciò si ottiene applicando un filtro che amplifica la differenza tra campioni consecutivi. La formula per la pre-enfasi è:  $y(t) = x(t) - \alpha x(t-1)$ , dove  $\alpha$  è il fattore di pre-enfasi, in genere intorno a 0.97.
- **Framing:** I segnali audio sono divisi in frame brevi (la *frame length*), in genere da 20 a 40 millisecondi. Ciò si basa sul presupposto che le frequenze in un segnale siano stazionarie per un breve periodo. Il framing aiuta ad analizzare il segnale in slot di tempo così piccoli. Il *frame stride* (o step) sposterà un frame e quello adiacente. Questi step potrebbero essere sequenziali o sovrapposti.
- **Windowing:** Ogni frame viene quindi sottoposto a windowing per ridurre al minimo le discontinuità ai confini del frame. Una funzione window comunemente utilizzata è la finestra di Hamming. Il windowing prepara il segnale per una trasformata di Fourier riducendo al minimo gli effetti alle estremità. L'immagine sottostante mostra tre frame (10, 20 e 30) e i campioni di tempo dopo il windowing (notare che la lunghezza del frame e il frame stride sono di 20 ms):



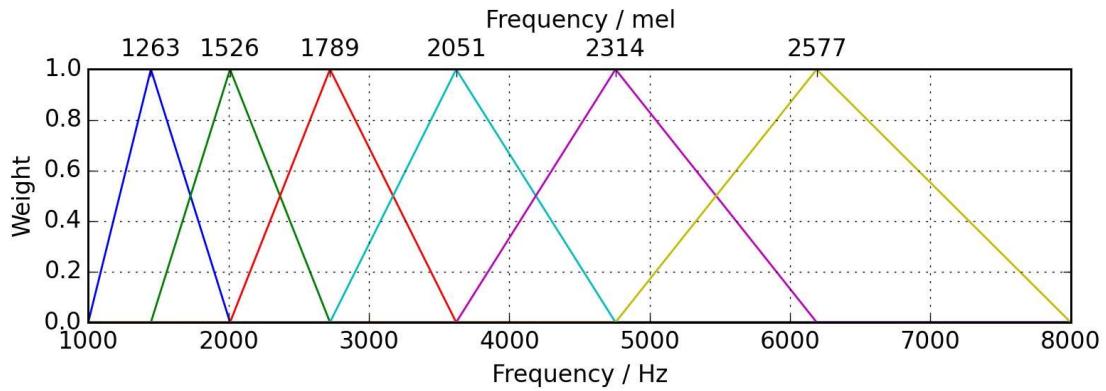
- **Fast Fourier Transform (FFT)** La Fast Fourier Transform (FFT) viene applicata a ciascun frame sottoposto a windowing per convertirlo dal dominio del tempo al dominio della frequenza. La FFT ci fornisce una rappresentazione a valori complessi che include sia informazioni di ampiezza che di fase. Tuttavia, per gli MFCC, solo l'ampiezza viene utilizzata per calcolare lo spettro di potenza. Lo spettro di potenza è il quadrato dello spettro di ampiezza e misura l'energia presente in ogni componente di frequenza.

Lo spettro di potenza  $P(f)$  di un segnale  $x(t)$  è definito come  $P(f) = |X(f)|^2$ , dove  $X(f)$  è la trasformata di Fourier di  $x(t)$ . Elevando al quadrato l'ampiezza della trasformata di Fourier, mettiamo in risalto le frequenze *più forti* rispetto a quelle *più deboli*, catturando così caratteristiche spettrali più rilevanti del segnale audio. Ciò è

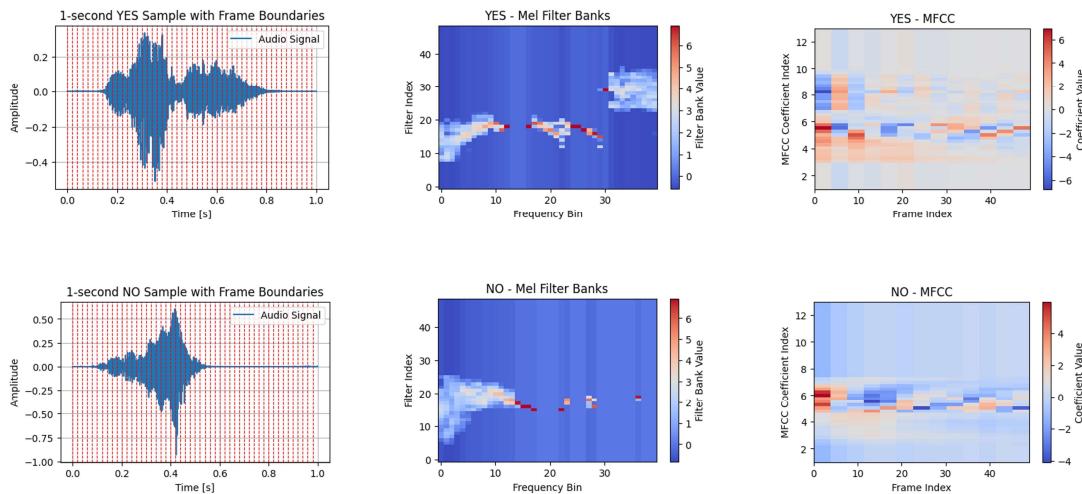
importante in applicazioni come la classificazione audio, il riconoscimento vocale e il Keyword Spotting (KWS), in cui l'attenzione è rivolta all'identificazione di distinti pattern di frequenza che caratterizzano diverse classi di audio o fonemi nel parlato.



- **Mel Filter Banks:** Il dominio di frequenza viene poi mappato sulla [scala Mel](#), che approssima la risposta dell'orecchio umano a diverse frequenze. L'idea è di estrarre più feature (più filtri) nelle frequenze più basse e meno in quelle alte. Pertanto, funziona bene sui suoni distinti dall'orecchio umano. In genere, da 20 a 40 filtri triangolari estraiano le energie di frequenza Mel. Queste energie vengono poi trasformate in logaritmo per convertire i fattori moltiplicativi in quelli additivi, rendendoli più adatti per un'ulteriore elaborazione.



- **Discrete Cosine Transform (DCT):** L'ultimo passaggio consiste nell'applicare la [Trasformata discreta del coseno \(DCT\)](#) alle energie del log Mel. La DCT aiuta a de-correlare le energie, comprimendo efficacemente i dati e mantenendo solo le feature più discriminanti. Di solito, vengono mantenuti i primi 12-13 coefficienti DCT, formando il vettore finale delle feature MFCC.



## Pratica con Python

Applichiamo quanto discusso lavorando su un campione audio reale. Aprire il notebook su Google CoLab ed estrarre le funzionalità MLCC sui campioni audio: [\[Open In Colab\]](#)

## Conclusione

*Quale Tecnica di Estrazione delle Feature dovremmo usare?*

I Mel-frequency Cepstral Coefficient (MFCC), le Mel-Frequency Energies (MFE), o lo Spettrogramma sono tecniche per rappresentare dati audio, che sono spesso utili in diversi contesti.

In generale, gli MFCC sono più focalizzati sulla cattura dell'inviluppo dello spettro di potenza, il che li rende meno sensibili ai dettagli spettrali a grana fine ma più robusti al rumore. Questo è spesso auspicabile per le attività relative al parlato. D'altro canto, gli spettrogrammi o MFE conservano informazioni di frequenza più dettagliate, il che può essere vantaggioso in attività che richiedono discriminazione basata su contenuti spettrali a grana fine.

**Gli MFCC sono particolarmente efficaci per:**

- Riconoscimento Vocale:** Gli MFCC sono eccellenti per identificare il contenuto fonetico nei segnali vocali.
- Identificazione del Parlante:** Possono essere utilizzati per distinguere tra diversi parlanti in base alle caratteristiche della voce.
- Riconoscimento delle Emozioni:** Gli MFCC possono catturare le sfumature nel parlato indicative di stati emotivi.
- Keyword Spotting:** Specialmente in TinyML, dove la bassa complessità computazionale e le piccole dimensioni delle funzionalità sono cruciali.

**Gli spettrogrammi o MFE sono spesso più adatti per:**

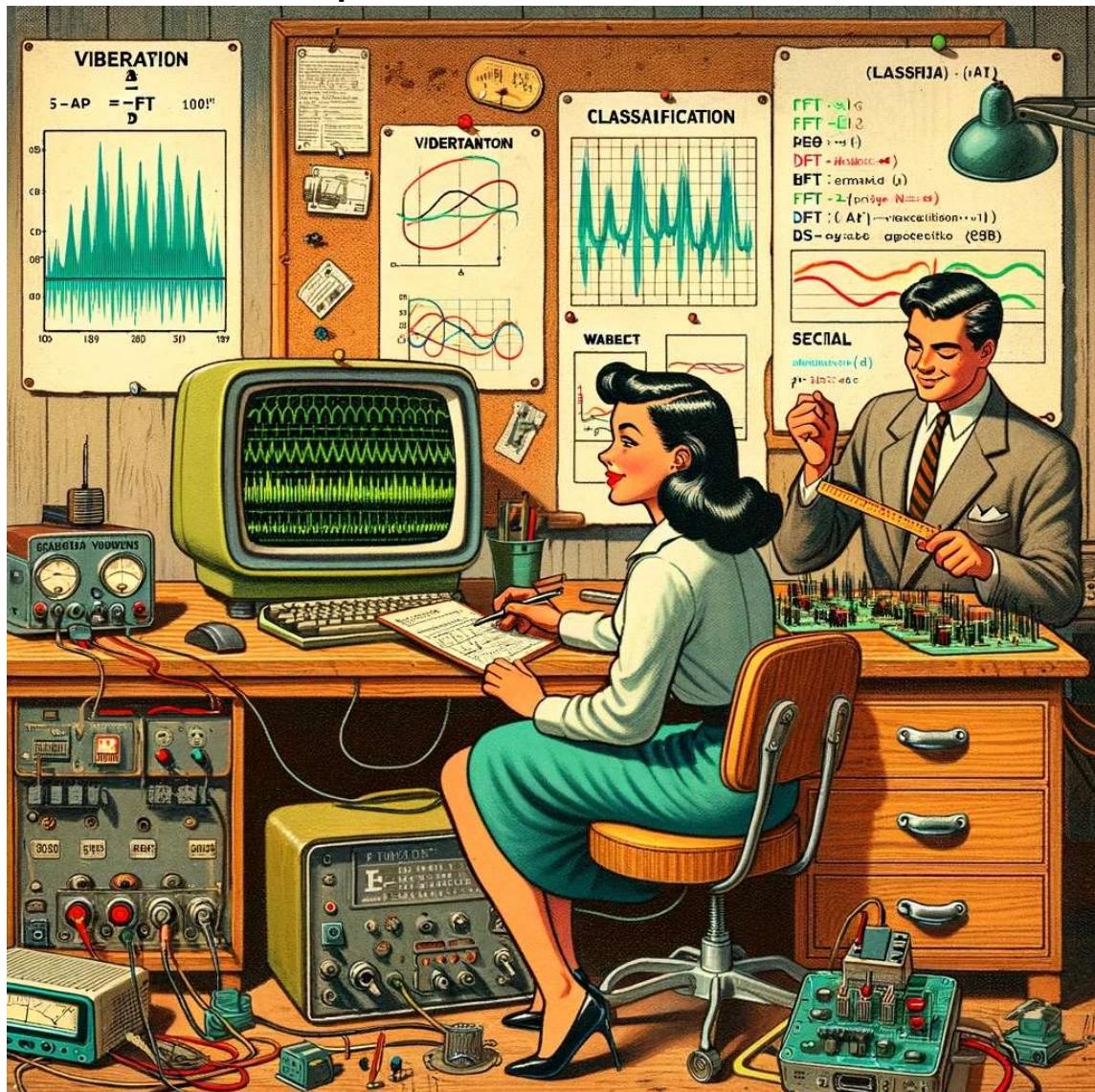
- Analisi Musicale:** Gli spettrogrammi possono catturare strutture armoniche e timbriche nella musica, il che è essenziale per attività come la classificazione del genere, il riconoscimento degli strumenti o la trascrizione musicale.
- Classificazione dei Suoni Ambientali:** Nel riconoscimento di suoni ambientali non vocali (ad esempio pioggia, vento, traffico), lo spettrogramma completo può fornire feature più discriminanti.
- Identificazione del Canto degli Uccelli:** I dettagli intricati dei richiami degli uccelli sono spesso meglio catturati utilizzando gli spettrogrammi.

4. **Elaborazione del Segnale Bioacustico:** In applicazioni come l'analisi dei richiami dei delfini o dei pipistrelli, le informazioni di frequenza a grana fine in uno spettrogramma possono essere essenziali.
5. **Audio Quality Assurance:** Gli spettrogrammi sono spesso utilizzati nell'analisi audio professionale per identificare rumori indesiderati, clic o altri artefatti.

## Risorse

- [Notebook Audio\\_Data\\_Analysis Colab](#)

## Blocco delle Feature Spettrali DSP



DALL-E 3 Prompt: Illustrazione in stile fumetto anni '50 di uno scienziato latino e una scienziata in una sala di ricerca sulle vibrazioni. L'uomo sta usando un righello per esaminare vecchi circuiti. La donna è al computer con grafici complessi sulle vibrazioni. Il tavolo di legno ha delle schede con dei sensori, in primo piano un accelerometro. Un classico computer con la parte posteriore arrotondata mostra l'IDE Arduino con codice per le assegnazioni dei pin LED e algoritmi di apprendimento automatico per il rilevamento del movimento. Il Serial Monitor visualizza FFT, classificazione, wavelet e DSP. Lampade vintage, strumenti e grafici FFT e Wavelet completano la scena.

## Panoramica

I progetti TinyML correlati al movimento (o alle vibrazioni) coinvolgono dati da IMU (solitamente **accelerometri** e **giroskopi**). Questi dataset di tipo temporale dovrebbero essere pre-elaborati prima di inserirli in un training di modello di apprendimento automatico, che è un'area impegnativa per l'apprendimento automatico embedded. Tuttavia, Edge

Impulse aiuta a superare questa complessità con la sua fase di pre-elaborazione dell'elaborazione del segnale digitale (DSP) e, più specificamente, il **Spectral Features Block** per i sensori inerziali.

Ma come funziona internamente? Analizziamolo nel dettaglio.

## Estrazione delle Feature di Revisione

L'estrazione di feature **caratteristiche** da un dataset catturato con sensori inerziali, come gli accelerometri, comporta l'elaborazione e l'analisi dei dati grezzi. Gli accelerometri misurano l'accelerazione di un oggetto lungo uno o più assi (in genere tre, indicati come X, Y e Z).

Queste misure possono essere utilizzate per comprendere vari aspetti del movimento dell'oggetto, come pattern di movimento e vibrazioni. Ecco una panoramica di alto livello del processo:

**Raccolta dati:** Per prima cosa, dobbiamo raccogliere i dati dagli accelerometri. A seconda dell'applicazione, i dati possono essere raccolti a diverse frequenze di campionamento. È essenziale assicurarsi che la frequenza di campionamento sia sufficientemente alta da catturare le dinamiche rilevanti del movimento studiato (la frequenza di campionamento dovrebbe essere almeno il doppio della massima frequenza presente nel segnale).

**Pre-elaborazione dei dati:** I dati grezzi dell'accelerometro possono essere rumorosi e contenere errori o informazioni irrilevanti. I passaggi di pre-elaborazione, come il filtraggio e la normalizzazione, possono aiutare a pulire e standardizzare i dati, rendendoli più adatti all'estrazione di feature.

Studio non esegue la normalizzazione o la standardizzazione, quindi a volte, quando si lavora con Sensor Fusion [gruppi di sensori], potrebbe essere necessario eseguire questo passaggio prima di caricare i dati in Studio. Ciò è particolarmente cruciale nei progetti di gruppi di sensori, come visto in questo tutorial, [Sensor Data Fusion with Spresense and CommonSense](#).

**Segmentazione:** A seconda della natura dei dati e dell'applicazione, potrebbe essere necessario dividere i dati in segmenti più piccoli o **finestre**. Ciò può aiutare a concentrarsi su eventi o attività specifici all'interno del dataset, rendendo l'estrazione di feature più gestibile e significativa. La scelta della **window size** [dimensione della finestra] e della sovrapposizione (**window span**) dipende dall'applicazione e dalla frequenza degli eventi di interesse. Come regola generale, dovremmo provare a catturare un paio di "cicli di dati".

**Estrazione delle feature:** Una volta che i dati sono stati pre-elaborati e segmentati, è possibile estrarre feature che descrivono le caratteristiche del movimento. Alcune feature tipiche estratte dai dati dell'accelerometro includono:

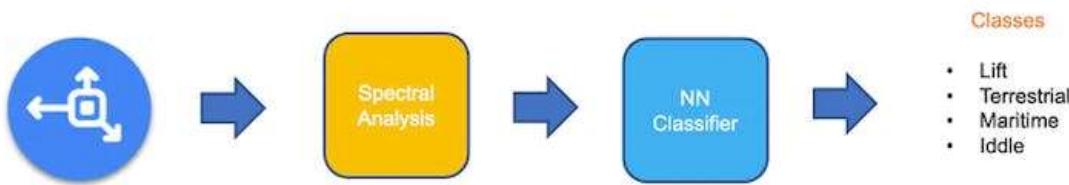
- Le feature del **Time-domain** descrivono le **proprietà statistiche** dei dati all'interno di ciascun segmento, come media, mediana, deviazione standard, asimmetria, curtosi e tasso di attraversamento dello zero.
- Le feature **Frequency-domain** si ottengono trasformando i dati nel dominio della frequenza utilizzando tecniche come la **Fast Fourier Transform (FFT)**. Alcune feature tipiche del dominio della frequenza includono lo spettro di potenza, l'energia spettrale, le frequenze dominanti (ampiezza e frequenza) e l'entropia spettrale.
- Le feature del dominio **Time-frequency** combinano le informazioni del dominio del tempo e della frequenza, come la **Short-Time Fourier Transform (STFT)** o la **Discrete Wavelet Transform (DWT)**. Possono fornire una comprensione più dettagliata di come il contenuto di frequenza del segnale cambia nel tempo.

In molti casi, il numero di feature estratte può essere elevato, il che può portare a un overfitting o a una maggiore complessità computazionale. Le tecniche di selezione delle feature, come le informazioni reciproche, i metodi basati sulla correlazione o l'analisi delle componenti principali (PCA), possono aiutare a identificare le feature più rilevanti per una

determinata applicazione e ridurre la dimensionalità del dataset. Studio può aiutare con tali calcoli rilevanti per le feature.

Esploriamo più in dettaglio un tipico progetto di classificazione del movimento TinyML trattato in questa serie di esercitazioni pratiche.

### Un progetto TinyML di Motion Classification

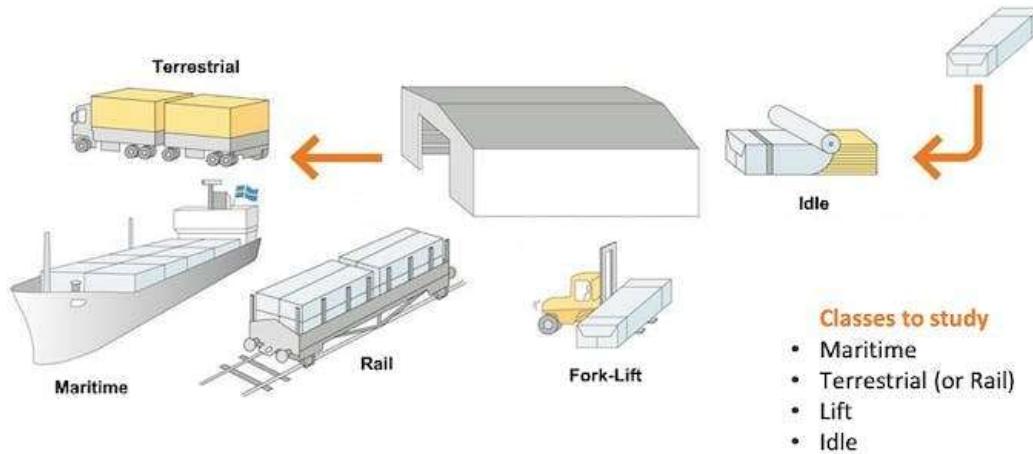


Nel progetto, *Motion Classification and Anomaly Detection*, abbiamo simulato sollecitazioni meccaniche nel trasporto, dove il nostro problema era classificare quattro classi di movimento:

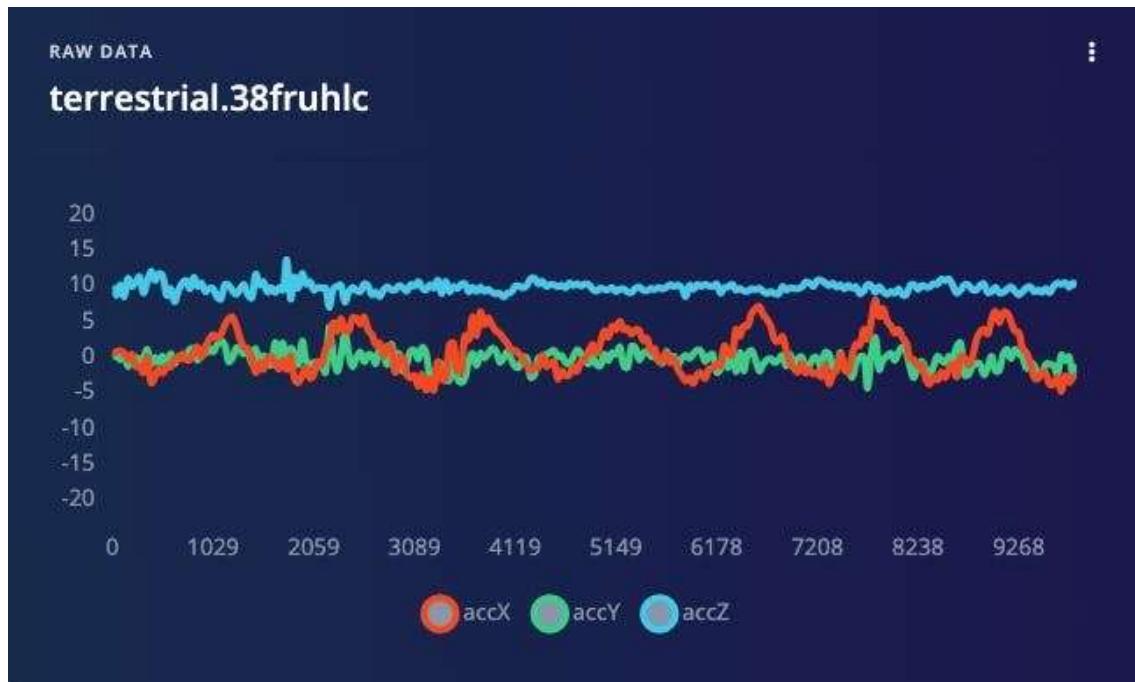
- **Maritime** (Pallet in navi)
- **Terrestrial** (pallet in un Camion o Treno)
- **Lift** [Sollevamento] (pallet movimentati da Carrello elevatore)
- **Idle** (pallet in Magazzini)

Gli accelerometri hanno fornito i dati sul pallet (o il container).

### Case Study: Mechanical Stresses in Transport



Di seguito è riportato un campione (dati grezzi) di 10 secondi, acquisito con una frequenza di campionamento di 50 Hz:

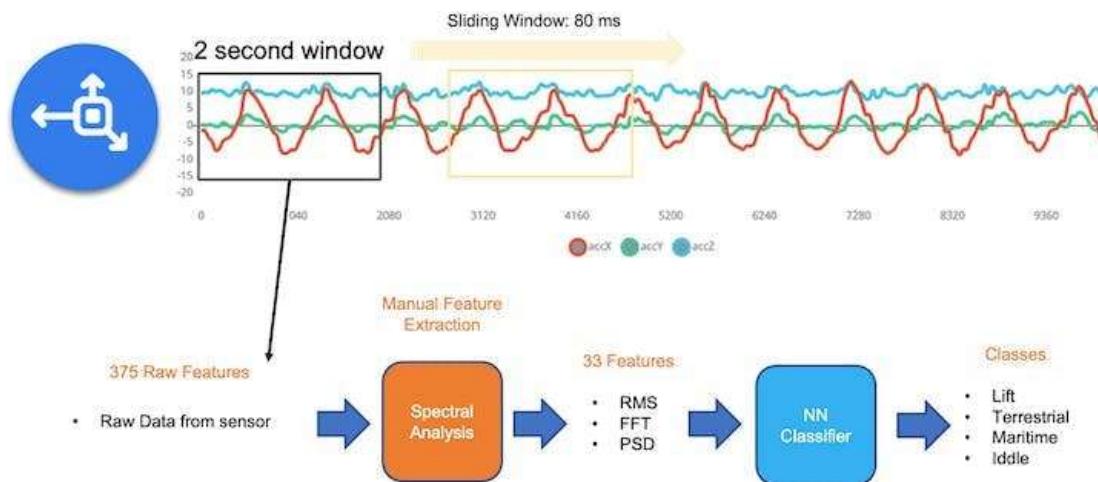


Il risultato è simile quando questa analisi viene eseguita su un altro set di dati con lo stesso principio, utilizzando una frequenza di campionamento diversa, 62.5Hz invece di 50Hz.

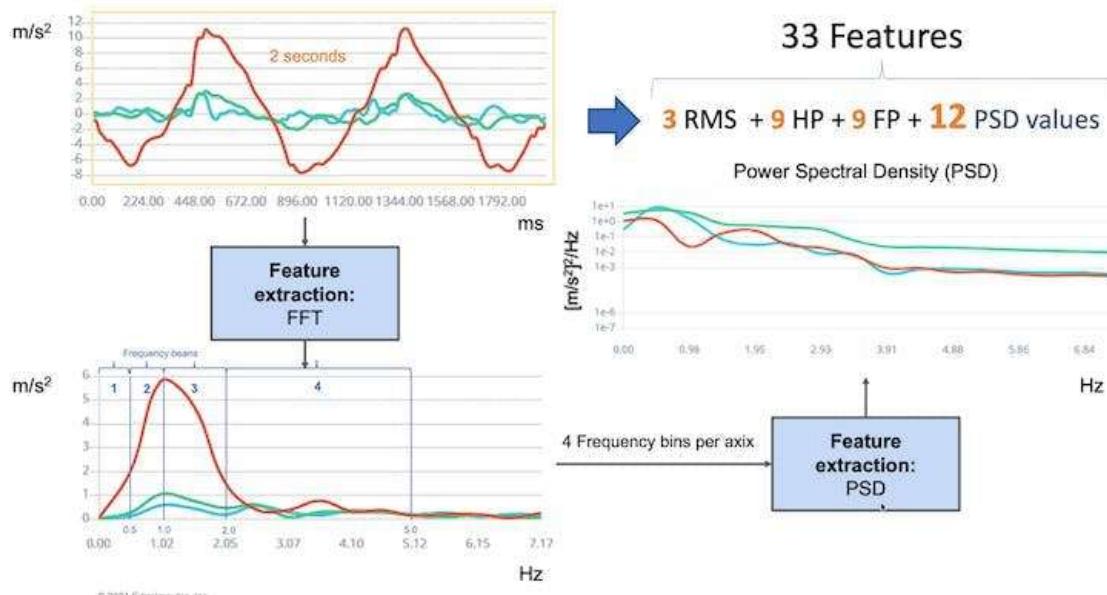
### Pre-elaborazione dei Dati

I dati grezzi acquisiti dall'accelerometro (dati di “serie temporali”) devono essere convertiti in “dati tabellari” utilizzando uno dei tipici metodi di estrazione delle feature descritti nell’ultima sezione.

Dovremmo segmentare i dati utilizzando una finestra scorrevole sui dati campione per l'estrazione delle feature. Il progetto ha acquisito i dati dell'accelerometro ogni 10 secondi con una frequenza di campionamento di 62.5 Hz. Una finestra di 2 secondi acquisisce 375 punti dati (3 assi x 2 secondi x 62.5 campioni). La finestra scorre ogni 80 ms, creando un dataset più grande in cui ogni istanza ha 375 “feature grezze”.



Su Studio, la versione precedente (V1) dello **Spectral Analysis Block** estraeva come caratteristiche del dominio del tempo solo l'RMS e, per il dominio della frequenza, i picchi e la frequenza (utilizzando FFT) e le caratteristiche di potenza (PSD) del segnale nel tempo, risultando in un set di dati tabellari fisso di 33 caratteristiche (11 per ogni asse),



Quelle 33 feature erano il tensore di input di un classificatore di reti neurali.

Nel 2022, Edge Impulse ha rilasciato la versione 2 dello Spectral Analysis block, che esploreremo qui.

### **Edge Impulse - Spectral Analysis Block V.2 funzionamento**

Nella Versione 2, le caratteristiche statistiche del dominio del tempo per asse/canale sono:

- RMS
- Skewness
- Kurtosis

E le caratteristiche spettrali del dominio della frequenza per asse/canale sono:

- Spectral Power [Potenza spettrale]
- Skewness [asimmetria] (nella prossima versione)
- Curtosi (nella prossima versione)

In questo [link](#) possiamo avere maggiori dettagli sull'estrazione delle feature.

Clonare il [progetto pubblico](#). Si può anche seguire la spiegazione, giocando col codice usando il mio Google CoLab Notebook: [Edge Impulse Spectral Analysis Block Notebook](#).

Si inizia importando le librerie:

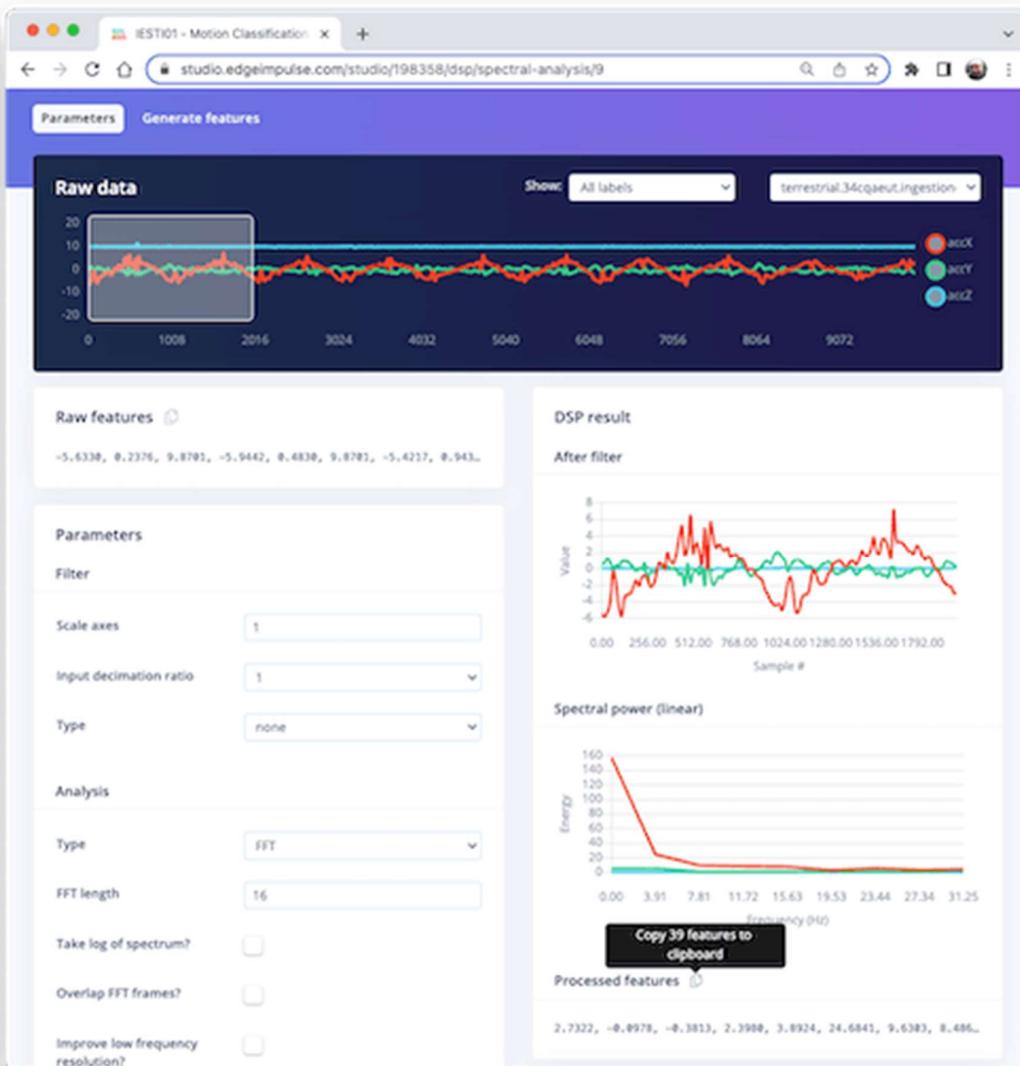
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
from scipy.stats import skew, kurtosis
from scipy import signal
from scipy.signal import welch
from scipy.stats import entropy
from sklearn import preprocessing
import pywt

plt.rcParams['figure.figsize'] = (12, 6)
plt.rcParams['lines.linewidth'] = 3
```

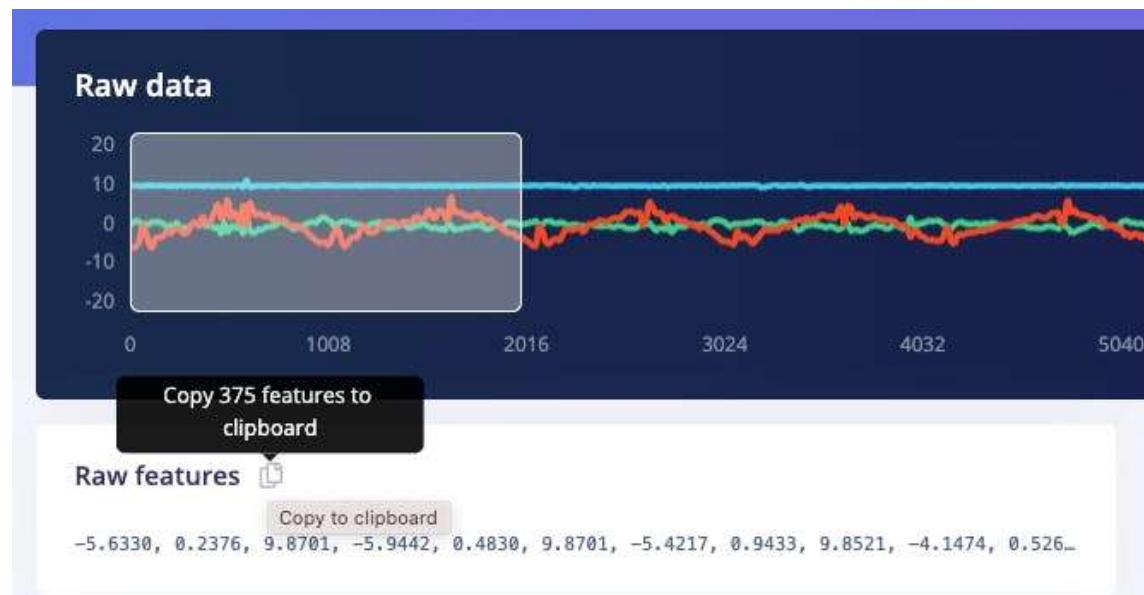
Dal progetto studiato, sceglieremo un campione di dati dagli accelerometri come di seguito:

- Dimensione della finestra di 2 secondi: [2,000] ms
- Frequenza di campionamento: [62.5] Hz
- Sceglieremo il filtro [None] (per semplicità) e una
- Lunghezza FFT: [16].

```
f = 62.5 # Hertz
wind_sec = 2 # seconds
FFT_Lenght = 16
axis = ['accX', 'accY', 'accZ']
n_sensors = len(axis)
```



Selezionando *Raw Features* nella scheda Studio Spectral Analysis, possiamo copiare tutti i 375 punti dati di una particolare finestra di 2 secondi negli appunti.



Incollare i punti dati in una nuova variabile `data`:

```
data=[-5.6330, 0.2376, 9.8701, -5.9442, 0.4830, 9.8701, -5.4217, ...]
No_raw_features = len(data)
N = int(No_raw_features/n_sensors)
```

Le feature grezze totali sono 375, ma lavoreremo con ogni asse singolarmente, dove N= 125 (numero di campioni per asse).

Vogliamo capire come Edge Impulse ottiene le feature elaborate.



Quindi, si devono anche incollare le feature elaborate su una variabile (per confrontare le feature calcolate in Python con quelle fornite da Studio):

```
features = [2.7322, -0.0978, -0.3813, 2.3980, 3.8924, 24.6841, 9.6303, ...]
N_feat = len(features)
N_feat_axis = int(N_feat/n_sensors)
```

Il numero totale di feature elaborate è 39, il che significa 13 feature/asse.

Osservando attentamente queste 13 feature, ne troveremo 3 per il dominio del tempo (RMS, Skewness e Kurtosis):

- [rms] [skew] [kurtosis]

e 10 per il dominio della frequenza (ci torneremo più avanti).

- [spectral skew][spectral kurtosis][Spectral Power 1] ... [Spectral Power 8]

### Suddivisione dei dati grezzi per sensore

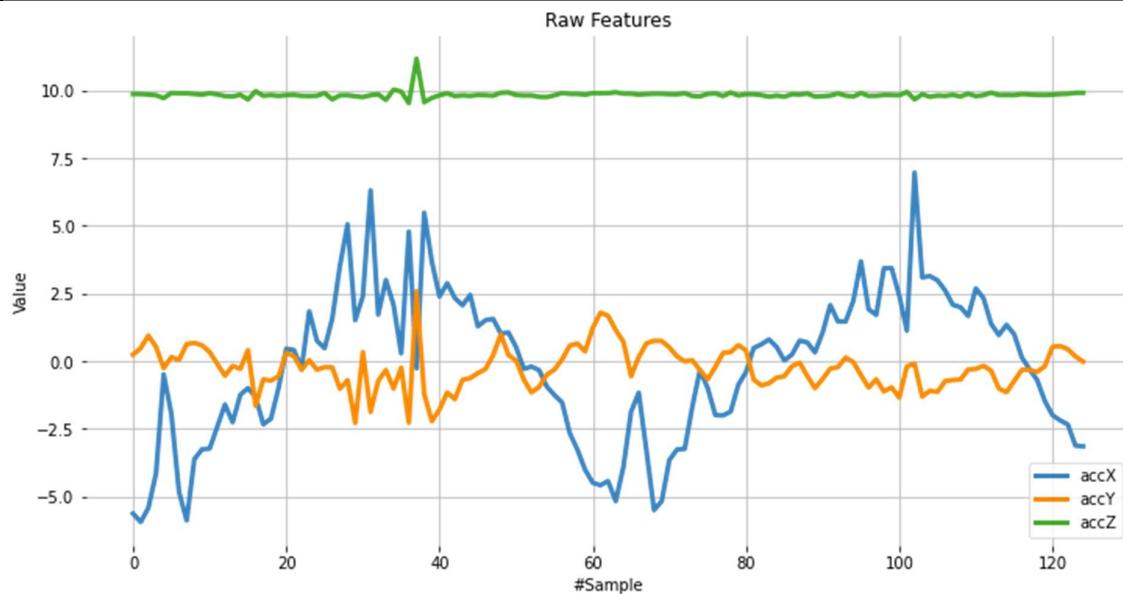
I dati hanno campioni da tutti gli assi; dividiamoli e tracciamo un grafico separatamente:

```
def plot_data(sensors, axis, title):
 [plt.plot(x, label=y) for x,y in zip(sensors, axis)]
 plt.legend(loc='lower right')
 plt.title(title)
 plt.xlabel('#Sample')
 plt.ylabel('Value')
 plt.box(False)
 plt.grid()
 plt.show()
```

```

accX = data[0::3]
accY = data[1::3]
accZ = data[2::3]
sensors = [accX, accY, accZ]
plot_data(sensors, axis, 'Raw Features')

```



### Sottrazione della media

Successivamente, dovremmo sottrarre la media dai *dati*. La sottrazione della media da un set di dati è una comune fase di pre-elaborazione dei dati in statistica e apprendimento automatico. Lo scopo della sottrazione della media dai dati è di centrare i dati attorno allo zero. Questo è importante perché può rivelare pattern e relazioni che potrebbero essere nascosti se i dati non sono centrati.

Ecco alcuni motivi specifici per cui la sottrazione della media può essere utile:

- Semplifica l'analisi: Centrando i dati, la media diventa zero, rendendo alcuni calcoli più semplici e facili da interpretare.
- Rimuove la distorsione: Se i dati sono distorti, sottraendo la media è possibile rimuoverli e consentire un'analisi più accurata.
- Può rivelare pattern: Centrare i dati può aiutare a scoprire pattern che potrebbero essere nascosti se i dati non sono centrati. Ad esempio, centrare i dati può aiutare a identificare i trend nel tempo se si analizza un set di dati di serie temporali.
- Può migliorare le prestazioni: in alcuni algoritmi di apprendimento automatico, centrare i dati può migliorare le prestazioni riducendo l'influenza dei valori anomali e rendendo i dati più facilmente confrontabili. Nel complesso, sottrarre la media è una tecnica semplice ma potente che può essere utilizzata per migliorare l'analisi e l'interpretazione dei dati.

```

dtmean = [(sum(x)/len(x)) for x in sensors]
[print('mean_'+x+'= ', round(y, 4)) for x,y in zip(axis, dtmean)][0]

accX = [(x - dtmean[0]) for x in accX]
accY = [(x - dtmean[1]) for x in accY]
accZ = [(x - dtmean[2]) for x in accZ]
sensors = [accX, accY, accZ]

plot_data(sensors, axis, 'Raw Features - Subtract the Mean')

```



## Feature Statistiche del Dominio del Tempo

### Calcolo RMS

Il valore RMS di un set di valori (o di una forma d'onda a tempo continuo) è la radice quadrata della media aritmetica dei quadrati dei valori o del quadrato della funzione che definisce la forma d'onda continua. In fisica, il valore RMS di una corrente elettrica è definito come il “valore della corrente continua che dissipava la stessa potenza in un resistore”.

Nel caso di un set di n valori  $\{x_1, x_2, \dots, x_n\}$ , l'RMS è:

$$x_{\text{RMS}} = \sqrt{\frac{1}{n} (x_1^2 + x_2^2 + \dots + x_n^2)} .$$

NOTARE che il valore RMS è diverso per i dati grezzi originali e dopo aver sottratto la media

```
Using numpy and standartized data (subtracting mean)
rms = [np.sqrt(np.mean(np.square(x))) for x in sensors]
```

Possiamo confrontare i valori RMS calcolati qui con quelli presentati da Edge Impulse:

```
[print('rms_+'x+'= ', round(y, 4)) for x,y in zip(axis, rms)][0]
print("\nCompare with Edge Impulse result features")
print(features[0:N_feat:N_feat_axis])
rms_accX= 2.7322
rms_accY= 0.7833
rms_accZ= 0.1383
```

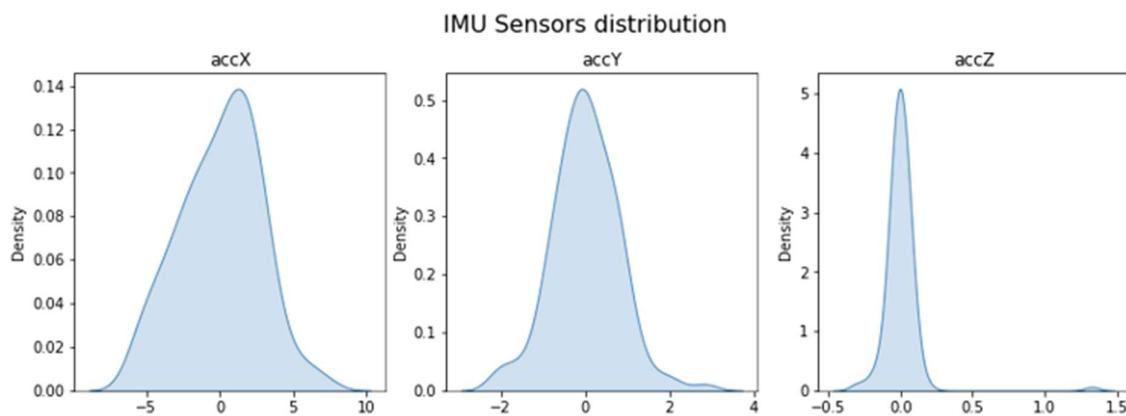
Rispetto alle caratteristiche dei risultati di Edge Impulse:

```
[2.7322, 0.7833, 0.1383]
```

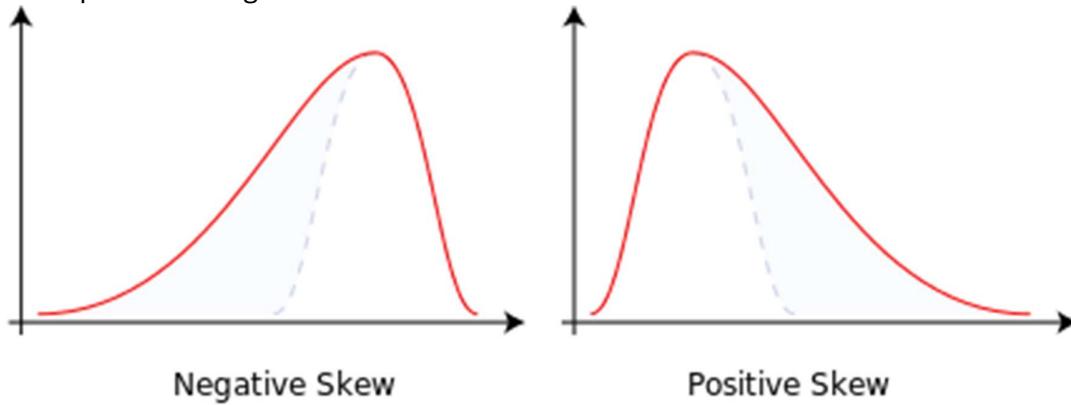
### Calcolo di asimmetria e curtosi

In statistica, asimmetria e curtosi sono due modi per misurare la **forma di una distribuzione**. Qui possiamo vedere la distribuzione dei valori del sensore:

```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(13, 4))
sns.kdeplot(accX, fill=True, ax=axes[0])
sns.kdeplot(accY, fill=True, ax=axes[1])
sns.kdeplot(accZ, fill=True, ax=axes[2])
axes[0].set_title('accX')
axes[1].set_title('accY')
axes[2].set_title('accZ')
plt.suptitle('IMU Sensors distribution', fontsize=16, y=1.02)
plt.show()
```



La **Skewness** [asimmetria] è una misura dell'asimmetria di una distribuzione. Questo valore può essere positivo o negativo.



- Un'asimmetria negativa indica che la coda si trova sul lato sinistro della distribuzione, che si estende verso valori più negativi.
- Un'asimmetria positiva indica che la coda si trova sul lato destro della distribuzione, che si estende verso valori più positivi.
- Un valore zero indica che non c'è alcuna asimmetria nella distribuzione, il che significa che la distribuzione è perfettamente simmetrica.

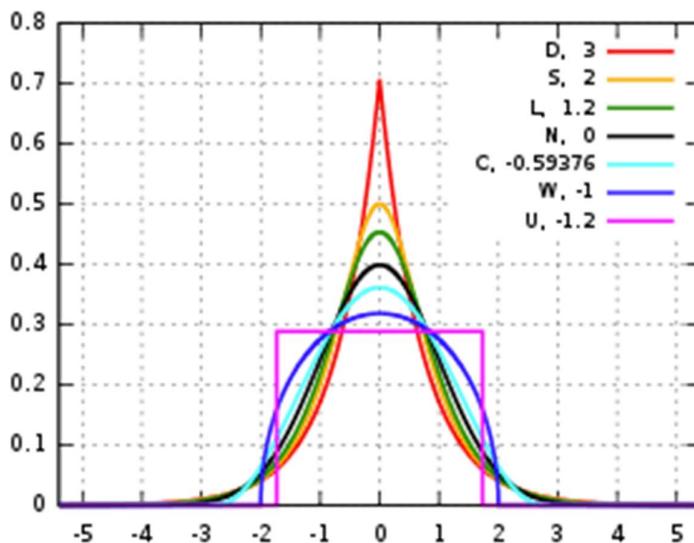
```
skew = [skew(x, bias=False) for x in sensors]
[print('skew_'+x+'= ', round(y, 4)) for x,y in zip(axis, skew)][0]
print("\nCompare with Edge Impulse result features")
features[1:N_feat:N_feat_axis]
```

```
skew_accX= -0.099
skew_accY= 0.1756
skew_accZ= 6.9463
```

Rispetto alle caratteristiche dei risultati di Edge Impulse:

```
[-0.0978, 0.1735, 6.8629]
```

La **Kurtosis** è una misura che indica se una distribuzione è a coda pesante o a coda leggera rispetto a una distribuzione normale.



- La curtosi di una distribuzione normale è zero.
- Se una data distribuzione ha una curtosi negativa, si dice che è platicurtica, il che significa che tende a produrre meno valori anomali e meno estremi rispetto alla distribuzione normale.
- Se una data distribuzione ha una curtosi positiva, si dice che è leptocurtica, il che significa che tende a produrre più valori anomali rispetto alla distribuzione normale.

```
kurt = [kurtosis(x, bias=False) for x in sensors]
[print('kurt_+' + x + '=' , round(y, 4)) for x,y in zip(axis, kurt)][0]
print("\nCompare with Edge Impulse result features")
features[2:N_feat:N_feat_axis]
kurt_accX= -0.3475
kurt_accY= 1.2673
kurt_accZ= 68.1123
```

Rispetto alle caratteristiche dei risultati di Edge Impulse:

```
[-0.3813, 1.1696, 65.3726]
```

## Feature spettrali

Il segnale filtrato viene trasmesso alla sezione di potenza spettrale, che calcola la **FFT** per generare le caratteristiche spettrali.

Poiché la finestra campionata è solitamente più grande della dimensione FFT, la finestra verrà suddivisa in frame (o “sotto-finestre”) e la FFT verrà calcolata su ogni frame.

**FFT length** - La dimensione della FFT. Questo determina il numero di bin FFT e la risoluzione dei picchi di frequenza che possono essere separati. Un numero basso significa che più segnali saranno mediati insieme nello stesso bin FFT, ma riduce anche il numero di feature e la dimensione del modello. Un numero alto separerà più segnali in bin separati, generando un modello più grande.

- Il numero totale di feature di potenza spettrale varierà a seconda di come si impostano i parametri del filtro e di FFT. Senza filtro, il numero di feature è 1/2 della lunghezza FFT.

## Spectral Power - Metodo di Welch

Dovremmo usare il [metodo di Welch](#) per dividere il segnale nel dominio della frequenza in “bin” e calcolare lo spettro di potenza per ogni bin. Questo metodo divide il segnale in segmenti sovrapposti, applica una funzione finestra a ogni segmento, calcola il periodogramma di ogni segmento usando DFT e ne fa la media per ottenere una stima più uniforme dello spettro di potenza.

```
Function used by Edge Impulse instead of scipy.signal.welch()
def welch_max_hold(fx, sampling_freq, nfft, n_overlap):
 n_overlap = int(n_overlap)
 spec_powers = [0 for _ in range(nfft//2+1)]
```

```

ix = 0
while ix <= len(fx):
 # Slicing truncates if end_idx > len, and rfft will auto-zero pad
 fft_out = np.abs(np.fft.rfft(fx[ix:ix+nfft], nfft))
 spec_powers = np.maximum(spec_powers, fft_out**2/nfft)
 ix = ix + (nfft-n_overlap)
return np.fft.rfftfreq(nfft, 1/sampling_freq), spec_powers

```

Applicazione della funzione di cui sopra a 3 segnali:

```

fax,Pax = welch_max_hold(accX, fs, FFT_Lenght, 0)
fay,Pay = welch_max_hold(accY, fs, FFT_Lenght, 0)
faz,Paz = welch_max_hold(accZ, fs, FFT_Lenght, 0)
specs = [Pax, Pay, Paz]

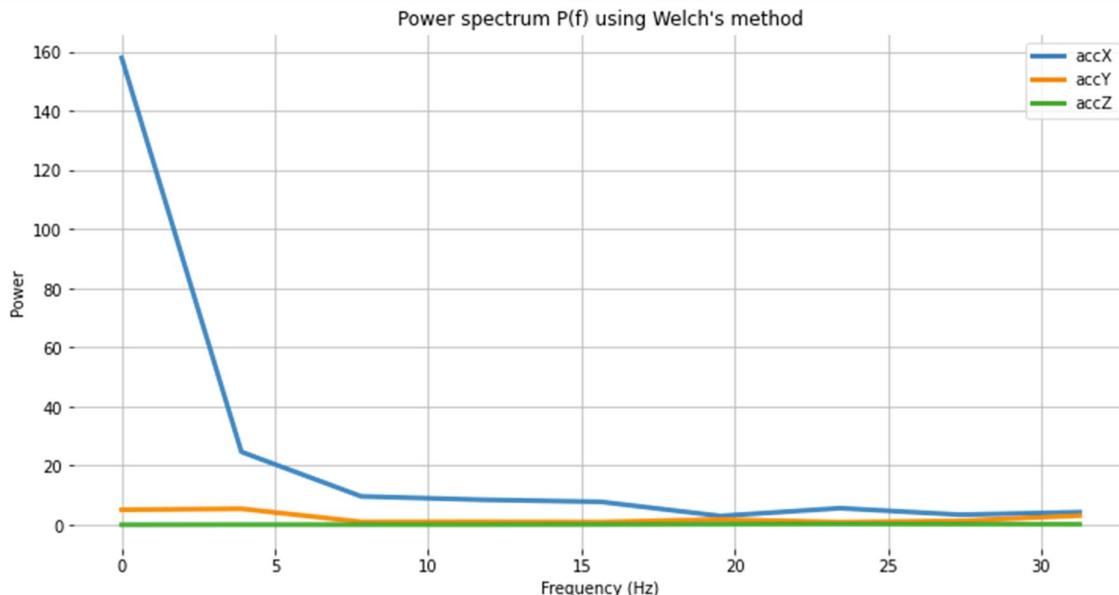
```

Possiamo tracciare lo spettro di potenza P(f):

```

plt.plot(fax,Pax, label='accX')
plt.plot(fay,Pay, label='accY')
plt.plot(faz,Paz, label='accZ')
plt.legend(loc='upper right')
plt.xlabel('Frequency (Hz)')
plt.ylabel('PSD [V**2/Hz]')
plt.ylabel('Power')
plt.title('Power spectrum P(f) using Welch's method')
plt.grid()
plt.box(False)
plt.show()

```



Oltre allo spettro di potenza, possiamo anche includere l'asimmetria e la curtosi delle feature nel dominio della frequenza (dovrebbero essere disponibili in una nuova versione):

```

spec_skew = [skew(x, bias=False) for x in specs]
spec_kurtosis = [kurtosis(x, bias=False) for x in specs]

```

Elenchiamo ora tutte le feature spettrali per asse e confrontarli con EI:

```

print("EI Processed Spectral features (accX): ")
print(features[3:N_feat_axis][0:])
print("\nCalculated features:")
print (round(spec_skew[0],4))
print (round(spec_kurtosis[0],4))
[print(round(x, 4)) for x in Pax[1:]][0]

```

EI Processed Spectral features (accX):

2.398, 3.8924, 24.6841, 9.6303, 8.4867, 7.7793, 2.9963, 5.6242, 3.4198, 4.2735

Calculated features:

2.9069 8.5569 24.6844 9.6304 8.4865 7.7794 2.9964 5.6242 3.4198 4.2736

```

print("EI Processed Spectral features (accY): ")
print(features[16:26][0:]) #13: 3+N_feat_axis; 26 = 2x N_feat_axis
print("\nCalculated features:")
print (round(spec_skew[1],4))

```

```
print (round(spec_kurtosis[1],4))
[print(round(x, 4)) for x in Pay[1:]] [0]
```

EI Processed Spectral features (accY):

0.9426, -0.8039, 5.429, 0.999, 1.0315, 0.9459, 1.8117, 0.9088, 1.3302, 3.112

Calculated features:

1.1426 -0.3886 5.4289 0.999 1.0315 0.9458 1.8116 0.9088 1.3301 3.1121

```
print("EI Processed Spectral features (accZ): ")
print(features[29:][0:]) #29: 3+(2*N_feat_axis);
print("\nCalculated features:")
print (round(spec_skew[2],4))
print (round(spec_kurtosis[2],4))
[print(round(x, 4)) for x in Paz[1:]] [0]
```

EI Processed Spectral features (accZ):

0.3117, -1.3812, 0.0606, 0.057, 0.0567, 0.0976, 0.194, 0.2574, 0.2083, 0.166

Calculated features:

0.3781 -1.4874 0.0606 0.057 0.0567 0.0976 0.194 0.2574 0.2083 0.166

## Dominio tempo-frequenza

### Wavelet

Wavelet è una tecnica potente per analizzare segnali con feature transitorie o bruschi cambiamenti, come picchi o bordi, che sono difficili da interpretare con i metodi tradizionali basati su Fourier.

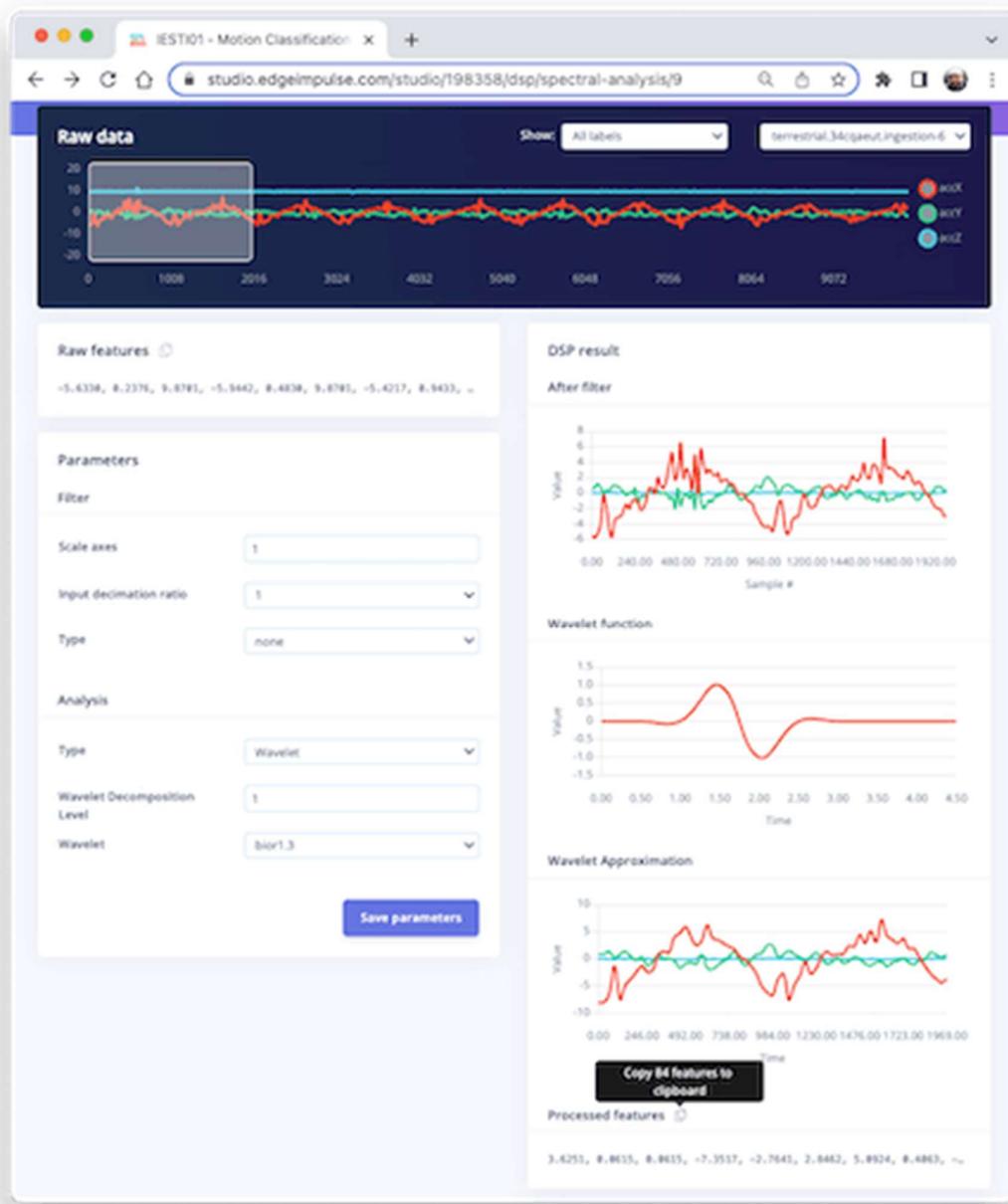
Le trasformazioni wavelet funzionano scomponendo un segnale in diverse componenti di frequenza e analizzandole individualmente. La trasformazione si ottiene convolvendo il segnale con una **funzione wavelet**, una piccola forma d'onda centrata su un tempo e una frequenza specifici. Questo processo scomponete efficacemente il segnale in diverse bande di frequenza, ciascuna delle quali può essere analizzata separatamente.

Uno dei vantaggi fondamentali delle trasformazioni wavelet è che consentono l'analisi tempo-frequenza, il che significa che possono rivelare il contenuto di frequenza di un segnale mentre cambia nel tempo. Ciò le rende particolarmente utili per analizzare segnali non stazionari, che variano nel tempo.

Le wavelet hanno molte applicazioni pratiche, tra cui la compressione di segnali e immagini, il denoising [rimozione del rumore], l'estrazione di feature e l'elaborazione di immagini.

Selezioniamo Wavelet sul blocco Spectral Features nello stesso progetto:

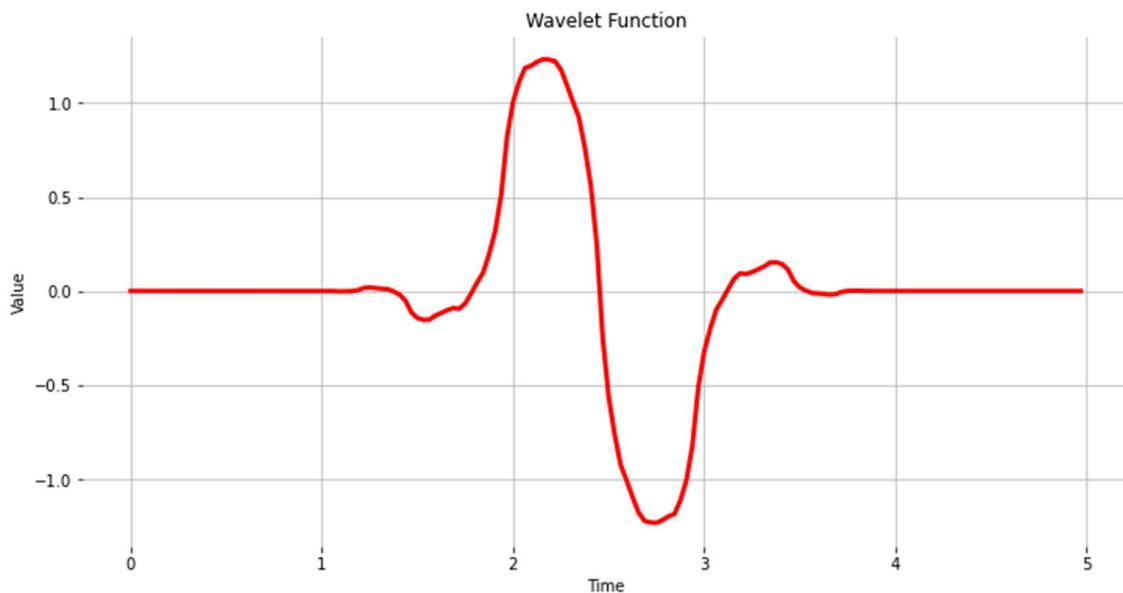
- Type: Wavelet
- Wavelet Decomposition Level: 1
- Wavelet: bior1.3



## La Funzione Wavelet

```
wavelet_name='bior1.3'
num_layer = 1

wavelet = pywt.Wavelet(wavelet_name)
[phi_d,psi_d,phi_r,psi_r,x] = wavelet.wavefun(level=5)
plt.plot(x, psi_d, color='red')
plt.title('Wavelet Function')
plt.ylabel('Value')
plt.xlabel('Time')
plt.grid()
plt.box(False)
plt.show()
```



Come abbiamo fatto prima, copiamo e incolliamo le Processed Features:

Copy 84 features to clipboard

Processed features

Copy to clipboard

3.6251, 0.0615, 0.0615, -7.3517, -2.7641, 2.8462, 5.0924, 0.4063, -0.2133, 3.8473, 15.032...

```
features = [3.6251, 0.0615, 0.0615, -7.3517, -2.7641, 2.8462, 5.0924, ...]
N_feat = len(features)
N_feat_axis = int(N_feat/n_sensors)
```

Edge Impulse calcola la [Discrete Wavelet Transform \(DWT\)](#) per ciascuno dei livelli di Wavelet Decomposition selezionati. Dopodiché, le feature verranno estratte.

Nel caso di **Wavelet**, le feature estratte sono *valori statistici di base*, *valori di incrocio* ed *entropia*. Ci sono, in totale, 14 feature per layer come di seguito:

- [11] Feature Statistiche: **n5**, **n25**, **n75**, **n95**, **media**, **mediana**, deviazione standard (**std**), varianza (**var**) radice quadrata media (**rms**), **curtosi** e asimmetria (**skew**).
- [2] Feature di attraversamento: Il tasso di attraversamento dello zero (**zcross**) e il tasso di attraversamento medio (**mcross**) sono rispettivamente i tempi in cui il segnale attraversa la linea di base ( $y = 0$ ) e il livello medio ( $y = u$ ) per unità di tempo
- [1] Feature di Complessità: L'**entropia** è una misura caratteristica della complessità del segnale

Tutti i 14 valori sopra indicati vengono calcolati per ogni Layer (incluso L0, il segnale originale)

- Il numero totale di feature varia a seconda di come si imposta il filtro e del numero di layer. Ad esempio, con il filtro [None] e Level[1], il numero di feature per asse sarà  $14 \times 2$  (L0 e L1) = 28. Per i tre assi, avremo un totale di 84 feature.

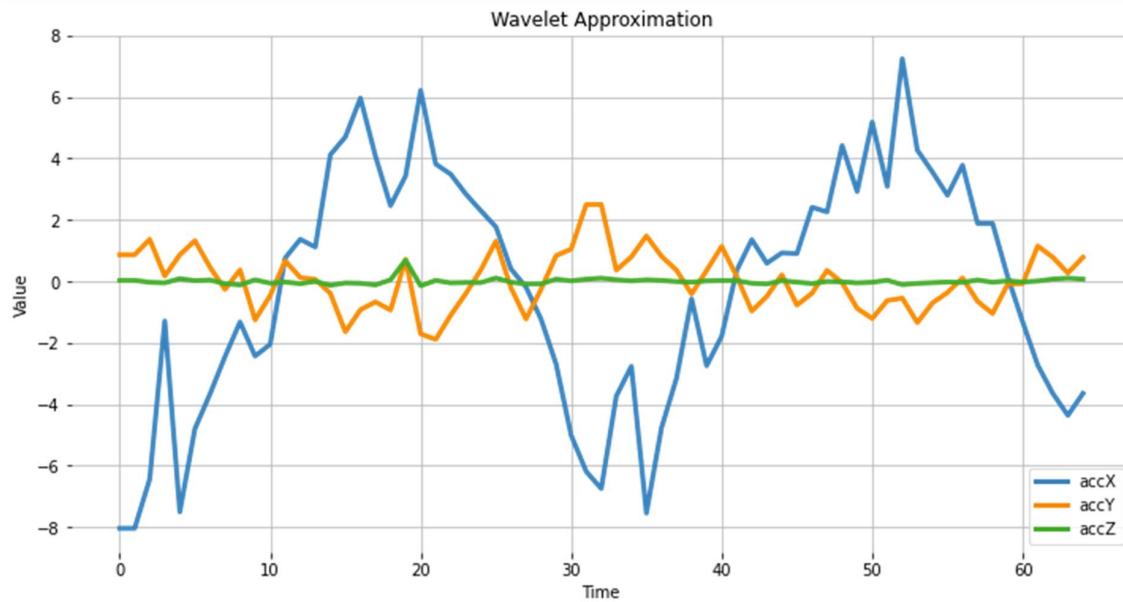
### **Analisi Wavelet**

L'analisi wavelet scomponete il segnale (**accX**, **accY**, e **accZ**) in diverse componenti di frequenza utilizzando un set di filtri, che separano queste componenti in componenti a bassa frequenza (parti del segnale che variano lentamente e contengono pattern a lungo termine), come **accX\_l1**, **accY\_l1**, **accZ\_l1** e componenti ad alta frequenza (parti del segnale che variano rapidamente e contengono modelli a breve termine), come **accX\_d1**, **accY\_d1**, **accZ\_d1**, consentendo l'estrazione di feature per ulteriori analisi o classificazioni.

Verranno utilizzati solo le componenti a bassa frequenza (coefficienti di approssimazione o cA). In questo esempio, assumiamo un solo livello (Single-level Discrete Wavelet Transform), in cui la funzione restituirà una tupla. Con una decomposizione multilivello, la “Multilevel 1D Discrete Wavelet Transform”, il risultato sarà un elenco (per i dettagli, vedere: [Discrete Wavelet Transform \(DWT\)](#))

```
(accX_11, accX_d1) = pywt.dwt(accX, wavelet_name)
(accY_11, accY_d1) = pywt.dwt(accY, wavelet_name)
(accZ_11, accZ_d1) = pywt.dwt(accZ, wavelet_name)
sensors_11 = [accX_11, accY_11, accZ_11]

Plot power spectrum versus frequency
plt.plot(accX_11, label='accX')
plt.plot(accY_11, label='accY')
plt.plot(accZ_11, label='accZ')
plt.legend(loc='lower right')
plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Wavelet Approximation')
plt.grid()
plt.box(False)
plt.show()
```



### Estrazione delle Feature

Cominciamo con le feature statistiche di base. Notare che applichiamo la funzione sia per i segnali originali che per i cAs risultanti dal DWT:

```
def calculate_statistics(signal):
 n5 = np.percentile(signal, 5)
 n25 = np.percentile(signal, 25)
 n75 = np.percentile(signal, 75)
 n95 = np.percentile(signal, 95)
 median = np.percentile(signal, 50)
 mean = np.mean(signal)
 std = np.std(signal)
 var = np.var(signal)
 rms = np.sqrt(np.mean(np.square(signal)))
 return [n5, n25, n75, n95, median, mean, std, var, rms]

stat_feat_10 = [calculate_statistics(x) for x in sensors]
stat_feat_11 = [calculate_statistics(x) for x in sensors_11]
```

### Asimmetria e Curtosi:

```
skew_10 = [skew(x, bias=False) for x in sensors]
skew_11 = [skew(x, bias=False) for x in sensors_11]
kurtosis_10 = [kurtosis(x, bias=False) for x in sensors]
kurtosis_11 = [kurtosis(x, bias=False) for x in sensors_11]
```

**Zero crossing (zcross)** è il numero di volte in cui il coefficiente wavelet attraversa l'asse dello zero. Può essere utilizzato per misurare il contenuto di frequenza del segnale poiché i segnali ad alta frequenza tendono ad avere più attraversamenti per lo zero rispetto ai segnali a bassa frequenza.

**Mean crossing (mcross)**, d'altra parte, è il numero di volte in cui il coefficiente wavelet attraversa la media del segnale. Può essere utilizzato per misurare l'ampiezza poiché i segnali ad alta ampiezza tendono ad avere più attraversamenti medi rispetto ai segnali a bassa ampiezza.

---

```

def getZeroCrossingRate(arr):
 my_array = np.array(arr)
 zcross = float("{0:.2f}".format(((my_array[:-1] * my_array[1:]) < 0).sum() / len(arr)))
 return zcross

def getMeanCrossingRate(arr):
 mcross = getZeroCrossingRate(np.array(arr) - np.mean(arr))
 return mcross

def calculate_crossings(list):
 zcross=[]
 mcross=[]
 for i in range(len(list)):
 zcross_i = getZeroCrossingRate(list[i])
 zcross.append(zcross_i)
 mcross_i = getMeanCrossingRate(list[i])
 mcross.append(mcross_i)
 return zcross, mcross

cross_10 = calculate_crossings(sensors)
cross_11 = calculate_crossings(sensors_11)

```

---

Nell'analisi wavelet, l'**entropia** si riferisce al grado di disordine o casualità nella distribuzione dei coefficienti wavelet. Qui, abbiamo utilizzato l'entropia di Shannon, che misura l'incertezza o la casualità di un segnale. Viene calcolata come la somma negativa delle probabilità dei diversi possibili risultati del segnale moltiplicata per il loro logaritmo in base 2. Nel contesto dell'analisi wavelet, l'entropia di Shannon può essere utilizzata per misurare la **complessità del segnale**, con valori più alti che indicano una maggiore complessità.

---

```

def calculate_entropy(signal, base=None):
 value, counts = np.unique(signal, return_counts=True)
 return entropy(counts, base=base)

entropy_10 = [calculate_entropy(x) for x in sensors]
entropy_11 = [calculate_entropy(x) for x in sensors_11]

```

---

Elenchiamo ora tutte le feature wavelet e creiamo un elenco per layer.

---

```

L1_features_names = ["L1-n5", "L1-n25", "L1-n75", "L1-n95", "L1-median", "L1-mean", "L1-sd",
 "L1-var", "L1-rms", "L1-skew", "L1-Kurtosis", "L1-zcross", "L1-mcross", "L1-entropy"]
]

L0_features_names = ["L0-n5", "L0-n25", "L0-n75", "L0-n95", "L0-median", "L0-mean", "L0-sd",
 "L0-var", "L0-rms", "L0-skew", "L0-Kurtosis", "L0-zcross", "L0-mcross", "L0-entropy"]
]

all_feat_10 = []
for i in range(len(axis)):
 feat_10 = stat_feat_10[i]+[skew_10[i]]+[kurtosis_10[i]]+[cross_10[0][i]]+[cross_10[1][i]]+[entropy_10[i]]
 [print(axis[i]+'+x+'= ', round(y, 4)) for x,y in zip(L0_features_names, feat_10)][0]
]
 all_feat_10.append(feat_10)
all_feat_10 = [item for sublist in all_feat_10 for item in sublist]
print(f"\nAll L0 Features = {len(all_feat_10)}")

all_feat_11 = []
for i in range(len(axis)):
feat_11 = stat_feat_11[i]+[skew_11[i]]+[kurtosis_11[i]]+[cross_11[0][i]]+[cross_11[1][i]]+[entropy_11[i]]
[print(axis[i]+'+x+'= ', round(y, 4)) for x,y in zip(L1_features_names, feat_11)][0]

```

---

```

all_feat_ll.append(feat_ll)
all_feat_ll = [item for sublist in all_feat_ll for item in sublist]
print(f"\nAll Ll Features = {len(all_feat_ll)}")

 accX L0-n5= -4.9364 accX L1-n5= -7.3516
 accX L0-n25= -1.8429 accX L1-n25= -2.7641
 accX L0-n75= 1.8842 accX L1-n75= 2.8462
 accX L0-n95= 3.8096 accX L1-n95= 5.0924
 accX L0-median= 0.4058 accX L1-median= 0.4064
 accX L0-mean= -0.0 accX L1-mean= -0.2133
 accX L0-std= 2.7322 accX L1-std= 3.8473
 accX L0-var= 7.4651 accX L1-var= 14.8015
 accX L0-rms= 2.7322 accX L1-rms= 3.8532
 accX L0-skew= -0.099 accX L1-skew= -0.2975
 accX L0-Kurtosis= -0.3475 accX L1-Kurtosis= -0.7631
 accX L0-zcross= 0.06 accX L1-zcross= 0.06
 accX L0-mcross= 0.06 accX L1-mcross= 0.06
 accX L0-entropy= 4.8283 accX L1-entropy= 4.1744
 accY L0-n5= -1.149 accY L1-n5= -1.3234
 accY L0-n25= -0.4475 accY L1-n25= -0.6492
 accY L0-n75= 0.4814 accY L1-n75= 0.7844
 accY L0-n95= 1.1491 accY L1-n95= 1.361
 accY L0-median= -0.0315 accY L1-median= 0.0659
 accY L0-mean= 0.0 accY L1-mean= 0.0276
 accY L0-std= 0.7833 accY L1-std= 0.9345
 accY L0-var= 0.6136 accY L1-var= 0.8732
 accY L0-rms= 0.7833 accY L1-rms= 0.9349
 accY L0-skew= 0.1756 accY L1-skew= 0.2874
 accY L0-Kurtosis= 1.2673 accY L1-Kurtosis= 0.0347
 accY L0-zcross= 0.29 accY L1-zcross= 0.31
 accY L0-mcross= 0.29 accY L1-mcross= 0.31
 accY L0-entropy= 4.8283 accY L1-entropy= 4.1317
 accZ L0-n5= -0.1242 accZ L1-n5= -0.1126
 accZ L0-n25= -0.0429 accZ L1-n25= -0.0493
 accZ L0-n75= 0.0349 accZ L1-n75= 0.0348
 accZ L0-n95= 0.0839 accZ L1-n95= 0.1022
 accZ L0-median= -0.0112 accZ L1-median= -0.0137
 accZ L0-mean= 0.0 accZ L1-mean= 0.0025
 accZ L0-std= 0.1383 accZ L1-std= 0.1053
 accZ L0-var= 0.0191 accZ L1-var= 0.0111
 accZ L0-rms= 0.1383 accZ L1-rms= 0.1053
 accZ L0-skew= 6.9463 accZ L1-skew= 4.4095
 accZ L0-Kurtosis= 68.1123 accZ L1-Kurtosis= 28.6586
 accZ L0-zcross= 0.35 accZ L1-zcross= 0.4
 accZ L0-mcross= 0.35 accZ L1-mcross= 0.37
 accZ L0-entropy= 4.5649 accZ L1-entropy= 4.1531

All L0 Features = 42 All L1 Features = 42

```

## Conclusione

Edge Impulse Studio è una potente piattaforma online in grado di gestire per noi l'attività di pre-elaborazione. Tuttavia, data la nostra prospettiva ingegneristica, vogliamo capire cosa sta succedendo “sotto il cofano”. Questa conoscenza ci aiuterà a trovare le migliori opzioni e gli iperparametri per ottimizzare i nostri progetti.

Daniel Situnayake ha scritto nel suo [blog](#): “I dati grezzi dei sensori sono altamente dimensionali e rumorosi. Gli algoritmi di elaborazione del segnale digitale ci aiutano a separare il segnale dal rumore. Il DSP è una parte essenziale dell’ingegneria embedded e molti processori edge hanno un’accelerazione integrata per il DSP. Come ingegnere ML, imparare il DSP di base dà dei superpoteri per gestire dati di serie temporali ad alta frequenza nei propri modelli”. Consiglio di leggere l'eccellente post di Dan nella sua interezza: [nn to cpp: What you need to know about porting deep learning models to the edge](#) [Tutto ciò che devi sapere sul porting dei modelli di deep learning verso l'edge].