UNIVERSITÀ
POLITECNICA
DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Elettronica

Studio, sviluppo e valutazione prestazionale di uno smart sensor supportato da edge computing

Study, development and performance evaluation of a smart sensor based on edge computing

Tesi di laurea di:
SIMONE GUALTIERI

Relatore:
Prof.ssa **SUSANNA SPINSANTE**

Correlatore:
Prof.ssa **STEFANIA CECCHI**

Anno Accademico 2021-2022

Abstract

La presente tesi verte sullo studio, lo sviluppo e la valutazione delle prestazioni di uno smart sensor, realizzato attraverso la piattaforma Arduino NANO 33 BLE Sense, per la raccolta e l'analisi di parametri ambientali, secondo il paradigma di Edge computing. In particolare, ci si focalizza sullo sviluppo e l'implementazione di una rete neurale a bordo della scheda, allo scopo di riconoscere determinati comandi vocali ed attuare di conseguenza una specifica azione, ad esempio leggere un sensore oppure accendere un led. Il sistema creato funzionerà in modo simile a "Hey Siri" o "OK Google" e sarà in grado di riconoscere parole chiave anche in presenza di rumori di fondo.

All'atto pratico, sfruttando un ampio database di parole e rumori di fondo messo a disposizione dalla stessa Google, dunque abbastanza strutturato, la scheda ha eccellenti prestazioni nella classificazione di quattro label (due keywords), con il modello di rete neurale considerato, anche se non ottime nella velocità di esecuzione. Le performance sono ancora soddisfacenti quando, invece di considerare un segnale vocale, si fa riferimento ad un movimento, rilevato dal sensore accelerometrico. Si ottengono buona velocità di risposta e accuratezza. Notevoli differenze si hanno invece al variare della frequenza di campionamento del sensore; non abbassandola, per cui non si ha degrado delle prestazioni, ma anzi un aumento nella velocità di risposta, ma aumentandola; la scheda mostra infatti i suoi limiti strutturali, soffrendo per il grande numero di campioni da elaborare ed in termini di velocità di risposta si vede un brusco decadimento delle prestazioni, con un dilatamento dei tempi non compatibile con applicazioni di interesse pratico. Si potrebbero sfruttare a pieno le potenzialità di Arduino e schede simili in ambito Edge, con una opportuna ottimizzazione del modello di rete neurale e delle risorse del sistema. La tesi in esame può essere certamente un punto di partenza per dispositivi simili, più precisi e performanti.

Indice

Capitolo 1

La piattaforma ARDUINO

1.1 Arduino Nano 33 BLE Sense

In questo capitolo si farà una veloce panoramica sulla scheda e sui sensori che verranno utilizzati nel prosieguo della trattazione, in quanto ormai Arduino è un marchio conosciuto a livello mondiale e le sue schede sono largamente utilizzate in tutto il mondo in svariati ambiti e non hanno certamente bisogno di troppe presentazioni. Per ulteriori approfondimenti si rimanda al sito ufficiale Arduino [7] [8].

La scheda Arduino Nano 33 BLE Sense (figura 1.1) è un'evoluzione della tradizionale Arduino Nano, ma con un processore molto più performante: l'nRF52840 di Nordic Semiconductors, 32 bit ARM® Cortex®-M4 CPU a 64 MHz. Questo permette la scrittura di codici più lunghi e con più variabili rispetto ad Arduino Uno (ha 1 MB di memoria di programma, 32 volte più grande rispetto Arduino Uno e una RAM 128 volte maggiore). Il processore include altre funzionalità come l'accoppiamento Bluetooth® tramite NFC e modalità ultra-low power.

La caratteristica principale di questa scheda è la possibilità di eseguire applicazioni di Edge Computing utilizzando TinyML. Questo grazie anche alla grande quantità di sensori integrati:

- Sensore inerziale a 9 assi (3D IMU)
- Sensore di temperatura e umidità
- Barometro
- Microfono
- Sensore di gesti, prossimità, colore e luminosità.

Figura 1.1 – Arduino Nano 33 BLE Sense

La scheda ha inoltre la possibilità di effettuare comunicazioni tramite Bluetooth® e Bluetooth® Low Energy (BLE). Ulteriori novità sono la porta micro-USB e la tensione di 3.3 V. Come per altre schede Arduino anche questa è dotata di led integrati, pin ingresso/uscita digitali ed analogici, connessioni UART, SPI e I2C. Una veloce panoramica è data in figura 1.2.

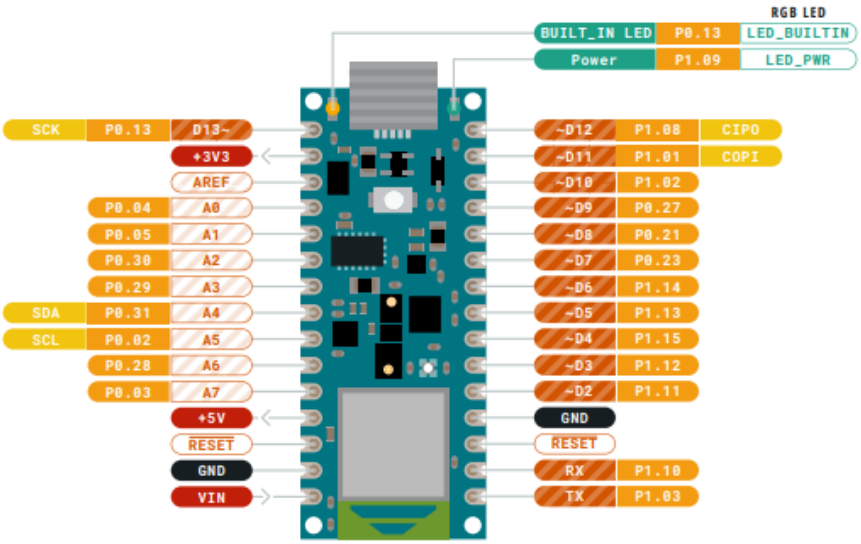


Figura 1.2 - Pinout

Per la programmazione si può utilizzare il solito Arduino Desktop IDE o, in alternativa, il più recente Web Editor online; per approfondimenti si rimanda al sito, dove è possibile anche trovare guide e lezioni sul linguaggio di programmazione con esempi pratici [8].

1.2 Sensori

1.2.1 IMU

IMU sta per **inertial measurement unit**. È un dispositivo elettronico che misura e segnala la accelerazione di un corpo, la velocità angolare e l'orientamento del corpo, utilizzando una combinazione di accelerometro, giroscopio e magnetometro. L'IMU integrata nel Nano BLE 33 Sense è la LSM9DS1 di ST Microelectronics che viene associata con la libreria Arduino_LSM9DS1 fornita da Arduino, contenente gli esempi per l'utilizzo base del sensore. Nello specifico verrà usata una libreria modificata che permette di variare i parametri più avanzati della IMU con maggiore praticità [10]. Il sensore comunica attraverso il protocollo I2C o SPI. La sua posizione è indicata in figura 1.3.

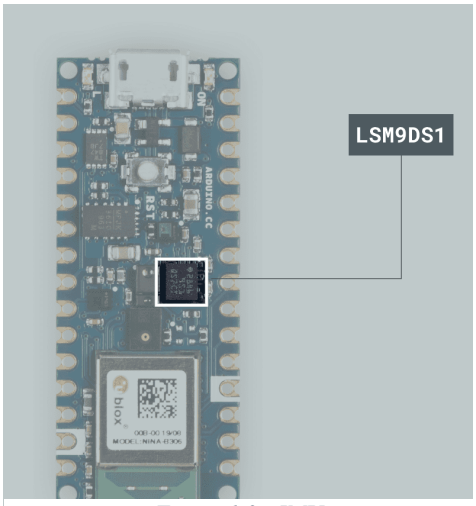


Figura 1.3 - IMU

Accelerometro

Un accelerometro è un dispositivo elettromeccanico usato per misurare le forze di accelerazione. Queste forze possono essere statiche, come la forza di gravità, oppure dinamiche, come durante i movimenti o le vibrazioni. Per funzioni più avanzate è necessario andare ad agire sui registri della libreria, per tali funzioni si rimanda al datasheet del sensore [9]. L'orientazione del sensore è indicata in figura 1.4.

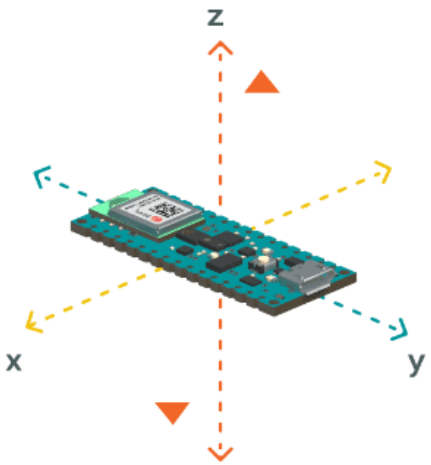


Figura 1.4 – Assi IMU

Giroscopio

Un giroscopio è un dispositivo che può misurare l'orientazione e la velocità angolare di un oggetto. In un certo senso un giroscopio è più avanzato di un accelerometro, in quanto può misurare l'inclinazione e l'orientamento laterale di un oggetto, mentre un accelerometro può misurare solo il suo movimento lineare. L'orientazione per questa scheda è indicata in figura 1.5.

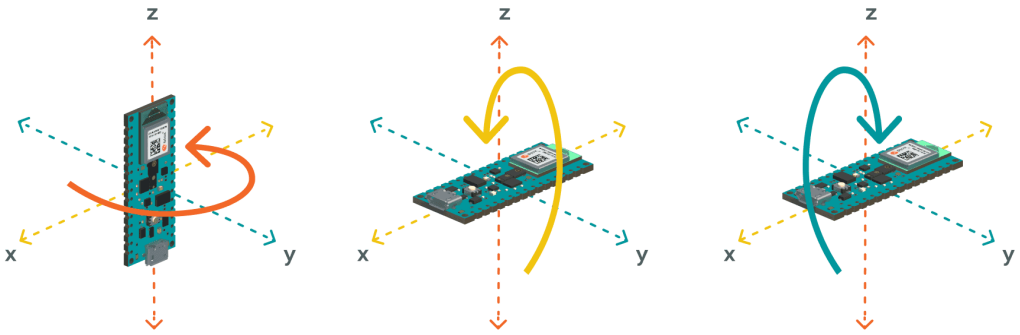


Figura 1.5 – Assi Giroscopio

I sensori giroscopici sono anche detti "sensori di velocità angolare". Questa viene misurata in gradi al secondo, variazione dell'angolo di rotazione nell'unità di tempo.

Magnetometro

Un magnetometro è un dispositivo in grado di misurare il campo magnetico, la sua forza, direzione e il cambiamento relativo ad una particolare posizione. La misura delle componenti del campo permette di definire il vettore campo magnetico nel punto in cui si sta effettuando la misura. L'orientazione del sensore è ritratta in figura 1.6.

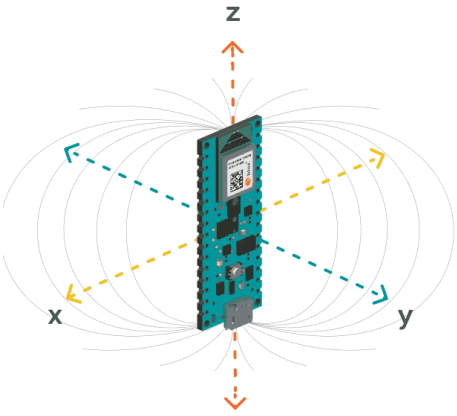


Figura 1.6 – Assi Magnetometro

1.2.2 Microfono

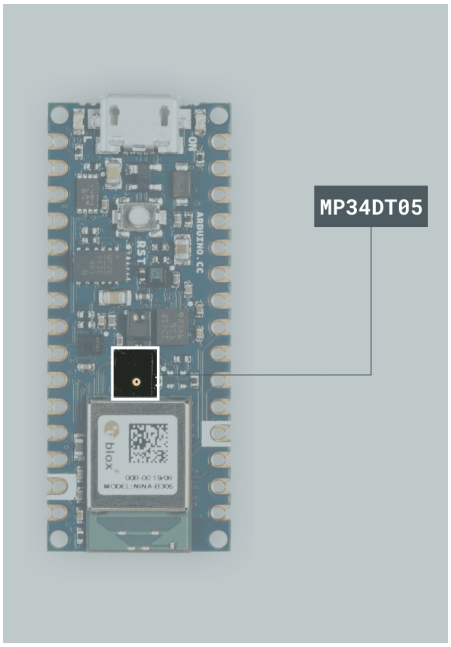


Figura 1.7 – Microfono

I microfoni sono componenti che convertono il suono fisico in segnali elettrici, che a loro volta possono essere poi convertiti in dati digitali. I microfoni sono comunemente usati nei terminali mobili, nei sistemi di riconoscimento vocale o persino nei dispositivi di input per giochi e realtà virtuale. Il sensore MP34DT05 è un microfono ultracompatto, omnidirezionale, a basso consumo, che utilizza il PDM (Pulse-Density Modulation) per rappresentare un segnale analogico con un segnale binario (figura 1.7). La modulazione a densità d'impulso consiste nella codifica di un segnale in una sequenza d'impulsi per cui l'informazione non è associata all'ampiezza degli stessi, ma alla frequenza con cui questi commutano tra livelli discreti (la loro densità appunto).Il microfono ha un SNR di 64 dB e un range di temperatura -40/85 °C, il che lo rende una buona scelta per applicazione in terminali mobili. Notebook e altri.

1.2.3 Sensore di Temperatura e Umidità

I sensori di temperatura sono componenti che convertono la temperatura fisica in dati digitali. Allo stesso modo, i sensori di umidità sono in grado di misurare i livelli di umidità atmosferica e tradurli in segnali elettrici. Pertanto, questi sensori sono essenziali per il monitoraggio ambientale, specialmente all'interno e intorno alle apparecchiature elettroniche sensibili.

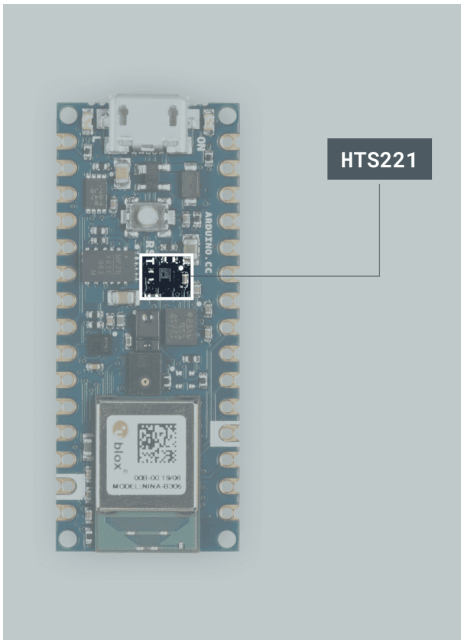


Figura 1.8 – Sensore di Temperatura e Umidità

L'HTS221 è un sensore ultracompatto per umidità relativa e temperatura (figura 1.8). Usa il protocollo I2C o SPI per comunicare e scambiare dati. Le sue caratteristiche sono:

- Precisione umidità: ± 3.5% rH, da 20 a +80% rH
- Intervallo umidità: da 0 a 100 %
- Precisione temperatura: ± 0.5 °C, da 15 a +40 °C
- Intervallo temperatura: da -40 a 120°C

Questi tipi di sensori vengono utilizzati più di quanto si pensi e si trovano in vari oggetti di uso quotidiano. Alcuni esempi di applicazione di questo sensore sono: condizionatori, umidificatori, frigoriferi, dispositivi indossabili, smart home e tanti altri. Purtroppo, il sensore istallato in questa scheda soffre di inaffidabilità man mano che la scheda rimane accesa, questo a causa dell'autoriscaldamento della scheda stessa. Per ovviare a questo problema si può ridurre al minimo la potenza del processore oppure cercare di ridurre al minimo l'influenza della scheda sul sensore.

1.3 Il tool COOLTERM

Usando il monitor seriale dell'IDE di Arduino si notano i limiti del tool, tra tutti l'impossibilità del salvataggio di dati. È necessario quindi andare a considerare l'uso di terminali alternativi che offrano maggiori funzionalità.

CoolTerm è una semplice applicazione di terminale per porta seriale pensata per hobbisti e professionisti con la necessità di scambiare dati con hardware collegato a porte seriali come servocontrollori, kit robotici, ricevitori GPS, microcontrollori, ecc. Il tool è disponibile per i sistemi operativi più comuni ed è costantemente aggiornato dallo sviluppatore. Il software vanta numerose funzionalità tra cui il salvataggio dei dati, la scelta del formato, l'aggiunta del timestamp, connessioni multiple e tanti altri. Nel seguito del paragrafo si farà una veloce

introduzione al tool, al fine di utilizzarlo per lo scopo della presente tesi; per approfondimenti si rimanda al sito dello sviluppatore [11].

Per prima cosa si collega l'Arduino al PC, con il codice già istallato; in questo caso il codice serve per estrarre i valori da accelerometro e giroscopio e salvarli in formato CSV, in modo da poter essere caricati su Edge Impulse. In aggiunta, tramite la libreria modificata è possibile modificare i parametri della IMU, si vedrà questa parte di codice nel capitolo 4. Il codice è mostrato in figura 1.9.

```
#include <Arduino_LSM9DS1.h>

const float accelerationThreshold = 2.5; // threshold of significant in G's
const int numSamples = 900;   /* se posto uguale al sample rate campiono per 1s, se volessi campionare per intervalli      <----
                                 diversi basta cambiare qusto valore (es: 119 a 119Hz vuol dire un intervallo di 1s) */

int samplesRead = numSamples;

void setup() {
  Serial.begin(115200);
  while (!Serial);

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

```
void loop() {
  float aX, aY, aZ/*, gX, gY, gZ*/;

  // wait for significant motion
  while (samplesRead == numSamples) {
    if (IMU.accelerationAvailable()) {
      // read the acceleration data
      IMU.readAcceleration(aX, aY, aZ);

      // sum up the absolutes
      float aSum = fabs(aX) + fabs(aY) + fabs(aZ);

      // check if it's above the threshold
      if (aSum >= accelerationThreshold) {
        // reset the sample read count
        samplesRead = 0;
        break;
      }
    }
  }
  // inizio conteggio timestamp
  unsigned long start_timestamp = millis();

  // check if the all the required samples have been read since
  // the last time the significant motion was detected
  while (samplesRead < numSamples) {

    // Take timestamp so we can hit our target frequency
  unsigned long timestamp = millis();
```

```
  // check if both new acceleration and gyroscope data is
  // available
  if (IMU.accelerationAvailable() && IMU.gyroscopeAvailable()) {
    // read the acceleration and gyroscope data
    IMU.readAcceleration(aX, aY, aZ);
    IMU.readGyroscope(gX, gY, gZ);

    samplesRead++;

   // print the data in CSV format
    Serial.print(timestamp - start_timestamp);
    Serial.print(",");
    Serial.print(aX, 3);
    Serial.print(',');
    Serial.print(aY, 3);
    Serial.print(',');
    Serial.print(aZ, 3);
    Serial.print(',');
    Serial.print(gX, 3);
    Serial.print(',');
    Serial.print(gY, 3);
    Serial.print(',');
    Serial.print(gZ, 3);
    Serial.println();
```

Figura 1.9 – Codice per raccolta dati accelerometro e giroscopio in CSV

A questo punto si apre CoolTerm, la schermata che si aprirà sarà quella in figura 1.10. Premendo sul pulsante Options si va a selezionare la porta e la frequenza del collegamento, si preme poi OK. Quindi lo scopo è salvare i dati in formato CSV in un file, fare allora clic su Connection -> Capture to text/binary file -> Start verrà richiesto nome e posizione in cui salvare il file e il formato (che deve essere CSV).

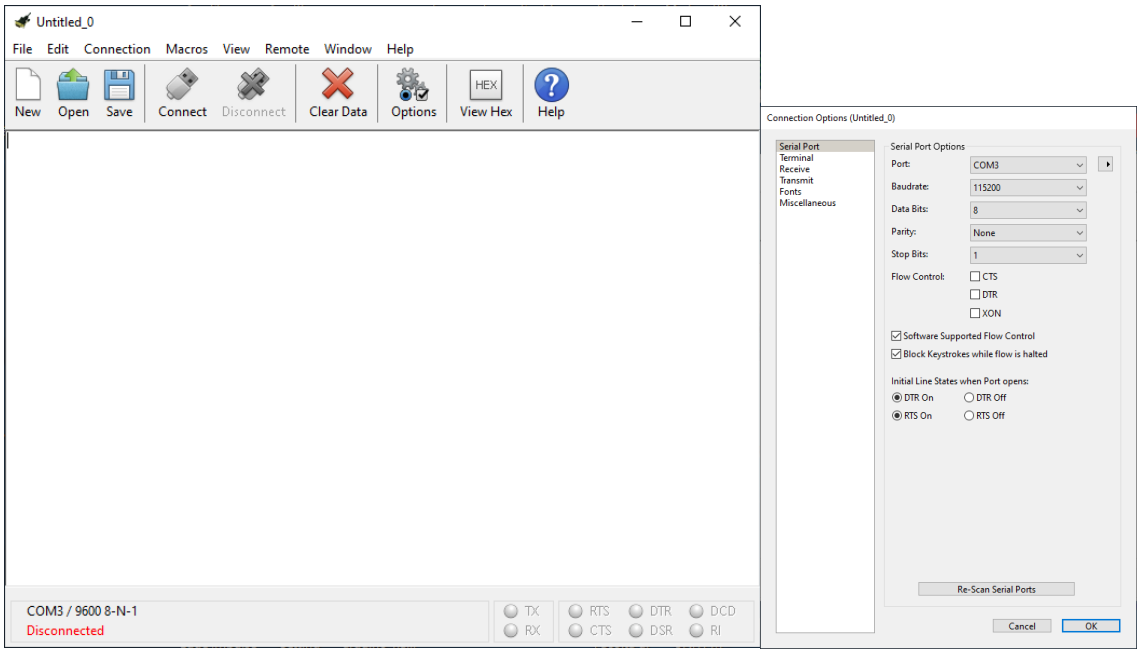


Figura 1.10 – Schermata principale CoolTerm

Da questo momento in poi tutto quello che verrà inviato dalla scheda al monitor seriale verrà salvato sul file Excel creato. Una volta cliccato sul taso Connect si inizia a registrare i dati. Quello che si avrà sarà allora (figura 1.11):

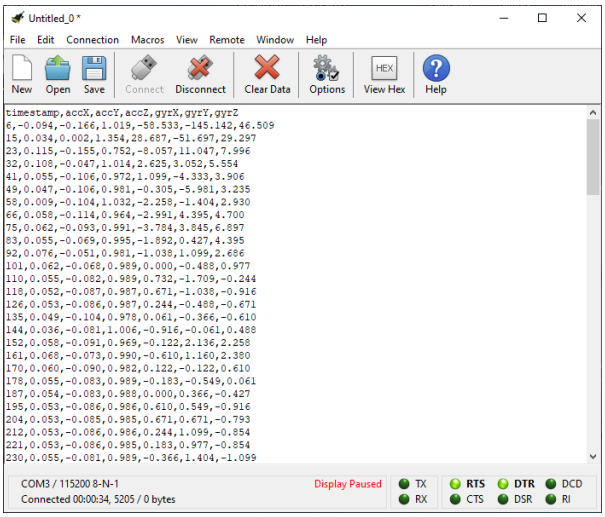


Figura 1.11 – Registrazione dati

Una volta che i dati di interesse sono stati salvati è sufficiente fermare la registrazione, il file sarà pronto per essere caricato su Edge Impulse.

Capitolo 2

Edge e Cloud Computing

2.1 Introduzione

L'**edge computing** è un paradigma che prevede di elaborare e analizzare i dati in prossimità della loro origine, riducendo così il tempo di latenza della rete e ottenendo un trasferimento efficiente dei dati. Tale aspetto ha un'importanza significativa nell'Internet of Things (IoT), nell'intelligenza artificiale e nel big data analytics. Affinché gli edge devices rispondano in modo intelligente, devono elaborare le informazioni in tempo reale e agire di conseguenza. Nell'Edge computing, non è necessario trasferire i dati a un altro server, come mostrato in figura 2.1.

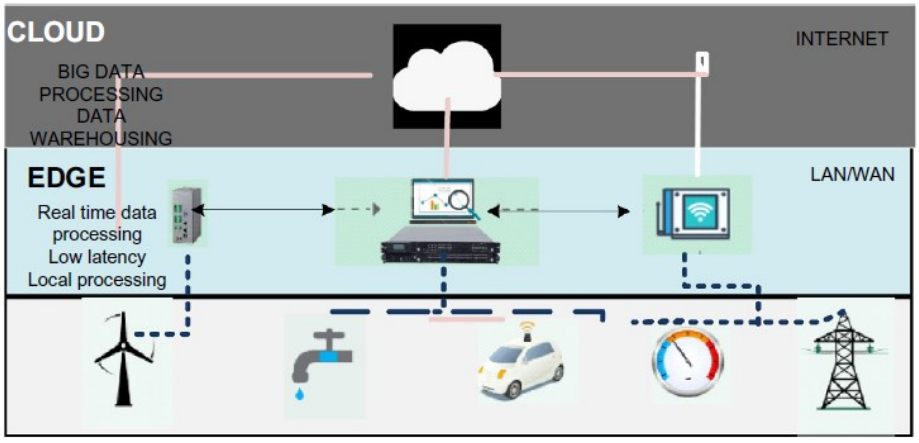


Figura 2.1 – Infrastruttura Edge-Cloud

Questo calcolo distribuito locale permette di utilizzare i dispositivi edge per fornire elaborazioni che risultino pertinenti nel contesto in cui sono inseriti. Esempi sono le telecamere per l'elaborazione delle immagini on-device, i dispositivi medici indossabili che trasmettono dati ai dispositivi finali tramite Bluetooth. Molte delle funzionalità di edge computing assomigliano alla tecnologia del cloud computing. Il **cloud computing** al contrario del primo è un paradigma che centralizza l'elaborazione dei dati in grandi server e dispositivi di elaborazione. Ma le caratteristiche uniche che rendono l'edge computing una tecnologia promettente sono diverse:

- **Distribuzioni geografiche dense**. L'edge computing avvicina i vantaggi del cloud all'utente installando varie piattaforme di elaborazione dati ai margini della rete.
- **Supporto alla mobilità**. Con l'aumento del numero di dispositivi mobili, il supporto alla mobilità è una delle necessità principali del presente. Sulla base del numero di dispositivi IoT connessi nel recente passato, può essere prevista la crescita del numero di dispositivi connessi secondo l'andamento mostrato in figura 2.2. L'edge computing aderisce al protocollo di separazione dell'ID del localizzatore (Locator ID Separation Protocol - LISP) e quindi fornisce supporto per la mobilità. Il protocollo LISP identifica separatamente la posizione e l'host con l'uso di un sistema di directory distribuito. La separazione dell'identità dell'host e della posizione è il meccanismo chiave che supporta la mobilità nell'edge computing.

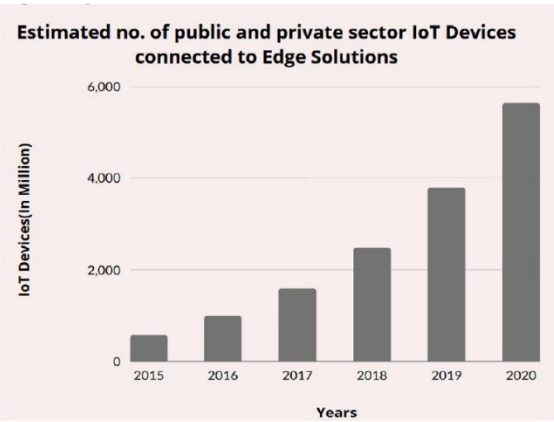


Figura 2.2 – Dispositivi IoT connessi tramite approccio edge

- **Consapevolezza della posizione**. La proprietà di location-awareness è un'altra importante caratteristica che consente all'edge computing di migliorare i servizi basati sulla posizione, come ad esempio i servizi di localizzazione real-time.
- **Prossimità**. I servizi di elaborazione e analisi dei dati si trovano vicino ai dispositivi edge, il che migliora l'esperienza e la risposta dell'utente. La disponibilità di questi servizi in stretta prossimità dell'utente fornisce informazioni sul contesto per migliori decisioni. Allo stesso modo, i fornitori di servizi possono anche analizzare e migliorare i propri servizi in base al feedback dell'utente relativo all'allocazione delle risorse.
- **Bassa latenza**. La caratteristica più importante del paradigma dell'edge computing è la bassa latenza. Questa funzionalità consente di utilizzare l'edge computing in modo efficiente in applicazioni ad alta intensità di risorse e sensibili al tempo. Un edge server che si trova in prossimità dei dispositivi edge fornisce la distribuzione necessaria per l'elaborazione e l'analisi dei dati.

In questa era dell'IoT, miliardi o addirittura trilioni di dispositivi perimetrali comunicano e generano un'enorme quantità di dati a velocità molto elevata. In futuro, lo scenario sarà più vasto e complesso da gestire. Il cloud tradizionale o i data center centralizzati possono incontrare

complessità a causa del carico computazionale e dell'accumulo centralizzato dei dati. Le principali sfide affrontate da tali modelli di data center centralizzati possono essere elencate di seguito:

- Elevata quantità di aggregazione dei dati e velocità dei dispositivi IoT
- Alta latenza
- Vincoli di larghezza di banda della rete
- Dispositivi IoT con risorse limitate
- Servizi ininterrotti con connettività intermittente al cloud (es. macchine autonome)
- Sfide di sicurezza e privacy
- Scalabilità (a causa del sempre crescente numero di dispositivi connessi)
- Dominio di poche mega aziende

2.1.1 Supporto dell'edge computing alle funzionalità dell'IoT

L'edge computing fornisce varie funzionalità di supporto ai dispositivi IoT. I vantaggi principali sono elencati in figura 2.3. Tuttavia, i seguenti sono alcuni degli aspetti più importanti del supporto dell'edge computing alle funzionalità IoT:

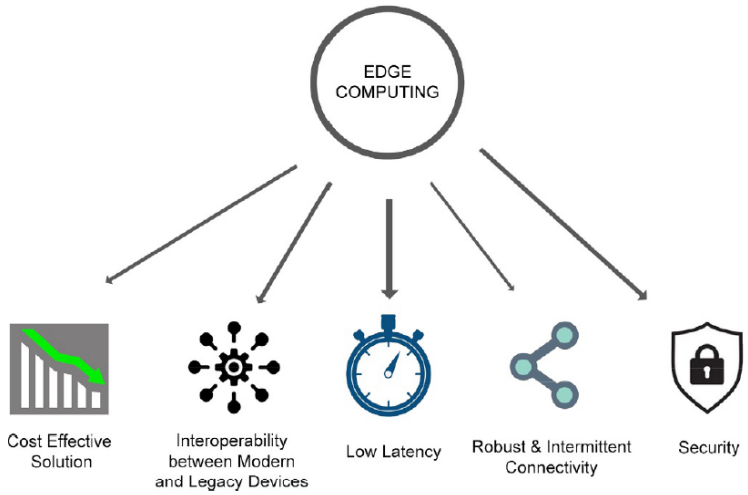


Figura 2.3 – vantaggi principali dell'edge computing

- **Gestione dei dispositivi**. Il gateway Edge può essere utilizzato per aggiornare gli attributi e le configurazioni dei dispositivi edge in locale invece della modalità cloud centralizzata in cui viene utilizzato un sistema di accodamento per gli aggiornamenti. È possibile distribuire risorse aggiuntive all'edge per identificare e ridurre un problema specifico e per fornire scalabilità del servizio.
- **Sicurezza**. L'edge server svolge un ruolo molto importante nella sicurezza dei dati. Qualsiasi misura di sicurezza può essere implementata all'edge server per proteggere

l'intero ecosistema edge. Con le diverse apparecchiature e dispositivi connessi all'IoT, le misure di sicurezza possono essere applicate all'edge per ridurre le complicazioni. La natura distribuita dell'edge garantisce una migliore gestione della sicurezza rispetto all'ambiente cloud centralizzato, poiché un attacco dannoso su un singolo nodo avrà un impatto minore. Conservare e trasportare meno informazioni non solo riduce il sovraccarico della rete, ma alleggerisce anche il sistema per una migliore gestione della sicurezza e autenticazione. I dati altamente sensibili possono essere trattati separatamente con un rigoroso meccanismo di sicurezza all'edge.

- **Messaggistica prioritaria**. Gli Edge Server possono gestire meglio i messaggi in base alla loro priorità. L'edge può elaborare, assegnare priorità e trasmettere tali dati più rapidamente rispetto a un server cloud centralizzato. Comunicando solo con l'edge locale e i gateway IoT, è possibile garantire una risposta più rapida. Successivamente, tali messaggi possono essere caricati nel cloud per la diffusione degli stessi su larga scala, se necessario.
- **Aggregazione dati**. Con l'aumento del numero di dispositivi IoT connessi, aumenta anche la quantità di dati replicati generati (es. letture di temperatura multiple da più sensori). Tali dati replicati devono essere filtrati prima di essere inviati al server centralizzato nel cloud. Dopo aver ricevuto più messaggi in un breve lasso di tempo, l'edge aggrega tutti i messaggi e genera un nuovo set di dati riepilogati, che dovrebbe contenere una panoramica di tutti i messaggi ricevuti. L'aggregazione dei dati all'edge aumenta significativamente l'efficienza del core riducendo il sovraccarico di elaborazione dei dati replicati. La latenza migliora, con meno dati da comunicare ed elaborare.

2.1.2 Cosa scegliere? Cloud o Edge Computing?

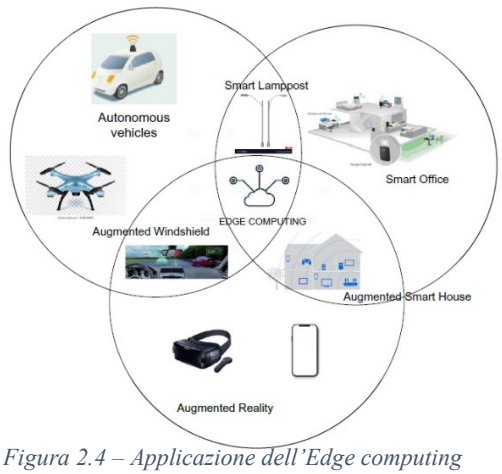


Figura 2.4 – Applicazione dell'Edge computing

Negli ultimi dieci anni, i fornitori di servizi cloud hanno creduto che tutto fosse su una traiettoria verso il cloud. Ma ora si sta lavorando sull'idea che sia meglio decentralizzare i carichi di lavoro ai margini della rete in modo da ottenere prestazioni migliori. Il cloud computing in genere entra in gioco quando le organizzazioni richiedono storage e potenza di calcolo per eseguire determinati applicazioni e processi, e/o gestione remota dei dati

(figura 2.4). Nel recente passato, si riteneva che il cloud fosse l'unica soluzione a tutto ciò che riguardasse i dati. Ma ora i principali fornitori di servizi cloud stanno mostrando interesse per l'idea di un carico di lavoro distribuito nei punti in cui i dati sono localizzati. Nel prossimo futuro, sarà quasi impossibile trasferire in modo tempestivo una così grande quantità di dati dai dispositivi finali ai data center remoti. La latenza sarà un problema, causato dalla distanza tra i dispositivi IoT edge e i data center nei modelli cloud centralizzati. La crescita esponenziale del numero di dispositivi IoT causerà una latenza elevata per tante applicazioni che includono comunicazioni end-to-end. La tabella in figura 2.5 da un confronto immediato tra i due approcci, permettendo di valutare vantaggi e svantaggi di una o l'altra tecnologia. In breve, ci sono carenze nel cloud computing tradizionale, che devono essere affrontate per fornire servizi efficienti e affrontare il cambio di paradigma nel futuro mondo IoT. L'edge computing potrebbe essere la

Criteria	Cloud computing	Edge computing
Basic feature	In cloud computing huge cloud data-centers are involved that might be many miles away from the user and end devices	Edge computing is applicable to everything which includes placing service provisioning, data, and analysis in proximity to users and devices
Utilization	Most of the current applications utilize cloud-based applications	Applications based on IoT, Virtual Reality, Augmented Reality, Smart homes and Smart cities, smart vehicles etc.
Data centers	A small number of huge cloud data centers	A large number of small edge data centers
Proximity to data centers	Usually data centers are far from IoT devices	Data-centers are in proximity to IoT devices
Latency	High because of distance between devices and data centers. The edge approach emphasizes reducing latency and providing more processing of data close to the source	Low because the devices are local to the data centers. May take time in transferring data from the Cloud
Bandwidth consumption	More data consumption, as first data need to be transferred to remote data centers	Less, as the data are processed to local edge servers
Security	More prone to en-route attack while data transmission	Less prone to attack because of less physical distance
Configuration requirement at IoT end devices	Devices with rich configuration	Less powerful end devices are required as applications can make better use of edge server's capability while applying artificial intelligence and machine learning algorithms
Energy consumption of end-devices	More energy consumption in computation	Minimizes energy consumption for battery-powered edge devices, such as phones as amount of computation at the edge is less. Devices didn't have to rely entirely on their own processors or drain phone batteries with intense computation

Figura 2.5 – Confronto tra Cloud ed Edge computing

soluzione possibile, ma solo l'integrazione di edge e cloud computing può far emergere il meglio di questi due mondi.

2.1.3 Vantaggi e potenzialità dell'Edge computing

La bassa latenza e l'ottimizzazione della larghezza di banda sono caratteristiche chiave che contribuiscono all'ascesa dell'edge computing. La quantità di dati generati dai dispositivi digitali sta aumentando in modo esponenziale. La consueta pratica di archiviazione dei dati nel cloud sta

diventando troppo costosa e troppo lenta per soddisfare le esigenze dell'utente finale. Il costo è un'altra considerazione chiave per l'edge computing. Il processo di filtraggio coinvolto nell'edge computing consente la trasmissione al cloud solo dei dati rilevanti, scartando i dati irrilevanti e riducendo i costi di rete. L'edge computing è più efficiente dal punto di vista computazionale e ritenuto più adatto per essere integrato con l'IoT per affrontare la privacy e la sicurezza degli utenti finali. Alcuni potenziali vantaggi dell'edge computing possono essere elencati di seguito:

- Bassa latenza
- Minor consumo di energia
- Dispositivi più semplici ed economici
- Disponibilità della larghezza di banda e gestione efficiente dei dati
- Sicurezza della rete
- Autonomia
- Privacy dei dati
- Filtraggio/priorità dei dati

L'edge computing prevede di avvicinare i servizi e le utilità forniti dal cloud computing ai dispositivi IoT per garantire un'elaborazione efficiente e in tempo reale. La minore latenza fornita dall'edge migliorerà l'autonomia nel prossimo futuro grazie a una migliore collaborazione con i campi dell'intelligenza artificiale e del machine learning. Pertanto, l'edge computing offre molti vantaggi, ma soffre anche di vari problemi tecnici e non, che andranno affrontati nell'immediato futuro per la crescita della tecnologia.

2.2 Smart sensors

Se si combina un sensore, un circuito di interfaccia analogica, un convertitore analogico- digitale (ADC) e un'interfaccia bus in un unico alloggiamento, si ottiene uno smart sensor. Nella figura 2.6 sono mostrati tre sensori intelligenti ibridi, che differiscono per il grado di integrazione. Nel terzo, il sensore è già combinato con un circuito di interfaccia che fornisce un flusso di bit. Solo l'interfaccia bus è ancora necessaria separatamente.

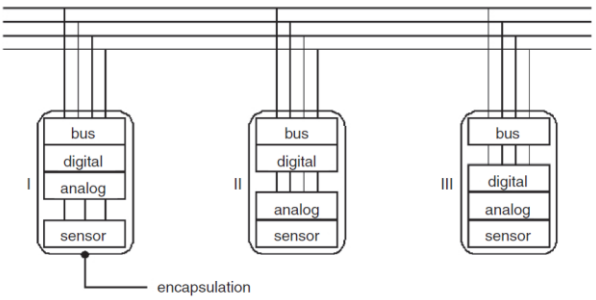


Figura 2.6 – Hybrid smart sensors

Integrando tutte le funzioni dal sensore all'interfaccia bus in un unico chip, si ottiene uno smart sensor integrato, come illustrato nella figura 2.7.

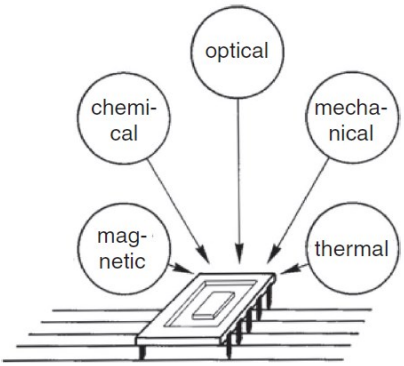


Figura 2.7 – Smart sensor integrato

Uno smart sensor integrato dovrebbe contenere tutti gli elementi necessari: uno o più sensori, amplificatori, un chopper e multiplexer, un convertitore AD, buffer, un'interfaccia bus, indirizzi, controllo e gestione dell'alimentazione. Questo è mostrato nella figura 2.8.

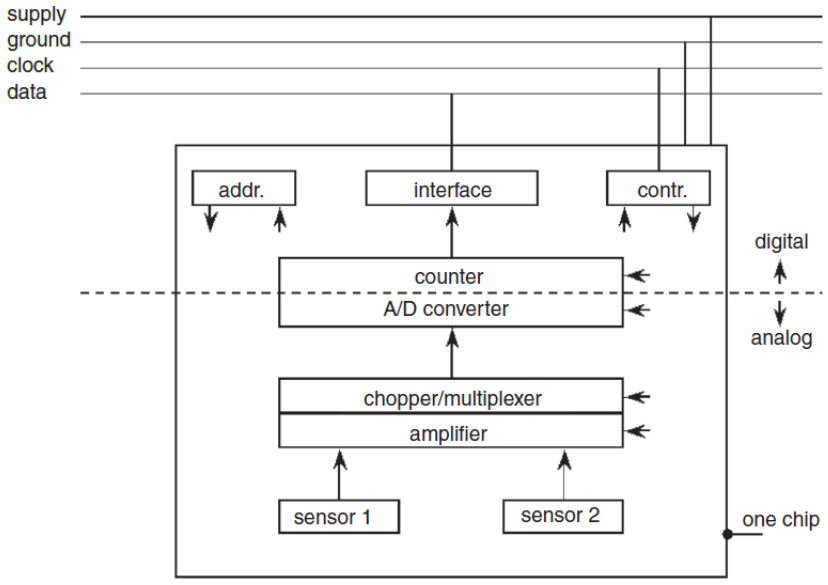


Figura 2.8 – Funzioni di uno smart sensor integrato

Sebbene l'integrazione completa di tutte le funzioni sia costosa, la produzione in serie del sensore risultante può mantenere ragionevole il costo per lo smart sensor integrato. Un altro vantaggio è che i costi di installazione dell'intero sistema di sensori possono essere ridotti drasticamente grazie alla semplice architettura modulare. Tuttavia, per realizzare tutte le funzioni su un chip, si deve prima integrare una varietà di sensori su un singolo chip. A tale scopo è in fase di sviluppo una tecnologia di microstrutturazione tridimensionale IC-compatible. Inoltre, deve essere sviluppata un'elettronica di interfaccia, adatta per l'integrazione sul chip del sensore.
La figura 2.9 illustra l'evoluzione dei sistemi di sensori intelligenti integrati con molti passaggi intermedi. Maggiore è il mercato dei sensori intelligenti di un certo tipo, maggiore è l'integrazione economicamente conveniente per quel tipo.

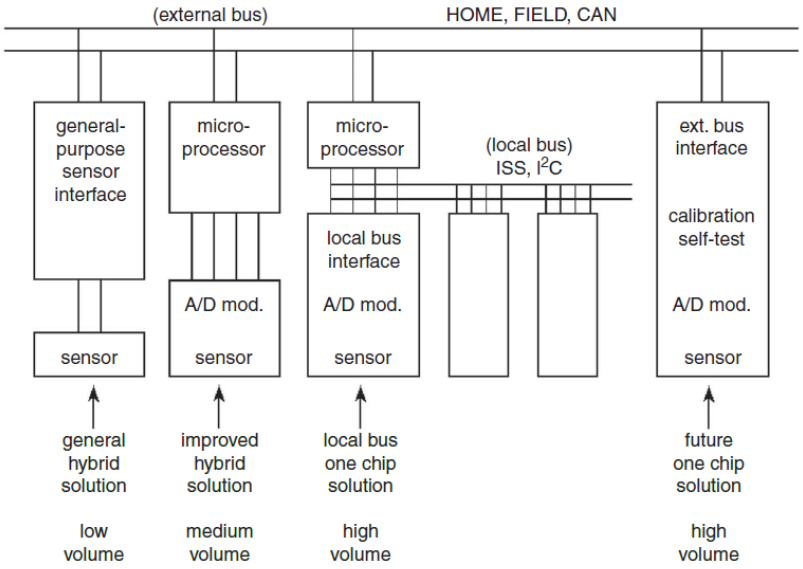


Figura 2.9 – Evoluzione dei sistemi smart sensor

Lo scopo finale è rappresentato nella figura 2.10. Essere in grado di integrare una fonte di alimentazione wireless e una comunicazione wireless, apre ad un concetto completamente nuovo di sensori. Molti sensori potrebbero quindi essere utilizzati in automobili, case, vestiti e tanti altri campi, per ottenere informazioni preziose.

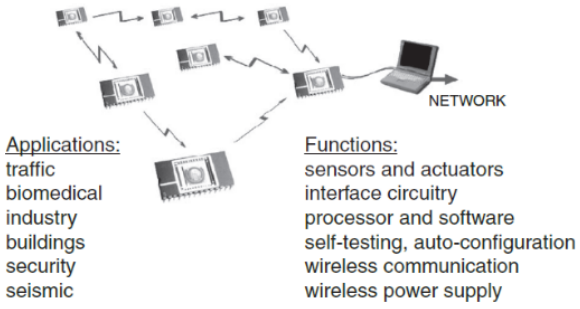


Figura 2.10 – Microsensori autonomi

Nei sistemi di smart sensors, le funzioni dei sensori e le loro interfacce sono combinate in un design globale. Queste funzioni includono il rilevamento, il condizionamento del segnale, la conversione da analogico a digitale, l'interfaccia bus e l'elaborazione dei dati. Inoltre, possono essere incluse ulteriori funzioni a un livello gerarchico superiore, come l'autotest, l'autocalibrazione, la valutazione e l'identificazione dei dati. In molti sensori fisici e chimici, la larghezza di banda dell'informazione è piuttosto piccola, molto più piccola di quella della parte elettronica del sistema. Ciò consente al progettista del sistema di utilizzare un unico sistema elettronico per supportare molti elementi di rilevamento, al fine di eseguire misurazioni multiple. Inoltre, può utilizzare l'eccedenza di tempo/larghezza di banda disponibile della parte elettronica per migliorare la precisione, l'affidabilità e la stabilità a lungo termine del sistema o per ridurre la dissipazione di potenza.

Quanto sopra ha reso gli smart sensors protagonisti dell'Edge computing e un elemento essenziale nell'IoT, in un certo senso si potrebbe dire che ormai l'IoT non potrebbe esistere senza smart sensors. Per ulteriori approfondimenti si rimanda a testi tecnici (es. [13] [14] [15]).

2.3 Protocolli di comunicazione

Si farà ora una breve panoramica dei protocolli di comunicazione più usati in questo tipo di sistemi. Essenzialmente sono tre, basati tutti su una comunicazione di tipo seriale: UART, I2C e SPI. Un esempio di implementazione è illustrato in figura 2.11.

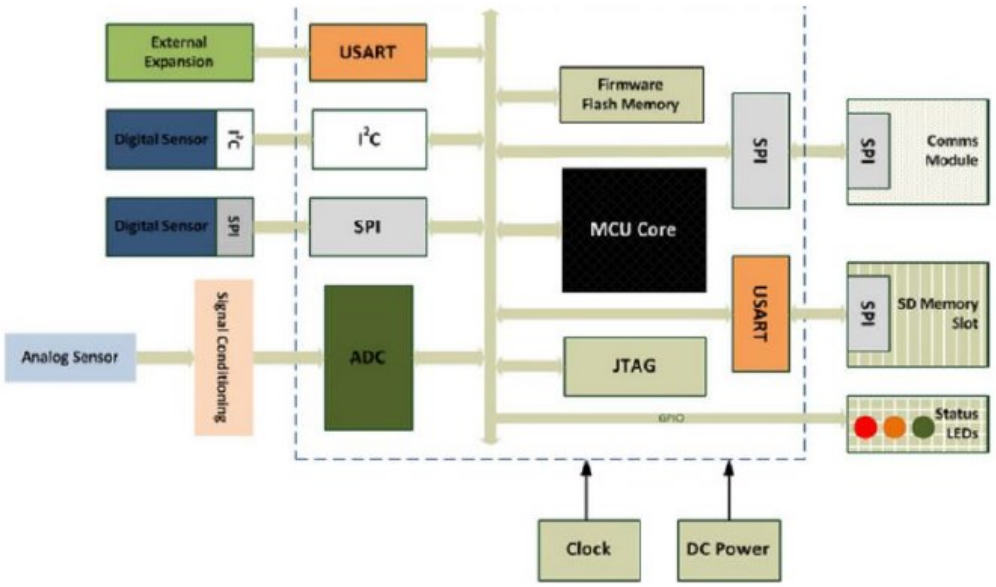


Figura 2.11 – UART, I2C e SPI

La comunicazione di tipo seriale è un protocollo molto usato nelle comunicazioni tra dispositivi e componenti all'interno di dispositivi elettronici; questo prevede l'invio di un bit alla volta, in sequenza, lungo il canale di trasmissione. Diversamente dalla comunicazione di tipo parallelo, in cui più bit vengono inviati contemporaneamente su un bus, questa è di più facile implementazione e per questo molto sfruttata.

L'**UART** (Universal Asynchronous Receiver/Transmitter) è un tipo di trasmissione seriale bidirezionale asincrona, ovvero non dipende da un segnale di clock sincronizzato tra due dispositivi, ma da un bit di inizio (1) e uno di fine (0). La comunicazione deve avvenire ad un preciso baud rate (rate trasmission dei simboli) concorde tra i due dispositivi. La trasmissione avviene in tre modalità differenti: *simplex*, i dati vengono trasmessi in una sola direzione; *half-duplex*, dati trasmessi in entrambe le direzioni ma non in contemporanea; *full-duplex*, i dati possono venire trasmessi e ricevuti nello stesso momento. La struttura del collegamento è illustrata in figura 2.12.

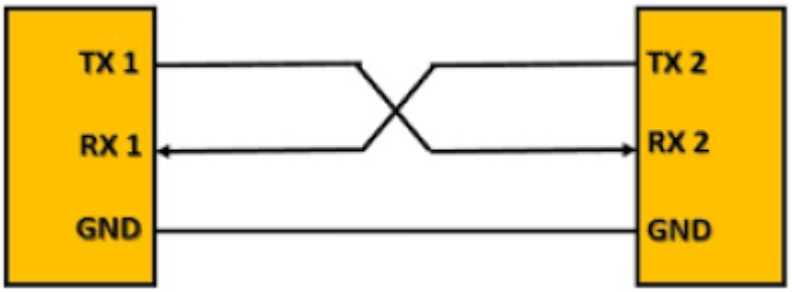


Figura 2.12 – UART

Nel collegamento fisico si utilizzano due linee di comunicazione: TX per la trasmissione, RX per la ricezione e una linea di terra comune (GND). I vantaggi di questo tipo di comunicazione sono la semplicità di operazione e implementazione, uniti alla non necessità di un clock per la sincronizzazione, a discapito di un basso rate di trasmissione. La velocità di trasmissione può essere notevolmente migliorata con il protocollo **USART**, che prevede l'ausilio di un clock per la sincronizzazione, e che permette di evitare la necessità di specificare un preciso baud rate di comunicazione. La velocità massima di questo tipo di protocollo è di circa 4 Mbps.

La comunicazione **I2C** (Inter Integrated Circuit) utilizza un bus seriale bidirezionale sincrono con due collegamenti fisici (fili), in cui vi sono almeno un master e uno o più slave. Le due linee di comunicazione sono definite come l'SDA (serial data line acceptance port) e l'SCL (serial clock line), la prima per la trasmissione e la ricezione dei dati, la seconda per la sincronizzazione

temporale, a cui va aggiunto un collegamento a terra (GND) e di alimentazione (figura 2.13). Il dispositivo che funge da master inizia e termina la conversazione stabilendo il clock, tutti gli altri dispositivi sono gli slaves, i quali vengono comandati a ricevere o trasmettere i dati secondo le disposizioni del master.

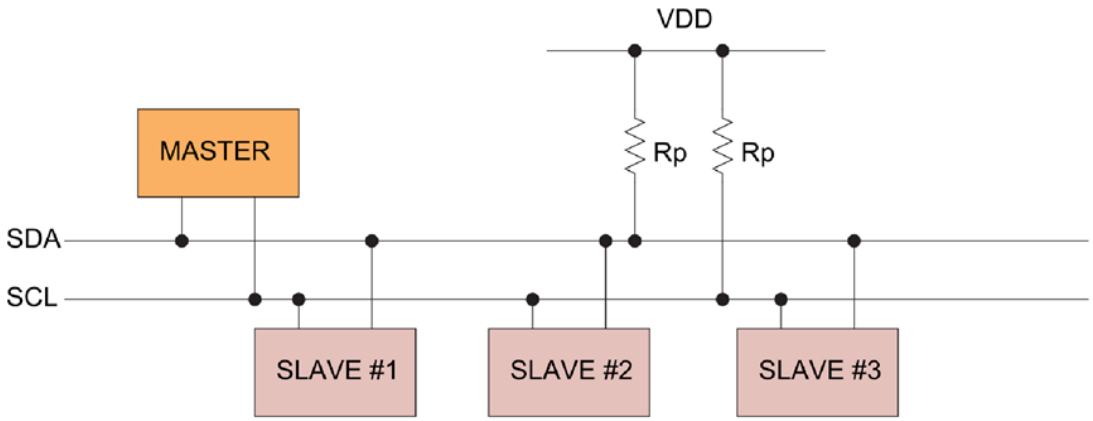


Figura 2.13 – I2C

I due punti di forza principali di questo protocollo sono la grande flessibilità e la possibilità di collegare più dispositivi con un unico bus a due fili a patto di non collegare un numero troppo elevato di dispositivi, che ne degraderebbero le prestazioni. Esistono due modalità di trasferimento, la standard, che può arrivare fino a 100 kbit/s e la fast mode che permette velocità fino a 400 kbit/s. Si aggiunge inoltre una terza modalità, più recente, che permette trasmissioni a 3,4 Mbit/s che amplia anche il numero massimo di dispositivi collegabili.

Il protocollo **SPI** (Serial Peripheral Interface), è un sistema di comunicazione seriale sincrono e full-duplex, tra un microcontrollore e un circuito integrato, o tra più microcontrollori. È il protocollo più veloce tra quelli analizzati, permette velocità fino a 100 Mbps; le linee di clock e di dati vengono condivise tra più dispositivi, i cui indirizzi di identificazione sono univoci. La comunicazione avviene tramite quattro porte (figura 2.14):

- MOSI (Master Data Output, Slave Data Input)
- MISO (Master Data Input, Slave Data Output)
- SCLK (clock digitale che dipende dal master)
- NSS (Slave Enabled Signal, emesso dal master per selezionare lo slave)

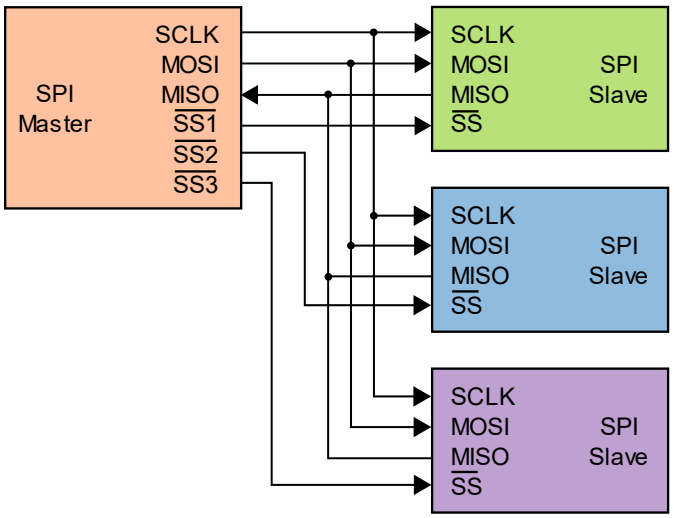


Figura 2.14 – SPI

La trasmissione dei dati sul bus si basa su un registro a scorrimento (shift register), di dimensione solitamente a 8 bit di cui sono dotati i singoli dispositivi, che siano master o slave. Il vantaggio dell'SPI rispetto all'I2C risiede nella non necessità di implementare un indirizzo ogni qual volta si voglia comunicare con uno slave. Rispetto all'UART invece, non prevede bit di inizio e fine conversazione, permettendo una trasmissione delle informazioni continuativa, senza interruzioni. Inoltre, è garantita la capacità di trasmettere e ricevere i dati nello stesso momento, avendo separati i collegamenti MISO e MOSI. Gli svantaggi, confrontando con l'I2C e l'UART, sono rispettivamente: l'assenza di un meccanismo che testimoni la ricezione dei dati e la mancanza di sistemi di rilevamento d'errore (bit di parità). Come è ovvio, dunque, la scelta del protocollo va fatta in base alle specifiche esigenze che una certa applicazione richiede. La maggior parte di schede e sensori ormai supportano più protocolli, lasciando libertà di scelta al progettista.

Capitolo 3

EMBEDDED MACHINE LEARNING E RETI NEURALI

3.1 Introduzione

Al giorno d'oggi, l'intelligenza artificiale (AI) è un campo fiorente con molte applicazioni e temi di ricerca attivi. Sfruttiamo le intelligenze artificiali per automatizzare il lavoro di routine, comprendere parole o immagini, effettuare diagnosi in ambito medico o supportare la ricerca scientifica.

La vera sfida dell'AI si è rivelata quella di risolvere compiti che, ironicamente, sono facili da svolgere per le persone, ma sono anche quelli più difficili da descrivere formalmente. Sono i problemi che risolviamo intuitivamente, che ci sembrano automatici, come riconoscere parole pronunciate o volti. Compiti astratti e formali che sono tra i più difficili per la mente umana sono i più facili da svolgere per un computer. Quest'ultimo è da tempo in grado di sconfiggere anche il miglior giocatore di scacchi, ma solo recentemente ha acquisito abilità come riconoscere oggetti o parole.

La soluzione a questi problemi è quella di permettere ai computer di imparare dall'esperienza e comprendere il mondo in termini di una gerarchia di concetti, in cui ogni concetto è definito attraverso la sua relazione con concetti più semplici. Raccogliendo conoscenze dall'esperienza, si evita la necessità per gli sviluppatori di specificare formalmente tutta la conoscenza di cui il computer necessita. La gerarchia dei concetti consente al computer di apprendere concetti complicati costruendoli da quelli più semplici. Se disegnassimo un grafico che mostri come questi concetti siano costruiti uno sopra l'altro, questo sarebbe "profondo", con numerosi livelli. Per questo motivo un approccio di questo tipo all'AI viene chiamato **deep learning**.

Le difficoltà incontrate dai sistemi che si basano su conoscenze hard-coded suggeriscono che i sistemi di AI necessitano della capacità di acquisire le proprie conoscenze estraendo modelli da dati grezzi. Questa capacità è nota con il termine di **machine learning.** L'introduzione dell'apprendimento automatico ha consentito ai computer di affrontare i problemi che

coinvolgono la conoscenza del mondo reale e di prendere decisioni che appaiono soggettive. Molte attività di intelligenza artificiale possono essere risolte progettando il giusto set di **features** (proprietà/caratteristiche) da estrarre per quell'attività, quindi fornendo queste funzionalità ad un algoritmo di machine learning. Tuttavia, per molte attività, è difficile sapere quali funzionalità dovrebbero essere estratte. Una soluzione a questo problema è utilizzare il machine learning per scoprire non solo la mappatura dalla rappresentazione in uscita ma anche la rappresentazione stessa. Questo approccio è conosciuto come **representation learning**. Una delle principali cause di difficoltà in molte applicazioni di intelligenza artificiale nel mondo reale è che molti fattori di variazione influenzano ogni singolo dato che siamo in grado di osservare (ad esempio i singoli pixel nell'immagine di un'auto rossa potrebbero essere molto vicini al nero di notte o in condizioni di luce sfavorevole). La maggior parte delle applicazioni ci richiede di districarsi tra i fattori di variazione e di scartare quelli che non ci interessano, dunque, può rivelarsi molto difficile estrarre caratteristiche di così alto livello di complessità da dati grezzi. Il deep learning risolve questo problema centrale nel representation learning introducendo rappresentazioni espresse in termini di altre rappresentazioni più semplici. Il deep learning consente al computer di creare concetti complessi da concetti più semplici. La figura 3.1 mostra come un sistema di deep learning può rappresentare il concetto di un'immagine di una persona combinando concetti più semplici, come angoli e contorni, che sono a loro volta definiti in termini di spigoli.

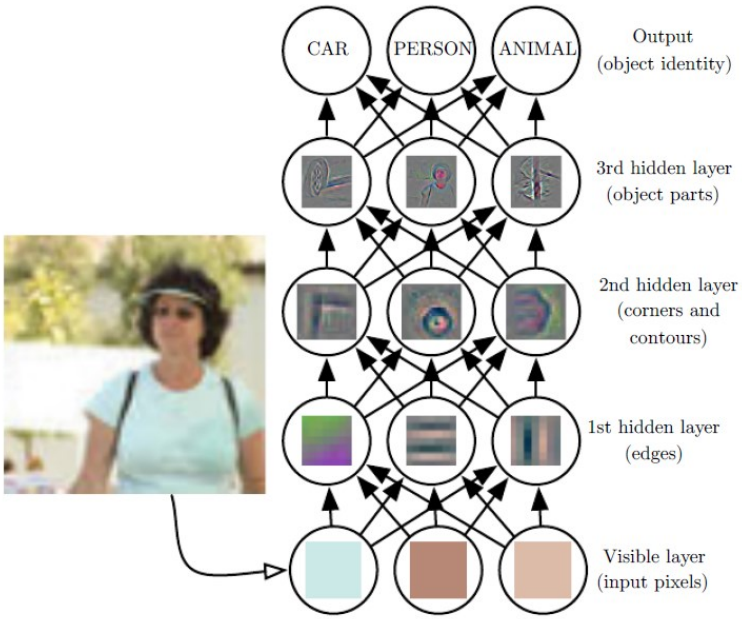


Figura 3.1 – Rappresentazione di un modello di deep learning

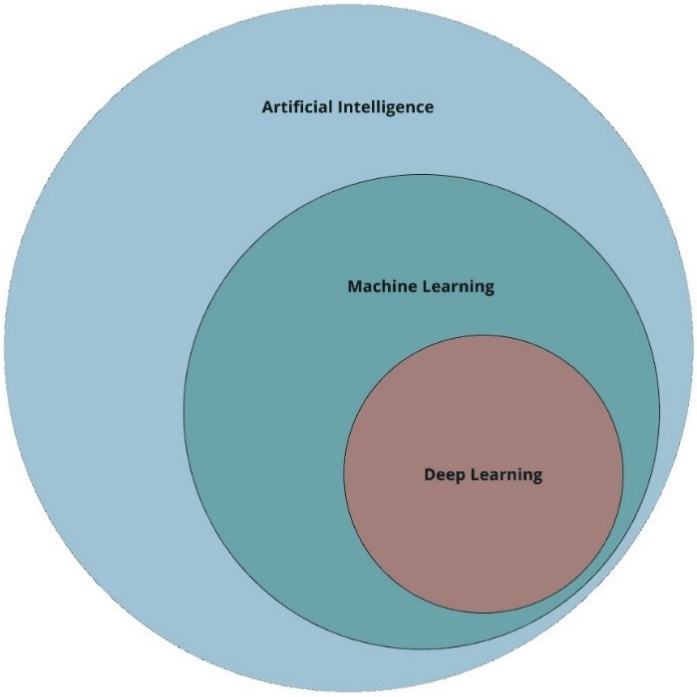
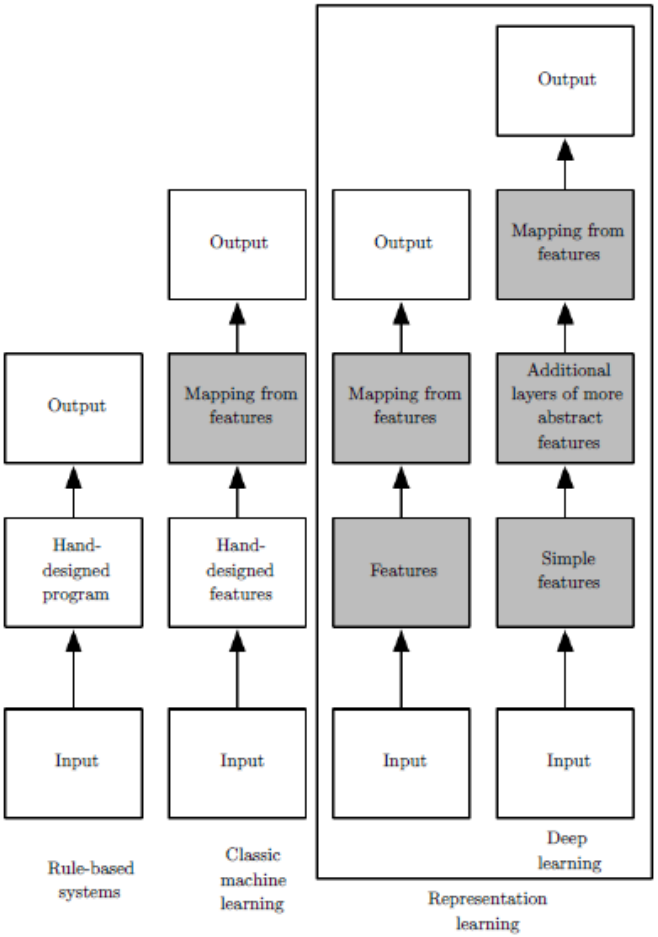


Figura 3.2 – Discipline dell'AI

La figura 3.2 illustra le relazioni esistenti tra le diverse discipline dell'AI, vediamo come il deep learning sia un tipo di machine learning che a sua volta è un approccio (non l'approccio) all'AI.

Figura 3.3 – Grafico di flusso delle diverse discipline dell'AI

I diagrammi in figura 3.3 mostrano come le diverse parti di un sistema AI si relazionino all'interno delle diverse discipline dell'intelligenza artificiale. Le caselle ombreggiate indicano i componenti che sono in grado di apprendere dai dati.



3.2 Fondamenti di Machine Learning

Il machine learning è essenzialmente una forma di statistica applicata con una maggiore enfasi sull'uso dei computer per stimare statisticamente funzioni complicate e una minore enfasi sulla dimostrazione degli intervalli di confidenza attorno a queste funzioni. Un algoritmo di machine learning è un algoritmo in grado di apprendere dai dati. Ma cosa intendiamo per apprendimento? Tom Mitchell nel 1997 ne diede una definizione:

“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E”

E cioè:

"Si dice che un programma per computer impari dall'esperienza E rispetto a qualche classe di compiti T e misura delle prestazioni P, se le sue prestazioni nei compiti in T, misurate da P, migliorano con l'esperienza E"

Si può immaginare un'ampia varietà di esperienze E, compiti T e misure di prestazione P, e in questa trattazione non facciamo alcun tentativo di fornire una definizione formale di ciò che può essere utilizzato per ciascuna di queste entità. Nel proseguo del paragrafo si forniranno invece descrizioni intuitive ed esempi di diversi tipi di compiti, misure delle prestazioni ed esperienze che possono essere utilizzate per costruire algoritmi di apprendimento automatico, con un approccio basilare.

3.2.1 Le Tasks, T

L'apprendimento automatico ci consente di affrontare compiti troppo difficili da risolvere con programmi classici scritti e progettati da esseri umani. In questa definizione relativamente formale della parola "compito", il processo di apprendimento stesso non è il compito, l'apprendimento è il nostro mezzo per raggiungere la capacità di svolgere il compito. Ad esempio, se vogliamo che un robot sia in grado di camminare, allora il compito è camminare. Potremmo programmare il robot per imparare a camminare, oppure potremmo tentare di scrivere direttamente un programma che specifichi come camminare manualmente.
Le attività di apprendimento automatico sono generalmente descritte in termini di come il sistema di machine learning dovrebbe elaborare un **esempio**. Un esempio è una raccolta di **caratteristiche/proprietà** (**features**) che sono state misurate quantitativamente da un oggetto o evento che vogliamo che il sistema di ML elabori. Tipicamente rappresentiamo un esempio come

un vettore $x \in R^n$ in cui ogni voce x_i del vettore è un'altra feature. Ad esempio, le caratteristiche di un'immagine sono solitamente i valori dei pixel nell'immagine.
Molti tipi di compiti possono essere risolti con il ML. Alcune delle attività di machine learning più comuni includono:

- **Classificazione**: In questo tipo di attività, al programma viene chiesto di specificare a quale delle k categorie appartiene un input. Un esempio di attività di classificazione è il riconoscimento degli oggetti, in cui l'input è un'immagine (di solito descritta come un insieme di valori di luminosità dei pixel) e l'output è un codice numerico che identifica l'oggetto nell'immagine. Il moderno riconoscimento degli oggetti è eseguito al meglio con il deep learning.
- **Classificazione con input mancanti**: La classificazione diventa più difficile se non si garantisce che ogni misura nel vettore di input sia sempre fornita. Questo tipo di situazione si presenta frequentemente nella diagnosi medica, perché molti tipi di test medici sono costosi o invasivi.
- **Regressione**: In questo tipo di attività, viene chiesto di prevedere un valore numerico dato un input. Questo tipo di attività è simile alla classificazione, tranne per il fatto che il formato dell'output è diverso. Applicazioni tipiche di questo tipo di compiti sono in abito economico (assicurazioni, titoli, trading)
- **Traduzione automatica**: in un'attività di traduzione automatica, l'input è già costituito da una sequenza di simboli in una lingua e il programma per computer deve convertirla in una sequenza di simboli in un'altra lingua. Questo è comunemente applicato alle lingue naturali, come la traduzione dall'inglese al francese. Il deep learning ha recentemente iniziato ad avere un impatto importante su questo tipo di attività, ed in maniera sempre più efficace.
- **Rilevamento di anomalie**: in questo tipo di attività, il programma per computer passa al setaccio una serie di eventi o oggetti e ne segnala alcuni come insoliti o atipici. Un esempio di rilevamento di anomalie è il rilevamento di frodi sulle carte di credito.

Altri compiti comuni che vengono svolti da algoritmi di ML sono: trascrizione, uscita strutturata, sintesi e campionamento, imputazione dei valori mancanti, denoising e tanti altri. Questo rende l'idea dell'ampio spettro di applicazione di questa tecnologia.

3.2.2 Misura delle Prestazioni, P

Per valutare le capacità di un algoritmo di ML, dobbiamo definire una misura quantitativa delle sue prestazioni. Di solito questa misura di prestazione P è specifica per il compito T svolto dal sistema.

Per attività quali classificazione, classificazione con input mancanti e trascrizione, spesso misuriamo l'**accuratezza** del modello. L'accuratezza è solo la proporzione di esempi per i quali il modello produce l'output corretto. Di solito siamo interessati alle prestazioni dell'algoritmo su dati che non abbia mai visto prima, poiché questo determina quanto bene funzionerà una volta distribuito nel mondo reale. Valutiamo queste misure prestazionali utilizzando un **test set** di dati separato dai dati utilizzati per l'addestramento del sistema.

La scelta della misura della performance può sembrare semplice e obiettiva, ma spesso è difficile scegliere una misura che corrisponda al meglio al comportamento desiderato del sistema. In alcuni casi, ciò è dovuto al fatto che è difficile decidere cosa misurare, questa scelta dipende molto dal tipo di applicazione che stiamo considerando. In altri casi, sappiamo quale quantità vorremmo idealmente misurare, ma misurarla non è pratico. In questi casi, si deve progettare un criterio alternativo che corrisponda comunque agli obiettivi di progettazione, o progettare una buona approssimazione al criterio desiderato.

3.2.3 Esperienza, E

Gli algoritmi di ML possono essere classificati in due grandi gruppi, come algoritmi **non supervisionati** o **supervisionati** in base al tipo di esperienza che possono avere durante il processo di apprendimento.

Gli algoritmi di apprendimento **senza supervisione** utilizzano un dataset contenente molte features; quindi, apprendono proprietà utili della struttura di questo set di dati, senza che queste siano state categorizzate precedentemente dallo sviluppatore.

Gli algoritmi di apprendimento **supervisionato** utilizzano un set di dati contenente funzionalità, ma ogni esempio è anche associato a un'etichetta o a un target. Il termine apprendimento supervisionato trae origine dal fatto che con questo metodo il target y è fornito da un "istruttore" che mostra al sistema cosa fare, dove arrivare. Nell'apprendimento non supervisionato, non c'è istruttore e l'algoritmo deve imparare a dare un senso ai dati senza questa guida esterna. Questa categorizzazione non è del tutto formale, questi due concetti non sono del tutto distinti, ma ci aiuta a suddividere molte delle cose si fanno con il ML.

Sono possibili altre varianti del paradigma dell'apprendimento. Ad esempio, nell'apprendimento semi-supervisionato, alcuni esempi includono una supervisione, altri no.
Alcuni algoritmi di ML non utilizzano un set di dati fisso. Ad esempio, gli algoritmi di **reinforcement learning** interagiscono con l'ambiente, cioè c'è un circuito di feedback tra il sistema di apprendimento e le sue esperienze. Questo genere di approccio è più complesso e va oltre lo scopo di questa trattazione.
La maggior parte degli algoritmi utilizza un solo **dataset**. Un dataset può essere descritto in molti modi. In tutti i casi, un dataset è una raccolta di esempi, che a loro volta sono raccolte di proprietà.

3.2.4 Capacità, Overfitting e Underfitting

La sfida centrale nel machine learning è quella di ottenere buoni risultati con input nuovi e inediti, non solo su quelli su cui è stato addestrato il nostro modello. La capacità di funzionare bene su input precedentemente non osservati è chiamata **generalizzazione**.
In genere, durante il training di un modello di machine learning, abbiamo accesso a un training set, possiamo calcolare una misura di errore sul training set chiamata errore di training e ridurre questo errore di training. Finora, quello che abbiamo descritto è semplicemente un problema di ottimizzazione. Ciò che separa il machine learning dall'ottimizzazione è che vogliamo che anche l'**errore di generalizzazione**, chiamato anche **errore di test**, sia il più basso possibile. L'errore di generalizzazione è definito come il valore atteso dell'errore su un nuovo input. Qui l'aspettativa è presa attraverso diversi possibili input, tratti dalla distribuzione degli input che ci aspettiamo che il sistema incontri nella pratica.
In genere stimiamo l'errore di generalizzazione di un modello di ML misurandone le prestazioni su un **test set** di esempi raccolti separatamente dal training set. Ci chiediamo come possiamo influenzare le prestazioni sul test set quando osserviamo solo il set di allenamento?
Il campo della teoria dell'apprendimento statistico fornisce alcune risposte. Se il training e il test set vengono raccolti in modo arbitrario, c'è davvero poco che possiamo fare. Se ci è permesso fare alcune ipotesi su come il training e il test set vengono raccolti, allora possiamo fare qualche progresso.
I dati di training e test sono generati da una distribuzione di probabilità sul dataset chiamata **processo di generazione dei dati**. Di solito si fanno una serie di ipotesi note collettivamente come **ipotesi i.i.d. (indipendenti ed identicamente distribuite).** Queste ipotesi prevedono che gli esempi in ciascun dataset siano indipendenti l'uno dall'altro e che il training set e il test set siano distribuiti in modo identico, tratti dalla stessa distribuzione di probabilità l'uno dell'altro. Questa ipotesi consente di descrivere il processo di generazione dei dati con una distribuzione di

probabilità su un singolo esempio. La stessa distribuzione viene quindi utilizzata per generare ogni esempio di training e ogni esempio di test. Questo quadro probabilistico e le ipotesi i.i.d. ci consentono di studiare matematicamente la relazione tra errore di training ed errore di test.
Una conseguenza immediata che possiamo osservare tra l'errore di training e di test è che l'errore di training atteso di un modello selezionato casualmente è uguale all'errore di test atteso di quel modello. I fattori che determinano le prestazioni di un algoritmo di ML sono la sua capacità di:

- Rendere l'errore di training piccolo.
- Rendere la differenza tra errore di training e test piccolo.

Questi due fattori corrispondono alle due sfide centrali del machine lerning: **underfitting** e **overfitting**. L'underfitting si verifica quando il modello non è in grado di ottenere un valore di errore sufficientemente basso sul training set. L'overfitting si verifica quando il divario tra l'errore di training e l'errore di test è troppo grande.
Possiamo controllare se un modello ha maggiori probabilità di overfit o underfit, alterando la sua **capacità**. In termini informali, la capacità di un modello è la sua abilità di adattarsi a un'ampia varietà di funzioni. I modelli con capacità ridotte potrebbero avere difficoltà a adattarsi al set di training. I modelli con capacità elevata possono incorrere nell'overfit memorizzando le proprietà del set di training che non risultano utili nel test set. Un modo per controllare la capacità di un algoritmo di apprendimento è scegliere il suo **hypothesis space**, l'insieme di funzioni che l'algoritmo di apprendimento può selezionare come soluzione.
Gli algoritmi generalmente funzionano al meglio quando la loro capacità è appropriata per la reale complessità dell'attività che devono eseguire e alla quantità di dati di addestramento che vengono forniti loro. I modelli con capacità insufficiente non sono in grado di svolgere compiti complessi. I modelli ad alta capacità possono risolvere compiti complessi, ma quando la loro capacità è superiore a quella necessaria per risolvere il compito richiesto possono incorrere in overfit. Tramite la figura 3.4 possiamo vedere in maniera qualitativa quanto sopra esposto, nel caso in cui l'algoritmo debba adattarsi ad una funzione quadratica sfruttando predizioni di tipo lineare, quadratico e di nono grado. Vediamo come la funzione lineare non sia in grado di seguire la curvatura (underfit); la funzione di nono grado riesce a trovare una soluzione accettabile, ma è in grado di trovarne infinite altre poiché ha a disposizione molti più parametri di quelli richiesti dall'esempio.

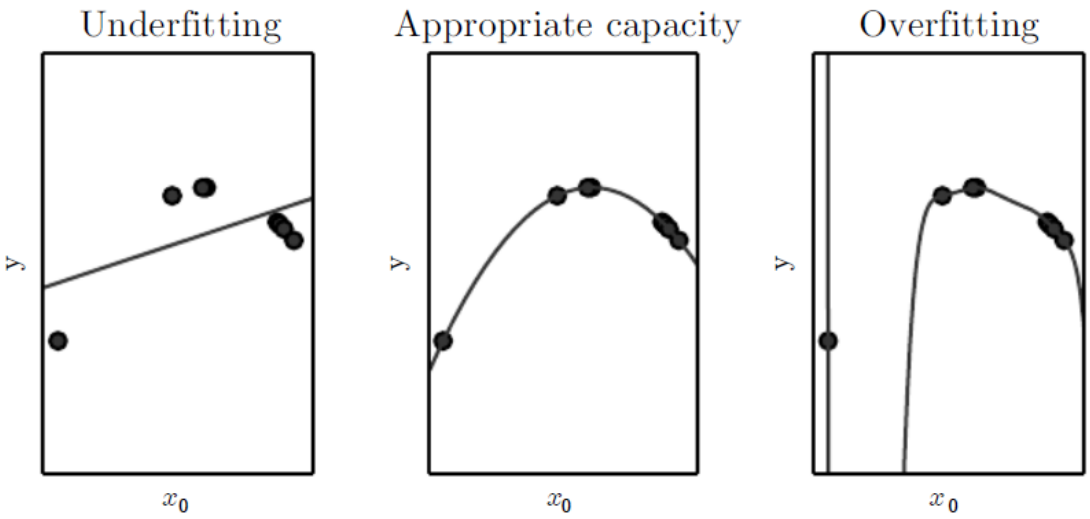


Figura 3.4 – Overfitting e Underfitting nel caso di una funzione quadratica

All'atto pratico ci sono diversi modi per variare a capacità del modello, essa non è determinata solo dalla scelta del modello. Il modello specifica quale famiglia di funzioni può scegliere l'algoritmo di apprendimento variando i parametri al fine di raggiungere un obiettivo di addestramento. Questa è chiamata **representational capacity** del modello. In molti casi, trovare la funzione migliore all'interno di questa famiglia è un problema di ottimizzazione molto difficile. Nella pratica, l'algoritmo di apprendimento non trova effettivamente la funzione migliore, ma semplicemente quella che riduce notevolmente l'errore di training**.** Queste limitazioni aggiuntive, come l'imperfezione dell'algoritmo di ottimizzazione, significano che la **capacità effettiva** dell'algoritmo di apprendimento può essere inferiore alla capacità rappresentativa della famiglia di modelli.

Nel caso del deep learning il problema di determinare la capacità di un modello è particolarmente difficile perché la capacità effettiva è limitata dalle potenzialità dell'algoritmo di ottimizzazione e abbiamo poca comprensione teorica dei problemi di ottimizzazione non convessa molto generali coinvolti nel deep learning. Un andamento tipico dell'errore di generalizzazione in funzione della capacità è mostrato in figura 3.5.

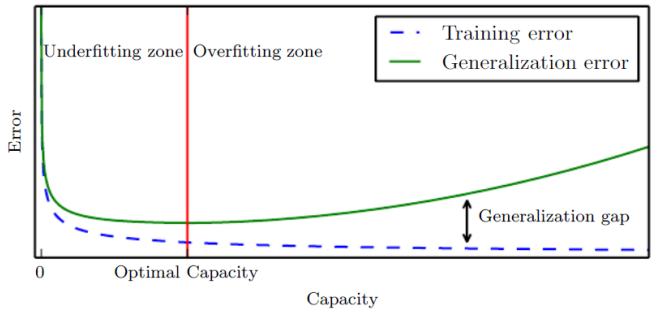


Figura 3.5 – Errore gi generalizzazione

I semplici algoritmi di machine learning sviluppati in questo campo lavorano molto bene in un’ampia varietà di problemi importanti. Tuttavia, non sono riusciti a risolvere i problemi centrali dell'AI, come il riconoscimento del parlato o degli oggetti, e lo sviluppo del deep learning è stato in parte motivato dall'incapacità degli algoritmi tradizionali di generalizzare bene su tali attività di intelligenza artificiale e in parte dalle dimensioni e dalla complessità sempre crescente dei dati trattati, che hanno reso il machine learning tradizionale inadatto.

3.3 Fondamenti di Reti Neurali

Il lavoro sulle reti neurali artificiali, comunemente chiamate reti neurali, è stato motivato inizialmente dall'osservazione del fatto che il cervello umano calcola in maniera completamente differente rispetto ai computer convenzionali. Il cervello è un computer altamente complesso, parallelo e non lineare. Esso ha la capacità di organizzare le sue strutture costituenti, chiamate neuroni in modo da effettuare certi calcoli in maniera molto più veloce rispetto al più veloce computer esistente oggi. Consideriamo ad esempio la visione umana, la quale è un compito di processo informativo. È funzione del sistema visivo fornire una rappresentazione dell'ambiente che ci circonda e, cosa più importante, fornire le informazioni di cui abbiamo bisogno per interagire con l'ambiente. Per essere precisi, il cervello svolge regolarmente compiti di riconoscimento percettivo (ad esempio, riconoscere un volto familiare incorporato in una scena sconosciuta) in circa 100-200 ms, mentre compiti di complessità molto minore richiedono molto più tempo su un computer potente. Nella sua forma più generale, una rete neurale è una macchina progettata per modellare il modo in cui il cervello esegue un particolare compito o funzione di interesse. Per ottenere buone prestazioni, le reti neurali utilizzano un'interconnessione massiccia di semplici celle di calcolo denominate "neuroni". Possiamo quindi fornire la seguente definizione di rete neurale vista come una macchina adattiva:

Una rete neurale è un processore distribuito massicciamente parallelo costituito da semplici unità di elaborazione che ha una naturale propensione a memorizzare la conoscenza e renderla disponibile per l'uso. Assomiglia al cervello sotto due aspetti:

1. *La conoscenza viene acquisita dalla rete dall'ambiente attraverso un processo di apprendimento.*
2. *Le forze di connessione interneuroniche, note come pesi sinaptici, vengono utilizzate per memorizzare le conoscenze acquisite.*

La procedura utilizzata per eseguire il processo di apprendimento è chiamata **algoritmo di apprendimento**, la cui funzione è quella di modificare i pesi sinaptici della rete in modo ordinato per raggiungere un obiettivo di progettazione desiderato.

La modifica dei pesi sinaptici fornisce il metodo tradizionale per la progettazione di reti neurali. Tale approccio è il più vicino alla teoria del filtro adattivo lineare, che è già ben consolidato e applicato con successo in molti campi diversi (Widrow e Stearns, 1985; Haykin, 2002). Tuttavia, è anche possibile per una rete neurale modificare la propria topologia, il che è motivato dal fatto che i neuroni nel cervello umano possono morire e possono crescere nuove connessioni sinaptiche.

3.3.1 Vantaggi delle Reti Neurali

È evidente che una rete neurale trae la sua potenza di calcolo attraverso, in primo luogo, la sua struttura distribuita massicciamente parallela e, in secondo luogo, la sua capacità di apprendere e quindi generalizzare. La **generalizzazione** si riferisce alla produzione da parte della rete neurale di output ragionevoli per input non incontrati durante il **training** (apprendimento). Queste due capacità di elaborazione delle informazioni consentono alle reti neurali di trovare buone soluzioni approssimate a problemi complessi (su larga scala) che sarebbero intrattabili. In pratica, tuttavia, le reti neurali non possono fornire la soluzione lavorando individualmente. Piuttosto, devono essere integrati con un approccio coerente di ingegneria dei sistemi. In particolare, un problema complesso di interesse viene scomposto in una serie di attività relativamente semplici e alle reti neurali viene assegnato un sottoinsieme delle attività che corrispondono alle loro capacità. È importante riconoscere, tuttavia, che abbiamo ancora molta strada da fare prima di poter costruire un'architettura che imiti il cervello umano.
Le reti neurali offrono le seguenti proprietà e capacità utili:

1. **Non linearità**. Un neurone artificiale può essere lineare o non lineare. Una rete neurale, costituita da un'interconnessione di neuroni non lineari, è essa stessa non lineare. Inoltre, la non linearità è di un tipo speciale, nel senso che è distribuita in tutta la rete. La non linearità è una proprietà molto importante, in particolare se il meccanismo fisico sottostante responsabile della generazione del segnale di ingresso (ad esempio il segnale vocale) è intrinsecamente non lineare.
2. **Mappatura ingresso-uscita**. Un paradigma di apprendimento molto usato, chiamato apprendimento supervisionato, comporta la modifica dei pesi sinaptici di una rete neurale applicando una serie di esempi di training etichettati o esempi di attività. Ogni esempio è costituito da un segnale di ingresso univoco e da una corrispondente risposta desiderata (obiettivo). Alla rete viene presentato un esempio prelevato a caso dall'insieme, e i pesi sinaptici (parametri liberi) della rete vengono modificati per ridurre al minimo la differenza tra la risposta desiderata e la risposta effettiva della rete prodotta dal segnale di ingresso secondo un opportuno criterio statistico. Il training della rete viene ripetuto per molti esempi del set, fino a quando la rete raggiunge uno stato stazionario in cui non ci sono ulteriori variazioni significative nei pesi sinaptici. Gli esempi di training precedentemente applicati possono essere riapplicati durante la sessione di allenamento, ma in una diversa ordine. Pertanto, la rete impara dagli esempi costruendo una mappatura input-output per il per il problema. Altri approcci sono utilizzati.

3. **Adattabilità**. Le reti neurali hanno una capacità intrinseca di adattare i loro pesi sinaptici ai cambiamenti nell'ambiente circostante. In particolare, una rete neurale addestrata per operare in un ambiente specifico può essere facilmente riallenata per far fronte a piccoli cambiamenti nelle condizioni ambientali operative. Inoltre, quando opera in un ambiente non stazionario (cioè in cui le statistiche cambiano nel tempo), una rete neurale può essere progettata per modificare i suoi pesi sinaptici in tempo reale. L'architettura naturale di una rete per la classificazione di pattern, l'elaborazione del segnale, e le applicazioni di controllo, insieme alla capacità adattiva della rete, la rendono uno strumento utile nella classificazione dei modelli adattivi, nell'elaborazione adattiva del segnale e nel controllo adattivo. Come regola generale, si può dire che più adattivo realizziamo un sistema, assicurando sempre che il sistema rimanga stabile, più solide saranno probabilmente le sue prestazioni quando il sistema dovrà funzionare in un ambiente non stazionario. Questo però non è sempre vero, anzi, può essere l'opposto. Ad esempio, un sistema adattativo con costanti di breve durata può cambiare rapidamente e quindi tendere a rispondere a disturbi spuri, provocando un drastico degrado delle prestazioni del sistema. Per realizzare tutti i vantaggi dell'adattabilità, le costanti temporali principali del sistema dovrebbero essere sufficientemente lente da consentire al sistema di ignorare disturbi spuri, e tuttavia abbastanza brevi da rispondere a cambiamenti significativi nell'ambiente.
4. **Risposta evidente**. Nel contesto della classificazione dei modelli, una rete neurale può essere progettata per fornire informazioni non solo su quale particolare modello selezionare, ma anche sulla fiducia nella decisione presa. Queste informazioni possono essere utilizzate per rifiutare schemi ambigui, qualora si presentino, e quindi migliorare le prestazioni di classificazione della rete.
5. **Informazioni contestuali**. La conoscenza è rappresentata dalla struttura stessa e dallo stato di attivazione di una rete neurale. Ogni neurone nella rete è potenzialmente influenzato dall'attività globale di tutti gli altri neuroni nella rete. Di conseguenza, le informazioni contestuali sono trattate naturalmente da una rete neurale.
6. **Tolleranza ai guasti**. Una rete neurale, implementata in forma hardware, ha il potenziale per essere intrinsecamente tollerante ai guasti o in grado di eseguire calcoli affidabili, nel senso che le sue prestazioni degradino con lentezza in condizioni operative avverse. Ad esempio, se un neurone o i suoi collegamenti di connessione sono danneggiati, la qualità nel richiamo di un pattern memorizzato è ridotta. Tuttavia, a causa della natura distribuita delle informazioni archiviate nella rete, il danno deve essere esteso prima che la risposta complessiva della rete venga seriamente deteriorata. Per essere certi che la rete neurale sia, tollerante ai guasti, potrebbe essere, in ogni caso, necessario

adottare misure correttive nella progettazione dell'algoritmo utilizzato per addestrare la rete

7. **Implementabilità VLSI**. La natura estremamente parallela di una rete neurale la rende potenzialmente veloce per il calcolo di determinate attività. Questa stessa caratteristica rende una rete neurale adatta per l'implementazione utilizzando la tecnologia VLSI (very-large-scale-integrated).
8. **Uniformità di analisi e progettazione**. Fondamentalmente, le reti neurali godono dell'universalità come processori di informazioni, nel senso che la stessa notazione viene utilizzata in tutti i domini che coinvolgono l'applicazione delle reti neurali. Questa caratteristica si manifesta in diversi modi:
    - I neuroni, in una forma o nell'altra, rappresentano un ingrediente comune a tutte le reti neurali.
    - Questo rende possibile condividere teorie e algoritmi di apprendimento in diverse applicazioni delle reti neurali.
    - Le reti modulari possono essere costruite attraverso una perfetta integrazione di moduli.
9. **Analogia neurobiologica**. La progettazione di una rete neurale è motivata dall'analogia con il cervello, che è la prova vivente che l'elaborazione parallela tollerante ai guasti non è solo fisicamente possibile, ma anche veloce e potente**.** I neurobiologi considerano le reti neurali (artificiali) come uno strumento di ricerca per l'interpretazione dei fenomeni neurobiologici. D'altra parte, gli ingegneri guardano alla neurobiologia per nuove idee per risolvere problemi più complessi di quelli basati su tecniche di progettazione cablate convenzionali.

3.3.2 Modello di un Neurone

Un **neurone** è un'unità di elaborazione delle informazioni fondamentale per il funzionamento di una rete neurale. Il diagramma a blocchi di figura 3.6 mostra il modello di un neurone, che costituisce la base per la progettazione di una grande famiglia di reti neurali. Identifichiamo tre elementi base del modello neurale:

1. Un insieme di **sinapsi**, ognuna delle quali è caratterizzata da un proprio peso. In particolare, un segnale x_j all'ingresso della sinapsi j collegata al neurone k viene moltiplicato per il peso sinaptico w_{kj}. A differenza del peso di una sinapsi nel

cervello, il peso sinaptico di un neurone artificiale può trovarsi in un intervallo che include valori negativi e positivi.

2. Un **sommatore** per sommare i segnali di ingresso, pesati dalle rispettive forze sinaptiche del neurone; le operazioni qui descritte costituiscono un combinatore lineare.
3. Una **funzione di attivazione** per limitare l'ampiezza dell'uscita di un neurone. La funzione di attivazione viene anche definita funzione di schiacciamento, in quanto schiaccia (limita) l'intervallo di ampiezza consentito del segnale di uscita a un valore finito.

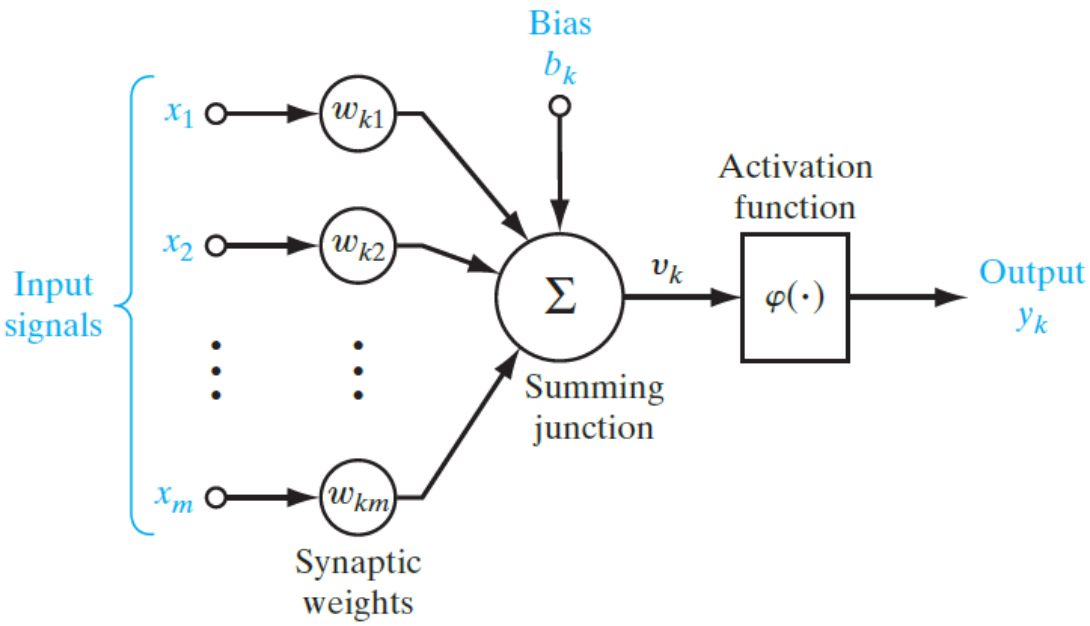
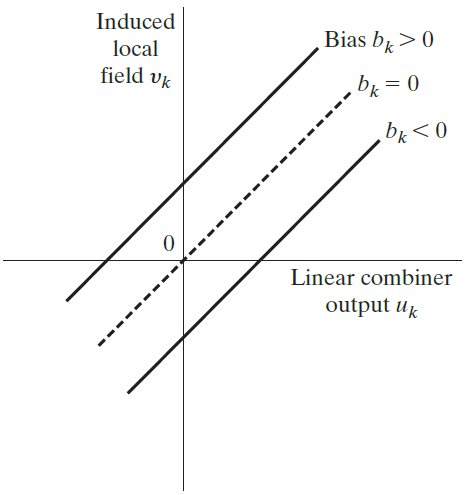


Figura 3.6 – Modello di un neurone

Il modello neurale di figura 3.6 include anche un bias applicato esternamente, indicato da b_k. Il bias b_k ha l'effetto di aumentare o diminuire l'input della funzione di attivazione, a seconda che sia rispettivamente positivo o negativo (figura 3.7). In termini matematici, possiamo descrivere il neurone k rappresentato in figura 3.6 scrivendo la coppia di equazioni:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \qquad e \qquad y_k = \varphi(u_k + b_k)$$

Figura 3.7 – Trasformazione affine prodotta dalla presenza di bias



3.3.3 Reti neurali viste come grafici diretti

Possiamo semplificare l'aspetto del modello utilizzando l'idea dei grafici di flusso del segnale senza sacrificare nessuno dei dettagli funzionali del modello. I grafici di flusso del segnale, con un insieme ben definito di regole, furono originariamente sviluppati da Mason (1953, 1956) per le reti lineari. Un **grafo di flusso del segnale** è una rete di collegamenti diretti (rami) che sono interconnessi in determinati punti chiamati **nodi**. Il grafico ha una funzione di trasferimento associata, o trasmittanza, che specifica il modo in cui il segnale y_k al nodo k dipende dal segnale x_j al nodo j. Il flusso dei segnali nelle varie parti del grafico è dettato dalle regole fondamentali dei grafici di flusso. Si possono distinguere due diversi tipi di collegamenti:

- **Collegamenti sinaptici**, il cui comportamento è governato da una relazione ingresso-uscita lineare. In particolare, il segnale di nodo x_j viene moltiplicato per il peso sinaptico w_{kj} per produrre il segnale di nodo y_k, come illustrato in figura. 3.8a.
- **collegamenti di attivazione**, il cui comportamento è regolato in generale da una relazione ingresso-uscita non lineare. Questa forma di relazione è illustrata in figura 3.8b, dove **φ**(·) è la funzione di attivazione non lineare.

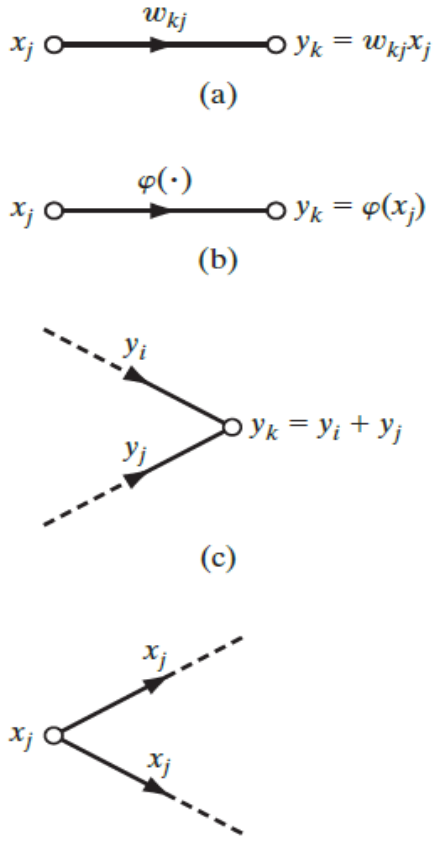


Figura 3.8 – Regole base per la costruzione del grafico di flusso

Ad esempio, utilizzando queste regole, possiamo costruire il grafico del flusso del segnale di figura 3.9 come modello di un neurone, corrispondente al diagramma a blocchi di figura 3.6.

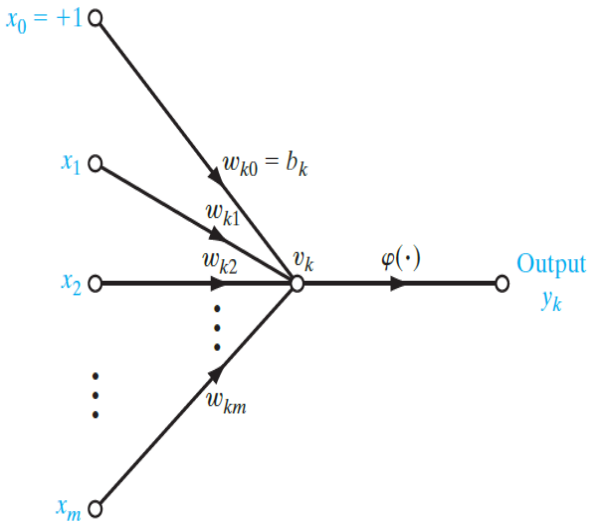


Figura 3.9 – Grafico di flusso di un neurone

Sulla base del grafico del flusso del segnale di figura 3.9 come modello di un neurone, possiamo ora offrire la seguente definizione matematica di una rete neurale:

Una rete neurale è un grafo diretto costituito da nodi con collegamenti sinaptici e di attivazione interconnessi ed è caratterizzato da quattro proprietà:

1. *Ogni neurone è rappresentato da un insieme di collegamenti sinaptici lineari, un bias applicato esternamente e un possibile collegamento di attivazione non lineare. Il bias è rappresentato da un collegamento sinaptico collegato a un input fissato a +1.*
2. *I collegamenti sinaptici di un neurone pesano i rispettivi segnali di ingresso.*
3. *La somma ponderata dei segnali di ingresso definisce il campo locale indotto del neurone in questione.*
4. *Il collegamento di attivazione schiaccia il campo locale indotto del neurone per produrre un output.*

Un grafo orientato, definito in questo modo, è **completo** nel senso che descrive non solo il flusso del segnale da neurone a neurone, ma anche il flusso del segnale all'interno di ciascun neurone. Possiamo utilizzare una forma ridotta di questo grafico omettendo i dettagli del flusso del segnale all'interno dei singoli neuroni. Un tale grafo orientato si dice **parzialmente completo**, viene definito grafo architettonico, descrivendo il layout della rete neurale (figura 3.10).

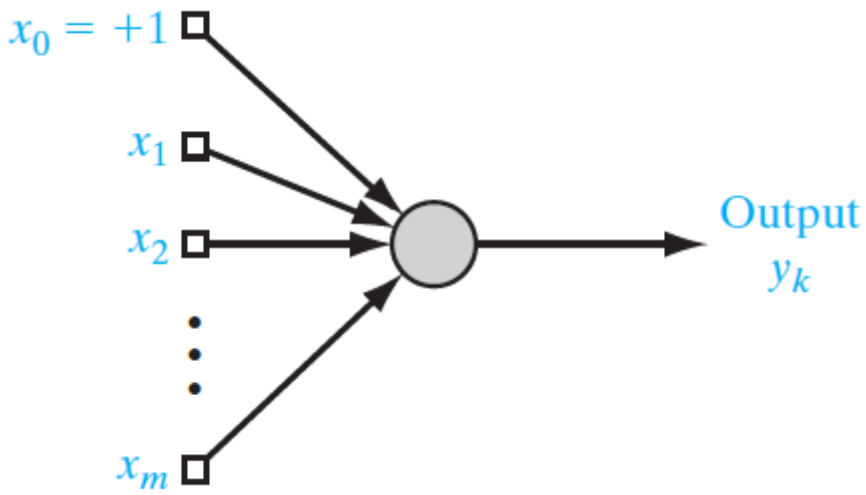


Figura 3.10 – Grafico Architettonico di un neurone

Per riassumere, abbiamo tre rappresentazioni grafiche di una rete neurale: diagramma a blocchi, grafo di flusso del segnale e grafo architettonico.

3.3.4 Feedback

Si dice che il feedback esiste in un sistema dinamico ogni volta che l'uscita di un elemento nel sistema influenza in parte l'ingresso applicato a quel particolare elemento, dando così origine a uno o più percorsi chiusi per la trasmissione di segnali attorno al sistema. In effetti, il feedback si verifica in quasi ogni parte del sistema nervoso di ogni animale (Freeman, 1975). Inoltre, svolge un ruolo importante nello studio di una classe speciale di reti neurali note come reti ricorrenti. La figura 3.11a mostra il grafico del flusso del segnale di un sistema di feedback ad anello singolo e una sua approssimazione (figura 3.11b).

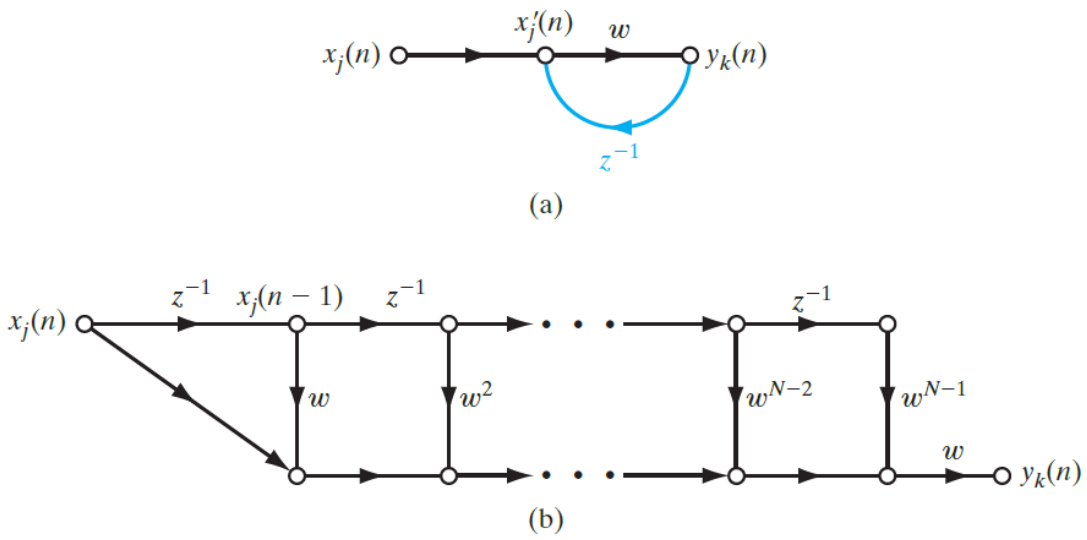


Figura 3.11 – Grafico di flusso di un sistema con feedback e una sua approssimazione

L'analisi del comportamento dinamico delle reti neurali che comporta l'applicazione del feedback è purtroppo complicata dal fatto che le unità di elaborazione utilizzate per la costruzione della rete sono generalmente non lineari. Ci limiteremo dunque solo a questi aspetti qualitativi, ulteriori approfondimenti andrebbero al di là dello scopo di questa introduzione.

3.3.5 Architettura di Rete

Il modo in cui sono strutturati i neuroni di una rete neurale è intimamente legato all'algoritmo di apprendimento utilizzato per addestrare la rete. Si può quindi parlare di algoritmi di apprendimento (regole) utilizzati nella progettazione di reti neurali come **strutturati**. In generale, possiamo identificare fondamentalmente tre classi diverse di architetture di rete:

1. **Reti Feedforward Single-layer**

In una rete neurale a strati, i neuroni sono organizzati sotto forma di strati. Nella forma più semplice, abbiamo uno **strato di input** di nodi sorgente che si proietta direttamente su uno **strato di output** di neuroni (nodi di calcolo), ma non viceversa. In altre parole, questa rete è strettamente di tipo **feedforward**. Un esempio di questo tipo di rete è illustrato in figura 3.12. Tale rete è chiamata rete a **single layer**, con la designazione "single layer" che si riferisce allo strato di output dei nodi di calcolo (neuroni). Non contiamo il livello di input dei nodi sorgente perché lì non viene eseguito alcun calcolo.

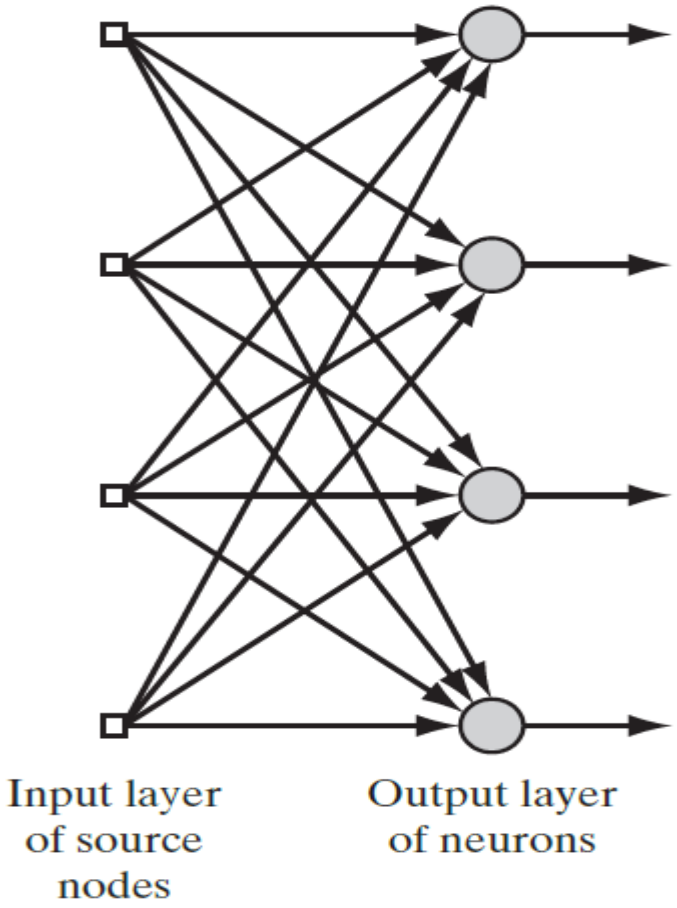


Figura 3.12 – Rete feedforward a single layer

2. **Reti Feedforward Multilayer**

La seconda classe di una rete neurale feedforward si distingue per la presenza di uno o più **hidden layers**, i cui nodi di calcolo sono corrispondentemente chiamati **hidden neurons**; il termine "hidden" (nascosto) si riferisce al fatto che questa parte della rete neurale non è vista direttamente né dall'input né dall'output della rete. La funzione dei

neuroni nascosti è di intervenire in qualche modo utile tra l'input esterno e l'output di rete. Aggiungendo uno o più livelli nascosti, la rete è abilitata ad estrarre statistiche di ordine superiore dal suo input. In un senso piuttosto ampio, la rete acquisisce una prospettiva globale nonostante la sua connettività locale, a causa dell'insieme aggiuntivo di connessioni sinaptiche e della dimensione extra delle interazioni neurali. L'insieme dei segnali di uscita dei neuroni nello strato di uscita (finale) della rete costituisce la risposta complessiva della rete al pattern di attivazione fornito dai nodi sorgente nello strato di ingresso (primo). Un esempio è illustrato in figura 3.13.

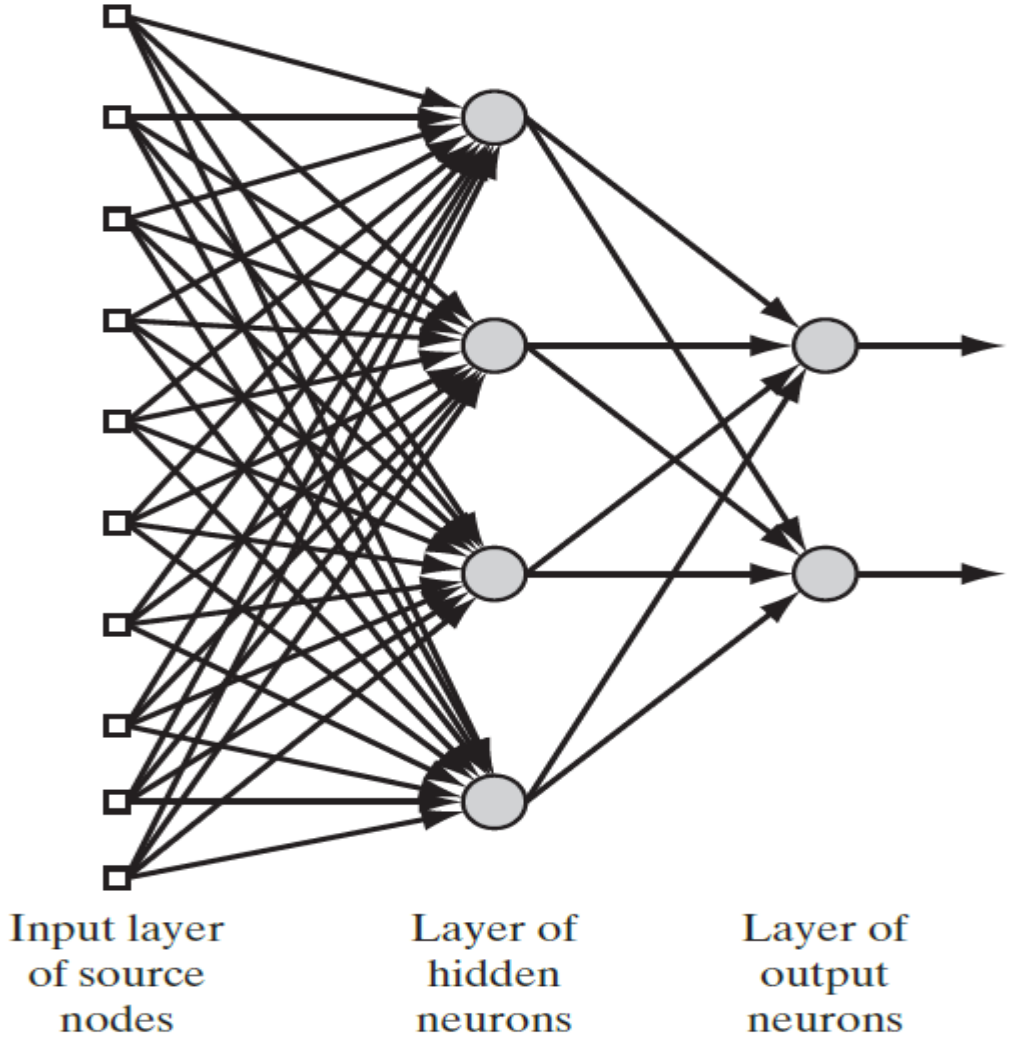


Figura 3.13 - Rete feedforward a più layer

Si dice che la rete neurale in figura 3.13 sia **completamente connessa** nel senso che ogni nodo in ogni strato della rete è connesso a ogni altro nodo nell'adiacente strato di forward. Se, invece, nella rete mancano alcuni collegamenti di comunicazione (connessioni sinaptiche), diciamo che la rete è **parzialmente connessa**.

3. Reti Ricorrenti

Una rete neurale ricorrente si distingue da una rete neurale feedforward in quanto ha almeno un ciclo di feedback. Ad esempio, una rete ricorrente può essere costituita da un singolo strato di neuroni con ciascun neurone che alimenta con il suo segnale di uscita gli ingressi di tutti gli altri neuroni, come illustrato nel grafico architettonico in figura 3.14. In figura 3.15 illustriamo un'altra classe di reti ricorrenti con neuroni nascosti. Le connessioni di feedback mostrate in figura 3.15 provengono dai neuroni nascosti così come dai neuroni di uscita.

Figura 3.14 – Rete ricorrente

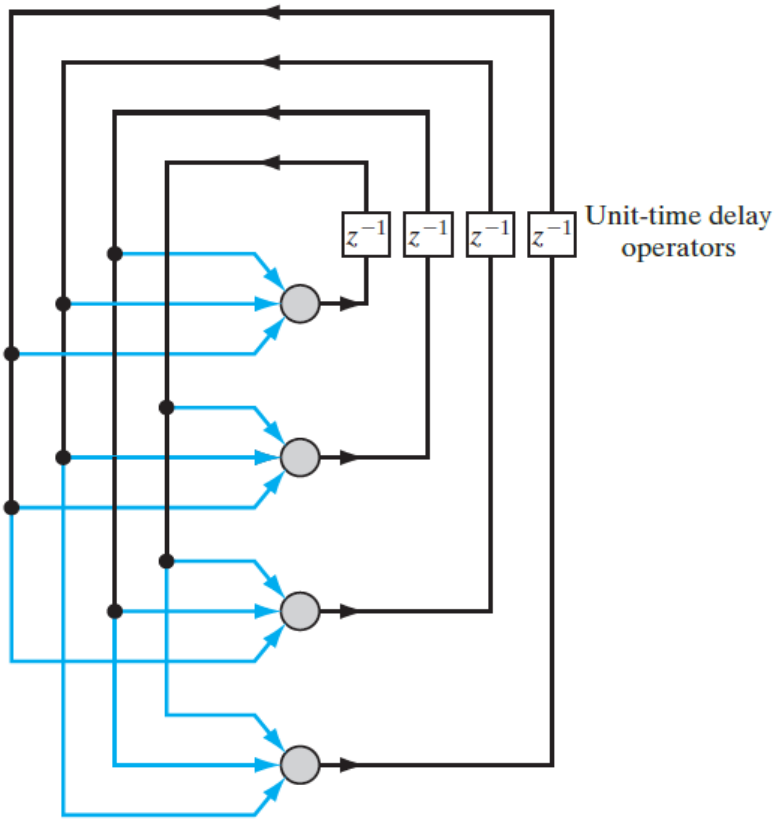
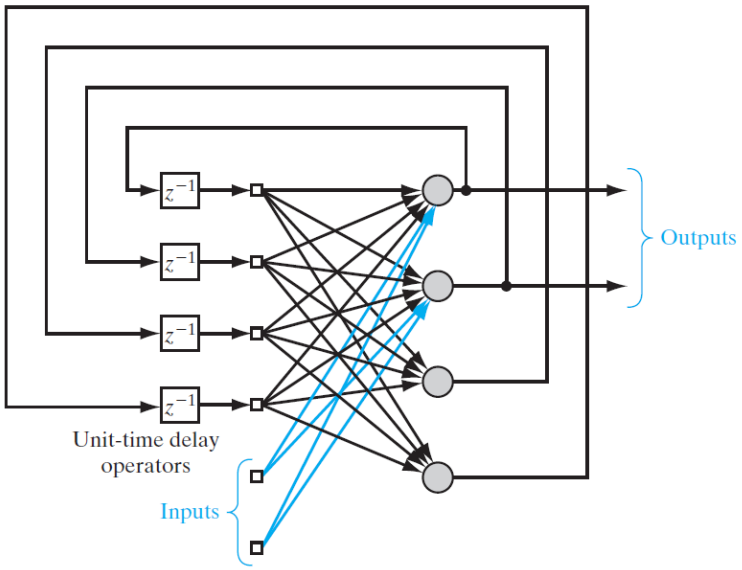


Figura 3.15 – Rete ricorrente con neuroni nascosti



3.3.6 Rappresentazione della conoscenza

Viene data la seguente definizione di conoscenza (Fischler e Firschein, 1987):

La conoscenza si riferisce alle informazioni archiviate o ai modelli utilizzati da una persona o da una macchina per interpretare, prevedere e rispondere in modo appropriato al mondo esterno.

Tipicamente, però, troviamo che le possibili forme di rappresentazione, dei parametri degli ingressi e della rete interna, sono molto diverse, il che tende a rendere lo sviluppo di una soluzione soddisfacente per mezzo di una rete neurale una vera sfida progettuale. Un compito importante per una rete neurale è apprendere un modello del mondo (ambiente) in cui è incorporata e mantenere il modello sufficientemente coerente con il mondo reale in modo da raggiungere gli obiettivi specificati della domanda di interesse. La conoscenza del mondo consiste in due tipi di informazioni:

1. Lo stato del mondo conosciuto, rappresentato dai fatti su ciò che è e ciò che è stato conosciuto; questa forma di conoscenza è denominata informazione preventiva.
2. Osservazioni (misure) del mondo, ottenute per mezzo di sensori progettati per sondare l'ambiente, in cui dovrebbe operare la rete neurale. Normalmente, queste osservazioni sono intrinsecamente rumorose, essendo soggette a errori dovuti al rumore del sensore e alle imperfezioni del sistema. In ogni caso, le osservazioni così ottenute forniscono l'insieme di informazioni, da cui sono tratti gli esempi utilizzati per addestrare la rete neurale.

Come visto nel capitolo precedente gli esempi possono essere etichettati o non etichettati. Si noti, tuttavia, che gli esempi etichettati possono essere impegnativi da raccogliere, poiché richiedono la disponibilità di un "insegnante" per fornire una risposta desiderata per ogni esempio etichettato. Al contrario, gli esempi non etichettati sono generalmente abbondanti in quanto non è necessaria la supervisione. Un insieme di coppie ingresso-uscita, con ciascuna coppia costituita da un segnale di ingresso e la corrispondente risposta desiderata, è indicato come un insieme di **dati di training**. Il percorso di apprendimento di una rete neurale segue passi comuni a quelli trattati per il più generale machine learning, cioè una fase in cui vengono forniti alla rete una serie di esempi di training da cui la rete apprende; una successiva fase di testing con esempi mai visti dalla rete da cui possiamo ricavare le vere prestazioni della rete. In una rete neurale di architettura specificata, la rappresentazione della conoscenza dell'ambiente circostante è definita dai valori assunti dai parametri liberi (cioè, pesi sinaptici e bias) della rete. La forma di questa rappresentazione costituisce il progetto stesso della rete neurale, e quindi è la chiave per le sue prestazioni.

Il tema di come la conoscenza sia effettivamente rappresentata all'interno di una rete artificiale è, tuttavia, molto complicato. Tuttavia, ci sono quattro regole per la rappresentazione della conoscenza che sono di natura generale di buon senso, come descritto di seguito.

1. Input simili da classi simili dovrebbero solitamente produrre rappresentazioni simili all'interno della rete e dovrebbero quindi essere classificati come appartenenti alla stessa classe.
2. Gli elementi da classificare come classi separate dovrebbero avere rappresentazioni ampiamente diverse nella rete.
3. Se una caratteristica particolare fosse importante, allora dovrebbe esserci un gran numero di neuroni coinvolti nella rappresentazione di quell'elemento nella rete.
4. Le informazioni preliminari e le invarianze dovrebbero essere integrate nella progettazione di una rete neurale ogni volta che sono disponibili, in modo da semplificare la progettazione della rete poiché non è necessario apprenderle.

La regola 4 è particolarmente importante perché la corretta aderenza ad essa si traduce in una rete neurale con una struttura specializzata. Questo è altamente auspicabile per diversi motivi:

- Le reti biologiche visive e uditive sono note per essere molto specializzate.
- Una rete neurale con una struttura specializzata di solito ha un numero inferiore di parametri liberi disponibili per la regolazione rispetto a una rete completamente connessa. Di conseguenza, la rete specializzata richiede un set di dati più piccolo per la formazione, apprende più velocemente e spesso generalizza meglio.
- La velocità di trasmissione delle informazioni attraverso una rete specializzata aumenta.
- Il costo di costruzione di una rete specializzata è ridotto a causa delle sue dimensioni ridotte rispetto a quelle della sua controparte completamente connessa.

Si noti, tuttavia, che l'incorporazione di conoscenze pregresse nella progettazione di una rete neurale limita l'applicazione della rete al particolare problema affrontato.

Il problema della rappresentazione della conoscenza in una rete neurale è direttamente correlato a quello dell'architettura di rete. sfortunatamente, non esiste una teoria ben sviluppata per ottimizzare l'architettura di una rete neurale necessaria per interagire con un ambiente di interesse, o per valutare il modo in cui i cambiamenti nell'architettura di rete influiscono sulla rappresentazione della conoscenza all'interno della rete. In effetti, risposte soddisfacenti a questi problemi si trovano solitamente attraverso uno studio sperimentale esauriente per una specifica applicazione di interesse, con il progettista della rete neurale che diventa una parte essenziale del ciclo di apprendimento strutturale.

3.3.7 Processo di apprendimento

Il processo ha considerazioni analoghe a quelle trattate per il macrogruppo del machine learning nel paragrafo 3.2 di cui le reti neurali sono una sottoclasse, andremo a valutare allora gli aspetti specifici di questa tecnologia, che la caratterizzano.

Apprendimento con insegnante

Le considerazioni sono analoghe a quelle già trattate nei paragrafi precedenti. La figura 3.16 illustra un diagramma a blocchi che rappresenta questa forma di apprendimento.

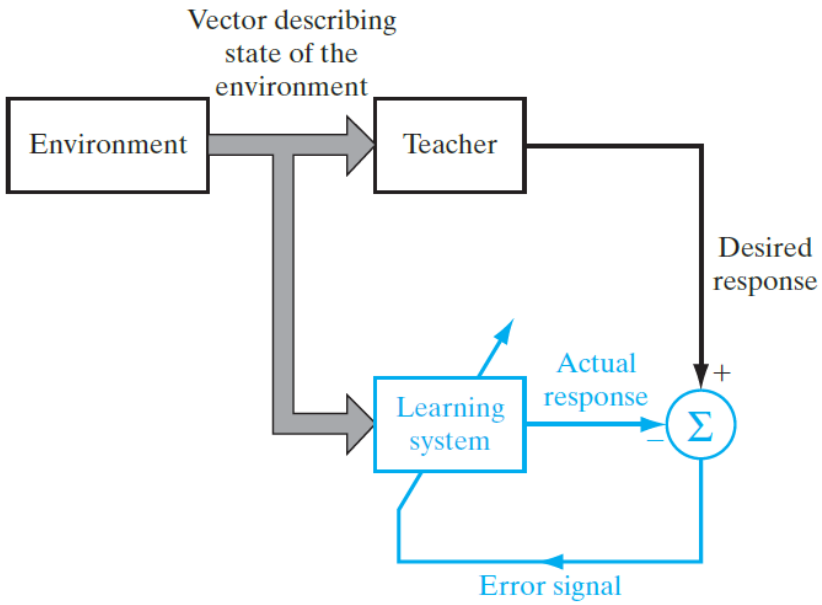


Figura 3.16 – Apprendimento con insegnante

Gli aggiustamenti vengono eseguiti in modo iterativo passo dopo passo con l'obiettivo di far sì che la rete neurale emuli l'insegnante; si presume che l'emulazione sia ottimale in un certo senso statistico. In questo modo, la conoscenza dell'ambiente a disposizione dell'insegnante viene trasferita alla rete neurale attraverso l'allenamento e memorizzata sotto forma di pesi sinaptici "fissi", che rappresentano memoria a lungo termine. Quando questa condizione viene raggiunta, possiamo fare a meno dell'insegnante e lasciare che la rete neurale si occupi dell'ambiente da sola. Dato un algoritmo progettato per ridurre al minimo la funzione errore, un insieme adeguato di esempi di input-output e un tempo sufficiente per eseguire l'addestramento, un sistema di apprendimento supervisionato è solitamente in grado di approssimare ragionevolmente bene una mappatura input-output sconosciuta.

Apprendimento senza insegnante

In questo secondo paradigma si individuano due sottocategorie:

1. Reinforcement Learning

Nell'apprendimento per rinforzo, l'apprendimento di una mappatura input-output viene eseguito attraverso l'interazione continua con l'ambiente al fine di ridurre al minimo un indice scalare di prestazioni. La figura 3.17 mostra il diagramma a blocchi di una forma di sistema di apprendimento per rinforzo costruito attorno a un **critic** che converte un **segnale di rinforzo primario** ricevuto dall'ambiente in un segnale di rinforzo di qualità superiore chiamato **segnale di rinforzo euristico**, entrambi ingressi scalari. L'obiettivo dell'apprendimento per rinforzo è ridurre al minimo una funzione cost-to-go, definita come l'aspettativa del costo cumulativo delle azioni intraprese in una sequenza di passaggi anziché semplicemente il costo immediato. La funzione del sistema di apprendimento è scoprire queste azioni e restituirle all'ambiente.

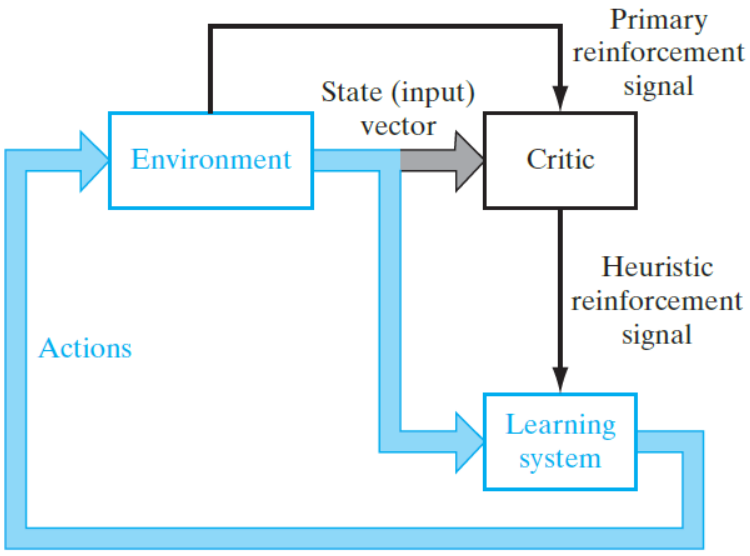


Figura 3.17 – Reinforcement learning

2. Apprendimento non supervisionato

Nelle reti neurali il concetto è lo stesso già trattato (figura 3.18). Nel dettaglio per eseguire l'apprendimento non supervisionato, possiamo utilizzare una regola di apprendimento competitivo. Ad esempio, possiamo utilizzare una rete neurale composta da due livelli: uno di input e uno competitivo. Il livello di input riceve i dati disponibili. Lo strato competitivo è costituito da neuroni che competono tra loro per l'"opportunità" di rispondere alle caratteristiche contenute nei dati di input.

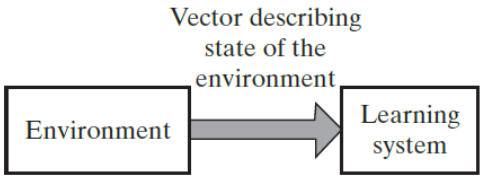


Figura 3.18 – Apprendimento non supervisionato

3.3.8 Learning Tasks

La scelta di una particolare regola di apprendimento è ovviamente influenzata dal learning task, la cui diversa natura testimonia l'universalità delle reti neurali.

Associazione del modello

Una memoria associativa è una memoria distribuita simile al cervello che apprende per associazione. L'associazione assume una di due forme: autoassociazione ed eteroassociazione. Nell'**autoassociazione**, una rete neurale è necessaria per memorizzare una serie di modelli (vettori) presentandoli ripetutamente alla rete. Successivamente alla rete viene presentata una descrizione parziale o una versione distorta (rumorosa) di un pattern originale in essa memorizzato e il compito è recuperare (richiamare) quel particolare pattern. L'**eteroassociazione** differisce dall'autoassociazione in quanto un insieme arbitrario di modelli di input è associato a un altro insieme arbitrario di modelli di output. L'autoassociazione implica l'uso dell'apprendimento non supervisionato, mentre il tipo di apprendimento coinvolto nell'eteroassociazione è supervisionato. Non è necessario entrare ulteriormente nel dettaglio della questione.

Riconoscimento del modello

Il riconoscimento del modello è formalmente definito come il processo mediante il quale un modello/segnale ricevuto viene assegnato a una delle classi prescritte. La rete neurale esegue il riconoscimento dei modelli effettuando prima una sessione di formazione durante la quale alla rete viene ripetutamente presentato un insieme di modelli di input insieme alla categoria a cui appartiene ciascun modello particolare. Successivamente, alla rete viene presentato un nuovo modello che non è mai stato visto prima, ma che appartiene alla stessa popolazione di modelli utilizzati per addestrare la rete. La rete è in grado di identificare la classe di quel particolare pattern grazie alle informazioni estratte dai dati di addestramento.

Il riconoscimento dei modelli eseguito da una rete neurale è di natura statistica, con i modelli rappresentati da punti in uno spazio decisionale multidimensionale. Lo spazio decisionale è suddiviso in regioni, ognuna delle quali è associata a una classe. I confini decisionali sono determinati dal processo di formazione. La costruzione di questi confini è resa statistica dalla variabilità intrinseca che esiste all'interno e tra le classi.

In termini generici, le macchine di riconoscimento di pattern che utilizzano reti neurali possono assumere una di due forme:

- La macchina è divisa in due parti, una rete non supervisionata per l'estrazione delle features e una rete supervisionata per la classificazione, come mostrato nel sistema ibridato di figura 3.19a. In termini concettuali, un pattern è rappresentato da un insieme di m osservabili, che possono essere visti come un punto x in uno **spazio di osservazione** (dati) **m-dimensionale**. L'estrazione delle caratteristiche è descritta da una trasformazione che mappa il punto x in un punto intermedio y in uno **spazio delle caratteristiche q-dimensionale** con q < m, come indicato in figura 3.19b. Questa trasformazione può essere vista come una riduzione della dimensionalità (cioè, compressione dei dati), il cui uso è giustificato in quanto semplifica il compito di classificazione. La classificazione stessa è descritta come una trasformazione che mappa il punto intermedio y in una delle classi in uno **spazio decisionale r-dimensionale**, dove r è il numero di classi da distinguere.

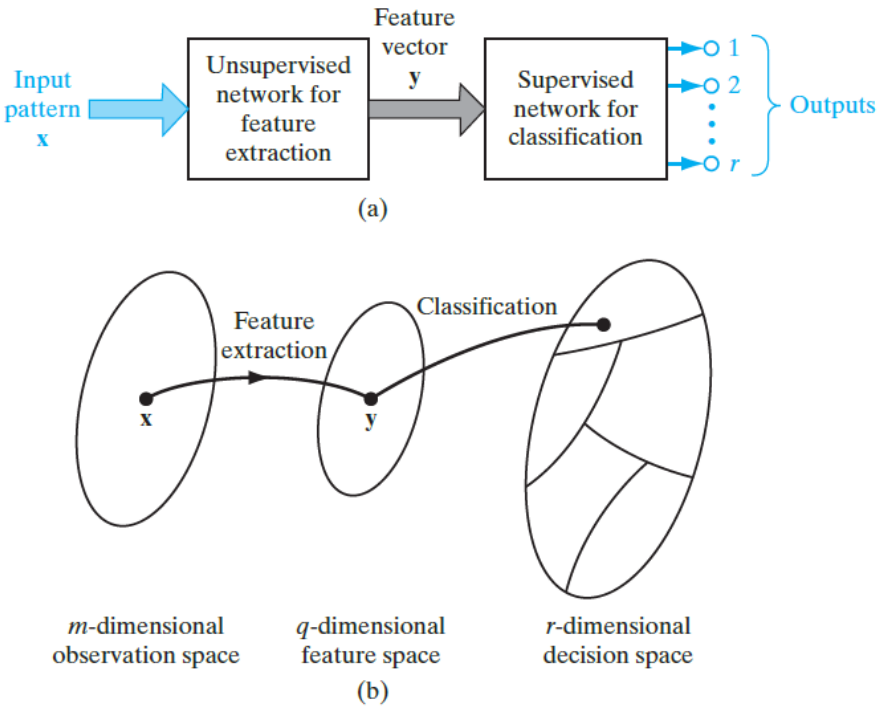


Figura 3.19 – Approccio classico alla classificazione dei pattern

- La macchina è progettata come una rete feedforward utilizzando un algoritmo di apprendimento supervisionato. In questo secondo approccio, il compito di estrazione delle caratteristiche è svolto dalle unità computazionali negli strati nascosti della rete.

3.4 Edge Impulse

Edge Impulse è una piattaforma online per lo sviluppo di progetti di machine learning su edge devices (figura 3.20). L'**edge machine learning** (Edge ML) è il processo di esecuzione di algoritmi di machine learning su dispositivi alla periferia di una rete per prendere decisioni e previsioni il più vicino possibile alla fonte di origine dei dati. Viene anche chiamata edge artificial intelligence o edge AI. Nel machine learning tradizionale, troviamo spesso server di grandi dimensioni che elaborano cumuli di dati raccolti da Internet per fornire alcuni vantaggi, come prevedere quale film guardare o etichettare automaticamente un video in un certo gruppo. Eseguendo algoritmi di ML su dispositivi edge come laptop, smartphone e sistemi embedded (come quelli che si trovano in smartwatch, lavatrici, automobili, robot di produzione, ecc.), possiamo produrre tali previsioni più velocemente e senza la necessità di trasmettere grandi quantità di dati attraverso una rete. Questo genere di approccio si sposa bene con il sempre crescente interesse ed espansione dell'Internet of Things (IoT), in quanto risolve tutta una serie di problemi legati alla tecnologia come:

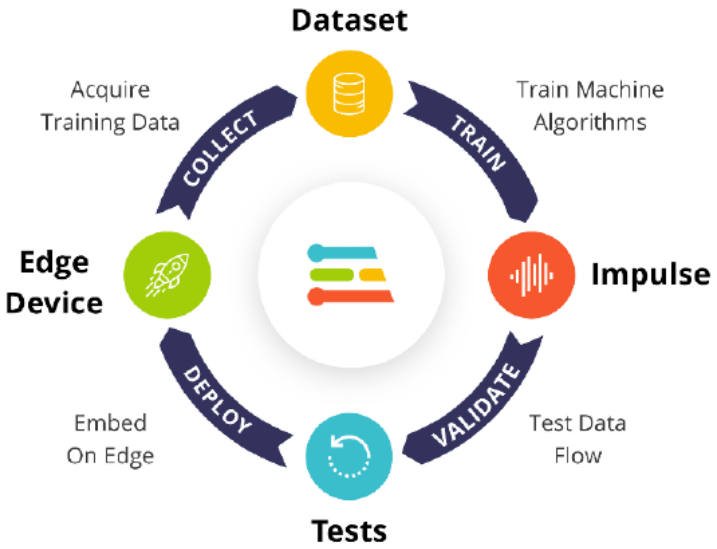


Figura 3.20 – Edge Impulse

- La trasmissione di dati di grandi dimensioni da sensori, come le immagini, per cui aumenta notevolmente la larghezza di banda richiesta alla rete.
- La trasmissione dei dati richiede una certa potenza.
- I sensori richiedono una connessione costante al server per fornire calcoli ML quasi in tempo reale.

Per contrastare la necessità di trasmettere grandi quantità di dati grezzi attraverso le reti, l'archiviazione dei dati e alcuni calcoli possono essere eseguiti su dispositivi più vicini all'utente o al sensore, noti appunto come "edge". L'edge computing include personal computer e smartphone oltre ai sistemi embedded (come quelli che comprendono l'IoT). Per rendere tutti questi dispositivi più intelligenti e meno dipendenti dai server back-end, ci rivolgiamo all'edge ML. Nella maggior parte dei casi, l'addestramento di un modello di ML è più impegnativo in

termini di calcolo rispetto all'esecuzione dell'inferenza. Di conseguenza, spesso ci affidiamo a potenti server per addestrare nuovi modelli. Ciò richiede la raccolta di dati sul campo per costruire un dataset e l'utilizzo di tale set di dati per addestrare il nostro modello, come mostrato in figura 3.21.

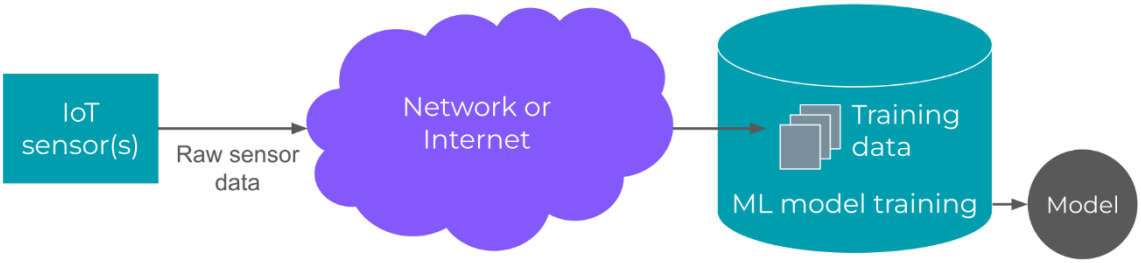


Figura 3.21 – Struttura per l'addestramento di un modello di ML

Una volta che abbiamo un modello addestrato, che è solo un modello matematico (sotto forma di una libreria software), possiamo distribuirlo al nostro sensore intelligente o ad un altro dispositivo edge. La logica su cui si basa la piattaforma Edge Impulse è proprio questa, essa mette a disposizione il tool per il progetto dell'algoritmo di ML e fornisce la potenza di calcolo dei propri server per effettuare il training, fornendo quindi un modello da implementare nel proprio dispositivo.

Edge Impulse mette a disposizione un ambiente di sviluppo molto intuitivo, anche per non addetti ai lavori o chi si approccia per la prima volta al machine learning. Sono supportati un gran numero di dispositivi, tra cui Arduino, Nordic Semiconductor, Raspberry, Synaptics, ma sono utilizzabili allo scopo anche smartphone o PC.

Andiamo ora a scoprire le diverse parti che compongono l'ambiente di sviluppo.

3.4.1 Dashboard

Dopo aver creato il progetto Edge Impulse Studio, si verrà indirizzati alla dashboard del progetto. La dashboard offre una rapida panoramica del progetto come l'ID progetto, il numero di dispositivi collegati, la quantità di dati raccolti, il metodo di etichettatura preferito, e altre proprietà modificabili. Si possono anche abilitare alcune funzionalità aggiuntive come la collaborazione, rendere pubblico il progetto e illustrare i tuoi progetti pubblici utilizzando i README di Markdown.

Le parti più interessanti di questa pagina sono:

Project info

Il widget delle informazioni sul progetto mostra le specifiche del progetto come l'ID progetto, il metodo di etichettatura e i calcoli della latenza per il dispositivo di destinazione (figura 3.22).

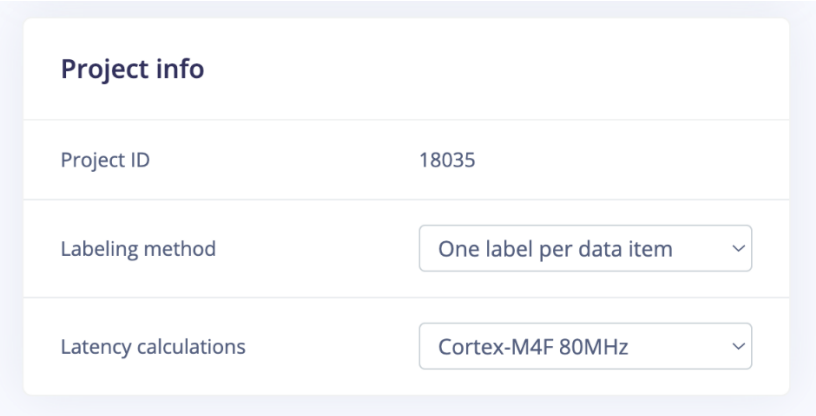


Figura 3.22 – Project info

Una funzione interessante è il componente di calcolo della latenza. Questo è un tempo approssimativo in millisecondi che il modello addestrato e le operazioni DSP impiegheranno durante l'inferenza in base al dispositivo di destinazione selezionato. Questo garantisce che le risorse di calcolo del dispositivo di destinazione non siano sottoutilizzate o sovrautilizzate. Inoltre, consente agli sviluppatori di risparmiare tempo associato a numerose iterazioni di inferenza avanti e indietro nello studio alla ricerca di modelli ottimali.

Block Outputs

Nella sezione Block Output si possono scaricare i risultati delle operazioni DSP e ML del proprio "impulso" (figura 3.23). Un **impulso** prende i dati grezzi e li suddivide in finestre più piccole; quindi, utilizza blocchi di elaborazione per estrarre delle features e di apprendimento per classificare i dati in ingresso. Le risorse scaricabili includono le funzionalità estratte, il modello Tensorflow salvato e modelli TensorFlow lite quantizzati e non quantizzati. Ciò è particolarmente

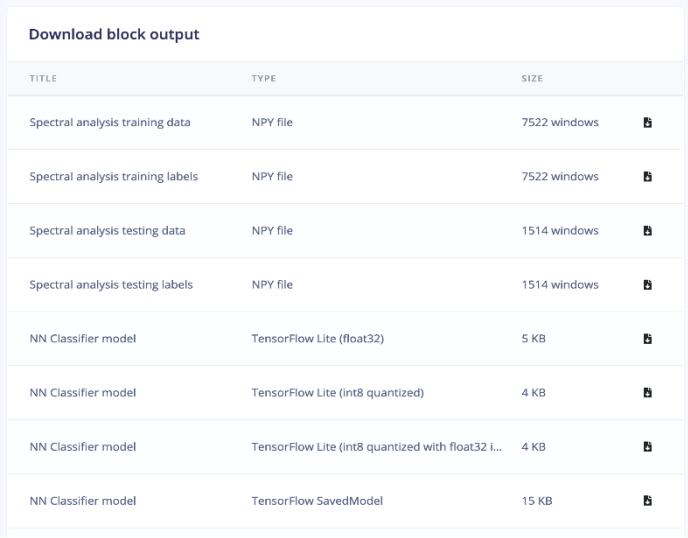


Figura 3.23 – Block Outputs

utile quando si desidera eseguire altre operazioni sui blocchi di output al di fuori di Edge Impulse Studio.

Danger Zone

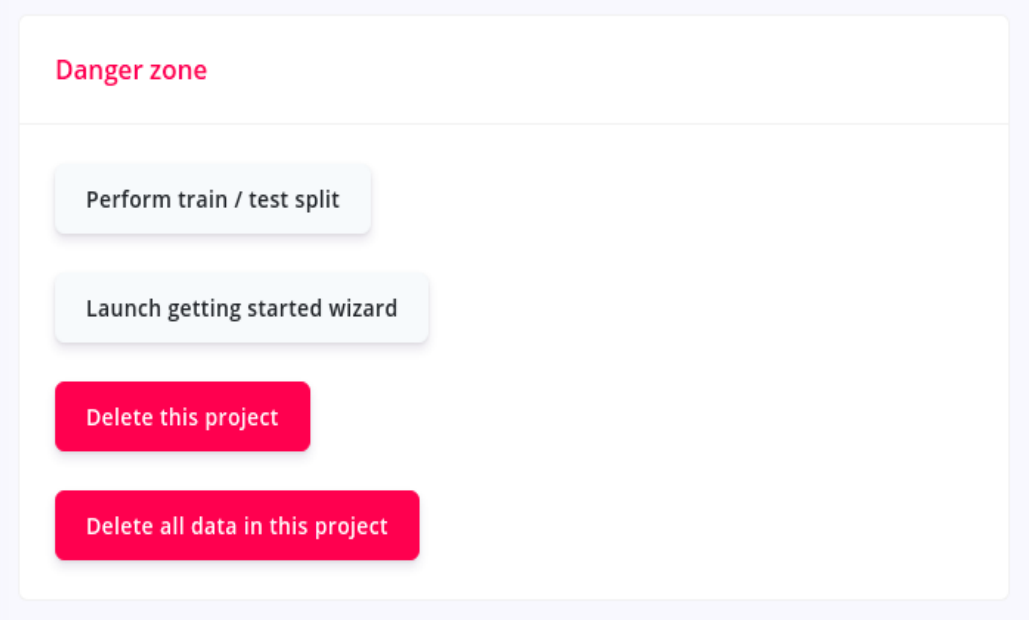


Figura 3.24 – Danger zone

In questo caso (figura 3.24) il tasto che ci interessa è quello train/test split. Questa azione ribilancia il dataset suddividendo automaticamente tutti i dati tra training set e testing set e reimposta le categorie per tutti i dati.

3.4.2 Devices

Nella scheda Dispositivi si trova un elenco di tutti i dispositivi connessi e una guida su come connettere nuovi dispositivi attualmente supportati da Edge Impulse (figura 3.25).

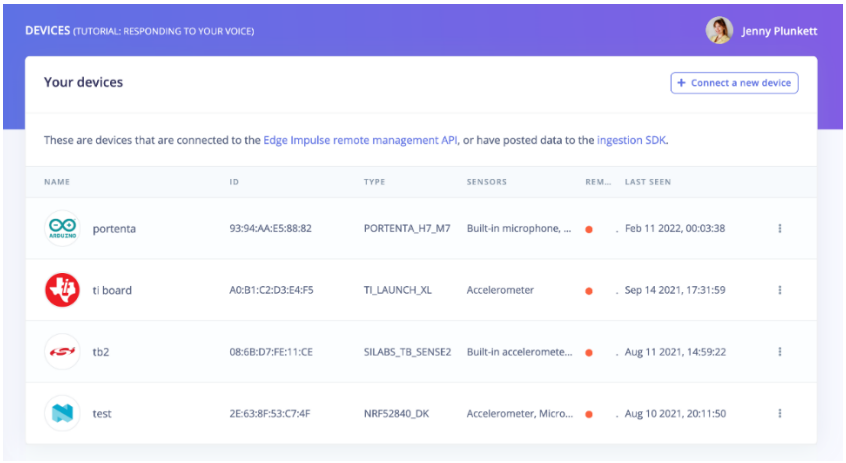


Figura 3.25 - Devices

3.4.3 Data acquisition

Tutti i dati raccolti per ogni progetto possono essere visualizzati in questa scheda (figura 3.26). Possiamo vedere come i dati sono stati suddivisi per train/test set, nonché la distribuzione dei dati per ciascuna classe nel dataset. Si possono anche inviare nuovi dati del sensore al progetto tramite caricamento file, WebUSB, Edge Impulse API o Edge Impulse CLI [3].

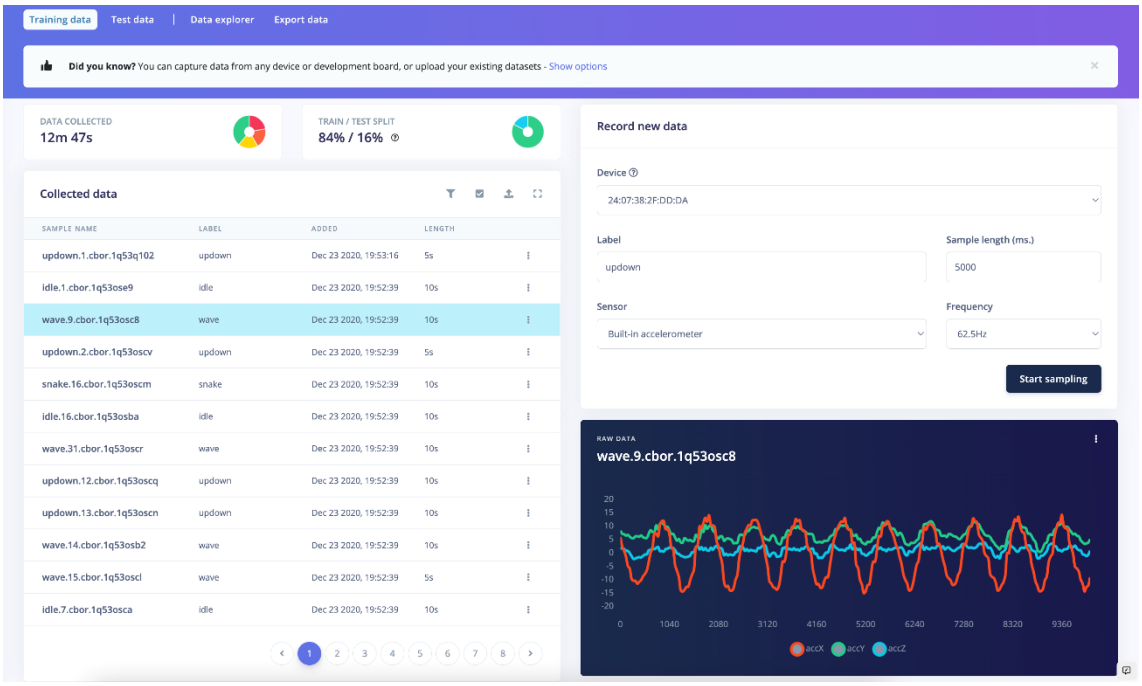


Figura 3.26 – Data acquisition

Aggiungere dati al progetto

Il pannello sulla destra permette di raccogliere dati direttamente da ogni piattaforma supportata attraverso:

- WebUSB
- Edge Impulse CLI daemon

Il WebUSB e l'Edge Impulse daemon funzionano con qualsiasi dispositivo completamente supportato eseguendo il flashing del firmware Edge Impulse sulla scheda (per approfondimento si rimanda al sito Edge impulse [3]).

Rapporto train/test del dataset

La suddivisione train/test è una tecnica per l'addestramento e la valutazione delle prestazioni di algoritmi di ML (figura 3.27). Indica come i dati vengono suddivisi tra i campioni di training e di test. Ad esempio, una divisione 80/20 indica che l'80% del dataset viene utilizzato per scopi di addestramento del modello mentre il 20% viene utilizzato per il test del modello. Questa sezione

mostra anche come vengono distribuiti i campioni di dati in ogni classe per prevenire set di dati sbilanciati che potrebbero introdurre bias durante l'addestramento del modello.

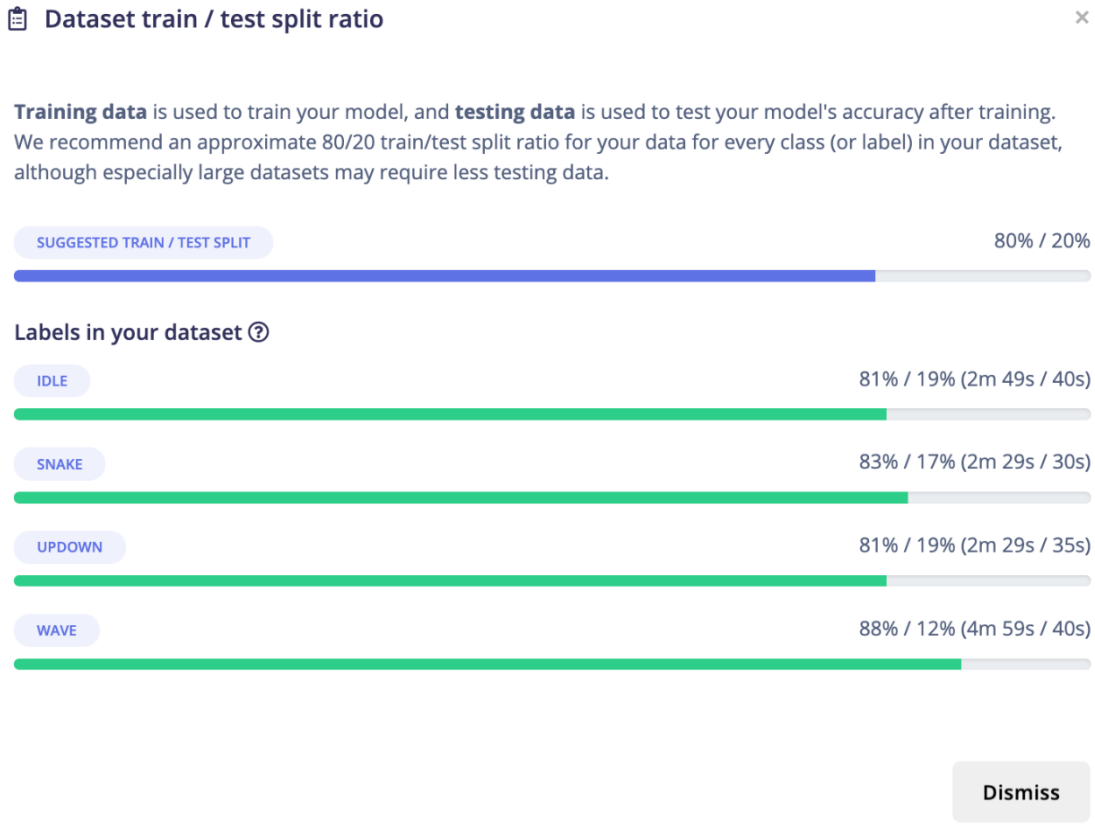


Figura 3.27 – Rapporto train/test del dataset

Dividere un campione

Una funzione che risulta molto utile durante la costruzione del dataset è quella che permette di dividere un campione, selezionare cioè parti di un campione che ci interessano di più (figura 3.28). Questa operazione è svolta automaticamente. Possiamo eseguire un movimento ripetutamente, o pronunciare una parola chiave più e più volte, e gli eventi vengono rilevati e possono essere memorizzati come singoli campioni. Ciò semplifica e rende molto più veloce la costruzione di un dataset di alta qualità di eventi discreti. Per farlo, si va su Acquisizione dati, si registrano alcuni nuovi dati, si fa clic e si seleziona Dividi campione. È possibile impostare la lunghezza della finestra e tutti gli eventi vengono rilevati automaticamente. Se si sta dividendo dati audio si può anche ascoltare gli eventi facendo clic sulla finestra, il lettore audio viene automaticamente popolato con quella divisione specifica.

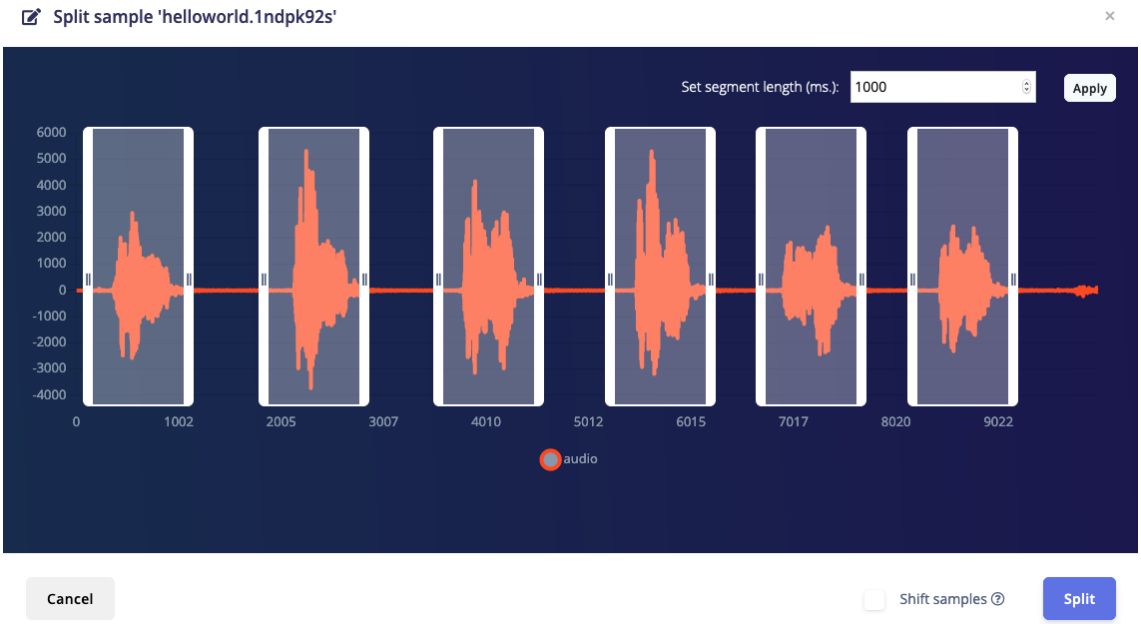


Figura 3.28 – funzione Split

I campioni vengono automaticamente centrati nella finestra, il che potrebbe causare problemi su alcuni modelli (la rete neurale potrebbe apprendere una scorciatoia in cui i dati al centro della finestra sono sempre associati a una determinata etichetta), quindi si può selezionare "Sposta campioni" per spostare automaticamente i dati un po' casualmente.

3.4.4 Data explorer

Il data explorer è uno strumento visivo per esplorare il dataset, trovare valori anomali o dati etichettati in modo errato e per aiutare a etichettare i dati senza etichetta (figura 3.29). L'explorer cerca prima di estrarre caratteristiche significative dai dati (attraverso l'elaborazione del segnale e l'embedding di reti neurali) e quindi utilizza un algoritmo di riduzione della dimensionalità per mappare queste caratteristiche in uno spazio 2D. Questo offre una panoramica ampia del dataset completo.

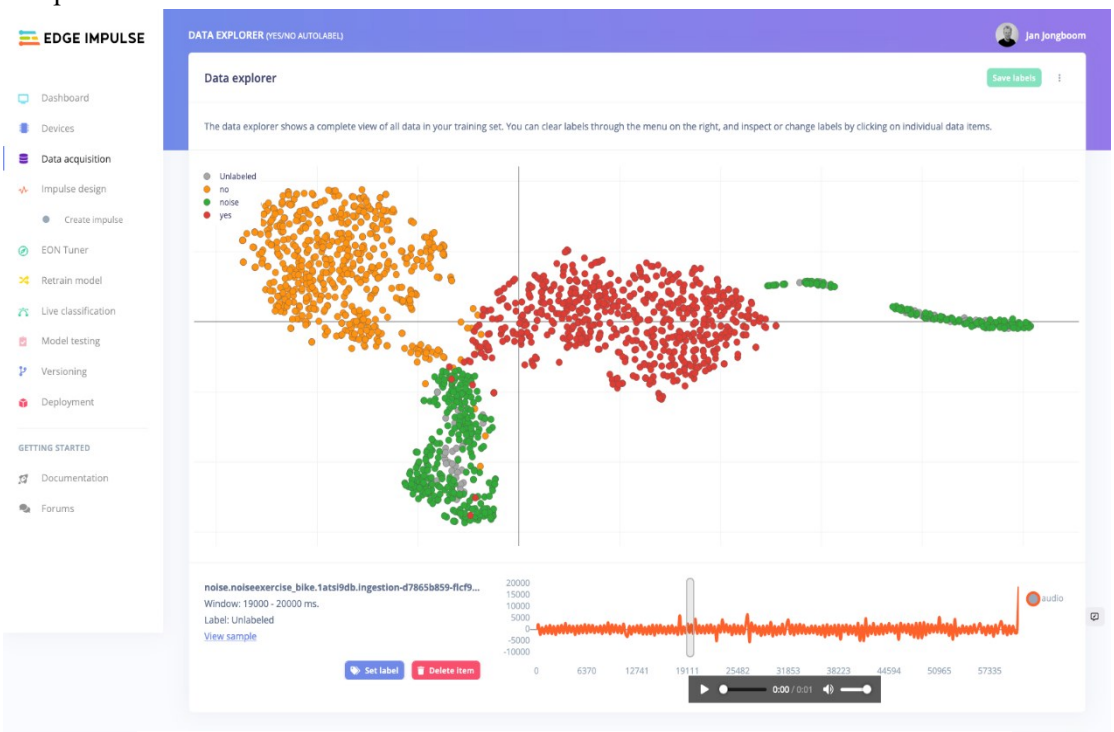


Figura 3.29 – Data explorer

3.4.5 Impulse Design

Questa schermata se vogliamo è il cuore di questa piattaforma, quella in cui andiamo a definire il modo in cui sarà strutturato il nostro algoritmo. Dopo aver raccolto i dati del progetto, ora possiamo creare il nostro impulso. Un impulso completo consisterà di tre blocchi di costruzione principali: blocco di input, blocco di elaborazione e un blocco di apprendimento. La figura 3.30 ci dà un esempio di impulso per la classificazione di movimenti tramite l'uso dei dati di un accelerometro.

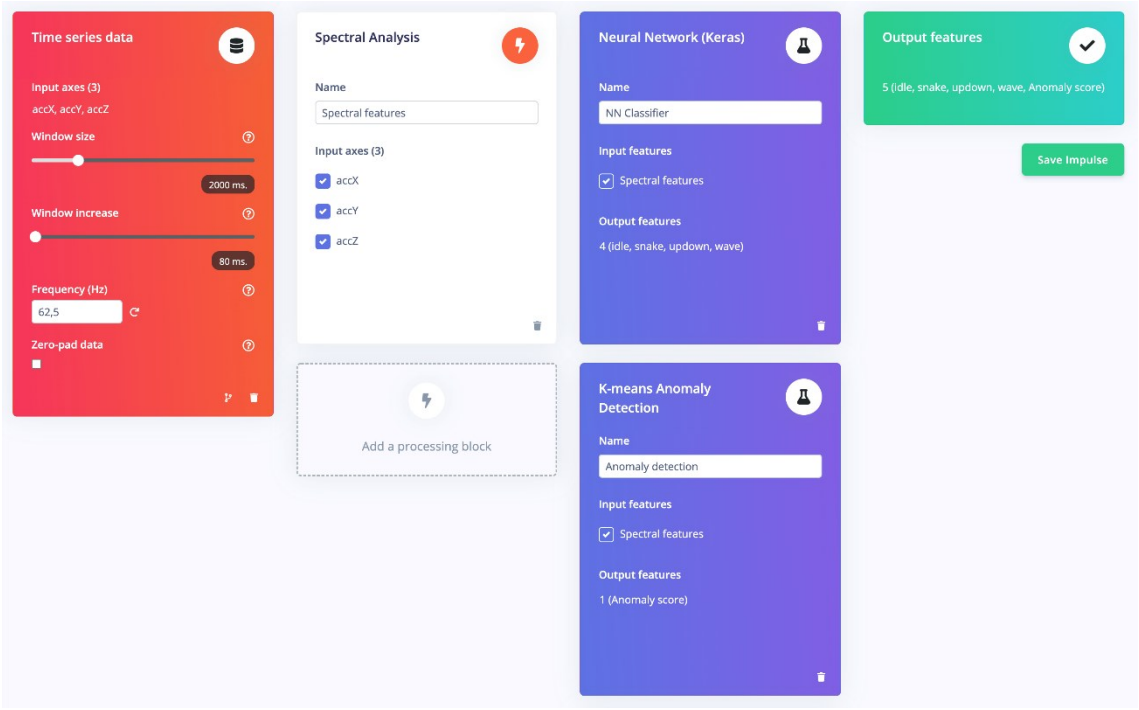


Figura 3.30 – Impulse Design

Blocco di Input

Il blocco di input indica il tipo di dati di input con cui si sta addestrando il modello. Possono essere serie temporali (audio, vibrazioni, movimenti) o immagini.

- Il campo degli **assi di input** elenca tutti gli assi a cui si fa riferimento dal dataset di training.
- La **dimensione della finestra** è la dimensione delle features grezze utilizzate per il training.
- L'**incremento della finestra** viene utilizzato per creare artificialmente più funzionalità (e alimentare il blocco di apprendimento con più informazioni).

In figura 3.31 il grafico vuole racchiudere il ruolo di ogni parametro

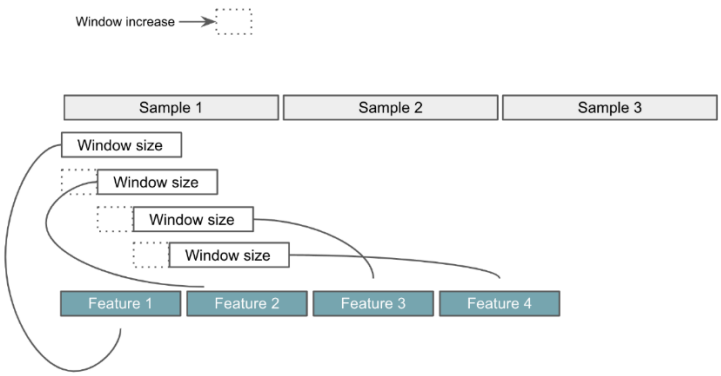


Figura 3.31 – Grafico descrittivo del ruolo dei parametri di un impulso

Blocchi di Elaborazione

Un blocco di elaborazione è fondamentalmente un estrattore di features. Consiste in operazioni DSP (Digital Signal Processing) utilizzate per estrarre funzionalità su cui il modello apprende. Queste operazioni variano a seconda del tipo di dati utilizzati nel progetto (figura 3.32).

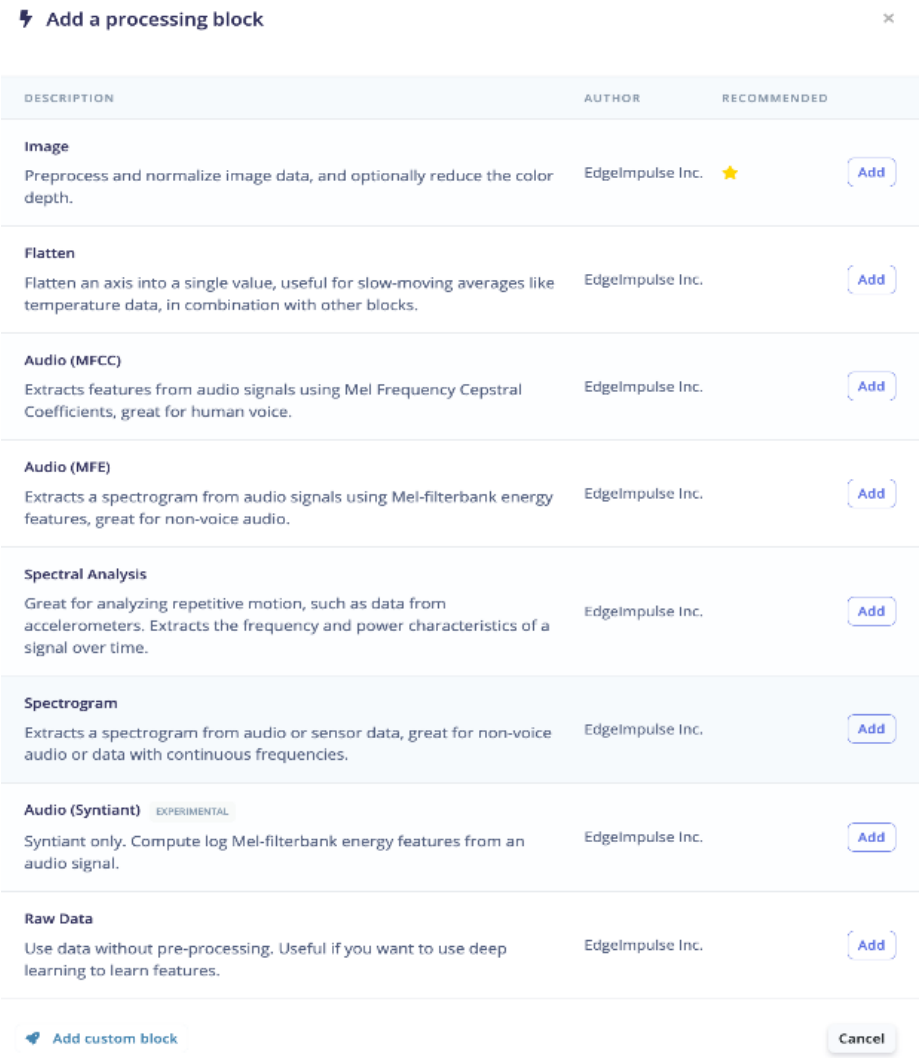


Figura 3.32 – Blocchi di elaborazione

Blocchi di Apprendimento

Un blocco di apprendimento è semplicemente una rete neurale addestrata per apprendere dai dati. I blocchi di apprendimento variano in base a ciò che si desidera che il modello faccia e al tipo di dati nel dataset di addestramento (figura 3.33). Gli algoritmi includono: classificazione, regressione, rilevamento di anomalie, transfer learning di immagini, rilevamento di parole chiave o rilevamento di oggetti.

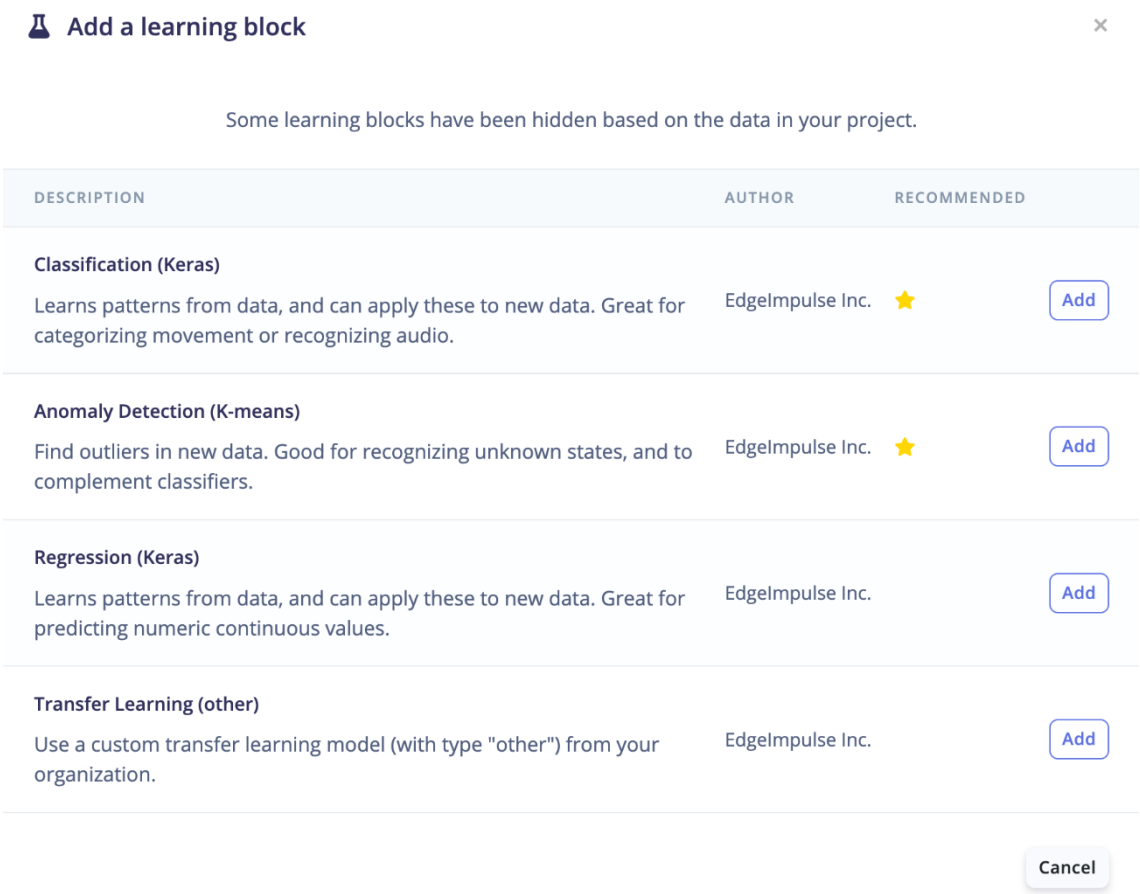


Figura 3.33 – Blocchi di apprendimento

Una volta creato l'impulso si andrà a specificare più nel dettaglio i parametri di ogni blocco e ad allenare la rete, a questo scopo Edge Impulse offre un insieme di tool che aiutano lo sviluppatore a farsi un'idea delle prestazioni della rete, capire come impostare al meglio i parametri e ad ottimizzare quest'ultima. Si vedranno più nel dettaglio questi aspetti nel capitolo successivo dedicato agli esperimenti.

3.4.6 Model Testing

Quando si raccolgono i dati, si divide il dataset in set di training e test. Il modello è stato addestrato solo con il set di training e il set di test viene utilizzato per convalidare/verificare le prestazioni del modello su dati che questo non ha mai visto. Ciò garantirà che il modello non abbia imparato a sovraadattarsi (overfitting) ai dati di training, il che è un evento comune. Cliccando su Test all il modello classificherà tutti i campioni del test set e fornirà un'accuratezza complessiva delle prestazioni del modello (figura 3.34).

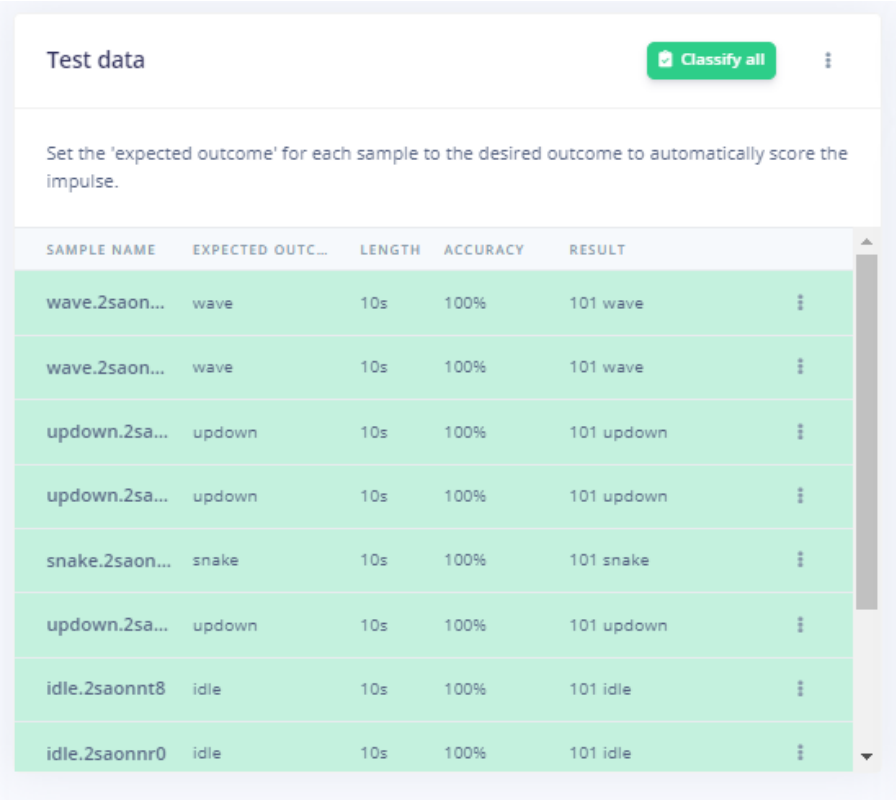


Figura 3.34 – Model Testing

Questa è anche accompagnata da una *confusion matrix* per mostrare come si comportano i modelli per ogni classe (figura 3.35). Per vedere una classificazione in dettaglio, si va al singolo campione che si desidera valutare e si clicca sui tre punti accanto ad esso, quindi si seleziona “mostra classificazione”. Si aprirà una nuova finestra che visualizzerà il risultato atteso e l'output previsto del modello con la sua precisione. Questa visualizzazione dettagliata può anche dare un suggerimento sul motivo per cui un elemento è stato classificato erroneamente (figura 3.36). Ogni blocco di apprendimento ha una soglia. Questa può essere la confidenza minima che una rete neurale deve avere o il punteggio massimo di anomalia prima che un campione venga

contrassegnato come anomalia. Si possono configurare queste soglie per modificare la sensibilità di questi blocchi di apprendimento.

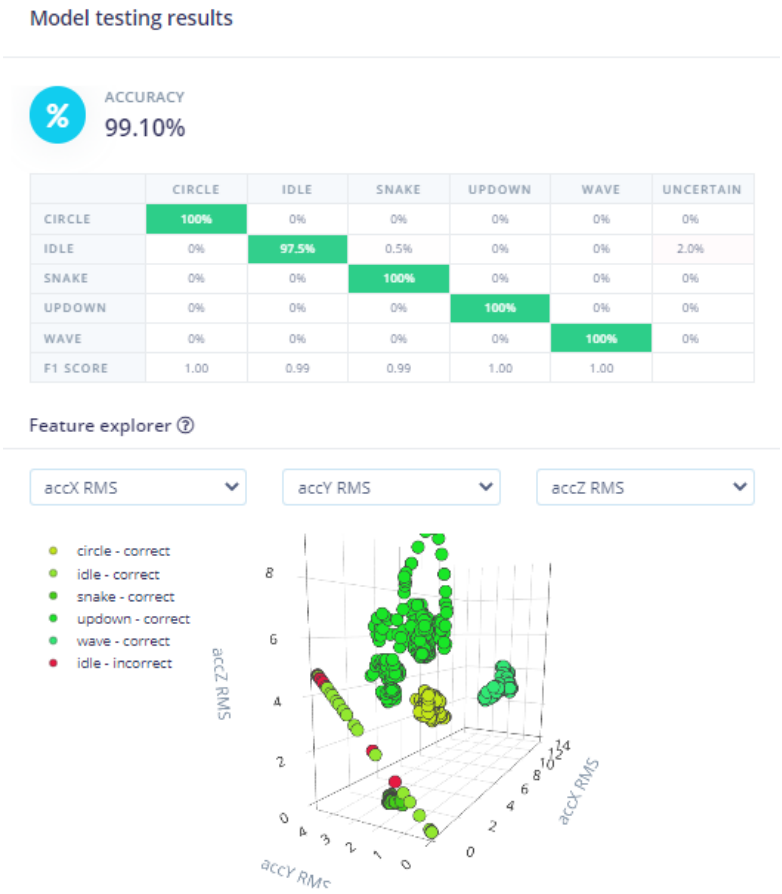


Figura 3.35 – Risultato di un Model testing

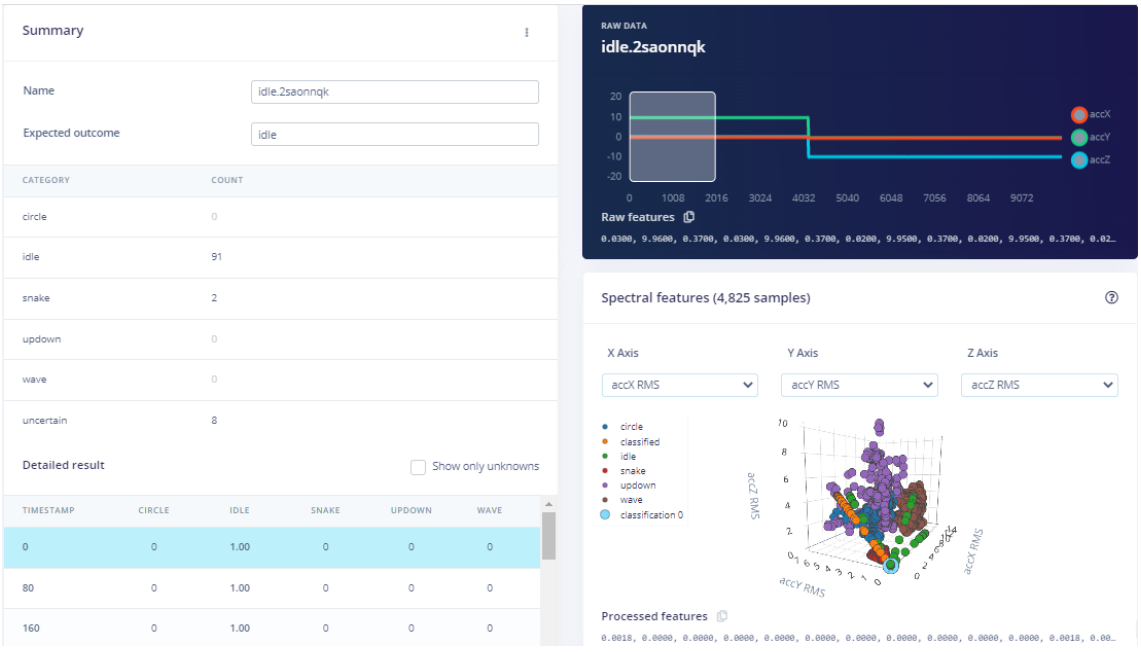


Figura 3.36 – Pagina mostra classificazione del Model testing

Live classification

La classificazione in tempo reale consente di convalidare il modello con i dati acquisiti direttamente da qualsiasi dispositivo o scheda di sviluppo supportata. Questo dà un'immagine delle prestazioni del modello con i dati del mondo reale (figura 3.37).

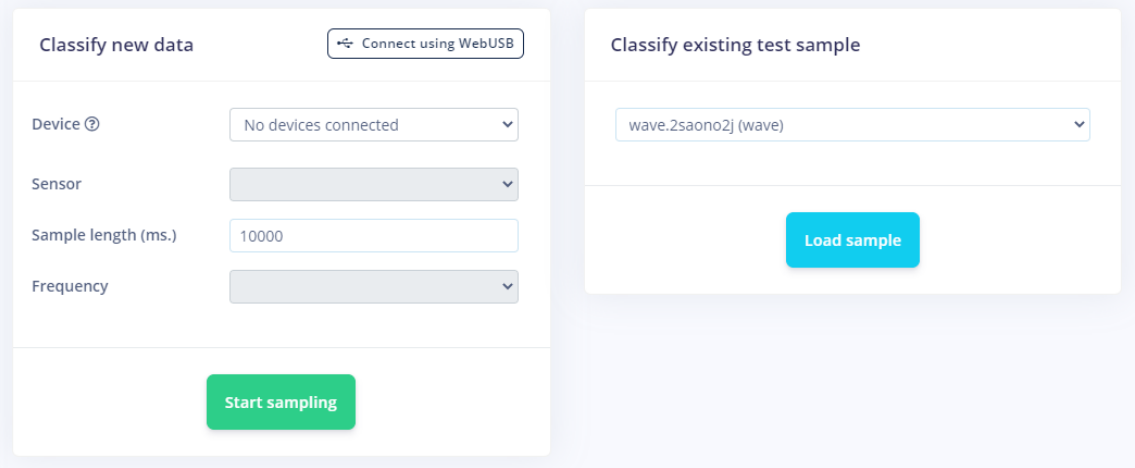


Figura 3.37 – Live classification

3.4.7 Deployment

Dopo aver addestrato e validato il modello è possibile implementarlo su qualsiasi dispositivo. Ciò fa funzionare il modello senza una connessione Internet, riduce al minimo la latenza e funziona con un consumo energetico minimo.

La pagina distribuzione comprende una varietà di opzioni di distribuzione tra cui scegliere a seconda del dispositivo di destinazione. Indipendentemente dal fatto che si stia utilizzando una scheda di sviluppo completamente supportata o meno, Edge Impulse fornisce opzioni di distribuzione tramite la libreria C++ che è possibile utilizzare per distribuire il modello su qualsiasi destinazione (a condizione che la destinazione disponga di una potenza di calcolo sufficiente per gestire l'attività).

Di seguito sono elencate le 4 categorie principali di opzioni di distribuzione attualmente supportate da Edge Impulse:

- Distribuisci come libreria personalizzabile.
- Distribuisci come pre-built firmware.
- Esegui direttamente su telefono o computer.
- Usa Edge Impulse per Linux per destinazioni Linux.

La prima opzione di distribuzione consente di trasformare l'impulso in un codice sorgente completamente ottimizzato che può essere ulteriormente personalizzato e integrato con la propria applicazione (figura 3.38). Questa opzione supporta le seguenti librerie (per il momento):

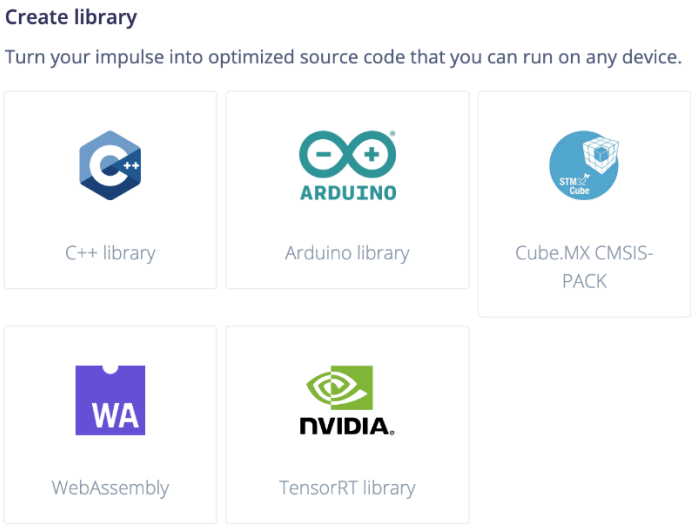


Figura 3.38 – Creazione di una libreria personalizzabile dell'impulso

Ad esempio, nel caso di interesse (libreria Arduino), è possibile eseguire l'impulso localmente generando una libreria Arduino che contiene tutti i blocchi definiti in precedenza, che è possibile importare come uno sketch nell'IDE ed eseguire.

La seconda opzione permette di utilizzare un file binario pronto per l'uso per una specifica scheda di sviluppo che raggruppa blocchi di elaborazione del segnale, configurazione e blocchi di apprendimento in un unico pacchetto. Questa opzione è attualmente disponibile solo per le schede di sviluppo completamente supportate, tra cui la scheda Arduino, come mostrato nell'immagine seguente (figura 3.39):

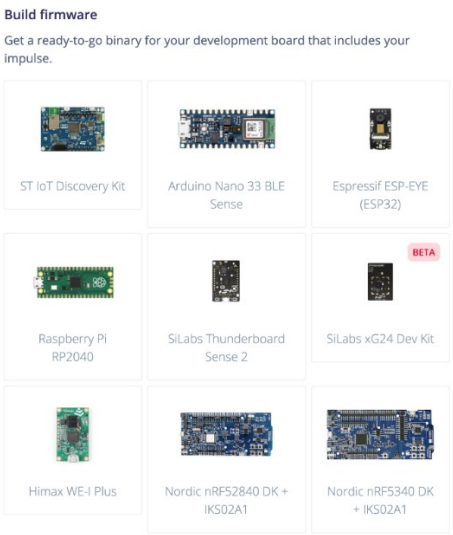


Figura 3.39 – Creazione di un file binario dell'impulso

Per distribuire il modello utilizzando il file binario, si seleziona il dispositivo di destinazione e si clicca su "Build". Si carica il firmware scaricato sul dispositivo, quindi si esegue il comando seguente:

```
edge-impulse-run-impulse
```

L'impulse runner mostra i risultati dell'impulso in esecuzione sulla scheda di sviluppo.
Quando si costruisce l'impulso per la distribuzione, Edge Impulse offre la possibilità di aggiungere un altro livello di ottimizzazione all'impulso utilizzando il compilatore EON (figura 3.40). Il compilatore EON consente di eseguire reti neurali con il 25-55% di RAM in meno e fino al 35% di flash in meno, pur mantenendo la stessa precisione, rispetto a TensorFlow Lite per microcontrollori.
Per avere un'idea di come l'impulso utilizzerà le risorse di calcolo del dispositivo di destinazione, Edge Impulse fornisce anche una stima di latenza, flash e RAM che l'impulso consumerà dal dispositivo di destinazione anche prima di distribuirlo localmente. Questo può davvero far risparmiare parecchio tempo di progettazione dovuto ad eventuali iterazioni ricorrenti ed esperimenti.

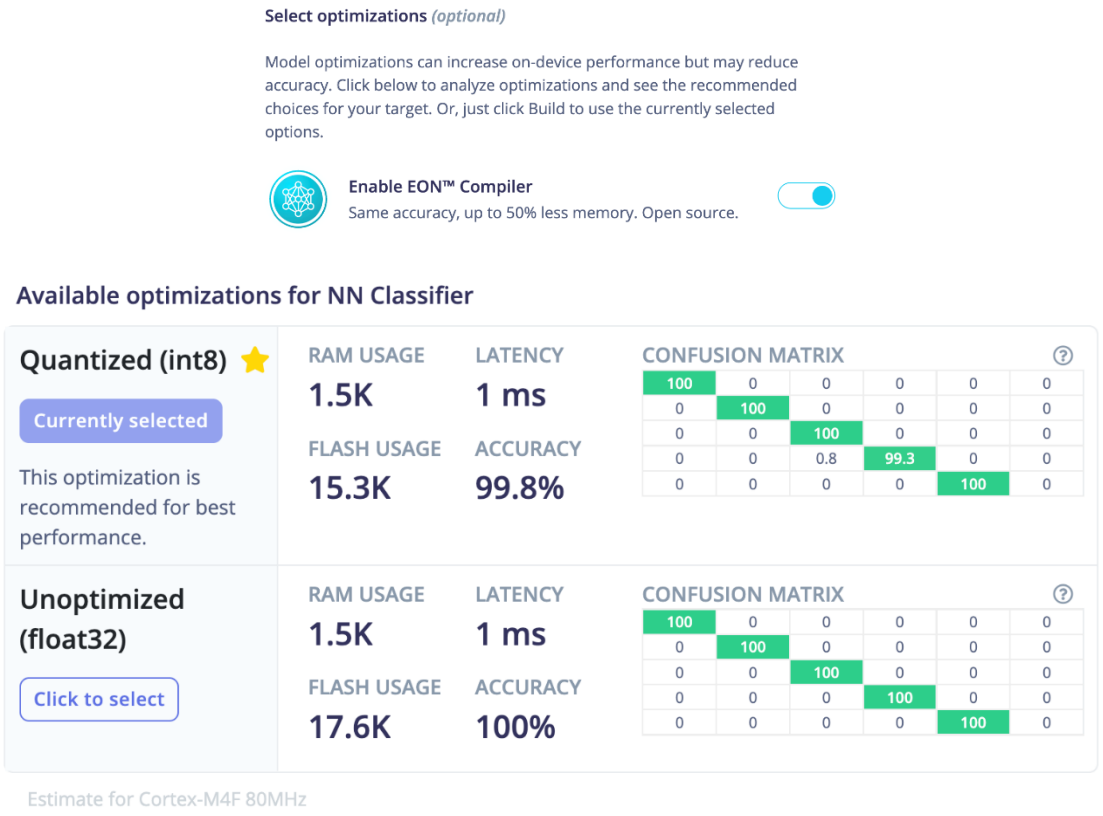


Figura 3.40 – Compilatore EON

È possibile anche selezionare se eseguire i modelli float32 non quantizzati o int8 quantizzati. I primi permettono un'accuratezza migliore dei risultati ma di contro richiedono una quantità di risorse più elevata, con una latenza mediamente maggiore; i secondi all'opposto richiedono meno risorse e hanno una latenza inferiore ma arrivano mediamente ad un'accuratezza più bassa. Le valutazioni vanno fatte nel caso specifico. La confusion matrix di cui sopra si basa solo sui dati di test per aiutare a sapere come si comporta il modello su dati del mondo reale sconosciuti. Può anche aiutare a dedurre se il modello si sia sovraadattato ai dati di training.

Una volta effettuate queste scelte si è pronti per caricare il modello sulla scheda e verificare quanto (tramite i tool di Edge impulse o altro) le prestazioni di questo si avvicinino a quelle teoriche che ci si aspetta.

Capitolo 4

Prove Sperimentali

4.1 Classificazione Vocale

4.1.1 Panoramica

In questo progetto si utilizzerà il machine learning per creare un sistema in grado di riconoscere eventi acustici, in particolare il parlato, attraverso la classificazione audio. Il sistema creato funzionerà in modo simile a "Hey Siri" o "OK, Google" e sarà in grado di riconoscere parole chiave o altri eventi acustici, anche in presenza di altri rumori di fondo o dialoghi di sottofondo. Si assocerà ad ogni parola l'accensione di un led in modo da avere un riscontro visivo della classificazione e un'azione da svolgere. In questo caso le parole chiave saranno YES e NO; alla prima si assocerà l'accensione del led di colore verde, alla seconda del rosso (con poca fantasia, ma è intuitivo). Inoltre, si avrà una label per il rumore di sottofondo e una per un set di parole casuali che dovranno essere etichettate come "sconosciute"; alla prima si assocerà l'accensione del led bianco, alle seconde quella del led blu. Oltre a ciò, al riconoscimento della parola YES l'Arduino dovrà restituire il valore misurato della temperatura, mentre al riconoscimento della parola NO il valore misurato di umidità. Per fare questo si sfrutteranno i sensori integrati che offre il NANO 33 BLE SENSE, cioè il microfono MP34DT05 ed il sensore di temperatura e umidità HTS221. In più, come base dati, si sfrutterà un ampio dataset messo a disposizione da Google per il progetto di reti neurali di questo tipo, con campioni di eventi acustici che sono un sottogruppo di quelli utilizzati da Google stessa per l'allenamento delle proprie reti neurali. Le parole comprendono uno spettro molto eterogeneo di campioni, essendo pronunciate da uomini, donne, ragazzi, ragazze, più o meno giovani ed in ambienti diversi, e nel database è compreso un insieme di tanti rumori di sottofondo, presi da ambienti vari [4].
Si vedrà quindi come raccogliere dati audio dal microfono a bordo della scheda NANO 33 BLE SENSE, utilizzare l'elaborazione del segnale per estrarre le informazioni più importanti e addestrare una deep neural network in grado di decidere se in una determinata clip audio è stata

riconosciuta una certa parola chiave. Infine, si caricherà il classificatore sul dispositivo embedded e si valuteranno le sue prestazioni.

4.1.2 Esperimento

Quando si va a progettare una classificazione di questo tipo bisogna tenere presente che alcune parole chiave sono più difficili da distinguere di altre, e in particolare le parole chiave con una sola sillaba (come "One") potrebbero portare a falsi positivi (ad es. quando si dice "Gone"). Questo è il motivo per cui Apple, Google e Amazon utilizzano tutte parole chiave di almeno tre sillabe ("Hey Siri", "OK, Google", "Alexa"). In questo caso Google le ha scelte per noi, ma è un fattore a cui prestare attenzione.

Una volta collegato Arduino alla piattaforma (tramite i tool messi a disposizione da Edge Impulse, cioè l'Edge Impulse CLI [6]) si entra nella facciata **Data acquisition.** A questo punto si hanno due possibilità:

1. *Raccogliere i dati dalla scheda*: in questo caso si va ad impostare la parola chiave come etichetta, la lunghezza del campione su 10 s, il sensore su "microfono" e la frequenza su 16 kHz. Quindi si fa clic su **Start sampling** e si inizia a pronunciare la parola chiave più e più volte (con pause nel mezzo). Viene raccolto allora un numero di campioni abbastanza elevato (più elevato è, meglio è, almeno 10 minuti per ciascuna label) per ogni label, cercando di avere campioni più eterogenei possibile; vuol dire ad esempio pronunciare le parole con cadenza diversa, velocità diversa, far pronunciare le parole a persone diverse ecc…, raccogliere rumori di fondo in diversi ambienti e così via. Una volta finito ci si ritroverà con un certo numero di campioni con andamento simile a quello in figura 4.1, in cui si vedono chiaramente le parole separate da "silenzio".

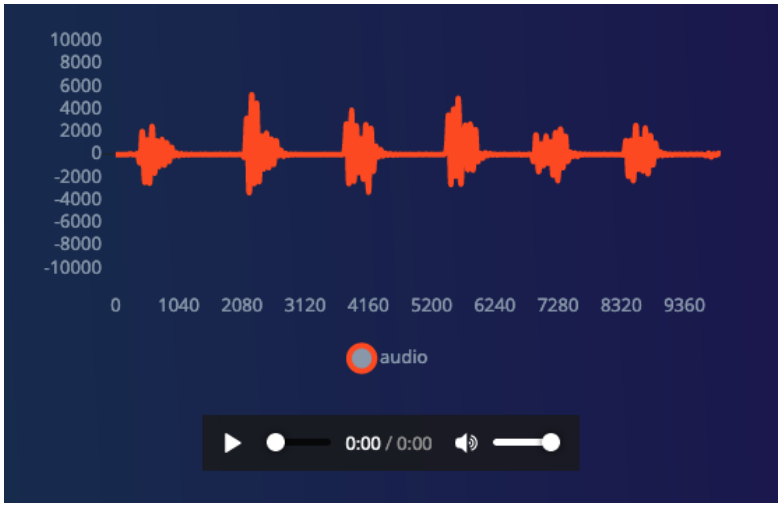


Figura 4.1 – Visualizzazione del campione acustico raccolto

Tuttavia, questi dati non sono ancora ideali per il Machine Learning. Dovranno essere ritagliate le parti in cui si trova parola chiave. Questo è importante perché quello che si vuole è che solo la parola chiave effettiva sia etichettata come tale e non accidentalmente il rumore o parole incomplete. Fortunatamente Edge Impulse ha la funzione Split che ci aiuta, permettendo di decidere la lunghezza della finestra e ritagliare le parti che ci interessano. Inoltre, si va a selezionare il pulsante *Shift samples* che centra le finestre in maniera random intorno al campione, in modo da avere una certa eterogeneità sulla centratura dei campioni (figura 4.2).

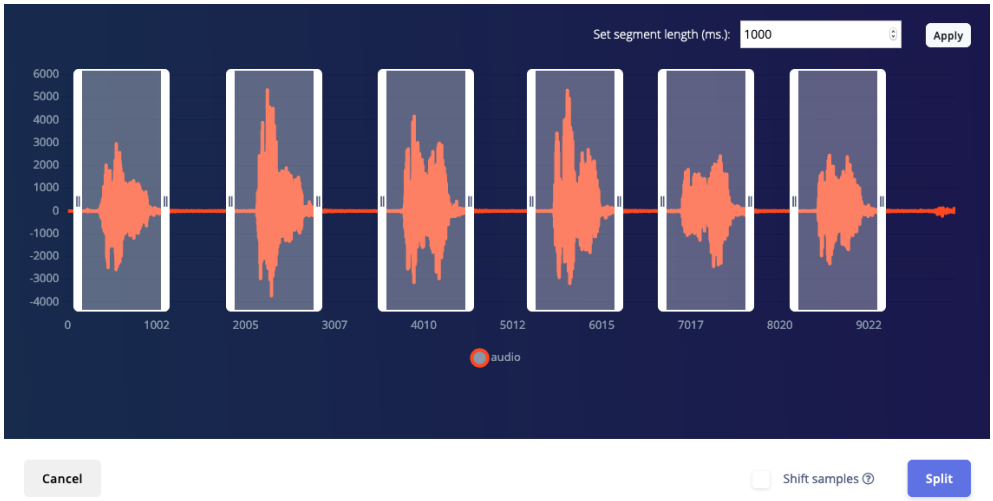
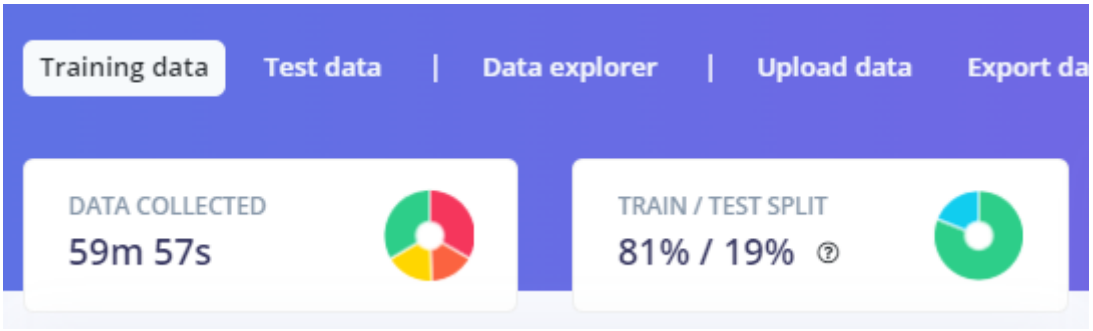


Figura 4.2 – Funzione Split

In questo caso si prendono finestre di durata 1 s.

2. Oppure, come nel progetto specifico *caricare i dati direttamente da file* (ottenuto ad esempio dal tool CoolTerm descritto nel capitolo 1) si può scegliere in che modo dividere i dati tra training e test e assegnare un'etichetta a ciascun campione.

A questo punto si effettua un ribilanciamento dei campioni raccolti tra training e test tramite il pulsante apposito, e ci si troverà in una situazione simile a quella mostrata in figura 4.3:



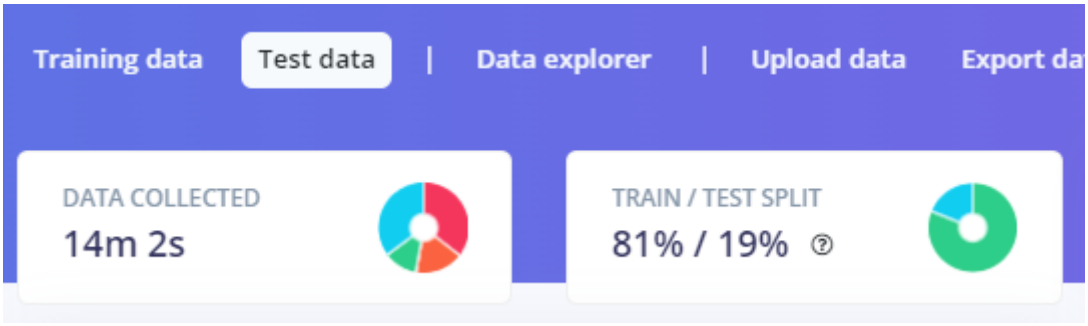


Figura 4.3 - Dataset

Sistemato il dataset, si può progettare l'impulso. Un impulso prende i dati grezzi, li divide in finestre più piccole, utilizza i blocchi di elaborazione del segnale per estrarre le features e quindi utilizza un blocco di apprendimento per classificare i nuovi dati. I blocchi di elaborazione del segnale restituiscono sempre gli stessi valori per lo stesso input e vengono utilizzati per semplificare l'elaborazione dei dati grezzi, mentre i blocchi di apprendimento apprendono dalle esperienze passate.

Per questo progetto è stato usato il blocco di elaborazione del segnale "MFCC". MFCC sta per Mel Frequency Cepstral Coefficients. Fondamentalmente è un modo per trasformare l'audio grezzo, che contiene una grande quantità di informazioni ridondanti, in una forma semplificata di più facile utilizzo. Questo risulta ottimo per il parlato umano. In dettaglio, le estrazioni delle features aggiungono un ulteriore passaggio al blocco MFE (che si può approfondire sul sito di Edge impulse [5]) dando come risultato una rappresentazione compressa dei banchi di filtri. Una trasformazione coseno discreta viene applicata su ciascun banco di filtri per estrarre i coefficienti

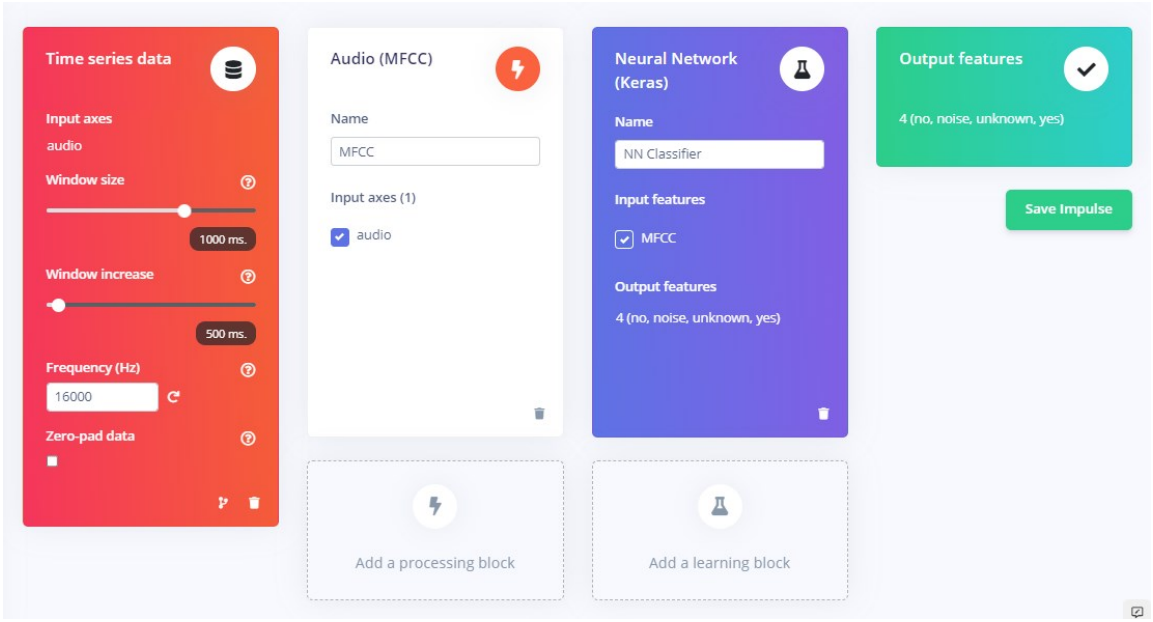


Figura 4.4 – Design impulso

cepstrali. Di solito vengono mantenuti 13 coefficienti, il resto viene scartato in quanto rappresenta cambiamenti rapidi non utili per il riconoscimento vocale. Si passano quindi questi dati audio semplificati al blocco di rete neurale Keras, che imparerà a distinguere tra le quattro classi di audio, lo si vedrà tra poco. La figura 4.4 mostra il design dell'impulso.

Assemblati gli elementi costitutivi dell'impulso, è il momento di configurarne ogni singola parte. Si entra nella scheda MFCC nel menù di navigazione a sinistra. Si vedrà una pagina simile a quella in figura 4.5:

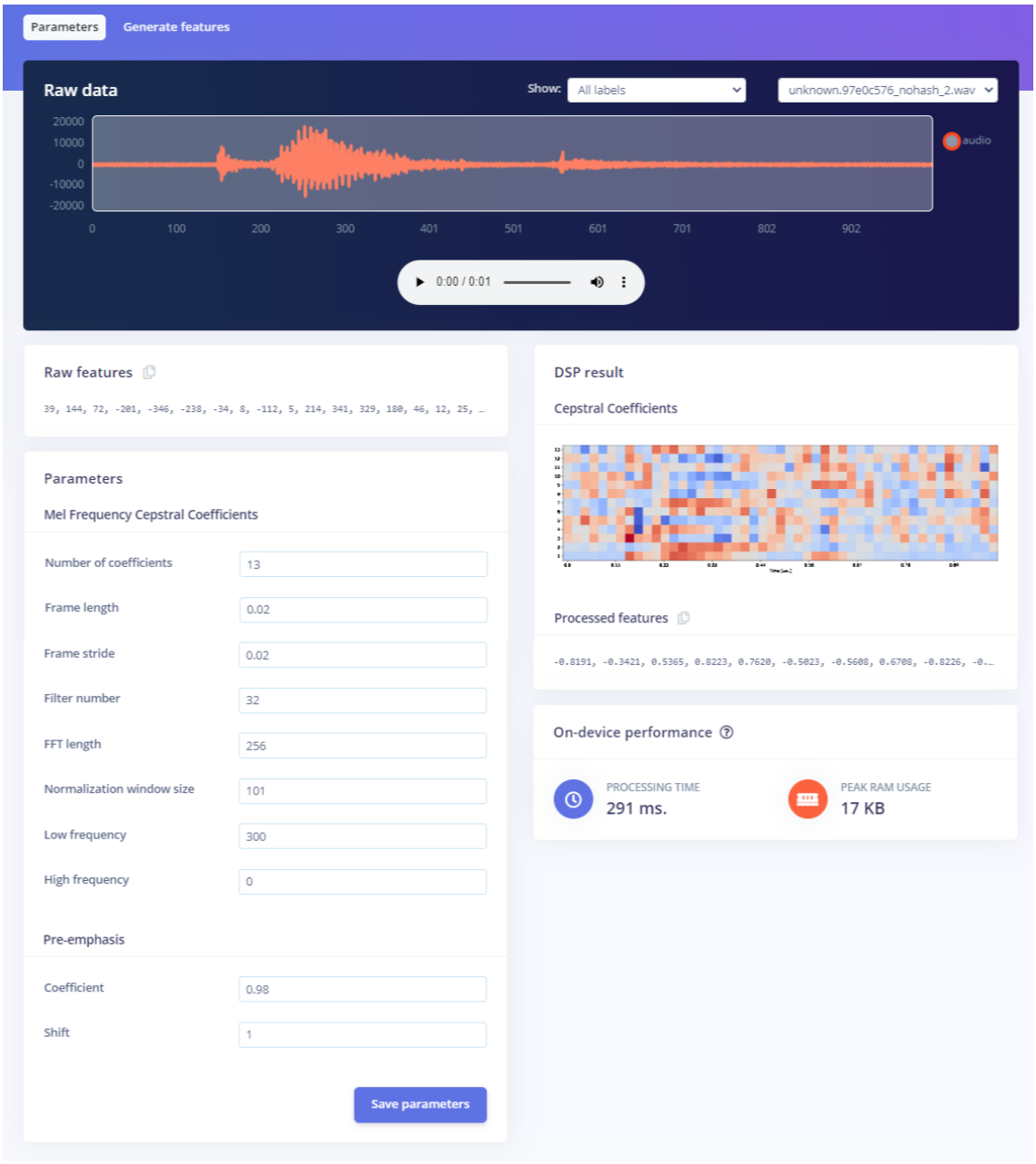


Figura 4.5 - MFCC

Questa pagina consente di configurare il blocco MFCC e consente di visualizzare in anteprima come verranno trasformati i dati. La parte destra della pagina mostra una visualizzazione dell'output dell'MFCC per un brano audio, che è uno spettrogramma. Nella parte bassa (figura 4.6) si possono vedere le performance previste per il blocco MFCC nel microcontrollore.

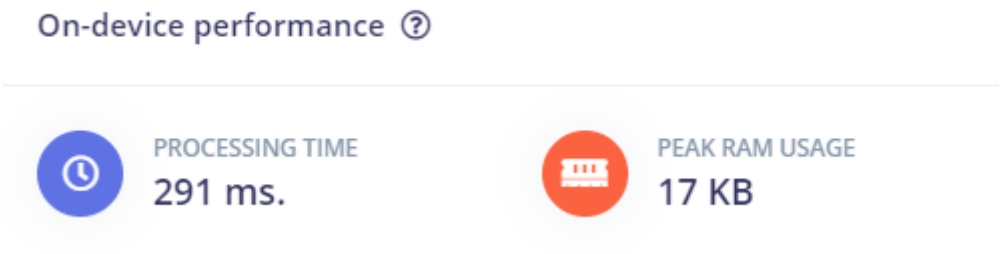


Figura 4.6 – On-device performance

Gli spettrogrammi generati dal blocco MFCC verranno inviati ad un'architettura di rete neurale particolarmente efficace nell'imparare a riconoscere i modelli in questo tipo di dati tabulari. Prima di addestrare la rete neurale, si devono generare blocchi MFCC per tutte le finestre audio. Per fare ciò, si clicca sul pulsante *Save parameters* che porterà automaticamente nella pagina **Generate features**; quindi, si clicca sul pulsante *Generate features*. Questo richiederà circa un minuto per essere completato.

Successivamente appare una delle funzionalità più utili in Edge Impulse: il **feature explorer** (figura 4.7).

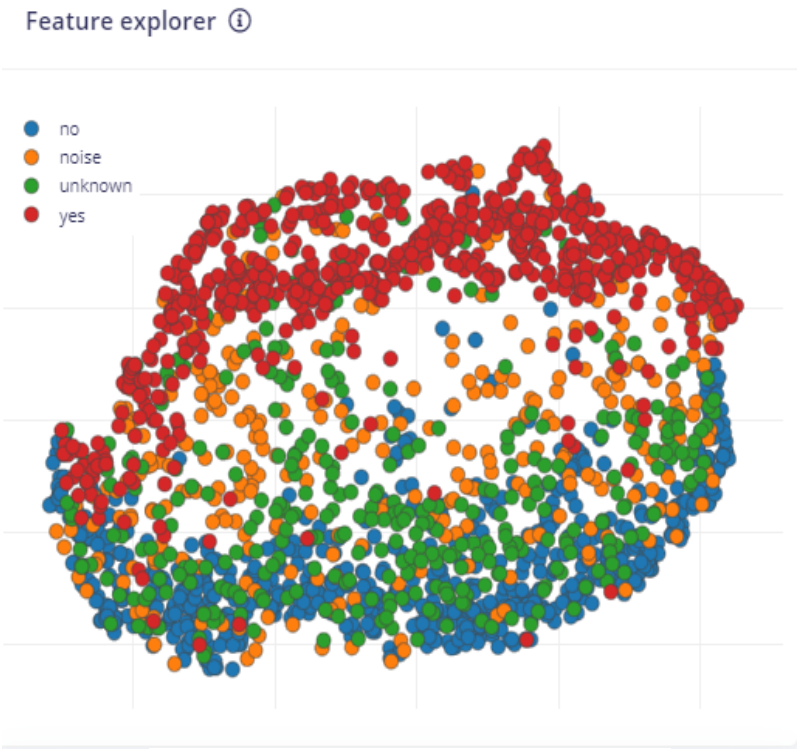


Figura 4.7 – Feature explorer

Questa è una rappresentazione 2D che mostra il dataset completo, con ogni elemento codificato con colori nella rispettiva etichetta. Permette di ingrandire ogni elemento, trovare anomalie (un elemento che si trova in un cluster sbagliato) e fare clic sugli elementi per ascoltare il campione. Questo è un ottimo modo per verificare se il dataset contiene elementi errati e per verificare se questo sia adatto al ML (le etichette dovrebbero essere ben distinte).

Con i dati processati è arrivato il momento di allenare la rete neurale. In questo caso la rete prende l'MFCC come ingresso e cerca di mappare questi ingressi in una delle quattro classi. Cliccando su *NN Classifier*, ci si trova davanti alla seguente pagina (figura 4.8):

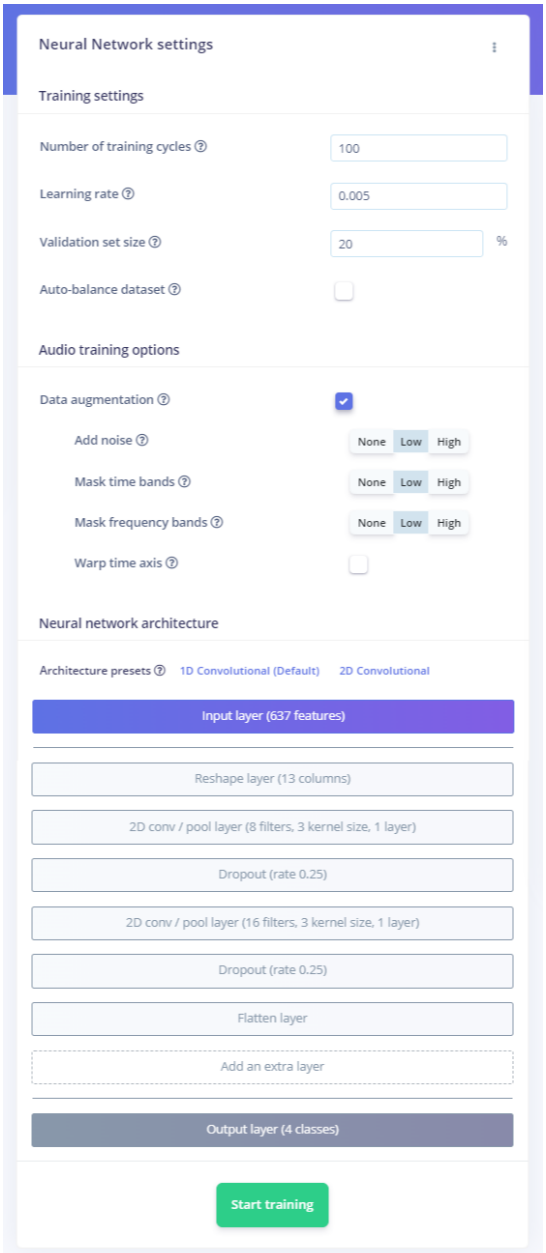


Figura 4.8 – NN Classifier

Si trova, come visto nei capitoli precedenti, la struttura di una rete neurale, formata dai diversi layer, che gradualmente trasforma gli ingressi originali in qualcosa di radicalmente differente, in questo caso in quattro valori, che corrispondono ai valori delle probabilità delle etichette.

Durante il training, lo stato interno dei neuroni viene gradualmente ottimizzato e perfezionato, in modo che la rete trasformi il suo input nel modo giusto, per produrre l'output corretto. Questo viene fatto inserendo un campione di dati di training, controllando quanto sia lontano l'output della rete dalla risposta corretta e regolando lo stato interno dei neuroni per rendere più probabile che venga prodotta una risposta corretta la volta successiva. Se fatto migliaia di volte, ciò si traduce in una rete addestrata. Ovviamente possono essere aggiunti o tolti layer in base alle esigenze del progetto che si considera. In questo caso un buon setting è quello in figura 4.8.

Inoltre, selezioniamo "Data augmentation". Quando è abilitato, i dati vengono mutati casualmente durante l'allenamento. Ad esempio, aggiungendo rumore, mascherando le bande di

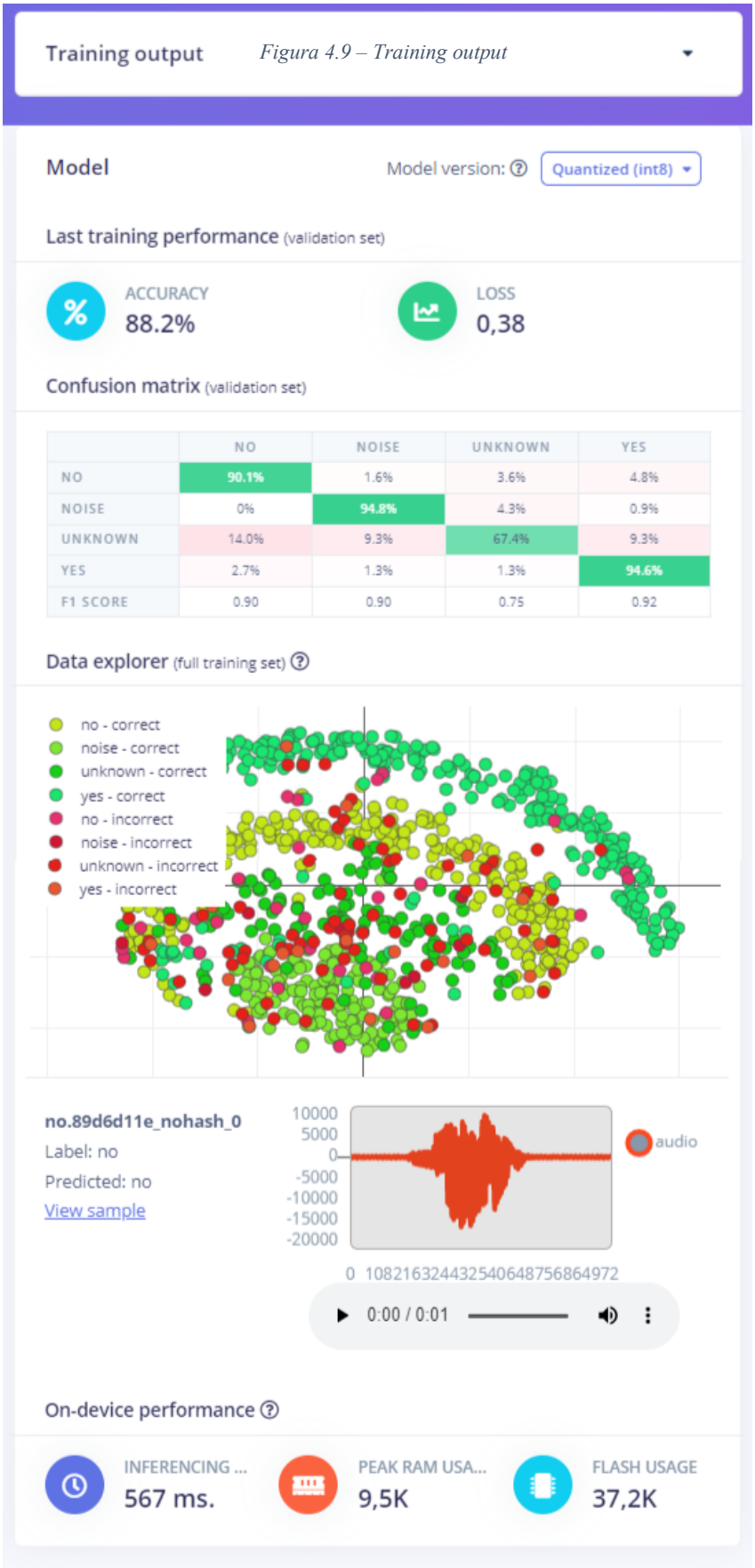


Figura 4.9 – Training output

frequenza o deformando l'asse del tempo. Questo è un modo molto rapido per far funzionare meglio il dataset nel mondo reale e impedisce l'overfitting della rete neurale (poiché i campioni di dati vengono modificati ad ogni ciclo di allenamento).

In base al tipo di training che verrà eseguito, esso richiederà qualche minuto. Una volta terminato il training avremo in uscita la schermata in figura 4.9. Si ottiene il data explorer similmente a come già visto, in questo caso però sono etichettati in rosso i campioni valutati in modo errato ed in verde quelli valutati correttamente. In più, con altre informazioni, si ottiene quello che la piattaforma chiama **Last training performance panel**,

che dà una valutazione quantitativa delle prestazioni della rete sui dati di training. Questi strumenti permettono di farsi un'idea di come funziona la rete, e di dove andare ad agire per migliorarne le prestazioni. È ovvio che più alti sono i valori meglio è, tuttavia, come visto nel capitolo 3, questo non è sempre vero; un'accuratezza del 100% potrebbe indicare con tutta probabilità che il modello sia in overfitting rispetto ai dati di training, cioè si sia troppo adattato a questi; per molte applicazioni una percentuale sopra 85% viene considerata più che buona. Le **On-device performance** ci dicono come funzionerebbe il modello sul dispositivo; l'intervallo di inferenza stima quanto tempo impiegherebbe il modello ad analizzare un secondo di dati sulla scheda selezionata.

Costruito e testato il modello sui dati di training, ora è importante testare il modello su dati che quest'ultimo non abbia mai processato, in modo da avere un'idea del suo comportamento nel mondo reale e verificare che il modello non sia in overfit. A questo scopo si apre la scheda **Model testing** e si sfrutta il 20% di dati che erano stati tenuti da parte come dati di test e che la rete non ha mai elaborato. Si clicca su *Classify all* e si attende il risultato (figura 4.10).

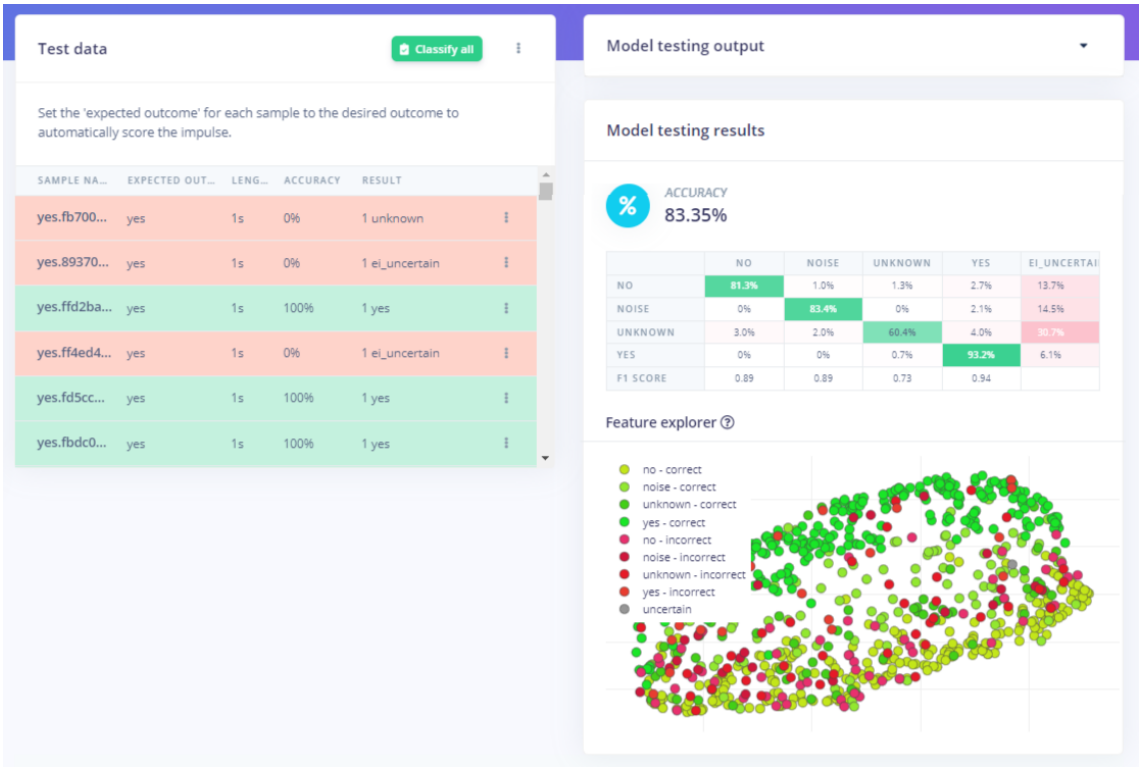


Figura 4.10 – Model testing

Si ottiene come prima una tabella con un indice delle prestazioni del dispositivo e un grafico con indicati in rosso i campioni classificati in modo errato, il che dà una chiara indicazione di quale dovrebbe essere il comportamento della rete. Si ottiene inoltre la tabella sulla sinistra, in cui si ha

una lista di tutti i campioni classificati con il valore di accuratezza e la rispettiva classificazione. Per approfondire un campione classificato erroneamente, basta cliccare sui tre punti (⋮) accanto a un campione e selezionare *Mostra classificazione*. Si viene quindi trasferiti alla visualizzazione della classificazione, che consente di ispezionare il campione e confrontare il campione con i dati di training. In questo modo si può controllare se si tratti effettivamente di un errore di classificazione o se i dati siano stati etichettati in modo errato. Da qui è possibile aggiornare l'etichetta (quando l'etichetta era sbagliata) o spostare l'elemento nel set di training per perfezionare il modello. È inevitabile che anche un modello di machine learning ben addestrato a volte classifichi erroneamente i suoi input. Quando si integra un modello in un'applicazione, bisogna tenere presente che non sempre darà la risposta corretta.
Ad esempio, se si stesse classificando audio, si potrebbero voler classificare diverse finestre di dati e calcolare la media dei risultati. Questo darà una migliore precisione complessiva rispetto al presupposto che ogni singolo risultato sia corretto.

Con l'impulso progettato, addestrato e verificato si procede a caricare il modello nella scheda. Ciò fa funzionare il modello senza una connessione internet, riduce al minimo la latenza e funziona con un consumo energetico minimo. Tra le diverse possibilità esaminate in precedenza, andremo a creare una libreria Arduino; una volta creata si ottiene una libreria con relativi esempi da caricare nell'IDE di Arduino. Gli esempi sono programmi che permettono di utilizzare la libreria creata, contenente l'impulso, all'interno del programma stesso e di poter quindi modificare in maniera relativamente semplice quest'ultimo per adattarlo agli scopi del progetto. Quello che si ottiene con le modifiche necessarie per il controllo dei led e della lettura dei sensori è ciò che segue:

```
// If your target is limited in memory remove this macro to save 10K RAM
#define EIDSP_QUANTIZE_FILTERBANK   0

/**
 * Define the number of slices per model window. E.g. a model window of 1000 ms
 * with slices per model window set to 4. Results in a slice size of 250 ms.
 * For more info: https://docs.edgeimpulse.com/docs/continuous-audio-sampling
 */
#define EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW 3                1

/* Includes ---------------------------------------------------------------- */
#include <PDM.h>
#include <Arduino_HTS221.h>    // <===
#include <Responding_to_the_voice_inferencing.h>

/** Audio buffers, pointers and selectors */
typedef struct {
    signed short *buffers[2];
    unsigned char buf_select;
    unsigned char buf_ready;
    unsigned int buf_count;
```

```
    unsigned int n_samples;
} inference_t;

static inference_t inference;
static bool record_ready = false;
static signed short *sampleBuffer;
static bool debug_nn = false; // Set this to true to see e.g. features generated from the raw
signal
static int print_results = -(EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW);

/**
 * @brief      Arduino setup function
 */
void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);

if (!HTS.begin()) {
    Serial.println("Failed to initialize humidity temperature sensor!");          // <===
    while (1);
  }

  // set LED's pin to output mode        // <===
  //pinMode(LED_BUILTIN, OUTPUT);
  pinMode(LEDR, OUTPUT);
  pinMode(LEDG, OUTPUT);
  pinMode(LEDB, OUTPUT);

// LED builtin attivo HIGH, LED RGB attivo LOW
  //digitalWrite(LED_BUILTIN, LOW);         // turn off the BUILTIN LED
  digitalWrite(LEDR, HIGH);                 // will turn the LED off
  digitalWrite(LEDG, HIGH);                 // will turn the LED off
  digitalWrite(LEDB, HIGH);                 // will turn the LED off

    Serial.println("Edge Impulse Inferencing Demo");

    // summary of inferencing settings (from model_metadata.h)
    ei_printf("Inferencing settings:\n");
    ei_printf("\tInterval: %.2f ms.\n", (float)EI_CLASSIFIER_INTERVAL_MS);
    ei_printf("\tFrame size: %d\n", EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE);
    ei_printf("\tSample length: %d ms.\n", EI_CLASSIFIER_RAW_SAMPLE_COUNT / 16);
    ei_printf("\tNo. of classes: %d\n", sizeof(ei_classifier_inferencing_categories) /
                                            sizeof(ei_classifier_inferencing_categories[0]));

    run_classifier_init();
    if (microphone_inference_start(EI_CLASSIFIER_SLICE_SIZE) == false) {
        ei_printf("ERR: Failed to setup audio sampling\r\n");
        return;
    }
}

/**
 * @brief      Arduino main function. Runs the inferencing loop.
 */
void loop()
{
    bool m = microphone_inference_record();
    if (!m) {
        ei_printf("ERR: Failed to record audio...\n");
        return;
```

```
    }

    signal_t signal;
    signal.total_length = EI_CLASSIFIER_SLICE_SIZE;
    signal.get_data = &microphone_audio_signal_get_data;
    ei_impulse_result_t result = {0};

    EI_IMPULSE_ERROR r = run_classifier_continuous(&signal, &result, debug_nn);
    if (r != EI_IMPULSE_OK) {
        ei_printf("ERR: Failed to run classifier (%d)\n", r);
        return;
    }

    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");
        for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
            ei_printf("    %s: %.5f\n", result.classification[ix].label,
                      result.classification[ix].value);
        }

    // print an empty line
    Serial.println();

// ------------ CONFERMA VISIVA DELLA CLASSIFICAZIONE ------------

    // NO keyword
    if(result.classification[0].value>0.6) {
      // RED ON
      digitalWrite(LEDR, LOW);            // will turn the LED on
      digitalWrite(LEDG, HIGH);           // will turn the LED off
      digitalWrite(LEDB, HIGH);           // will turn the LED off
      /*delay(1000);

      // read humidity
      float humidity = HTS.readHumidity();
      // print humidity
      Serial.print("Humidity = ");
      Serial.print(humidity);
      Serial.println(" %");
      // print an empty line
      Serial.println();
    }

    // NOISE keyword
    else if(result.classification[1].value>0.6) {
      // WHITE ON
      digitalWrite(LEDR, LOW);            // will turn the LED on
      digitalWrite(LEDG, LOW);            // will turn the LED on
      digitalWrite(LEDB, LOW);            // will turn the LED on
      /*delay(1000);
    }

    // UNKNOWN keyword
    else if(result.classification[2].value>0.6) {
      // BLUE ON
      digitalWrite(LEDR, HIGH);            // will turn the LED off
      digitalWrite(LEDG, HIGH);            // will turn the LED off
```

```
      digitalWrite(LEDB, LOW);            // will turn the LED on
      /*delay(1000);
    }

    // YES keyword
    else if(result.classification[3].value>0.6) {
      // GREEN ON
      digitalWrite(LEDR, HIGH);            // will turn the LED off
      digitalWrite(LEDG, LOW);            // will turn the LED on
      digitalWrite(LEDB, HIGH);            // will turn the LED off
      /*delay(1000);

      // read temperature
      float temperature = HTS.readTemperature();
      // print temperature
      Serial.print("Temperature = ");
      Serial.print(temperature);
      Serial.println(" °C");
      // print an empty line
      Serial.println();
    }

    // ---------------------------------------------------------------

#if EI_CLASSIFIER_HAS_ANOMALY == 1
        ei_printf("    anomaly score: %.3f\n", result.anomaly);
#endif

        print_results = 0;
    }
}

/**
 * @brief      PDM buffer full callback
 *             Get data and call audio thread callback
 */
static void pdm_data_ready_inference_callback(void)
{
    int bytesAvailable = PDM.available();

    // read into the sample buffer
    int bytesRead = PDM.read((char *)&sampleBuffer[0], bytesAvailable);

    if (record_ready == true) {
        for (int i = 0; i<bytesRead>> 1; i++) {
            inference.buffers[inference.buf_select][inference.buf_count++] = sampleBuffer[i];

            if (inference.buf_count >= inference.n_samples) {
                inference.buf_select ^= 1;
                inference.buf_count = 0;
                inference.buf_ready = 1;
            }
        }
    }
}

/**
 * @brief      Init inferencing struct and setup/start PDM
 *
 * @param[in]  n_samples  The n samples
 *
```

```
 * @return     { description_of_the_return_value }
 */
static bool microphone_inference_start(uint32_t n_samples)
{
    inference.buffers[0] = (signed short *)malloc(n_samples * sizeof(signed short));

    if (inference.buffers[0] == NULL) {
        return false;
    }

    inference.buffers[1] = (signed short *)malloc(n_samples * sizeof(signed short));

    if (inference.buffers[1] == NULL) {
        free(inference.buffers[0]);
        return false;
    }

    sampleBuffer = (signed short *)malloc((n_samples >> 1) * sizeof(signed short));

    if (sampleBuffer == NULL) {
        free(inference.buffers[0]);
        free(inference.buffers[1]);
        return false;
    }

    inference.buf_select = 0;
    inference.buf_count = 0;
    inference.n_samples = n_samples;
    inference.buf_ready = 0;

    // configure the data receive callback
    PDM.onReceive(&pdm_data_ready_inference_callback);

    PDM.setBufferSize((n_samples >> 1) * sizeof(int16_t));

    // initialize PDM with:
    // - one channel (mono mode)
    // - a 16 kHz sample rate
    if (!PDM.begin(1, EI_CLASSIFIER_FREQUENCY)) {
        ei_printf("Failed to start PDM!");
    }

    // set the gain, defaults to 20
    PDM.setGain(127);

    record_ready = true;

    return true;
}

/**
 * @brief      Wait on new data
 *
 * @return     True when finished
 */
static bool microphone_inference_record(void)
{
    bool ret = true;

    if (inference.buf_ready == 1) {
        ei_printf(
```

```
            "Error sample buffer overrun. Decrease the number of slices per model window "
            "(EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)\n");
        ret = false;
    }

    while (inference.buf_ready == 0) {
        delay(1);
    }

    inference.buf_ready = 0;

    return ret;
}

/**
 * Get raw audio signal data
 */
static int microphone_audio_signal_get_data(size_t offset, size_t length, float *out_ptr)
{
    numpy::int16_to_float(&inference.buffers[inference.buf_select ^ 1][offset], out_ptr,
length);

    return 0;
}

/**
 * @brief      Stop PDM and release buffers
 */
static void microphone_inference_end(void)
{
    PDM.end();
    free(inference.buffers[0]);
    free(inference.buffers[1]);
    free(sampleBuffer);
}

#if !defined(EI_CLASSIFIER_SENSOR) || EI_CLASSIFIER_SENSOR != EI_CLASSIFIER_SENSOR_MICROPHONE
#error "Invalid model for current sensor."
#endif
```

Ci sono alcune righe di codice interessanti, ad esempio il riquadro 1 dice che per ogni finestra del modello verranno prese 3 “fettine”, cioè per ogni intervallo di campionamento del segnale (1 s nel modello), vengono prese 3 finestre, quindi una all’inizio, una la centro ed una alla fine i cui risultati nella classificazione verranno mediati per dare come output la classificazione finale; questo approccio può aiutare a migliorare l’accuratezza generale del modello. Il secondo riquadro è la parte di codice che effettivamente dà i valori di uscita dell’inferenza, stampa infatti le variabili che contengono il tempo di risposta del DSP, classificazione e anomalia; inoltre, stampa il valore delle probabilità delle quattro etichette risultanti dall’inferenza. L’ultimo riquadro è una modifica effettuata al codice in modo da sfruttare i risultati della classificazione; il valore tra parentesi è legato alla specifica etichetta, mentre il valore a destra è la soglia (probabilità) impostata oltre la quale deve essere effettuata l’azione compresa nell’*if()*. È intuitivo che spostare la soglia influenzi

in un certo modo le prestazioni della rete, il valore 0.6/0.7 è un buon compromesso in questo caso (e in molti casi).

4.1.3 Valutazione prestazioni

Il modello caricato sulla scheda è stato testato nella pratica per verificarne le prestazioni. Il test consiste nel provare per ogni label 20 ripetizioni delle parole, in ordine randomizzato (ripetendo cioè a caso tra yes, no o unknown per un totale di 20 volte ognuna), ripetendo il test con diverse persone. I risultati dicono che il modello non è influenzato in maniera rilevante dalla persona che pronuncia la parola, per ogni persona si sono ottenuti i seguenti valori di accuracy della classificazione:

YES: 18/20 (90%)

NO: 18/20 (90%)

UNKNOWN: 16/20 (80%)

I valori peggiori si ottengono dunque con la terza etichetta che spesso sbaglia non nel riconoscimento della parola specifica ma scambiando la parola pronunciata per yes o no, questo a causa della scelta delle due keyword che non sono state scelte tra le più ottimali per questo genere di funzione. Da notare il valore di inferenza su un secondo di campione che è abbastanza alto (567 ms, figura 4.9). In figura 4.11 si vede il risultato della classificazione attraverso il monitor dell'IDE di Arduino.

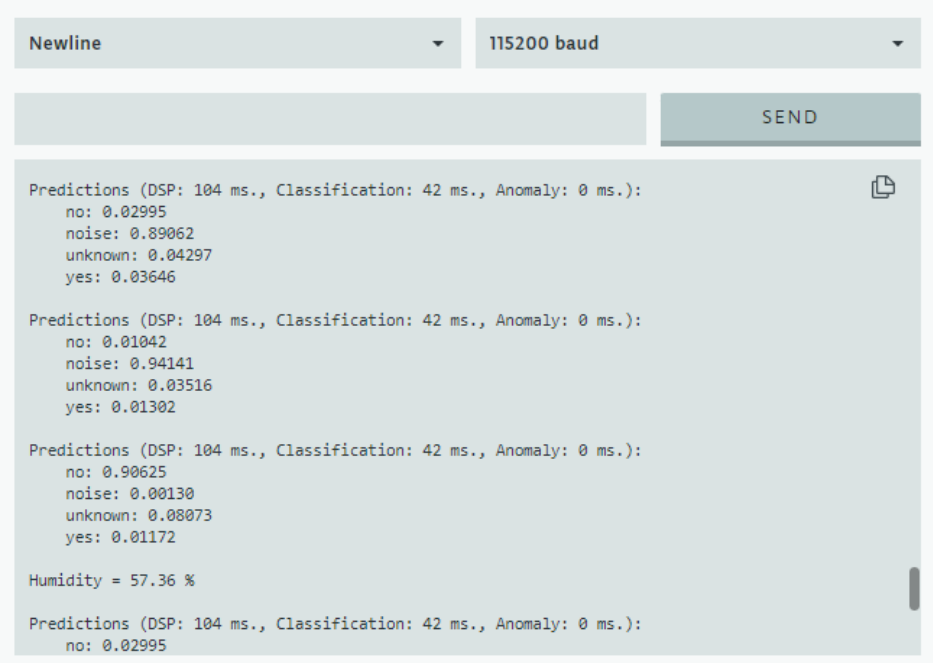


Figura 4.11 – Risultato inferenza da Monitor Arduino

Si può dire dunque che in generale il modello dia risultati più che buoni, riesce a riconoscere parole pronunciate da diverse persone senza troppa discrepanza tra le prestazioni, inoltre riconosce le due parole chiave con un'accuratezza eccellente e non possiamo dire che non sia lo stesso con le parole sconosciute; infatti, il riconoscimento è già buono e possiamo dedurre che con una scelta più accurata delle keyword le prestazioni dell'etichetta salirebbero ulteriormente. Il ritardo teorico nell'esecuzione dell'inferenza è alto ed anche all'atto pratico tra il ricevimento della parola e l'accensione del led passa circa un secondo. Questo non è però necessariamente un problema per gli scopi per cui è pensato questo progetto, cioè un utilizzo della scheda come smart sensor; se si pensa ad esempio ad un utilizzo come sensore in una smart home, per il controllo di luci, riscaldamento o altro, il tempo di attesa di un secondo non è sicuramente un problema. Certamente in altre applicazioni questo può essere un limite di questo tipo di schede, ma va valutato di caso in caso, considerando anche le possibili ottimizzazioni del modello che potrebbero dare più margine di manovra.

4.2 Classificazione Movimenti

4.2.1 Panoramica

In questo progetto verrà usato il ML per costruire un sistema di riconoscimento dei movimenti che funzioni su un sistema embedded come Arduino. In questo caso verrà aggiunto un ulteriore blocco di apprendimento, l'**Anomaly detection** che verrà trattata con maggiore dettaglio più avanti.

L'algoritmo dovrà classificare tre gesti (wave, snake e updown) e la situazione di riposo (idle), per fare questo si utilizzerà la IMU integrata nella scheda (LSM9DS1); si avrà come conferma visiva della classificazione l'accensione del led integrato: blu per wave, verde per snake, rosso per updown e niente per idle. Verranno valutate le prestazioni del modello con soggetti diversi e variando la frequenza di campionamento della IMU.

4.2.2 Esperimento

Si procede in maniera analoga a prima, come azione iniziale si raccolgono i dati per creare il dataset; dunque, si collega la scheda e si entra nella pagina *Data acquisition*. I campioni posso essere raccolti tramite CoolTerm o dal tool messo a disposizione da Edge Impulse. Nel secondo caso si seleziona come sensore l'accelerometro, una lunghezza del campione di 10 secondi, una frequenza di campionamento di 62.5 Hz e si associa una label. Un buon obbiettivo è avere circa

otto/dieci minuti di dati per ogni etichetta, cercando come già detto di avere campioni più eterogenei possibile. La situazione sarà all'incirca quella in figura 4.12.

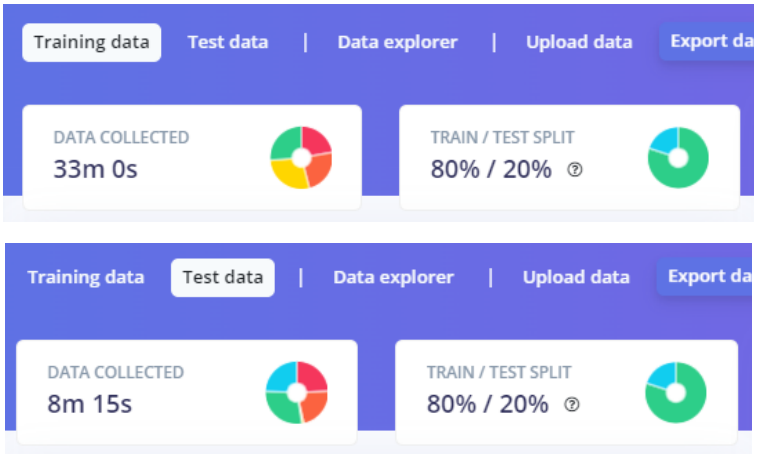


Figura 4.12 - Dataset

Una volta creato il dataset e dopo averlo riequilibrato è il momento di creare l'impulso. Per questo progetto si utilizza il blocco *Spectral analysis* per il blocco DSP, che applica un filtro, esegue l'analisi spettrale sul segnale ed estrae i dati di frequenza e potenza spettrale ed è ottimo per segnali con schemi ripetitivi. Per quanto riguarda i blocchi di apprendimento si aggiunge la Neural Network (Keras) come nel progetto precedente, che questa volta prenderà in ingresso le features spettrali; come blocco aggiuntivo c'è il blocco Anomaly detection (K-means) per il rilevamento delle anomalie; si imposta la lunghezza della finestra a 2 s e l'incremento a 80 ms e si salva l'impulso. Il risultato sarà l'impulso di figura 4.13.

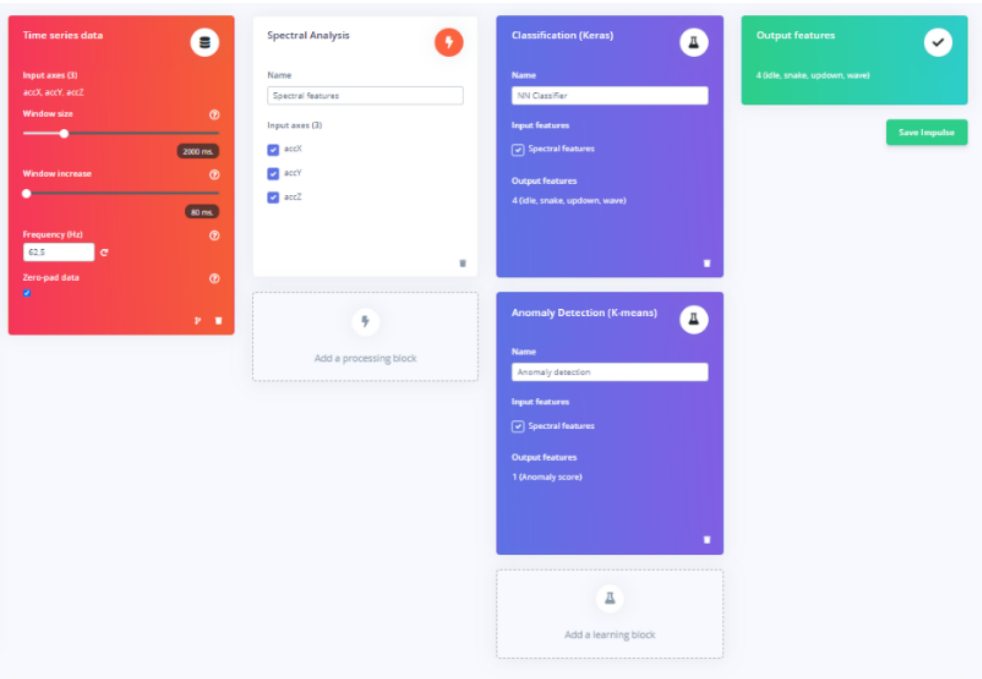


Figura 4.13 – Impulse design

È il momento di configurare i vari blocchi, il primo è lo spectral features, la schermata offre diverse funzioni, tra cui la possibile implementazione di un filtro e della lunghezza della FFT. Sulla destra si possono trovare grafici che mostrano le elaborazioni che vengono eseguite sul segnale. La situazione sarà dunque quella di figura 4.14.

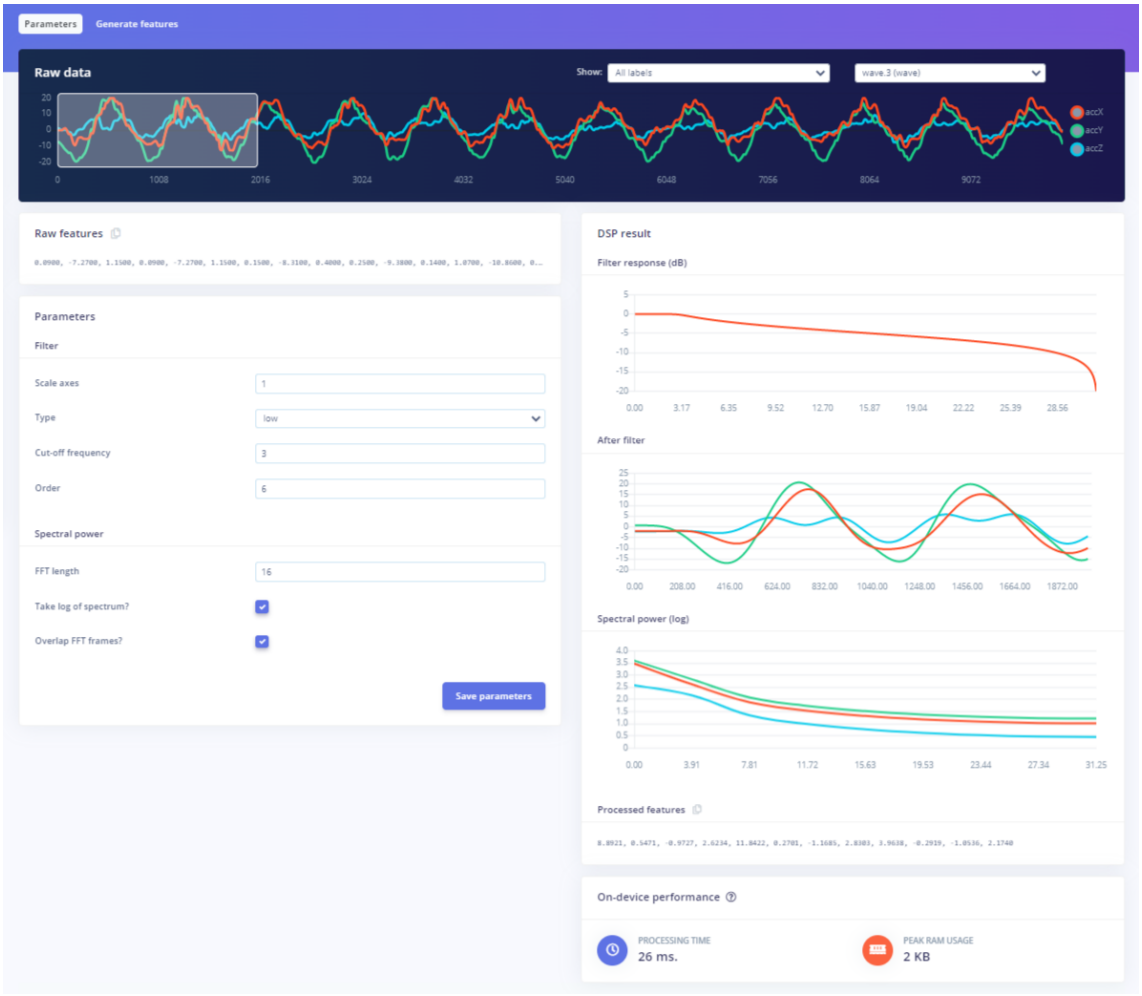


Figura 4.14 – Spectral features

Quindi si salva e si generano le features, stando attenti questa volta a selezionare l'opzione **Calculate feature importance** che tornerà utile tra poco. Il risultato è mostrato in figura 4.15.

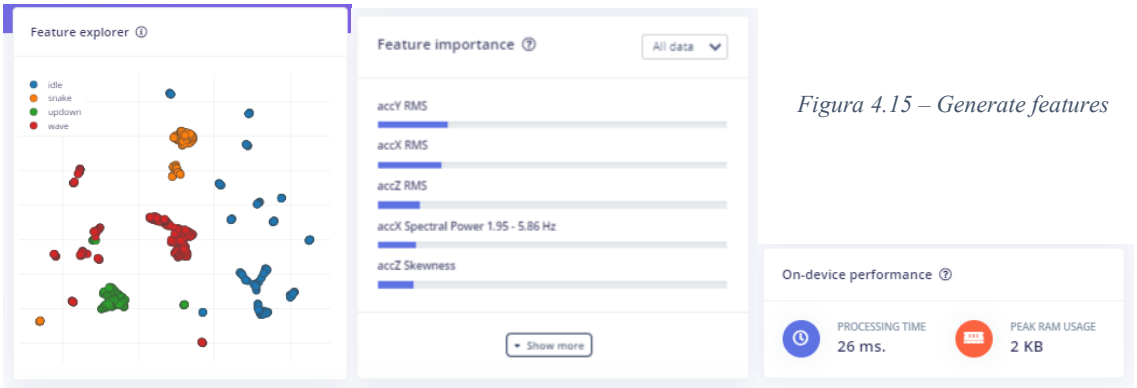


Figura 4.15 – Generate features

Si vede chiaramente come le features siano ben separate, anche se in un certo modo le due etichette updown e wave in alcuni punti si sovrappongono; in generale è un buon dataset di partenza e vale la pena proseguire con la creazione del modello. Allenando la rete neurale si ottiene la figura 4.16.

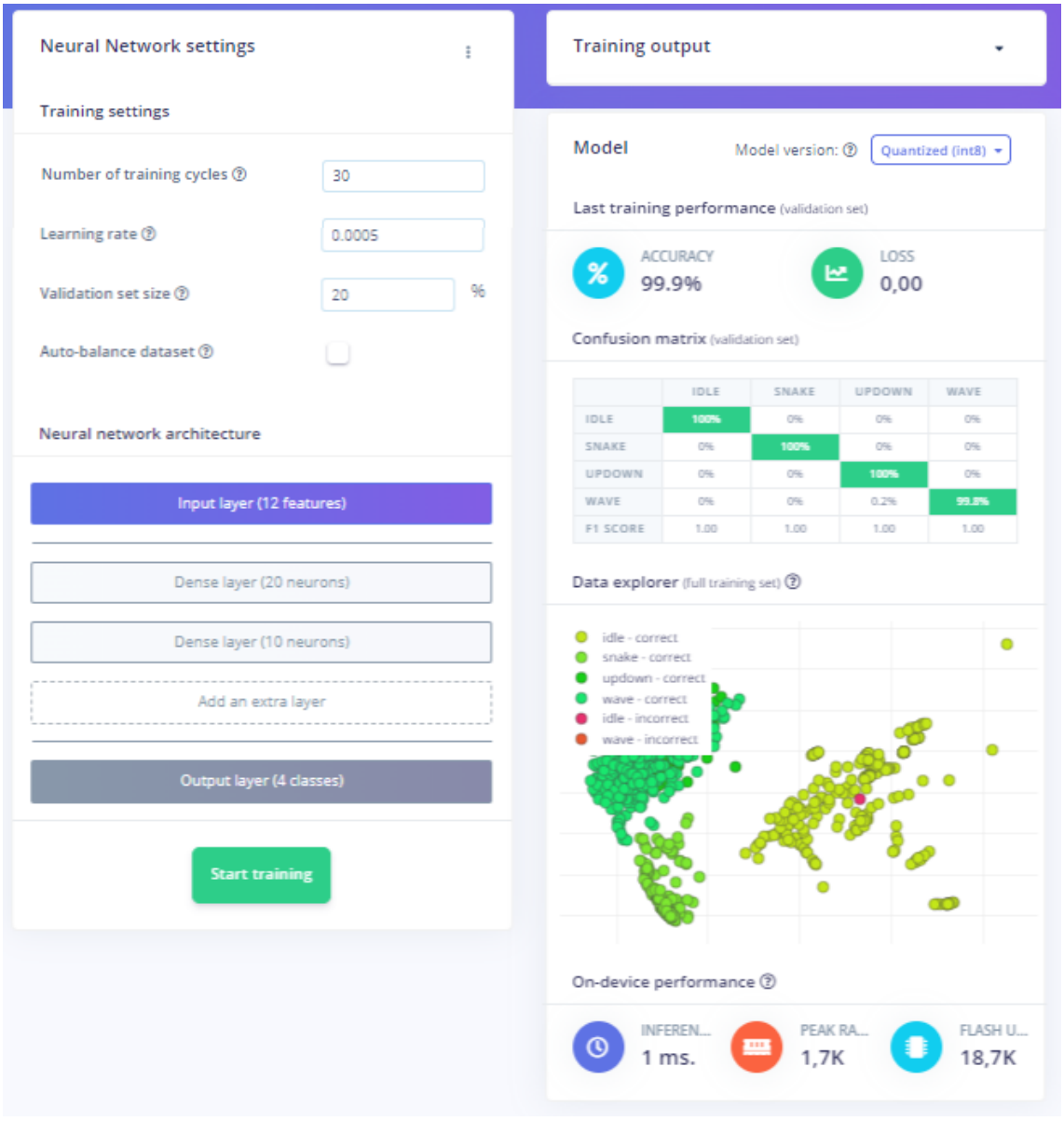


Figura 4.16 – NN settings

Come si vede dai valori e dai grafici, l'accuratezza del modello è decisamente elevata, questo può far pensare sicuramente ad una situazione di overfitting, che dovrà essere verificata. Da notare anche come il tempo di inferenza sia decisamente più basso rispetto al caso precedente, indicando come dati di questo tipo siano meno impegnativi da elaborare.

Le reti neurali funzionano molto bene per i compiti per i quali sono pensate, ma hanno un grosso difetto, sono terribili nel gestire dati che non abbiano mai visto prima, completamente diversi da

quelli per cui sono state allenate. Quello che queste reti fanno quando si trovano in una situazione del genere è di classificare questo segnale con una delle etichette che conoscono, in maniera del tutto imprevedibile. Per aiutare la rete sotto questo aspetto si aggiunge una seconda rete neurale, l'**Anomaly detection**, la quale crea dei cluster intorno ai dati di training e compara i campioni in arrivo con questi cluster. Se la distanza da questi cluster è troppo elevata questo campione viene contrassegnato come anomalia e dunque si ha un'allerta che la classificazione della rete non è affidabile. Per impostare i parametri di questo blocco si sfrutta il calcolo sull'importanza delle features eseguito in precedenza; per fare questo basta cliccare su **Select suggested axes** e iniziare il training della rete. Si otterrà la figura 4.17 che segue:

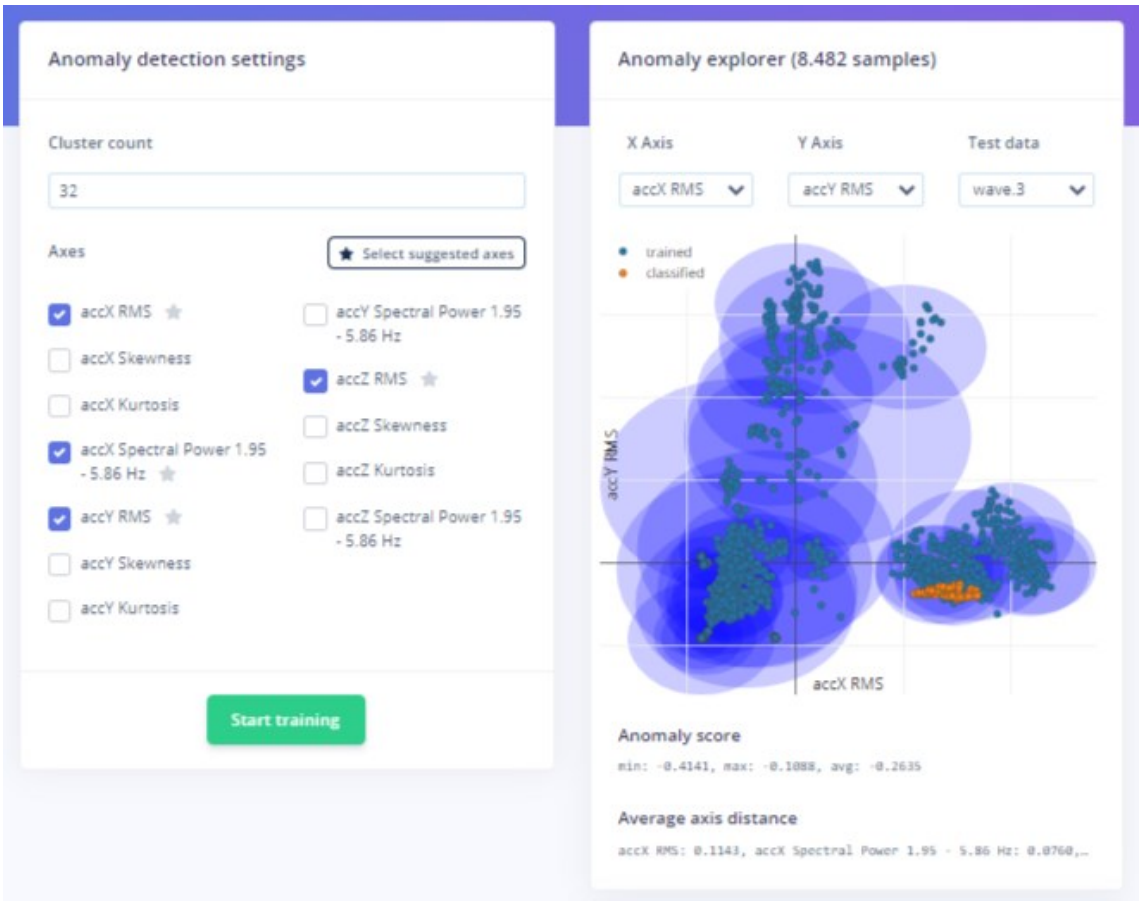


Figura 4.17 – Anomaly detection

A questo punto non resta che verificare come si comporta la rete con dati sconosciuti, basta andare su *Model testing* e far partire il test, i risultati seguono in figura 4.18. Come si può vedere le prestazioni sono ottime anche in questo caso; dunque, non sembra esserci una situazione di overfit. La sola cosa che resta da fare è creare la libreria Arduino, modificare il codice, caricarlo sulla scheda e verificarne le prestazioni sul campo.

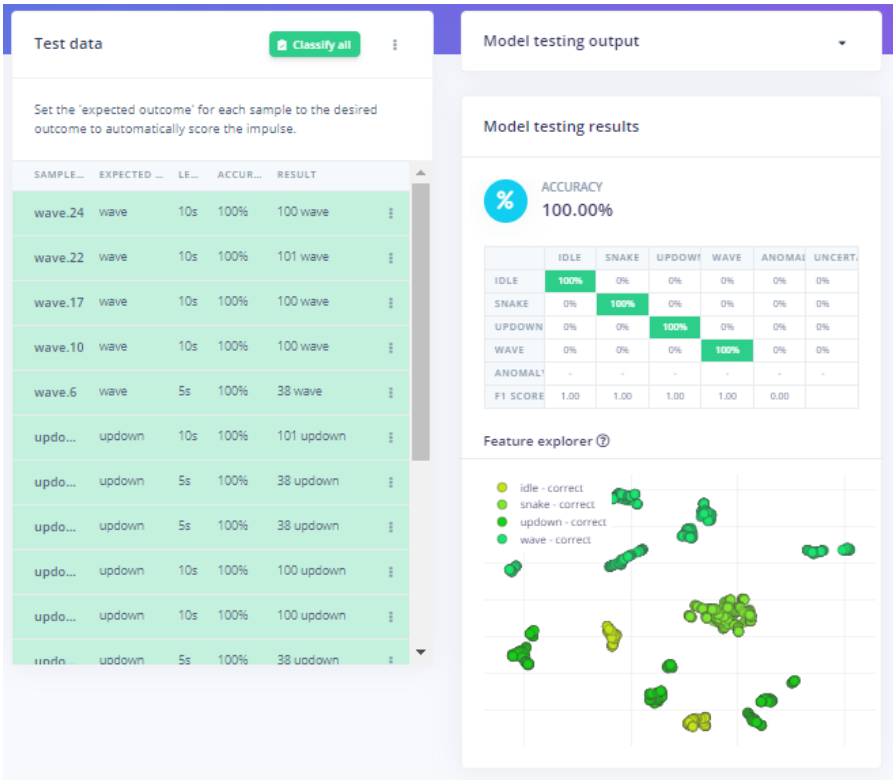


Figura 4.18 – Model testing

4.2.3 Valutazione prestazioni

Ai fini di valutare le prestazioni del sistema la prima cosa da fare è modificare il codice tramite l'IDE di Arduino. Questo per far si che la scheda esegua le azioni richieste in base ai risultati dell'inferenza, in questo caso solo l'accensione del colore del led specifico, e per variare i parametri che ci interessano ai fini della valutazione delle prestazioni. Per brevità verranno descritte solo le parti più interessanti.

Le righe di codice in figura 4.19 permettono, tramite la libreria modificata descritta nel paragrafo 1.3, di intervenire sui parametri più importanti della IMU, come la scala di misura, la frequenza di campionamento e le unità di misura di accelerometro, giroscopio e magnetometro.

La figura 4.20 mostra invece una struttura di codice che ha lo scopo di smussare i risultati dell'inferenza, nel senso che il risultato non viene preso sulla singola finestra catturata in un certo istante temporale, ma vengono classificate un certo numero di finestre successive, viene definita una soglia di confidenza che ognuna deve avere, una soglia per l'anomalia e il numero di classificazioni successive che devono essere uguali, sul totale delle letture, affinché la classificazione complessiva assegni il risultato ad una certa label. In questo caso si è trovato un

buon compromesso con 10 letture totali, 7/8 letture successive uguali e soglie per anomalia e confidenza pari rispettivamente a 0.3 e 0.8. Un esempio di output è illustrato in figura 4.21.

```
94
95      // IMU PARAMETERS SETTING ------------------------------------------------------------------------------------------------------------------
96
97    // Accellerometer
98    IMU.setAccelFS(2);
99    IMU.setAccelODR(3);       //  <----
100 /*****************************************************************************************************************************************
101 *******  FS  Full Scale         range 0:±2g | 1:±24g | 2: ±4g | 3: ±8g  (default=2)                                                    ******
102 *******  ODR Output Data Rate   range 0:off | 1:10Hz | 2:50Hz | 3:119Hz | 4:238Hz | 5:476Hz, (default=3)(not working 6:952Hz) ******
103 *****************************************************************************************************************************************/
104
105   // Gyroscope  (se accelerometro e giroscopio sono accesi l'ODR di entrambi è dato da quello del girsoscopio)
106   IMU.setGyroFS(3);
107   IMU.setGyroODR (5);   //  <----
108 /*****************************************************************************************************************************************
109 *********  FS  Full Scale       setting 0: ±245°/s | 1: ±500°/s | 2: ±1000°/s | 3: ±2000°/s       ***************************
110 *********  ODR Output Data Rate setting 0:off | 1:15Hz | 2:60Hz | 3:119Hz | 4:238Hz | 5:476Hz, (not working 6:952Hz)   *******
111 *****************************************************************************************************************************************/
112
113 // Magnetometer         // <===
114 IMU.setMagnetFS(0);
115 IMU.setMagnetODR(6);
116 /*****************************************************************************************************************************************
117 ****  FS  Full Scale        range (0=±400 | 1=±800 | 2=±1200 | 3=±1600  (µT)                                                          *****
118 ****  ODR Output Data Rate  range (6,7,8)=(40,80,400)Hz | not available on all chips (0..5): (0.625,1.25,2.5,5.0,10,20)Hz *****
119 *****************************************************************************************************************************************/
120
121 // Unit
122 IMU.accelUnit=  METERPERSECOND2;    // or  METERPERSECOND2
123 IMU.gyroUnit= DEGREEPERSECOND;  // or  DEGREEPERSECOND  RADIANSPERSECOND  REVSPERMINUTE  REVSPERSECOND
124 IMU.magnetUnit = MICROTESLA;  //   GAUSS   MICROTESLA   NANOTESLA                // <===
125
```

Figura 4.19 – Controllo frequenza e range IMU

```
177   // ----------------------------------------------------------------------------------------    <---
178
179   // This is a structure that smoothens the output result
180   // With the default settings 70% of readings should be the same before classifying.
181   ei_classifier_smooth_t smooth;
182   ei_classifier_smooth_init(&smooth, 10 /* no. of readings */, 8 /* min. readings the same */, 0.8 /* min. confidence */, 0.3 /* max anomaly */);
183                             // default setting (10, 7, 0.8, 0.3)
184   // ----------------------------------------------------------------------------------------
```

Figura 4.20 – Struttura di codice che smussa il risultato dell'inferenza

```
Output   Monitor seriale ×

Messaggio(CTRL + Invio per inviare il messaggio a 'Arduino Nano 33 BLE' su 'COM3')

Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): idle  [ 10, 0, 0, 0, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): idle  [ 10, 0, 0, 0, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): idle  [ 10, 0, 0, 0, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): idle  [ 10, 0, 0, 0, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): idle  [ 10, 0, 0, 0, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): idle  [ 10, 0, 0, 0, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): idle  [ 10, 0, 0, 0, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): idle  [ 10, 0, 0, 0, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): idle  [ 10, 0, 0, 0, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): idle  [ 10, 0, 0, 0, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): idle  [ 9, 0, 0, 0, 1, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): idle  [ 8, 1, 0, 0, 1, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): uncertain  [ 7, 2, 0, 0, 1, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): uncertain  [ 6, 2, 1, 0, 1, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): uncertain  [ 5, 2, 2, 0, 1, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): uncertain  [ 4, 2, 2, 1, 1, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): uncertain  [ 3, 2, 2, 2, 1, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): uncertain  [ 2, 2, 2, 3, 1, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): uncertain  [ 1, 2, 2, 4, 1, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): uncertain  [ 0, 2, 2, 5, 1, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): uncertain  [ 0, 2, 2, 6, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 3 ms.): uncertain  [ 0, 1, 2, 7, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 3 ms.): wave  [ 0, 0, 2, 8, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 3 ms.): wave  [ 0, 0, 1, 9, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): wave  [ 0, 0, 0, 10, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): wave  [ 0, 0, 0, 10, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): wave  [ 0, 0, 0, 10, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): wave  [ 0, 0, 0, 10, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): wave  [ 0, 0, 0, 10, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): wave  [ 0, 0, 0, 10, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): wave  [ 0, 0, 0, 10, 0, 0, ]
Predictions (DSP: 81 ms., Classification: 0 ms., Anomaly: 1 ms.): wave  [ 0, 0, 0, 10, 0, 0, ]
Predictions (DSP: 82 ms., Classification: 0 ms., Anomaly: 1 ms.): wave  [ 0, 0, 0, 10, 0, 0, ]
```

Figura 4.21 – Output inferenza smooth

```
198         // Run the classifier
199         ei_impulse_result_t result = { 0 };
200
201         err = run_classifier(&signal, &result, debug_nn);
202         if (err != EI_IMPULSE_OK) {
203             ei_printf("ERR: Failed to run classifier (%d)\n", err);
204             return;
205         }
206
207         // ------------ CONFERMA VISIVA DELLA CLASSIFICAZIONE ------------
208         // label: 0 IDLE, 1 SNAKE, 2 UPDOWN, 3 WAVE
209
210       // IDLE motion
211       if(smooth.count[0]>=8) {
212         /*// WHITE ON
213         digitalWrite(LEDR, LOW);            // will turn the LED on
214         digitalWrite(LEDG, HIGH);           // will turn the LED off
215         digitalWrite(LEDB, HIGH);           // will turn the LED off
216         delay(1000); */
```

Figura 4.22 – Classificatore e Attuatore

Nella figura 4.22 è presente la struttura di codice che effettua effettivamente la classificazione e sotto similmente al caso vocale, è utilizzato un *if()* che sfrutta l'uscita della classificazione, impostando una soglia, per effettuare un azione di qualche tipo (l'accensione del led in questo caso).

Un primo test richiede di provare la classificazione con diversi soggetti; nello specifico eseguire in sequenza random venti classificazioni per ogni label (dunque sessanta in tutto) e valutare per ognuna l'accuratezza. I risultati dicono che l'accuratezza è pressoché indipendente dal soggetto (segno che il dataset è abbastanza diversificato) e sono:

SNAKE: 19/20 (95%)

UPDOWN: 19/20 (95%)

WAVE: 17/20 (85%)

Il che è, in generale, un risultato soddisfacente. Capita spesso però che il modello assegni a movimenti casuali un'etichetta, segno del fatto che è necessario ampliare il dataset e ottimizzare la rete di riconoscimento delle anomalie. In ogni caso, considerando la velocità di risposta e l'accuratezza del modello, i risultati sono sicuramente buoni.

In un secondo test si sono valutate le prestazioni al variare della frequenza di campionamento con cui venivano raccolti i campioni per il dataset e a cui avrebbe lavorato successivamente il modello sulla scheda. Si sono considerate due frequenze, la più bassa e la più alta selezionabili sulla

scheda, 15 e 250 Hz. Nel primo caso le prestazioni, per quanto riguarda l'accuratezza, non hanno registrato variazioni significative, ma si è avuto un aumento nella velocità di elaborazione e classificazione. Ciò è sicuramente da imputarsi al minor numero di campioni da elaborare per ogni finestra; ci si sarebbe potuto aspettare un degradamento delle prestazioni dovuto alla scarsità di campioni, ma ciò non è avvenuto; questo probabilmente perché i movimenti considerati, per quanto possano essere fatti velocemente e "sporcati" apposta, sono abbastanza lenti e regolari; quindi, ciò permette di abbassare notevolmente la frequenza di campionamento senza notare peggioramenti evidenti delle prestazioni.

Per quanto riguarda la seconda frequenza in esame non si sono avuti peggioramenti consistenti in termini di accuratezza, ma si sono avuti in termini di elaborazione del segnale, arrivando forse a toccare l'effettivo limite di questa scheda. Per essere più precisi, infatti, il processamento del segnale (DSP), in questo caso, richiede un tempo pari a circa 500 ms, rendendo evidente il netto peggioramento dal punto divista della velocità di esecuzione. Questo ovviamente limita molto le possibili applicazioni del modello a questa frequenza, portando la scelta di utilizzo in situazioni per cui la frequenza di campionamento non debba essere troppo elevata.

Per provare a migliorare le prestazioni è stato tolto il pre-processamento del segnale. La velocità di elaborazione si è impennata arrivando a 8 ms ma l'accuratezza è diminuita considerevolmente; non tanto nel riconoscimento dei movimenti quanto nel riconoscere segnali aleatori come movimenti; in posizioni di riposo, tenendo la scheda ferma con il braccio alzato, la scheda riconosce piccoli movimenti involontari come un movimento volontario. Questo probabilmente è dovuto alla mancanza del pre-processamento del segnale che non mitiga più gli andamenti spuri del segnale che arriva in ingresso alla rete neurale, e senza quel filtraggio la rete è molto più esposta a questo tipo di errori. Dunque, le considerazioni fatte poco sopra a proposito delle applicazioni rimangono valide.

Considerazioni finali

I risultati ottenuti possono considerarsi in generale soddisfacenti. La scheda ha un buon comportamento in termini di accuratezza nel riconoscimento del gesto o della parola chiave, in entrambe le situazioni. Mentre nel caso vocale questa mostra in generale una buona accuratezza ma una velocità di risposta non ottimale, nel secondo caso entrambi questi parametri danno risultati convincenti. In conclusione, Arduino Nano 33 BLE Sense e schede analoghe possono essere un buono strumento, e punto di partenza, per l'utilizzo di questa tipologia di device come dispositivi di embedded machine learning applicati ai margini di una rete (Edge devices). Se non è suggerito l'utilizzo di questo genere di dispositivi (a meno di non utilizzare device con maggiore potenza computazionale) in applicazioni in cui è richiesta una grande velocità di risposta, come auto autonome, riconoscimento facciale, il loro contesto di applicazione potrebbe sicuramente essere quello di Smart Home, domotica, sensing ambientale e tutte quelle applicazioni in cui la massima velocità di risposta non è il parametro principale sul quale valutare le prestazioni ottenibili.

Bibliografia

[1] I. Goodfellow, Y. Bengio, A. Courville, “Deep Learning”, 2016, The MIT Press

[2] S. Haykin, “Neural Networks and Learning Machines”, 2008, Pearson – Prentice Hall

[3] www.edgeimpulse.com

[4] docs.edgeimpulse.com/docs/pre-built-datasets/keyword-spotting

[5] docs.edgeimpulse.com/docs/edge-impulse-studio/processing-blocks/audio-mfe

[6] docs.edgeimpulse.com/docs/edge-impulse-cli/cli-overview

[7] docs.arduino.cc/hardware/nano-33-ble-sense

[8] www.arduino.cc/

[9]https://content.arduino.cc/assets/Nano_BLE_Sense_lsm9ds1.pdf?_gl=1*1in36a6*_ga*MTAzMDA0NDg4Ni4xNjU1MzkwNTM2*_ga_NEXN8H46L5*MTY2NTYwNzEwMi42NS4xLjE2NjU2MTM5NjguMC4wLjA

[10] github.com/FemmeVerbeek/Arduino_LSM9DS1

[11] freeware.the-meiers.org/

[12] R. Pandey, S. K. Khatri, N. K. Singh, P. Verma, “Artificial Intelligent and Machine Learning for EDGE Computing”, 2022, Elsevier – Academic Press

[13] D. P. Agrawal, “Embedded Sensor Systems”, 2017, Springer

[14] F. Al-Turjman, “Cognitive Sensors and IoT”, 2017, CRC Press

[15] G. C. M. Meijer, “Smart Sensor Systems”, 2008, Wiley