Università degli Studi della Calabria

Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistemistica

---

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea

Analisi, progettazione ed implementazione di un sistema di edge intelligence per la classificazione dell'audio ambientale: un approccio basato su Sentiment Analysis e tinyML

Relatore	Candidato
Ing. Giancarlo Fortino	Paolo Fusca
Ing. Claudio Savaglio	Matr.207030
Correlatore	
Ing. Pietro Manzoni	

Anno Accademico 2021-2022

Sommario

La Sentiment Analysis (SA), intesa come trattamento computazionale di opinioni e sentimenti, è un campo di ricerca che si sta sviluppando in risposta alla crescente domanda del mercato di creare servizi e prodotti customer-oriented. In questa direzione sono state adattate molte delle tecnologie esistenti, sopratutto nel campo dell'informatica, con lo scopo di creare sistemi in grado di recepire la soggettività dei dati. L'introduzione del Machine Learning (ML) all'interno delle attività di analisi dei dati ha rappresentato una rilevante innovazione: infatti, le ricerche degli ultimi venti anni hanno dimostrato come le macchine, utilizzando questo tipo di algoritmi, possono imparare praticamente qualsiasi cosa, quindi anche a riconoscere emozioni nei testi, nella voce o nelle immagini. In particolare, algoritmi di ML possono essere inseriti in entrambe le fasi del processo di SA, ovvero la raccolta dei dati e l'analisi degli stessi. Di recente però, il Tiny Machine Learning(tinyML), adattamento delle tecniche convenzionali di ML per sistemi con ridotte capacità computazionali, ha aperto nuove strade che prima sembravano impercorribili. Una di queste è la possibilità di creare sensori intelligenti, capaci cioè di processare in loco dati catturati dall'ambiente circostante, e impiegarli nell'ambito della SA. Il sistema presentato in questa tesi mira a offrire delle indicazioni sul grado di apprezzamento di opere d'arte in un museo, animali in un parco naturale o in generale qualsiasi servizio in esposizione ad un pubblico, utilizzando sensori intelligenti che analizzano le parole pronunciate dagli utenti, catturandole con appositi microfoni posizionati. L'idea progettuale sperimenta le possibilità offerte dal software Edge Impulse, una sistema di nuova generazione che supporta tutte le fasi della creazione di un modello (a partire dalla raccolta dati allo sviluppo per microcontrollori) permettendo di utilizzare blocchi di processamento per ognuna di esse, o creare i propri, attraverso l'utilizzo di Python e TensorFlow per i blocchi di apprendimento. Le prestazioni del sistema sviluppato sono state valutate sia in termini di complessità computazionale che affidabilità dei risultati, mostrando che algoritmi molto leggeri(supportati da microcontrollori) possono essere impiegati nell'analisi dei sentimenti con qualità paragonabile a quella di calcolatori ordinari.

Indice

Capitolo 1

Introduzione

1.1 Analisi del contesto,identificazione del problema, motivazioni

L'analisi del livello di soddisfazione di un prodotto o servizio offerto è un lavoro fondamentale per garantire l'apprezzamento generale del pubblico e per il quale si adoperano gli strumenti e le tecnologie più avanzate, questo con il fine di ottimizzare i risultati e renderli il più possibile attendibili. Questi risultati possono essere utilizzati o venduti a terzi così da migliorare l'esperienza dei futuri utenti, oppure utilizzati per fini pubblicitari e tanti altri scopi. Questa analisi è solitamente preceduta da una fase di raccolta dati che ha un'importanza fondamentale sui risultati del processo. Tale raccolta può essere eseguita con diversi mezzi e strumenti, dai più tradizionali questionari di soddisfazione a veri e propri strumenti tecnologici e software avanzati. Chiunque utilizza uno smartphone partecipa quotidianamente alla raccolta di questi dati anche senza saperlo e anche per questo è uno degli argomenti di dibattito odierni più controversi che ha spinto le autorità a introdurre delle leggi per la tutela della privacy e il rallentamento della crescita di tale fenomeno. I dati sono quindi l'oro del nuovo millennio e la loro raccolta rappresenta un business con grandi potenzialità oltre che un opportunità per migliorare la qualità dei servizi. La raccolta dei dati è fondamentale anche nel processo di **SA**, con il quale si intende l'analisi del grado di apprezzamento di un soggetto, azienda, prodotto o qualsiasi altro oggetto esposto , fisicamente o virtualmente, al pubblico. I dati dell'apprezzamento possono essere molteplici e di forma diversa, possono essere raccolti esplicitamente, si immagini ad esempio la richiesta di inserire una votazione dopo essere stati in un ristorante, o implicitamente, ad esempio la condivisione di un commento su un social network. Enti pubblici e privati nella società odierna sono motivati alla raccolta feedback puntando al massimo, sia in termini di quantità che di qualità, per migliorare i propri prodotti, essere competitivi sul mercato o innalzare la propria reputazione sociale. Come analisi nell'ambito del SA infatti si intende anche capire le idee che un gruppo di soggetti hanno nei riguardi di un entità e le emozioni che in loro su-

scita. Offrire degli strumenti per migliorare l'attuazione di questo processo è l'obiettivo primario di molte ricerche scientifiche, si prova sempre dunque non solo a sfruttare nuovi strumenti di elaborazione ma anche a trovare nuove fonti di dati o sfruttare meglio quelle già utilizzate. Riconoscere i sentimenti o le emozioni, positive e negative, di un utente è un lavoro complesso e difficilmente può essere svolto con software tradizionali o non appropriati. In luoghi come musei, zoo, acquari, parchi giochi è spesso difficile capire il grado di apprezzamento del pubblico dopo la fruizione del servizio e di solito si possono trovare delle pulsantiere in cui si indica il grado di apprezzamento o dei rapidi questionari da compilare. Trovare delle alternativa valide non è facile e una di queste potrebbe essere una raccolta di dati differente, cioè senza una richiesta diretta, quindi ad esempio annotare quante persone si fermano ad osservare una certa opera d'arte, l'analisi delle espressioni facciali o dei commenti espressi dal pubblico, il tutto sempre nel rispetto della privacy delle leggi in vigore. Mettere in atto queste nuove strategie è possibile solo grazie allo sviluppo dell'intelligenza artificiale, sopratutto nell'apprendimento automatico, che ha portato la risoluzione di questi ed altri problemi su un nuovo livello. La proposta di questa tesi rappresenta un' applicazione differente di tecniche di ML e SA e una rielaborazione originale di diversi problemi già affrontati in altri studi. Infatti il tinyML ha avuto poche applicazioni, se non nessuna, nel campo del SA. Tutte le sperimentazioni in questo campo sono eseguite attraverso delle elaborazioni successive alla cattura dei dati, ad esempio ricercando commenti sui social network, o in ambiti diversi, oppure con schede ad alto consumo che non possono essere utilizzate nello scenario proposto, come l'utilizzo di modelli di deep learning eccessivamente grandi. Restando invece nell'ambito delle schede a basso consumo una soluzione alternativa al sistema proposto, e sicuramente implementabile con le tecnologie odierne, è l'uso del cloud o di un server remoto per elaborazioni e inferenze, ma questo risulterebbe in un uso eccessivo di risorse di rete e in eventuale latenza che potrebbe modificare il funzionamento del sistema. Quindi per questo motivo si è scelto di provare a sfruttare le potenzialità dei microcontrollori e in particolare del tinyML che si è sviluppato così tanto proprio per i risultati ottenuti negli ultimi anni.

1.2 Obiettivi

L'obiettivo principale di questo lavoro di tesi è la valutazione dell'efficacia di sistemi di SA implementati in prossimità dei punti di interesse (le opere d'arte di un museo o i punti di osservazione nei parchi naturali) e direttamente su dispositivi embedded, nel percepire ed identificare le emozioni del pubblico. Queste informazioni saranno poi utili a coloro che hanno interesse nella loro gestione per capirne il grado di apprezzamento, quindi tali dati potranno essere sfruttati per migliorare l'esperienza di un utente futuro. Per la realizzazione di un tale sistema, attraverso strumenti di intelligenza artificiale, nell'ambito del TinyML, verrà creato un software in grado di analizzare dati audio e riconoscere un dizionario di parole che saranno associate a sentimenti positivi e negativi, dopodichè questo modello sarà eseguito su una scheda a basso consumo, equipaggiata con un microfono, per valutarne le prestazioni in termini di qualità e complessità. I

risultati dell'applicazione del tinyML applicato nel campo della SA verranno quindi valutati ricreando scenari applicativi verosimili. L'obiettivo ultimo di questo lavoro è mettere a disposizione queste valutazioni che saranno determinanti per l'applicabilità della soluzione e potrebbero quindi essere la base per nuove ricerche.

1.3 Struttura della tesi

Il lavoro di tesi è organizzato nei seguenti capitoli. Nel Capitolo 2 verranno presentate le tecnologie utilizzate per l'elaborazione del processo, hardware e software, in relazione al contesto applicativo di riferimento. Nel Capitolo 3 è presente la descrizione del sistema sviluppato e in particolare dell'architettura e delle principali scelte progettuali. Nel Capitolo 4 è contenuta la descrizione dell'implementazione vera e propria del sistema, mentre i testi eseguiti e i risultati delle prove successive all'implementazione sono riportati nel Capitolo 5. Conclusioni ed eventuali sviluppi futuri sono infine raccolti nel Capitolo 6.

Capitolo 2

Background, tecnologie utilizzate e stato dell'arte

2.1 Tecnologie esistenti

La diffusione degli strumenti di ML e la necessità di offrire prodotti client-oriented porta sempre più gruppi di ricerca ad investire risorse nel miglioramento delle tecnologie esistenti e nella loro fusione con queste nuove esigenze. Sono ormai numerosi i progetti che si possono trovare a riguardo così come i framework e gli strumenti che si possono utilizzare. In questa sezione viene eseguita una panoramica di alcuni progetti rilevanti per comprendere i progressi e lo stato attuale di queste tecnologie, per quindi capire quale sia la base dei risultati su cui si basa il lavoro proposto e di conseguenza mettere in evidenza l'originalità dei risultati ottenuti.

2.1.1 SA con deep learning

Per quanto riguarda la SA gli strumenti che si possono trovare sono ormai numerosi , ma quando ci si riferisce a questo ambito viene però inteso la maggior parte delle volte il campo della Web SA. Lavori come [1], [2] ,[3] ad esempio hanno analizzato i progressi raggiunti analizzando vari progetti a riguardo, questi hanno dimostrato come è possibile applicare l'**intelligenza artificiale** e il Deep Learning in questo ambito per l'analisi di dati provenienti da social network diversi, come Twitter, Facebook o Instagram e prelevare commenti o frasi condivise per valutare il pensiero più comune intorno ad un entità. In particolare le tecnologie moderne allo stato dell'arte attuale hanno risolto problemi molto complessi. Nel campo della SA in questi lavori sono state eseguite delle panoramiche dello stato attuale e dei progressi. Si vede qui anche una comparazione tra vari sistemi per risolvere questi problemi mostrando anche i traguardi nei principali campi della SA. In particolare da questi studi viene rilevato che con gli strumenti odierni le analisi possono essere eseguite su 3 livelli in modo molto accurato: livello di documento, di frase e Aspect-based, cioè l'analisi delle opinioni riguardo ad un'entità su diversi

aspetti all'interno di una frase. Altri progetti sono stati portati avanti per studiare task secondari come analisi dei sentimenti, riconoscimento del sarcasmo o SA multilingua. Il deep learning ha dimostrato di funzionare molto bene in questo tipo di progetti e i risultati a riguardo hanno raggiunto traguardi rilevanti. Altre sottosezioni della SA sono invece ancora in via di sviluppo come la visual SA e l'audio classification nella SA. Sono stati selezionati quindi dei progetti che simboleggiano questi progressi,ad esempio per quanto riguarda la prima, in [4] viene mostrata un'implementazione possibile riguardo il riconoscimento di sentimenti nelle immagini portando ad un miglioramento di precedenti modelli proposti e raggiungendo anche un 91% di precisione. Questi due tipi vengono spesso usati insieme e diversi studi hanno rivelato che la combinazione tra classificazione audio e video funziona molto bene come viene illustrato in [5]. Qui prendendo dei video da Youtube vengono eseguite delle analisi per verificare poi quanto i risultati siano attendibili, unendo le analisi del video con quelle dell'audio.

2.1.2 Classificazione audio con tinyML

Riguardo alle tecnologie odierne e i progetti analizzati si è visto invece che la classificazione di dati audio attraverso tinyML applicato alla SA è un campo poco esplorato. In generale invece la classificazione audio in schede a basso consumo, quindi con l'uso di tinyml, è un argomento studiato molto negli ultimi anni e che sta portando a risultati significativi che possono essere ritrovati in diversi studi ma anche nelle tecnologie che utilizziamo nella vita di tutti i giorni. Essendo le tecnologie e i framework in aumento il lavoro di sviluppo di modelli di neural network per questo tipo di device è sempre più diffuso, infatti sviluppare questo tipo di applicazioni è sempre più facile così come la sperimentazione di nuove soluzioni. Esempi lampanti si ritrovano negli assistenti locali, come Google o Alexa, quando si fa l'uso della "parola di risveglio", infatti in questo caso il device deve restare sempre attivo eseguendo inferenza continua nell'attesa di riconoscere la parola. Risulta chiaro che questo lavoro di inferenza deve essere eseguito in locale, quindi sul dispositivo, poichè altrimenti significherebbe un abuso della rete così come un appesantimento dei server remoti. Questi esempi riscontrabili nella vita reale fanno capire quanto queste tecnologie siano ormai avanzate. Nuovi studi vengono eseguiti continuamente per sperimentare nuove tecniche per migliorare i risultati correnti, ad esempio in [6] gli autori hanno mostrato un esempio della classificazione della pronuncia di parole attraverso una scheda a basso consumo, in questo caso Arduino 33-BLE, con l'uso del framework Edge Impulse. In questo studio viene mostrato quindi come può essere implementato il classificatore che nel caso proposto riconosce tre parole con un'accuratezza fino al 98.75%. Sono stati raggiunti risultati significativi anche nella sperimentazione di nuove tecniche, come in [7] in cui viene proposta una nuova strategia per la ricerca della configurazione migliore di rete ,attraverso un particolare setup, ma sopratutto a basso consumo. [8] è uno studio che rappresenta un'importante punto di partenza per il lavoro qui proposto, qui vengono comparati i metodi più tradizionali di SA basati sull'audio classification con un nuovo modo di concepire il problema. I metodi tradizionali si basano sul riportare il contenuto audio in testo e poi analizzarlo , mentre qui si riporta una tecnica basata sul keyword spotting della singola parola, mostrando

che l'espressione del sentimento la maggior parte delle volte si basa un dizionario ridotto di parole che sono rappresentative di una certa emozione e che anche isolate possono guidare nella estrapolazione del sentimento. Questo dizionario di parole è stato creato attraverso un algoritmo che prendendo in input dati di commenti, recensione e frasi del mondo reale identifica quali parole accomunano i vari sentimenti. Infine questa tecnica ha dimostrato di avere buoni risultati applicativi. Dall'analisi di questi studi emerge quindi che si sta progredendo anche riguardo la sentiment analyis per dati audio ma i sistemi per ora esistenti svolgono attività non avanzate o comunque non sono applicabili in tutti i campi. I sistemy tinyMl per dati audio, ad esempio, non sono sviluppati per la SA e hanno limitazioni come ad esempio il dizionario limitato. In questo lavoro si proverà dunque a superare queste limitazioni.

2.2 Analisi delle componenti teoriche e delle tecnologie utilizzate

L'idea progettuale descritta utilizza concetti informatici e matematici che verranno descritti in questo sezione, insieme ai framework ed alle tecnologie utilizzate. Questo progetto si sviluppa infatti partendo da un studio approfondito dell'intelligenza artificiale e in particolare di ML e **tinyML**, facendo seguire, nella fase applicativa, lo studio degli strumenti scelti da utilizzare.

Vi sono infatti svariati software e framework che possono essere utilizzati per l'implementazione di algoritmi di ML. Framework come Keras, Caffe, TensorFlow implementano le operazioni matematiche che sono alla base del ML e consentono all'utente di creare programmi interagendo solamente con linguaggi ad alto livello come Python. Per quanto riguarda invece lo sviluppo su microcontroller verranno usati principalmente TensorFlowLite e il tool **EdgeImpulse**, un software recente e innovativo che offre un ambiente sul web per eseguire tutte le operazioni riguardanti lo sviluppo di modelli di embedded ML, dalla raccolta dati al caricamento dell'algoritmo sul microcontroller. Verranno utilizzati concetti basilari sui sistemi **Arduino** e programmazione C, infine si discuterà su concetti dell'Internet of Things per la comunicazione del microcontroller con un eventuale server.

2.2.1 L'intelligenza artificiale

L'intelligenza artificiale,IA è lo studio di costruire o programmare i computer per abilitarli a fare ciò che le menti umane possono fare [9], è una disciplina appartenente all'informatica che si è evoluta a pari passo con la nascita dei primi elaboratori informatici, arrivando ad imporre il suo utilizzo sia in ambito di processi aziendali che nella vita quotidiana. Il test di Turing ha segnato la nascita dell'intelligenza artificiale poiché per la prima volta poneva il problema riguardo alla capacità di pensiero di una macchina. Questo è tutt'ora utilizzato per definire quelli che sono gli obiettivi dell'intelligenza artificiale e consiste nel porre un umano davanti al computer e vedere se sia in grado, facendo delle domande, di capire se si stratta di un uomo o un computer. Dalla formula-

zione di questo test ad oggi questa branca dell'informatica si è molto evoluta e sono tanti gli studi eseguiti alla ricerca di continui miglioramenti, andando spesso anche incontro a pesanti critiche di carattere etico-filosofiche. L'IA evolvendosi si è poi diramata in diverse discipline minori e provando a fare una classificazione, in accordo con Gerevini [10], possiamo distinguere come aree di interesse dell'IA:

- Voce;
- Linguaggio naturale;
- Ragionamento automatico;
- Conoscenza;
- Apprendimento;
- Robotica;
- Visione.

2.2.2 Apprendimento automatico

L'apprendimento automatico o **ML**, è una delle aree di specializzazione dell'intelligenza artificiale che più ha avuto successo negli anni, divenuta oggetto di studio di tanti ricercatori, tra informatici e matematici, è l'area di conoscenza in cui si pone il lavoro di questa testi. Un algoritmo di ML è un processo computazionale che usa dati di input per realizzare un compito desiderato senza essere letteralmente programmato [11]. Fondamentale è il processo di adattamento o training durante il quale l'algoritmo, attraverso dati di training, si autoconfigura per rispondere a input già visti o nuovi tentando di associare ogni input all'output desiderato e quindi eseguendo un processo di generalizzazione. Per capire i vantaggi dell'approccio ML si immagini di creare un programma che generalizzi e riconosca il comportamento dell'onda sonora associata alla pronuncia di una parola, si intuisce subito come sarebbe un lavoro molto complesso che probabilmente genererebbe un codice pesante e poco funzionale. In generale è noto come gli algoritmi di ML, in particolare di **Deep Learning**, funzionano bene nel processo di generalizzazione e all'aumentare dei dati come si vede in figura 2.3.

Per apprendere questi algoritmi utilizzano delle collezioni di dati, **dataset**, che possono essere create o catturate. Andando ad approfondire il concetto se ne analizzano le diverse categorie, solitamente si distinguono infatti tre tipologie di apprendimento automatico:

- **Apprendimento supervisionato**;
- **Apprendimento non supervisionato**;
- **Apprendimento per rinforzo**.

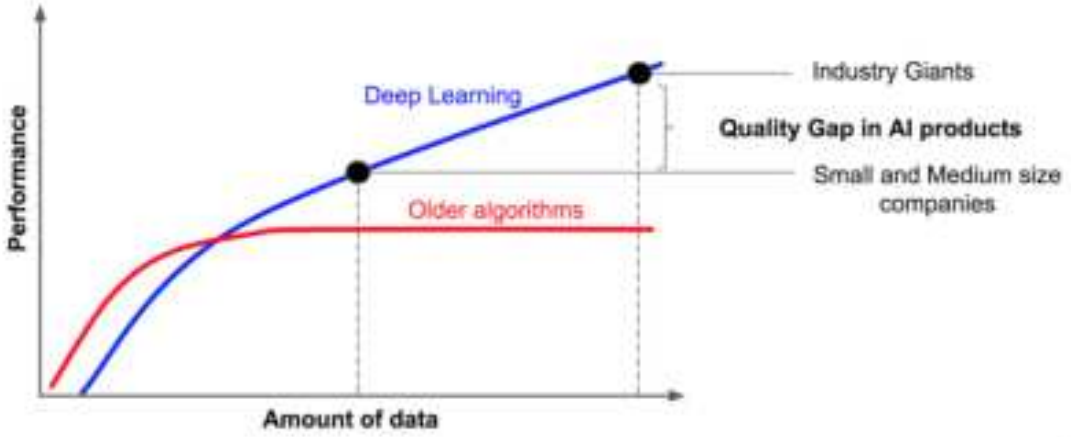


Figura 2.1: *Deep Learning vs algoritmi obsoleti*

Un classico e semplice esempio di apprendimento supervisionato è la classificazione di un input in etichette definite a priori, ad esempio un algoritmo che riesce a distinguere quando un'immagine che riceve in input è un cane e un gatto. Questo tipo di classificazione si contrappone a quella tipica dell'apprendimento non supervisionato di cui un esempio tipico è il clustering. In questo tipo di classificazione l 'algoritmo studiando i dati iniziali definisce dei cluster dividendo i vari dati in base alle loro caratteristiche, quindi non vi sono etichette definite a priori ma gruppi di input raggruppati per similarità. Infine l'ultima tecnica , apprendimento per rinforzo, è quella che si differenzia di più dalle altre. E' lo scenario in cui si vuole addestrare un agente autonomo al conseguimento di un determinato obiettivo tramite passi sequenziali che dipendono dal sistema circostante e lo modificano. L'addestramento in modo approssimativo procede a valutare la bontà di una delle azioni sequenziali per migliorare quelle future. Queste tecniche trovano praticamente applicazioni qualsiasi ambito e la tipologia di apprendimento in cui si colloca l'idea progettuale proposta è di tipo supervisionato. Per approfondire il concetto di apprendimento automatico bisogna prima analizzarne altri come quelli di **modello** e **caratteristiche**. Un modello di apprendimento automatico è in sostanza una rappresentazione matematica che riceve dei dati in input che servono al modello per imparare a riconoscere, appunto, le caratteristiche dei dati per eseguire, ad esempio, una classificazione. In particolare vengono solitamente distinti modelli che rappresentano l'apprendimento supervisionato o non supervisionato. Per quanto riguarda l'apprendimento supervisionato vi è:

- Modello di regressione: Vengono definiti di questa tipologia i modelli che non non hanno un insieme finito di valori in output, quindi ad esempio modelli di previsione che possono restituire in output qualsiasi valore numerico;

- Modelli di classificazione: Al contrario dei precedenti hanno un'insieme finito di classi che possono restituire in output.

Ci sono poi i modelli che vengono perlopiù inseriti nella categoria di apprendimento non supervisionato:

- Modello di clustering: modelli che presi in input dei dati li raggruppano per valori simili delle caratteristiche;
- Modello di associazione: servono per trovare associazioni tra dati, ad esempio sapere se una persona con certe caratteristiche comprerà un oggetto piuttosto che un altro;
- Modello di riduzione dimensionale: modelli che vengono usati per generalizzare i dati e quindi diminuirne la complessità.

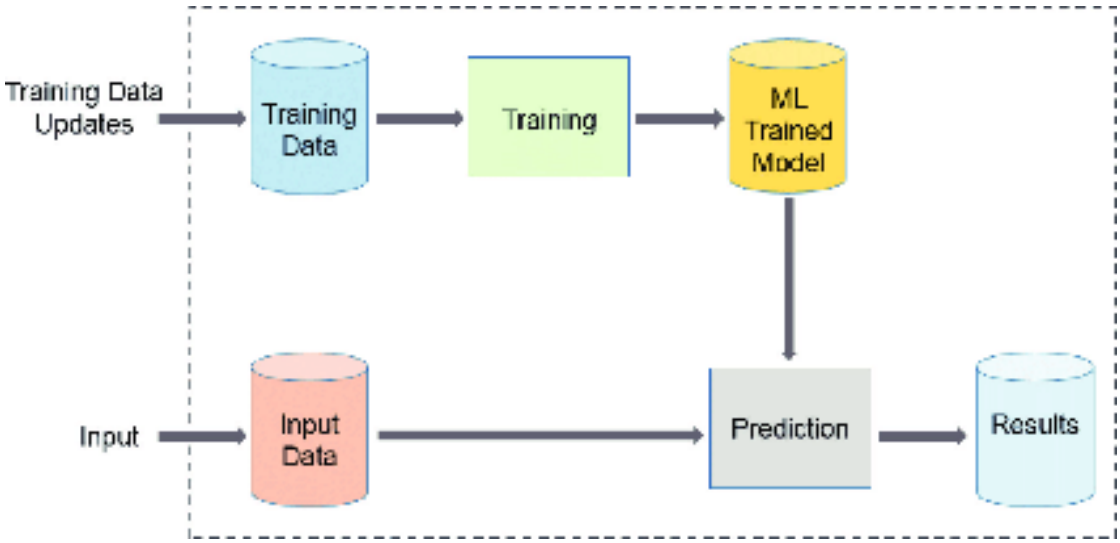


Figura 2.2: *Generico workflow per un modello*

L' **addestramento** di un modello si può vedere dunque come la scelta dei parametri migliori. Questi sono dei valori numerici che rappresentano il modello stesso determinandone i risultati. I parametri vengono scelti in base ai risultati e attraverso questi si sceglie se continuare o meno l'addestramento. I modelli infatti possono avere risultati accurati o meno e la metrica solitamente più utilizzata per misurare la qualità è l'accuratezza. Queste metriche poi si differenziano nel modo in cui vengono utilizzare a seconda del modello. Dunque i passaggi necessari per la creazione di un modello di ML possono essere visti come:

- Studio del problema;

- Raccolta dati;
- Preparazione dei dati;
- Scelta del modello;
- Addestramento del modello;
- Valutazione del modello.

Nella maggior parte dei casi questi passaggi vengono ripetuti tornando indietro, ad esempio per aggiornare il dataset o rieseguire l'addestramento. In figura 2.2 possiamo vedere il generico workflow che riguarda un modello, utilizzando prima un insieme di dati per addestrare il modello, dopodichè vengono eseguite le inferenze su altri dati sconosciuti.

2.2.3 tinyML

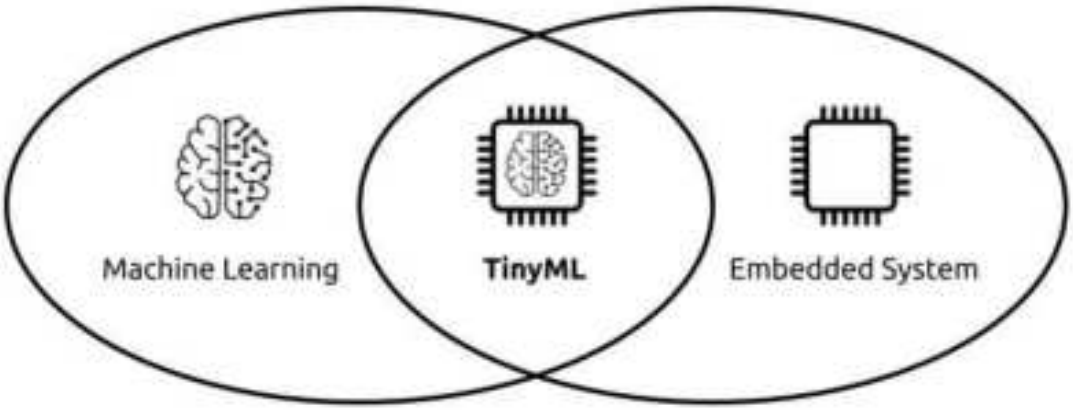


Figura 2.3: *tinyML*

Con tinyML si intende quella branca dell'apprendimento automatico con un uso ridotto o minimo delle risorse di una macchina che solitamente è rappresentata da un sistema embedded. Questo tipo di approccio si è diffuso in un modo talmente rapido che le risorse a disposizione aumentano da un giorno all'altro. Questo è dovuto all'utilità che il tinyMl ha trovato in ogni campo e ai vantaggi che ne derivano rispetto ad un approccio tradizionale. Si può trovare molta documentazione di ricerca in merito come in [12] e [13]. Solitamente infatti i sistemi si basano su device **IOT**(Internet of things) intelligenti o sensori che comunicano col cloud. Il classico esempio del perchè del TinyMl riguarda i soliti device intelligenti come Alexa. In questi casi non si può usare la parola di risveglio e aspettare che i dati mandati al cloud vengano elaborati e restituiti indietro. La cosa migliore ovviamente sembra eseguire l'algoritmo di ML in loco. Stesso discorso si può

fare ad esempio per un'immagine o qualsiasi altro dato sui cui si vuole applicare il ML, ad esempio un sistema di un ospedale installato per rilevare anomalie del paziente deve essere sempre attivo e rilevare subito un'eventuale anomalia, mentre nel caso tradizionale con il cloud potrebbe anche esserci un problema con la rete e il sistema diverrebbe praticamente inutile. Questi sistemi devono poter essere attivi sempre, quindi oltre ad essere molto piccoli devono garantire un uso ridotto di energia, infatti i microcontrollori solitamente consumano energia nell'ordine dei microwatt o milliwatt, mentre le CPU normali nell'ordine dei watt. I vantaggi del tinyMl sono molti e dipendono anche dai casi d'uso ma dei vantaggi universalmente riconosciuti sono:

- Bassa latenza: Come già detto l'esecuzione delle inferenze in locale è più veloce rispetto a una esecuzione remota e il software è più leggero, quindi questo porta ad un ottenimento dell'output con bassa latenza;
- Basso consumo: I microcontroller hanno un consumo molto ridotto di energia essendo anche progettate per essere schede a basso consumo;
- Basso consumo di banda: Non usando la rete non ne interferiscono col suo normale funzionamento evitando l'invio continuo di dati;
- Privacy: Essendo l'esecuzione locale è chiaro che non bisogna inviare i dati a nessun server con il rischio che chi non ne ha diritto entri in possesso di dati sensibili.

Ovviamente la scelta migliore sarebbe di addestrare il modello sul cloud o comunque non sul microcontroller e poi trasferirlo. Riuscire però a superare le sfide proposte dal tinyMl non è facile poichè queste comprendono la minimizzazione nell'uso di memoria e quindi la conversione di algoritmi dell'ordine dei megabyte o gigabyte in algoritmi che spesso non superano l'ordine dei kilobyte. In più i microcontroller utilizzati non supportano solitamente i numeri in virgola mobile. Superare queste sfide al giorno d'oggi è possibile ma bisogna tenere conto che la complessità dei problemi risolvibili con algoritmi ottimizzati in questo modo è ridotta. Ad esempio problemi affrontabili possono essere il riconoscimento vocale di parole, il riconoscimento di gesti con un accelerometro o problemi di anomaly detection. E' possibile constatare come molte energie sono state spese in questa direzione, ad esempio framework come TensorFlow che sono stati adattati a questi microcontroller nella versione TensorFlow Lite e nuovi microcontroller sviluppati per supportare queste tecnologie come la scheda Portenta di Arduino.

2.2.4 Reti neurali

Le reti neurali artificiali sono nate nel momento in cui l'uomo a ha provato a imitare il funzionamento dei neuroni del cervello umano attraverso una macchina. Più precisamente nel 1943 gli scienziati McCulloch e Pitts descrivono teoricamente la prima rete neurale elementare che lavorava su dati binari. Le reti neurali artificiali imitano i neuroni e le sinapsi del cervello umano, così come la capacità che hanno i neuroni a ricevere più sinapsi contemporaneamente. Ma il concetto forse più importante imitato è quello della flessibilità dei collegamenti e quindi delle sinapsi. Infatti il cervello umano presenta

dei collegamenti sinaptici fitti ma flessibili che variano a seconda della stimolazione che ricevono.

Il Perceptron

La prima rete neurale sviluppata fu a partire dal **Perceptron**, questo è un classificatore binario rappresentato matematicamente come:

$$f(x) = \chi(\langle w, x \rangle + b)$$

Qui il vettore input viene moltiplicato scalarmente con il vettore pesi w e dopodichè viene sommato il bias b che può essere visto come la soglia di attivazione. La funzione $\chi(y)$ può rappresentare diverse funzioni come la funzione di Heaviside. In sostanza si tratta di un classificatore binario che deve essere addestrato attraverso un determinato algoritmo che presi dei dati in input addestra il classificatore. Questo algoritmo conosciuto con il nome di algoritmo di Perceptron è dimostrato convergere nel caso i dati possano essere divisi linearmente. Il classificatore può essere visto infatti come una retta o un **iperpiano**, o meglio, c'è un associazione binaria tra il classificatore e l'iperpiano che rappresenta. Immaginando il caso bidimensionale quando l'iperpiano è una retta, se i punti possono essere divisi linearmente allora questo algoritmo troverà come classificatore finale la retta che suddivide questi punti. In figura 2.4 si può vedere una rappresentazione generica

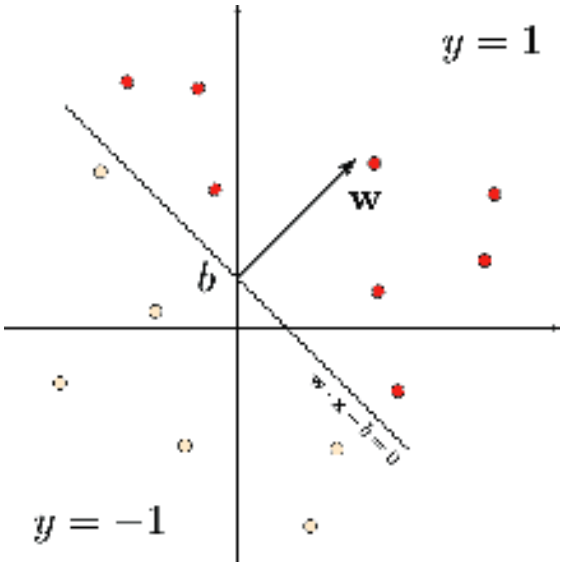


Figura 2.4: *Rappresentazione di un generico iperpiano rappresentante un classificatore lineare*

dell'iperpiano che classifica i punti a seconda dell'output y , se questo è positivo i punti stanno nella parte superiore o positiva dell'iperpiano e se negativo nella parte inferiore. Come si può intuire dalla figura ad esempio anche un punto bianco potrebbe trovarsi tra

i punti rossi in modo che la retta non può più suddividerli ma potrebbe esserci bisogno di un classificatore non lineare che possa assumere la forma di una curva.

Costruzione di una rete neurale

Nonostante il successo del perceptron questo interesse nel suo studio durò poco tempo ma mise le basi per gli studi futuri, i quali potendo contare su macchine molto più potenti rispetto a quando nacque il perceptron resero possibile l'addestramento di reti che erano formate da più livelli e quindi che non si limitavano a risolvere problemi lineari. Quando più neuroni vengono aggregati allora si forma una rete neurale, che aumenta la sua complessità quanto più la rete diviene fitta. Quando un neurone riceve in input più ingressi questi vengono sommati con il peso dell'arco che attraversano e il neurone va ad attivarsi se questa somma supera una certa soglia che dipende dalla funzione di attivazione, nel caso di n neuroni e rete fully connected allora il valore in output del neurone m nel livello successivo è:

$$y_m = Activation(W_{m1X1} + W_{m2X2} + W_{m3X3} + ...W_{mnXn} + b_m)$$

La figura 2.5 mostra una rete generica con dei livelli nascosti che in questo caso sono 2. Se

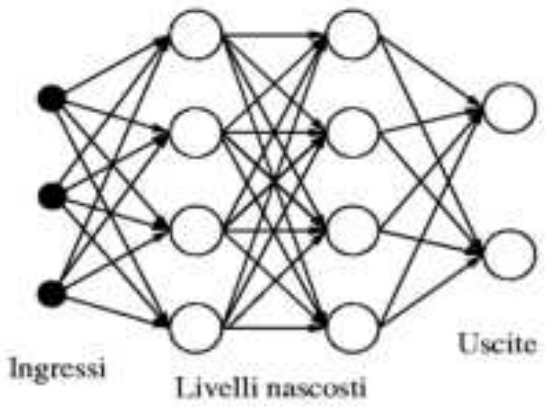


Figura 2.5: *Rete neurale con due livelli nascosti*

i neuroni propagano informazioni solo in avanti allora sono chiamate reti **FeedForward**. I neuroni non sono collegati con neuroni dello stesso livello. Il neurone può assumere diverse forme come quella del perceptron precedentemente descritto nel qual caso la rete prenderà il nome di MLP(multilayer perceptron). Come abbiamo detto però ogni neurone rappresenta un iperpiano ed è quindi lineare e se ,come in questo caso si hanno dei neuroni in and tra di loro , allora un punto che rispetta l'equazione di più iperpiani risulterà nella zona di intersezione tra di loro,da questo si capisce che vengono superati

i limiti del neurone singolo. Interessante notare come con i valori dei neuroni di ogni livello, in una rete completamente connessa, si possono ottenere con una moltiplicazione tra matrici tra la matrice dei pesi w di ogni arco moltiplicata per il vettore dei valori dei neuroni del livello precedente. Dopo che viene definita la struttura della rete neurale con numero di nodi per livelli e numero di livelli, si passa ad una fase essenziale e cioè l'addestramento della rete. Il tipico addestramento di una rete neurale utilizza il concetto di gradiente di una speciale funzione di errore che vuole essere minimizzata. Ma come insegnare ad una rete gli errori che sta commettendo e sopratutto come propagare l'errore a tutti i livelli è un lavoro che innesca diversi procedimenti matematici piuttosto complessi. Il primo concetto che deve essere introdotto è la propagazione all'indietro, o **backpropagation**, la quale si serve del concetto di errore rispetto al caso ideale. In poche parole viene calcolato l'errore su tutti i nodi di output e questo valore equivale a quanto il risultato della rete neurale attuale si differenzia dal risultato ideale su un determinato dato input. Una funzione che si usa spesso per calcolare questo errore è la squared-error function che equivale a:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

Questo valore è elevato al quadrato poichè deve rappresentare una distanza tra due punti e l'errore totale della rete è la somma degli errori così calcolati per ogni nodo output, ma ci sono anche altre funzioni di errore che spesso vengono usate come Mean-squared-error, Categorical-crossentrophy o altre ancora. Il primo passo che si fa a questo punto è di correggere i valori dei pesi nei nodi output seguendo il valore del gradiente e quindi spostando il valore dei vari pesi w in modo da minimizzare la funzione di errore. Quindi prendendo i nodi nel livello output, e i rispettivi pesi che conducono a questo nodo, osservo quanto cambia il valore di errore quando cambiano questi pesi e dopo aver scelto un **learning rate**, che è numero solitamente molto basso nell'ordine dei centesimi o millesimi, mi sposto nella direzione contraria a quella del gradiente per minimizzare l'errore. Eseguire la backpropagation vuol dire portare questo cambiamento fino ai livelli iniziali, quindi spostandosi al penultimo livello so che ogni nodo influisce sul risultato di più nodi nel livello finale, quindi la derivata che si andrà a calcolare in questo livello prende i risultati delle derivate calcolati nell'ultimo livello e così via fino ad arrivare ai nodi iniziali aggiustando via via tutti i pesi.

Algoritmi di ottimizzazione

In figura 2.6 viene raffigurata una semplice funzione di errore a due variabili che mostra come può essere il percorso per arrivare al minimo della funzione di errore che segue sempre la **discesa del gradiente**. Da questo disegno si può evincere anche come questa discesa può essere non sempre esatta poichè si può ricadere in minimi locali, punti di stallo e così via. Per questo il learning rate è molto importante e una sua giusta calibrazione può evitare errori di questo tipo ma che in ogni caso non si possono evitare, sono state sviluppate però altre tecniche più avanzate per ovviare a questo tipo di problemi come ad esempio l'introduzione della derivata seconda per valutare anche la velocità con

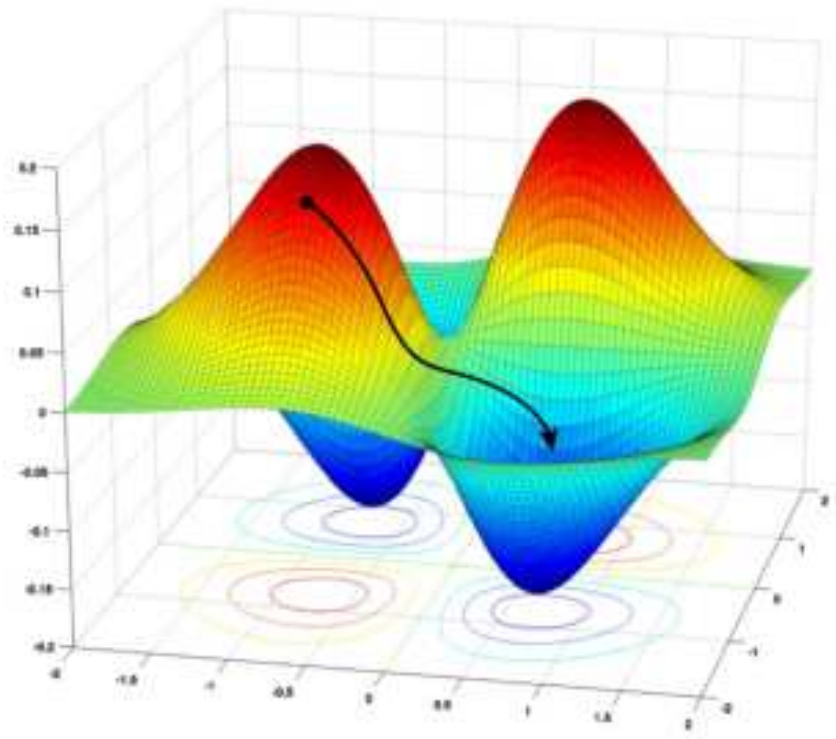


Figura 2.6: *Possibile percorso di ricerca del minimo su una funzione di errore*

cui cambia la pendenza. Esistono poi diversi algoritmi di discesa del gradiente, tra questi ci sono stochastic gradient descendent, batch gradient descendent, mini-batch gradient descendent. La differenza principale tra i tre è rappresentata dai dati che vengono usati nei diversi stage di allenamento: Nel primo caso ogni stage utilizza un solo dato di allenamento per ricalcolare i parametri, questo offre buone prestazioni di memoria ma peggiori per la velocità ed è stocastico perché esegue una stima probabilistica su quello che può essere il valore del gradiente, molto spesso è difficile che raggiunge il vero minimo ma finisce per fluttuarci intorno. Il secondo invece calcola i parametri in ogni stage su tutti i dati di allenamento, quindi è meno pesante ma ha un'occupazione di memoria rilevante. Infine l'ultimo esegue ogni stage su un batch ridotto di dati scelto casualmente, quindi offre prestazioni in equilibrio tra la prima e la seconda tecnica, spesso come il primo caso anche questo finisce per fluttuare intorno al minimo poiché prende un piccolo sottoinsieme di tutti i dati ogni volta. Esistono poi tanti altri algoritmi di ottimizzazione nati appunto per ovviare a problemi di questo tipo, anche se quello forse più usato è **ADAM**. ADAM è l'acronimo di Adaptive Moment Estimation ed è stato presentato nel 2015 in [14]. ADAM funziona molto bene con quantità elevate di dati ed offre ottime prestazioni computazionali, l'algoritmo utilizza la media esponenziale e ponderata dei gradienti calcolati precedentemente per velocizzare la discesa del gradiente

verso il minimo. La forza di questo algoritmo sta nel fatto che la discesa esegue piccoli passi quando si trova in un punto vicino al minimo globale, ma esegue passi più ampi quando si trova vicino a un minimo locale e quindi evita situazioni indesiderate come invece potrebbero fare gli algoritmi precedenti. E' stato dimostrato come anche questo algoritmo al crescere dei dati offre le prestazioni migliori in termini di efficienza rispetto agli altri ottimizzatori.

Funzioni di attivazione

Le funzioni di attivazione come anticipato nelle pagine precedenti servono far rispondere il neurone solo in presenza di una determinata soglia di stimolazione. L'esempio di una funzione di attivazione semplice è quella lineare come ad esempio $f(x) = x$ e il cui grafico è ovviamente rappresentato da una semplice retta. Sigmoid è un'altra funzione che è molto utile nel caso si vogliano rappresentare probabilità per la sua capacità di offrire un risultato che sta sempre tra 0 e 1, nello stesso modo in cui si rappresenta la probabilità. Quando invece la probabilità è divisa in più elementi, quindi ad esempio nella classificazione, la funzione **softmax** suddivide la probabilità in un numero di classi maggiore di 2, questa funzione infatti è di fatto usata nell'ultimo livello delle reti neurali che risolvono problemi di classificazione. La funzione tangente può anche essere usata come funzione di attivazione ed ha una forma sigmoide come l'altra che ricorda una s, la differenza sta però nel range in cui vengono mappati i risultati che va da -1 a 1. Altre funzioni di attivazione sono state testate negli anni e in particolare una che ha raggiunto più successo di tutte ed è praticamente usata in quasi tutti i problemi che fanno uso di reti convoluzionali è la **ReLu** che sta per Rectified Linear Unit. La funzione semplicemente mappa tutti i valori che non sono positivi in zero, questo è stato dimostrato dare ottimi risultati in alcuni tipi di problemi ma allo stesso tempo annulla qualsiasi valore negativo, quindi potrebbe non mappare i valori negativi in modo appropriato. Per provare a risolvere questo problema è nata la funzione LeakyRelu che mappa tutti i valori negativi moltiplicandoli per un fattore, che dovrebbe essere tra 0 e 1, e estendendo quindi il codominio della funzione fino a $-\infty$. In figura 2.7 si possono vedere gli andamenti grafici delle funzioni di attivazione descritte.

2.2.5 Reti neurali convoluzionali

Le reti neurali convoluzionali hanno tutte le caratteristiche delle reti tradizionali, con l'unica differenza notevole tra le CNN e le reti neurali artificiali tradizionali è che le reti neurali convoluzionali sono utilizzati principalmente nel campo del riconoscimento dei pattern[15]. Questo tipo di reti avvicina ancora maggiormente le capacità delle macchine a quelle del cervello umano, in particolare modo poichè sono state costruite imitando le modalità di riconoscimento che utilizza il cervello. La corteccia visiva infatti non riconosce interamente ciò che vede ma analizza piccole aree unendo poi i risultati e ottenendo un'immagine completa di ciò che vede. Il funzionamento delle reti convoluzionali invece si basa sull'individuazione delle features, o caratteristiche, rilevanti in una certa sequenza di dati e il loro addestramento consiste nel fornire alla rete un certo set

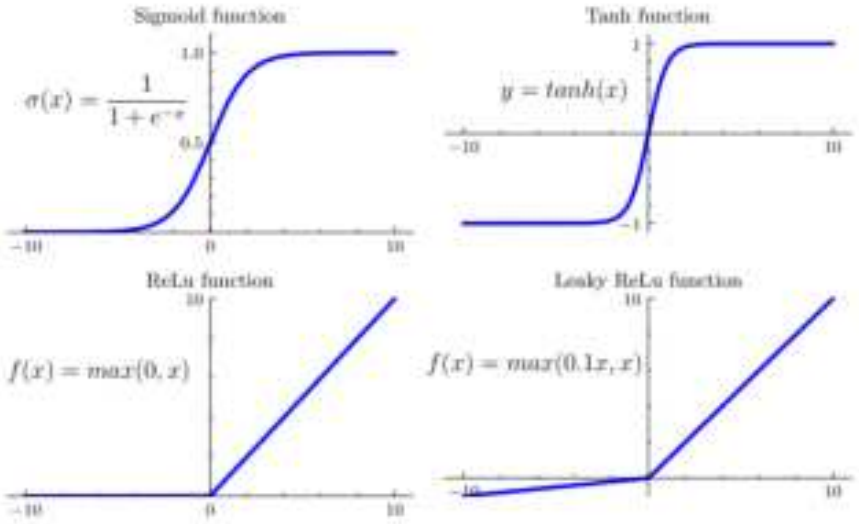


Figura 2.7: *Alcune funzioni di attivazione non lineari*

di dati di allenamento in cui riconoscendo i pattern ricorrenti e associandoli alle classi la rete imparerà a classificare in futuro dati sconosciuti. Molto importante è quindi che la rete riesca a vedere bene e riconoscere queste caratteristiche e per fare ciò si usano delle tecniche di preprocessing con l'intento di metterle in risalto. Un altro vantaggio importante delle reti convoluzionali è l'addestramento di queste reti insegna loro anche a riconoscere le dipendenze dei dati spaziali e temporali, quindi non si ottiene il risultato solo in base , ad esempio, ai vari bit di un'immagine, i quali potrebbero non avere nessun nesso preso singolarmente, ma potrebbero avere un legame se visti nell'insieme. In più per la capacità che hanno di rilevare le caratteristiche rilevanti i dati da processare vengono di molto ridotti ponendo l'attenzione solo su ciò che davvero è importante. Grazie a queste caratteristiche le CNN sono divenute le reti neurali più studiate e utilizzate. In figura 2.8 si può osservare una rete convoluzionale per riconoscere immagini di veicoli, queste reti sono formate da diverse tipologie di livelli che si susseguono. Solitamente sono composte da una parte che serve al rilevamento delle features e gli ultimi livelli che servono alla classificazione.

Livello convoluzionale

Nelle reti convoluzionali vengono usati diversi tipi di livelli di processamento, il più importante è il livello convoluzionale. Esistono diversi livelli di questo tipo che si differenziano in base alla dimensionalità con cui operano, sicuramente il tipo più diffuso è 2D, ma per venire incontro ad altre esigenze sono stati pensati anche i livelli 1D e 2D. La dimensionalità però non indica quella del filtro ma del modo in cui si muove il filtro, ad esempio nei livelli convoluzionali 1D il filtro può avere due dimensionalità,

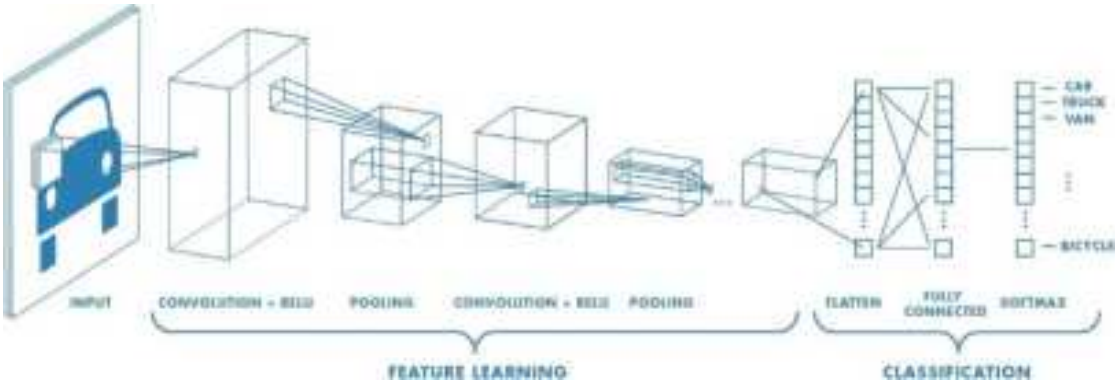


Figura 2.8: *Esempio di rete convoluzionale, fonte [16]*

quindi un array bidimensionale, ma può muoversi solo lungo un asse. In figura 2.9 si

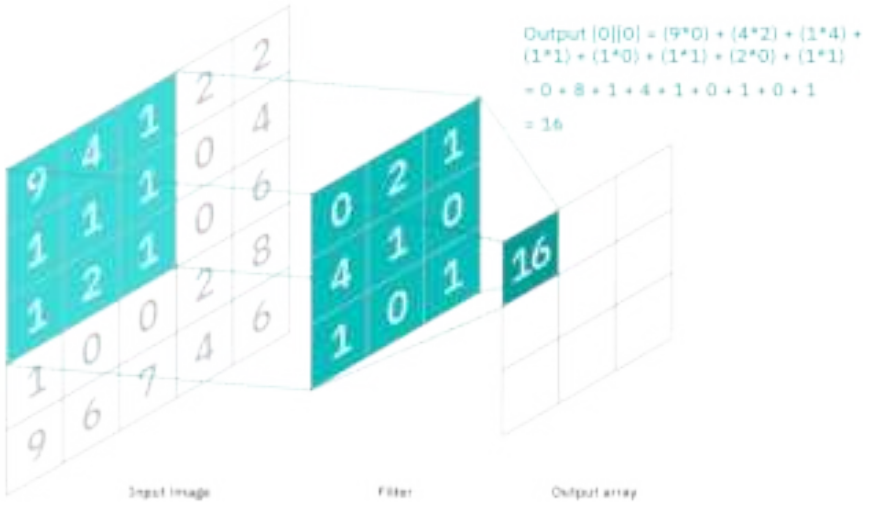


Figura 2.9: *Funzionamento di un filtro in una CNN, fonte [17]*

vede il funzionamento intuitivo dei filtri nel caso di uno spostamento(stride)=1 e senza uso di padding, prendendo in esempio un'immagine $5x5$ e un filtro $3x3$ si ottiene in output un $3x3$ poichè il filtro scorre sull'immagine muovendosi prima sull'asse orizzontale e poi verticale quando non incontra più movimento. Ogni volta che la matrice si posa su una porzione di immagine il valore del filtro corrispondente viene moltiplicato con il valore dell'immagine in quella posizione, i valori vengono sommati e per ogni posizione del filtro si ottiene una valore dell'array finale. Le immagini hanno solitamente però una dimensione in più che è il numero di canali(RGB). Il filtro in questo caso ha 3 dimensioni ma si muove solo sugli assi x e y e viene applicato contemporaneamente su tutti e tre gli strati e i valori per i 3 canali sommati dando lo stesso in output un vettore

bidimensionale(figura 2.10. Quando la dimensionalità del filtro non corrisponde a quella dei dati si possono usare delle tecniche di padding per farle combaciare aggiungendo ad esempio righe o colonne di zeri. Ci sono poi quelli di tipo 1D che solitamente sono usati per i dati in dipendenza dal tempo, come ad esempio dati audio o 3D usati per analisi IRM o stima di distanze. I livelli convoluzionali funzionando così riescono a catturare le features ad alto livello, per esempio, parlando di immagini, vengono catturate nei primi livelli caratteristiche come archi,linee e tratti semplici che nei livelli successivi vengono composi tra di loro per rilevare forme sempre più complesse, per questo motivo la profondità della rete è molto importante per problemi complessi.

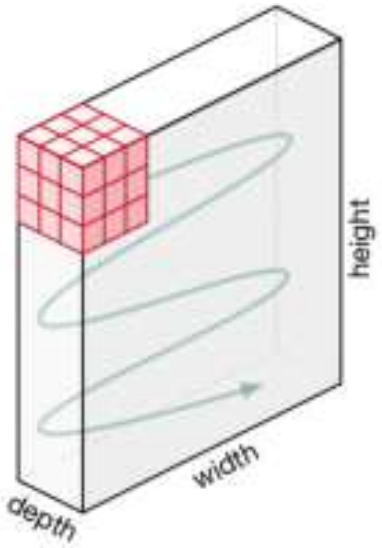


Figura 2.10: *Spostamento del filtro 2D su tre canali, fonte [16]*

Livello di pooling

Questo tipo di livello serve principalmente per estrarre le features rilevanti poichè attraverso una compressione dei dati l'output presenterà solamente le caratteristiche che più pesano nella classificazione. In più con questo tipo di compressione, così come il livello convoluzionale, riduce i dati che devono essere processati. L'output di questo livello dipende dai dati e da alcuni parametri come la grandezza del pool e la grandezza di ogni passo eseguito. Esistono tre tipi di livelli di pooling principalmente usati:

- Max-Pooling: Prende, nello spazio di pool, il valore maggiore. E' quello più usato ed anche quello che funziona meglio poichè elimina eventuali dati di disturbo e allo stesso tempo riduce la grandezza dei dati;
- Average-Pooling: Nello spazio di pool prende tutti i valori ed esegue una media, dando quindi peso a tutti i valori;

- Min-Pooling: Come il Max-Pooling ma prendendo il valore minore. Offre quasi sempre prestazioni peggiori se comparato con gli altri due.

I livelli di pooling solitamente seguono quelli convoluzionali, infatti il livello convoluzionale vero e proprio è solitamente rappresentato dall'unione di questi due. Nelle reti

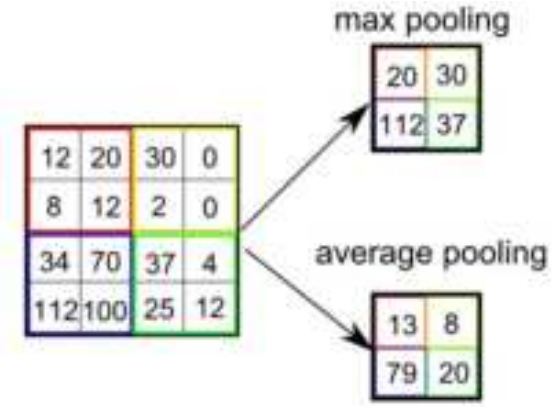


Figura 2.11: *Applicazione di due diverse tipologie di pooling, fonte [16]*

convoluzionali vengono usati anche i livelli completamente connessi, questi di solito sono posti nel livello finale ed utilizzano la funziona di attivazione max-pool per la classificazione e i dati prima di arrivare a questo livello vengono appiattiti per poter essere leggibili.

2.2.6 Raccolta e operazioni sui dati

I dati sono una delle componenti essenziali per ottenere un modello di ML. Per come sono state ideati questi algoritmi apprendono dai dati creati o già esistenti in natura, e dopo aver visto abbastanza dati la rete si può definire addestrata, anche se non esiste un limite effettivo della durata di un addestramento. I dati possono essere immagini, file audio, parole, testi e così via e prima di essere dati al computer viene eseguita una fase di **preprocessing**, cioè vengono eseguite delle operazioni sui dati con lo scopo di renderli maggiormente leggibili alla macchina o migliorarne la qualità. Dopo aver ottenuto una quantità considerevole di dati questi non saranno usati tutti per l'addestramento ma suddivisi in:

- Training Set : Questo set di dati è l'unico che davvero il modello "vede" l'addestramento. Quindi deve essere ben costruito per fare in modo che il modello sappia rispondere a tutti i dati che vede nel mondo reale;

- Validation set: Viene usato durante l'addestramento ma ha una funzione diversa dal training set, questo set è usato per settare gli iperparametri ad alto livello e vedere ad ogni stage se l'addestramento sta andando nella direzione giusta e che quindi sappia generalizzare correttamente;

- Test Set: Sono i dati che vengono utilizzati quando l'addestramento è terminato per vedere le prestazioni effettive e le risposte del modello a dati nuovi.

La suddivisione dei dati nei tre insieme è una scelta arbitraria, anche se solitamente viene usato un 80% dei dati per il training set, il 20% per il testset per poi suddividere di nuovo il primo in training set e validation set.

Preprocessing dei dati ed estrazione delle features

Dopo la creazione del dataset i dati non sono pronti ad essere consegnati alla rete neurale, o meglio, si potrebbe anche farlo ma con risultati che non sono sicuramente ottimali. A seconda dei dati da gestire infatti vengono applicate delle operazioni sui dati. Se sono immagini ad esempio possono essere ridimensionate, dati audio possono essere ritagliati in una certa finestra o settati a una certa frequenza e così via. Ci sono poi delle operazioni più complesse che possono essere eseguite sui dati, ad esempio le immagini potrebbe essere anche consegnate in forma grezza alla rete neurale, ma dati audio ad esempio, offrono caratteristiche migliori se osservati nel dominio della frequenza o se trasformati in modo intelligente. Queste trasformazioni dipendono poi dal contesto di utilizzo, infatti analizzare rumori è diverso dall'analisi delle parole pronunciate da un uomo, calcolare lo spettrogramma può funzionare bene per gli audio che non contengono voce mentre il calcolo dei **Mels frequency cepstral coefficients**(MFCC) è utile per estrarre le features dalla voce umana. Quest'ultimo utilizza lo spettrogramma durante il suo processo applicando anche altri tipi di calcoli. In particolare le fasi principali che

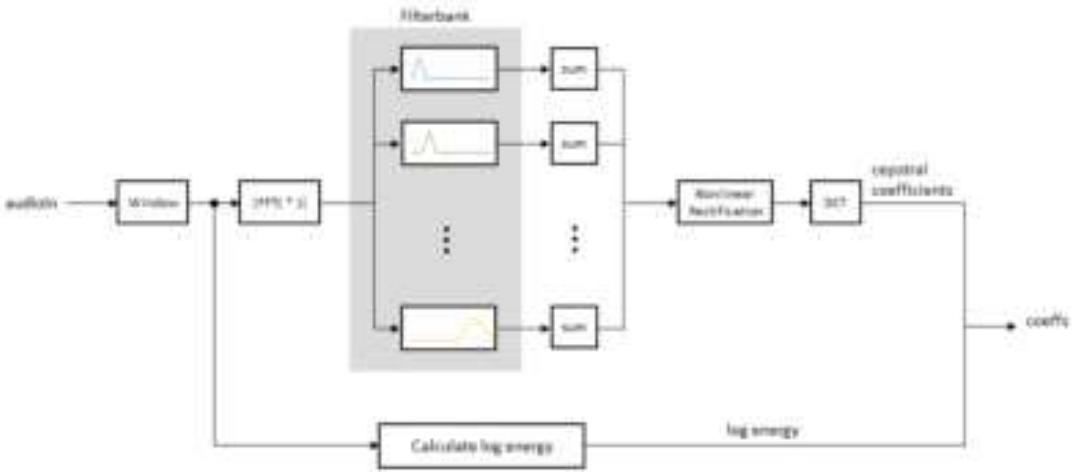


Figura 2.12: *MFCC workflow, fonte [18]*

compongono questa estrazione sono:

- Windowing: In questa fase il sample audio viene diviso in frame di una certa lunghezza che solitamente si trova sui 25 ms che è una lunghezza non breve per

evitare che non contenga abbastanza informazione e non lunga per evitare che queste informazioni varino molto durante la finestra;

- FFT: La Fast Fourier Transformation è applicata ad ogni frame creato nel passo precedente, quindi il sample viene portato dal dominio del tempo al dominio della frequenza. I frame nel tempo comporranno così lo spettrogramma, che avrà un numero di features pari alle features di frequenza calcolate per ogni frame moltiplicato per il numero di frame;
- In questo passo viene applicata una banca di filtri, triangolari poichè hanno questa forma come si vede in figura 2.14, rispettando la scala mel, questa è una scala speciale che si differenzia dalla classica misurata in hertz poichè rappresenta l'ascoltatore umano che all'aumentare che all'aumentare della frequenza non capta più la differenza. Applicando questi filtri, quindi moltiplicando il valore di un filtro per l'ampiezza dello spettro, si prelevano o modificano le informazioni che interessano. L'idea è quindi di applicare dei filtri su ogni spettro di potenza per fare in modo che venga dato più peso alle basse frequenze rispetto alle alte, rispettando quindi questa scala;
- Una DCT(Discrete Cosine Transformation) è applicata ad ogni filterbank per estrarre i coefficienti cepstrali.

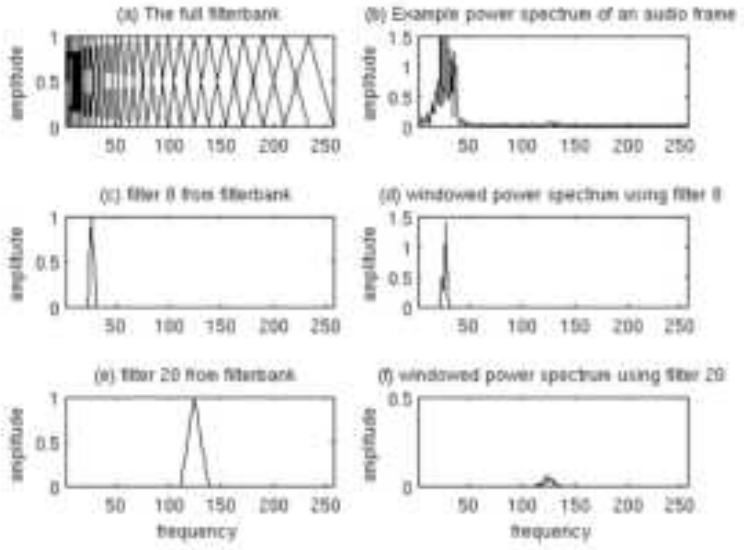


Figura 2.13: *Applicazione di due filtri ad uno spettro di potenza, fonte [19]*

In figura 2.13 è possibile vedere l'applicazione del processo ad una pronuncia della frase "Hello World"

Figura 2.14: *Cepstral coefficients della frase "hello world" , fonte [20]*

Data Augmentation

Le tecniche di data augmentation sono utilizzate per diversi scopi, come migliorare la qualità e la varietà dei dati. Sono molto utili ad esempio nel caso di un dataset povero che ha bisogno di essere migliorato. Questo tipo di tecniche non inventano nessun nuovo dato, ma prendono i dati già esistenti che con l'applicazione di alcune trasformazioni diventano nuovi dati utili all'addestramento. Ci sono diversi tipi augmentation e ovviamente dipendono dalla forma dei dati, che possono essere immagini, audio o altro. Per quanto riguarda le immagini è facile pensare a dei modi per creare nuovi dati, questo deve essere sempre fatto con l'idea di addestrare il modello a dati di qualsiasi tipo, ad esempio non è detto che in un'immagine il soggetto sia sempre in una posizione o al centro dell'immagine. Alcune tecniche di augmentation per le immagini sono:

- Flip: Equivale a prendere lo speculare di un'immagine;
- Rotazione: Ruotare l'immagine di diversi gradi;
- Traslazione: Spostare l'immagine in modo arbitrario;
- Ritaglio: Prende una porzione dell'immagine;
- Disturbo: tecniche di disturbo possono essere l'aggiunta di pixel bianchi e neri in modo casuale nell'immagine o il disturbo gaussiano che è la distorsione di feature che appaiono frequentemente.

Queste sono solo una piccola parte delle tecniche applicabili e i tool odierni offrono una varietà elevata di possibilità. Altre tecniche riguardano altri tipi di dati come audio, per i quali ad esempio è possibile applicare il disturbo gaussiano o altre tecniche avanzate che operano sullo spettrogramma. In figura 2.15 si possono osservare alcune tecniche di data augmentation applicate alla stessa immagine. Per quanto riguarda i dati audio, e in particolare la **Speech recognition**, tre tecniche vengono usate solitamente per l'augmentation di dati [8], queste tecniche vengono applicate sul mel-frequency cepstrum.

- Time warping: Ipotizzando il tempo sull'asse orizzontale e la frequenza sull'asse verticale, viene allora inserito un disturbo in un certo range di tempo;

Figura 2.15: *Applicazione di diverse tecniche di data augmentation, fonte [21]*

- Frequency masking: Mascherare alcune frequenze in un certo range;
- Time masking: Mascherare alcuni istanti di tempo in un certo. range.

2.2.7 Allenamento di un modello

Il training è il processo che fa ottenere il modello vero e proprio. Vengono usati per eseguirlo tutti i blocchi di processamento e le risorse che si pensano necessari. Nella pratica è l'esecuzione delle tecniche descritte nella sezione 2.2.4. Esistono molti tool che possono eseguire le operazioni di addestramento e mostrare in output i risultati in tempo reale per controllarne l'andamento. Prima di iniziare il training è importate settare nel modo opportuno alcune variabili, primo fra tutti il numero di **epoche**. Il numero di epoche corrisponde esattamente all'esecuzione di un solo passo in avanti e un solo passo di propagazione all'indietro. Solitamente viene inteso come l'allenamento della rete con una sola iterazione su tutto il dataset. Prima del training devono essere anche scelti il tasso di apprendimento, che come spiegato rappresenta la grandezza del passo di movimento in direzione opposta al gradiente, e quanti dei dati del training set si vogliono usare per il validation set. Una volta in esecuzione è opportuno monitorare la situazione e la differenza di prestazioni tra l'accuratezza sul validation set e quella sul training set. E' importante non far avanzare training troppo tempo per evitare problemi di **overfitting**, quando cioè la rete al posto di imparare memorizza, se succede allora non sarà in grado di generalizzare il problema. Si può capire la situazione in figura 2.16 quando nel terzo caso il modello conosce a memoria i dati del training set proposti ma non sa generalizzare. Evitare l'overfitting dipende da tante cose come la varietà dei dati, il numero di livelli e neuroni e così via, però un'importante metrica per evitarla durante l'addestramento è valutare l'accuratezza sul validation set e sul training set, queste non si vuole siano uguale ma si vuole invece che durante l'addestramento crescano

Figura 2.16: *Tre esempi caratteristici di classificazione, fonte [22]*

proporzionalmente, quando invece non si vede più il miglioramento per il validation set mentre continua a vedersi sul training set allora si sta andando incontro all'overfitting. Le prestazioni durante una certa fase dell'allenamento oltre che dall'accuratezza possono essere valutate sul fattore di perdita. Se si vuole ottenere un modello ben funzionante l'andamento di questo valore dovrebbe seguire una curva inversa a quella dell'accuratezza che quindi ci fa capire che l'allenamento dovrebbe essere terminato quando questa curva segue un andamento costante, quantomeno sul validation set.

Metriche di valutazione per modelli di classificazione

Una volta terminato l'allenamento viene usato finalmente il test set per la valutazione del modello o dei modelli. I risultati sono solitamente illustrati in una **matrice di confusione** come in figura 2.17, che avere tante righe e colonne quanto è il numero di classi. Ci sono delle metriche specifiche che sono usate solitamente per modelli di classificazione:

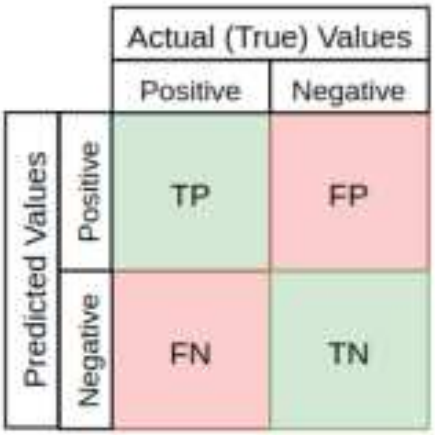


Figura 2.17: *Esempio di una matrice di confusione, fonte [23]*

- $Accuratezza = \frac{TP}{T}$
- $Precisione = \frac{TP}{TP+FP}$
- $Recall = \frac{TP}{TP+FN}$
- $F1score = 2 * \frac{Precisioe*Recall}{Precisione+Recall}$

L'accuratezza è la metrica fin ora discussa per la valutazione del modello, però non è quella migliore. L'accuratezza è il rapporto tra le classificazioni corrette del modello sul totale delle classificazioni. Si immagini di avere due classi A e B con 100/100 classificazioni corrette per la classe A e 1/2 classificazioni corrette per la classe B. In questo caso l'accuratezza è elevatissima ma il modello invece non funziona bene. Infatti per la classe B vengono sbagliate metà delle classificazioni. E' per questo che vengono usate le altre metriche. In particolare l'accuratezza offre scarsi risultati quando il dataset non è bilanciato rispetto al numero di oggetti per classe. La precisione dice invece tra tutte le volte che il modello ha classificato un dato in una classe quante di queste volte era davvero di quella classe. La recall invece indica degli oggetti reali di una classe quanti il modello riesce a trovare. Ad esempio se il modello in presenza di 50 oggetti di classe A e 50 di classe B classifica 99 oggetti nella classe A questo ha una recall elevata per la classe A ma una precisione pessima, mentre se l'ultimo oggetto appartenente alla classe B viene classificato nella classe B allora per questa classe vi è una precisione ottima ma una recall pessima. L'ultima metrica unisce semplicemente il concetto di precisione e di recall ed è molto utile quando si vogliono valutare le prestazioni di un modello di classificazione.

2.2.8 Ottimizzazioni di un modello

Come discusso nella sezione 2.2.3 in alcuni casi si vuole che questi modelli funzionino in macchine con capacità limitata. Dopo aver creato un modello si possono applicare alcune tecniche [24] [25] che sono state sviluppate per ridurre la complessità di questi algoritmi. Esistono poi diversi tool che permettono queste trasformazioni come TensoFlowLite, Aimet, CoreMl,PyThorch e altri.

Quantizzazione

Lo sviluppo delle prime reti neurali fu indirizzato totalmente al loro funzionamento e quindi all'ottenimento di risultati attendibili, per questo motivo fu scelto fin da subito di utilizzare numeri a virgola mobile a 32 bit. Si è notato che se questi numeri vengono sostituiti con altri a bassa precisione allora la rete neurale tende a rispondere come il cervello umano. Infatti se ad esempio si sta processando un'immagine la sua rappresentazione con numeri a bassa precisione tenderebbe ad essere un'immagine più sfocata rispetto a quella originale, quindi lavorare con questo tipo di numeri vuol dire in pratica aggiungere un nuovo livello di disturbo ai dati e infatti si vuole che le reti neurali sappiano rispondere bene anche in presenza di disturbo. Si osservi la figura 2.18, in questo

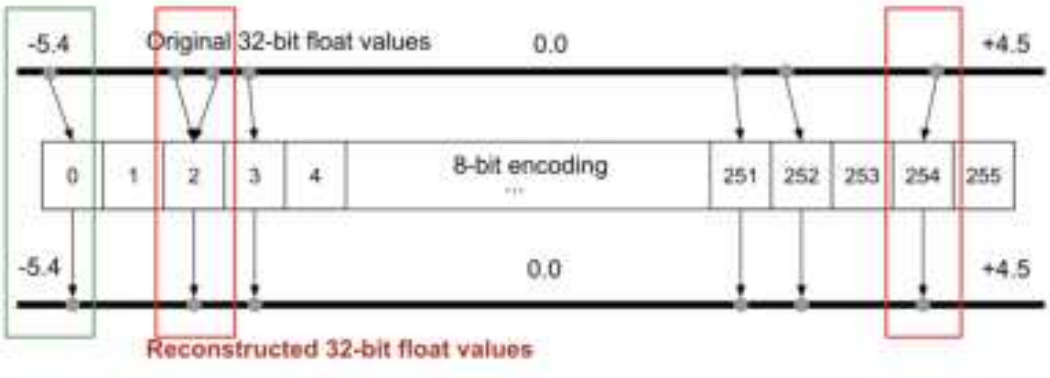


Figura 2.18: *Esempio di quantizzazione, fonte [25]*

caso si vogliono rappresentare numeri da -5.4 a 5.5 . Se lo si vuole fare con interi a 8 bit si hanno a disposizione 255 numeri, quindi posso rappresentare un numero diverso dopo ogni 0.038. Il problema di questo approccio è chiaramente la perdita di informazioni, ad esempio -5.4 e -5.41 saranno mappati con il numero 0, è però il prezzo da pagare per avere una riduzione significativa della complessità, se si opera su microcontroller sono molti di più i vantaggi che si ottengono in questo modo.

Pruning

Un'altra tecnica che si differenzia dalla quantizzazione è il pruning. Con questa tecnica invece di ridurre il peso dai dati vengono direttamente ridotti i dati. Si intende chia-

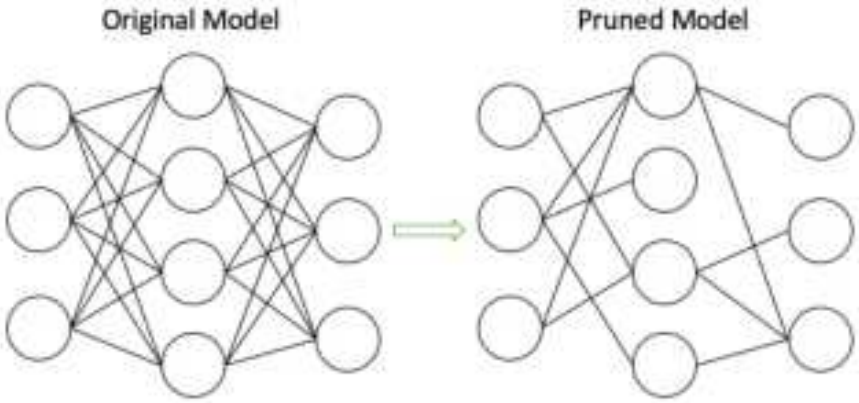


Figura 2.19: *Esempio di pruning, fonte [26]*

ramente dalla figura 2.19 che vengono eliminati alcuni collegamenti, e rispettivi pesi,

tra i neuroni, solitamente in modo casuale. Questa tecnica oltre ad offrire una buona riduzione della memoria in alcuni casi aiuta a contrastare l'overfitting [27]

Weight clustering

Questa tecnica consiste nell'individuare un certo numero di centroidi ed associare ogni peso al cluster di appartenenza. E' stato dimostrato come questa tecnica offre ottimi risultati di riduzione di memoria ma comunque una piccola riduzione delle prestazioni computazionali[28]. Si vede in 2.20 come vengono scelti quattro centroidi e suddivisi i

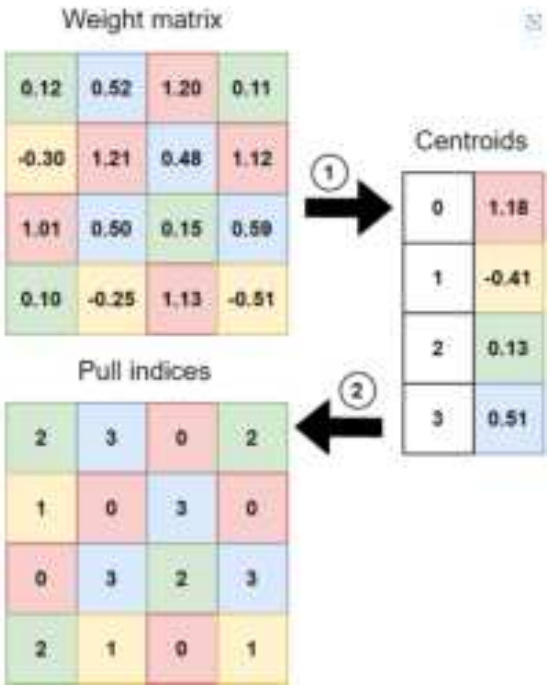


Figura 2.20: *Esempio di Weight clustering, fonte [29]*

pesi in quattro cluster, al posto di ogni peso si memorizza quindi l'indice del centroide, se si hanno quattro cluster in un livello allora verranno usati solo 2 bit per memorizzare gli indici.

Encoding

E' una tecnica semplice ma efficace per ridurre il numero di bit usati per la rappresentazione. viene usata una rappresentazione binaria. Si esegue l'encoding dei pesi usando la tecnica di Huffman, quindi utilizzando la frequenza con cui appaiono i pesi, dopodichè dall'albero ottenuto la codifica per ogni peso è il percorso in bit dalla radice alla foglia.

Compilazione

Si tratta di portare il modello nel formato migliore per la scheda target, ad esempio una scheda potrebbe eseguire con maggior efficienza il codice c rispetto a quello python. In generale viene sempre eseguita un'operazione di questo tipo quando si devono usare questi modelli in un microcontroller o su sistemi operativi portatili come Android ed è eseguita di norma come l'ultima operazione tra quelle di ottimizzazione.

2.2.9 Tecnologie software e hardware utilizzate

2.2.10 TensorFlow

TensorFlow è un sistema di apprendimento automatico che opera su su larga scala e in ambienti eterogenei,consente agli sviluppatori di sperimentare nuove ottimizzazioni e algoritmi di formazione,supporta una varietà di applicazioni, con particolare attenzione alla formazione e all'inferenza sul lavoro di rete neurali profonde [30]. Ad oggi è forse il tool più usato in questo ambito, per l'apprendimento cosi come in alcuni dei sistemi più innovativi come Gmail, Google cloud speech o Google search. La forza principale di TensorFlow è che offre davvero un ambiente semplice da usare, anche per i programmatori meno esperti. Permette infatti di creare un modello di ML in poche righe di codice e lavorando ad alto livello, senza bisogno di approfondire la matematica e i calcoli che stanno sotto. Ciò non vuol dire che un utente esperto non possa interessarsi del livello matematico, infatti TensorFlow offre la possibilità anche di operare con queste operazioni se lo si desidera. E' scritto in C++, Python e CUDA e permette di interfacciarsi con le API di Keras per la costruzione di blocchi per i modelli di ML. In figura 2.21 si

```
import tensorflow as tf
import numpy as np

model1 = tf.keras.Sequential([
    tf.keras.layers.Dense(4),
    tf.keras.layers.Dense(4)
])
```

Figura 2.21: *Creazione di un semplice modello con TensorFlow*

vede la creazione di un modello e quanto è semplice aggiungere livelli con TensorFlow. Un'importante caratteristica di cui tenere conto è la portabilità dei modelli che possono essere creati con TensorFlow, questo è importante quando devono essere creati modelli per microcontroller o macchine differenti da quella dove il modello è stato creato.

TensorFlow Lite

TensorFlow Lite è un set di strumenti che abilita l'apprendimento automatico sul dispositivo aiutando gli sviluppatori a eseguire i loro modelli su dispositivi mobili, embedded ed edge [31]. Fornisce supporto per Android e iOS dispositivi e Linux incorporato. TensorFlow Lite per microcontrollori (TFLite Micro) è un adattamento recente di TFLite (a metà 2019), dedicato all'esecuzione di modelli ML microcontrollori. TFlite stesso fornisce API in diversi linguaggi, come Java, Swift, Python e C++. TFlite Micro utilizza l'API C++, in particolare C++11, che riutilizza gran parte della base di codice generale di TensorFlow. Questo ToolKit presenta principalmente due strumenti fondamentali: il convertitore (TFLite Converter) e l'interprete (TFLite Interpreter) che può essere installato indipendentemente su il dispositivo su cui vogliamo fare inferenza. Il convertitore ha il compito principale di ottimizzare il modello riducendo le suo dimensioni e aumentando la suo velocità di esecuzione[32]. Come per la creazione del modello con TensorFlow

```
import tensorflow as tf
import pathlib

# from keras model
converter = tf.lite.TFLiteConverter.from_keras_model(model)
# or from tf saved model
converter = tf.lite.TFLiteConverter.from_saved_model(tf_path_model)
# last from concrete functions
converter = tf.lite.TFLiteConverter.from_concrete_funcions(tf_path_concrete_functions)

# start conversion
tflite_model = converter.converter()

# save model
tflite_model_file = pathlib.Path('./my_path')
tflite_model_file.write_bytes(tflite_model)
```

Figura 2.22: *Conversione di un modello con TensorFlow Lite 2*

anche con TFLite bastano pochi semplici passi per convertire e/o ottimizzare il modello. In figura 2.22 si possono vedere i pochi passi necessari alla conversione di un modello.

2.2.11 Edge Impulse

Edge Impulse è una piattaforma di sviluppo per l'apprendimento automatico su dispositivi edge, per gli sviluppatori e aziende. L'azienda ha la missione di offrire a ogni sviluppatore e produttore di dispositivi la migliore esperienza di sviluppo e distribuzione per l'apprendimento automatico all'edge, concentrandosi su applicazioni di sensori, audio e visione artificiale. Con Edge Impulse, sviluppatori di software, ingegneri ed esperti di dominio possono risolvere problemi reali utilizzando l'apprendimento automatico su dispositivi edge. Edge Impulse offre la massima efficienza e velocità su un'ampia gamma di hardware, dagli MCU alle CPU. [33] Attraverso Edge Impulse si possono creare

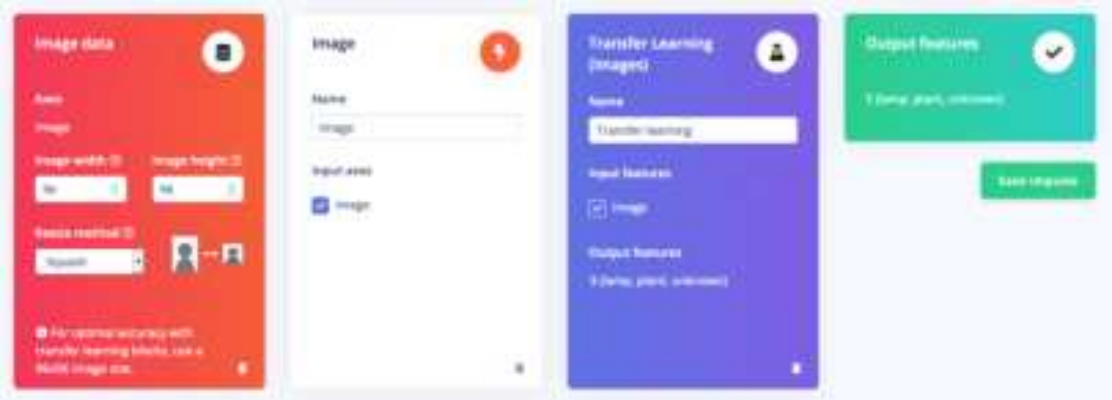


Figura 2.23: *Creazione di un impulso, fonte [34]*

modelli di intelligenza artificiale seguendo l'interfaccia intuitiva che il software mette a disposizione. Permette innanzitutto di collegare qualsiasi tipo di sensore alla piattaforma, quindi si possono usare sensori del pc, di uno smartphone o una scheda dedicata come un microcontroller. Edge Impulse segue la raccolta dei dati aiutando l'utente a suddividerli in dataset e mostrando sempre le informazioni sul dataset, in più permette di lavorare sui dati per eseguire operazioni come ridimensionameto di immagini o ritaglio di dati audio. Un concetto importante di Edge Impulse, da cui poi prende il nome il software stesso, è quello di impulso. L'impulso è un insieme di blocchi di processamento che possono essere eseguiti in modo sequenziale o parallelo con altri blocchi. La creazione dell'impulso decide il workflow che deve essere seguito nella lavorazione dei dati fino all'output, ad esempio in figura 2.23 si può vedere un esempio di impulso per il processamento di immagini. L'immagine viene prima adattata al contesto scelto e dopodichè viene mandata in input ad un blocco di ML, in questo caso di transfer learning per ottenere nell'ultimo blocco la classificazione in tre classi. Ogni blocco dell'impulso può essere modificato approfonditamente, in più Edge Impulse offre degli strumenti di analisi importanti come il feature explorer, che permette di analizzare i dati prima di inviarli al classificatore e mostra una rappresentazione grafica dei dati che vengono clusterizzati a seconda delle features grezze, figura 2.24. A seconda del blocco di ML scelto vengono messe a disposizione alcune opzioni di lavoro. Ad esempio se viene scelto un modello di rete neurale per la classificazione Edge Impulse mette a disposizione un'interfaccia user-friendly per creare la rete neurale (2.25) indicando i parametri graficamente. Per gli utenti esperti è possibile visualizzare e modificare il codice python per avere un controllo a basso livello. Dopo l'addestramento si può testate il modello in una pagina dedicata al testing oppure in una modalità di live classification. Essendo sviluppato per ML su dispositivi edge Edge impulse offre diverse opzioni per l'esportazione del modello creato, può essere creato il firmware per diversi dispositivi oppure l'esportazione del modello in una libreria creata appositamente per diversi linguaggi di programmazione, infine può essere eseguito l'impulso anche su uno smartphone o direttamente sul computer.

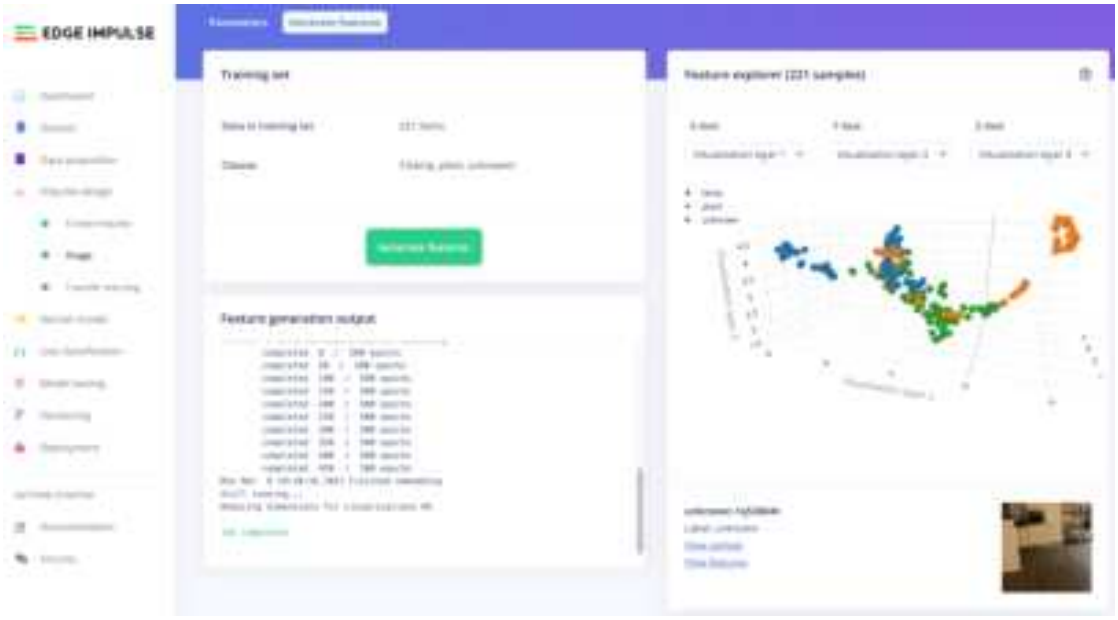


Figura 2.24: *Interfaccia di Edge Impulse: Feature explorer, fonte [34]*

Figura 2.25: *Creazione grafica di una rete neurale in Edge Impulse*

2.2.12 Arduino Portenta e Arduino vision shield

Portenta H7 esegue simultaneamente codice di alto livello e attività in tempo reale. Il design include due processori in grado di eseguire attività in parallelo. Ad esempio, è possibile eseguire il codice compilato da Arduino insieme a quello di MicroPython e

avere entrambi i core per comunicare tra loro. La funzionalità Portenta è duplice, può essere eseguita come qualsiasi altra scheda microcontrollor incorporata o come processore principale di un computer incorporato. Portenta può eseguire facilmente i processi creati con TensorFlow Lite,si potrebbe avere uno dei core che calcola al volo un algoritmo di visione artificiale, mentre l'altro potrebbe eseguire operazioni di basso livello come il controllo di un motore o fungere da interfaccia utente. Il processore principale di H7 è il dual core STM32H747 che include un Cortex M7 a 480 MHz e un Cortex M4 a 240 MHz. I due core comunicano tramite un meccanismo di chiamata di procedura remota che consente di chiamare le funzioni sull'altro processore senza problemi[35]. **Portenta Vision Shield** offre funzionalità di livello industriale alla scheda Arduino Portenta. Questo componente aggiuntivo hardware consente di eseguire applicazioni di visione artificiale integrate, connettersi in modalità wireless o tramite Ethernet ad Arduino Cloud o ad un'infrastruttura e attivare un sistema al rilevamento di eventi sonori. Vision Shield è stato progettato per funzionare con Arduino Portenta H7[35]. Per la programmazione dello sketch è stato utilizzato **Arduino Ide** per le facilities messe a disposizione del programmatore.

2.2.13 SA

I sentimenti e le emozioni umane rappresentano un tratto distintivo dell'uomo che da sempre lo aiuta a relazionarsi con le altre persone e con il mondo circostante. Parallelamente allo sviluppo delle tecnologie e del mondo digitale sono aumentati sempre più i motivi per cui si rende utile sviluppare sistemi per l'identificazione dei sentimenti.
L'analisi delle reazioni degli utenti all'utilizzo di un prodotto, così come alla vista di un'opera d'arte, è stata eseguita per molto tempo in modo approssimativo, basandosi su ciò che era umanamente tangibile e quindi limitato. Eventi che hanno rivoluzionato il marketing sono stati l'avvento e la diffusione dell'e-commerce con la possibilità di inserire feedback come voti e commenti ad un prodotto, avvantaggiando un altro acquirente nel processo di scelta ma dando allo stesso tempo al venditore dei dati accurati su ciò che viene apprezzato o meno. Questa filosofia, che è ormai possibile riscontrare in ogni servizio esposto al pubblico, sia virtualmente che fisicamente, con l'evoluzione tecnologica ha trovato sempre più applicazioni. Dal punto di vista degli erogatori di un servizio sapere ciò che viene apprezzato o meno è fondamentale per migliorare tale servizio e nel mondo digitale queste informazioni sono ormai dappertutto, sopratutto nei social network nei quali la gente esprime liberamente opinioni su qualsiasi argomento. Ma la SA ha svariate applicazioni, addirittura nei dibattiti politici con i quali potremmo capire le opinioni delle persone su determinati candidati alle elezioni o partiti politici. I risultati delle elezioni possono essere previsti anche da post sulla politica. [1]. I commenti,le recensioni e i feedback espliciti rivolti a un determinato servizio o entità rappresentano solo una parte delle informazioni sulle opinioni quando si possono trovare migliaia di dati aggiuntivi annidati nel web. Avere strumenti per analizzare queste informazioni è l'obiettivo della SA. La SA [36] utilizza l'elaborazione del linguaggio naturale (NLP), l'analisi del testo e le tecniche computazionali per automatizzare l'estrazione o la classificazione del sentimento. Non c'è però una definizione universale del termine e infatti

si fa a volte riferimento alla SA con il nome di **Opinion mining** , invece altri testi riferiscono che l' Opinion Mining estrae e analizza l'opinione delle persone su un'entità mentre la SA identifica il sentimento espresso in un testo, quindi lo analizza.[1] In questo lavoro di tesi ci si riferisce alla SA includendo anche il concetto di Opinion Mining. Si possono distinguere principalmente quattro tipi di obiettivi di analisi:

- Valutare una certa informazione che può essere posizionata in una scala di valori che vanno dal valore massimo negativo al valore massimo positivo. Quando sono solamente due valori si tratta di una classificazione binaria ;
- Valutare una certa informazione e distinguerne l'emozione dell'utente, ad esempio dicendo se si tratta di una sentimento di rabbia,paura,felicità,simpatia ecc..;
- Valutare l'intento di un utente. Ad esempio capire se egli è propenso all'acquisto di un prodotto;
- Valutare il pensiero di un utente rispetto a diversi aspetti di un'entità. Ad esempio capire cosa il cliente pensa delle diverse componenti di uno smartphone come la fotocamera, la batteria, il display e così via.

Per tutte le tipologie di analisi possono essere utilizzati svariati approcci. Negli ultimi anni si è molto diffuso l'approccio con l'utilizzo del ML, sia di tipo supervisionato che non, in generale si può vedere una semplice rappresentazione di questi approcci in figura 2.26

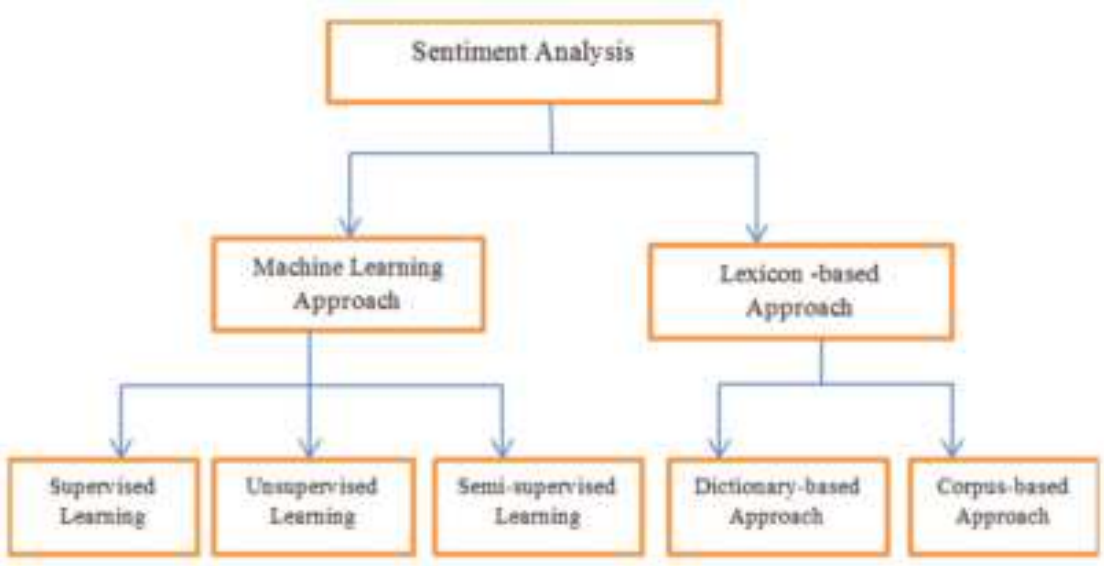


Figura 2.26: *Differenti approcci alla SA, fonte [37]*

Diversi approcci che ricadono nell'ambito del ML possono essere utilizzati quando si è in presenza di dati testuali[37]. Un 'altro tipo di approccio, sopratutto quando non

si è in presenza di questo tipo di dati, è l'utilizzo del ML direttamente nel processo di raccolta dati, ad esempio per la raccolta di dati audio su cui poi eseguire analisi dei sentimenti.

Capitolo 3

Progettazione del sistema

Come già anticipato, il sistema sviluppato in questo lavoro di tesi è rivolto a contesti come parchi naturali, musei, mostre o spazi espositivi solitamente affollati, al chiuso o all'aperto, e in cui i rumori di sottofondo possono molto influire sui dati audio. Posizio-

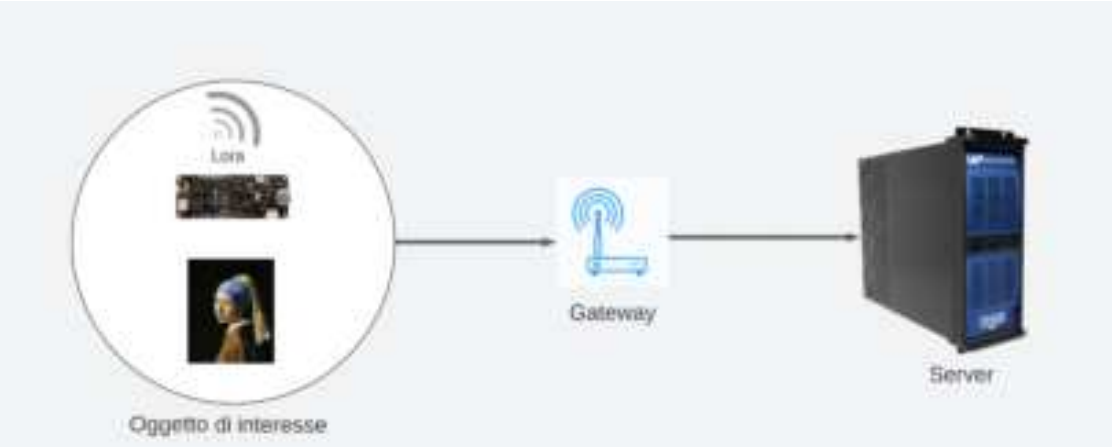


Figura 3.1: *Architettura del sistema sviluppato*

nando i sensori intelligenti in prossimità dei punti di interesse, come ad esempio le varie opere d'arte se ci riferiamo al contesto del museo o gli animali se siamo nel contesto del parco naturale, si vuole catturare parole chiave che siano indicative sul grado di apprezzamento di quanto il pubblico vede. Nel caso specifico del progetto si fa riferimento ad un dizionario di 12 parole, metà italiane e metà spagnole, scelte arbitrariamente per rappresentare due insiemi di sentimenti, uno positivo e l'altro negativo. Un microcontroller dotato di un microfono cattura l'audio in un certo range spaziale e lo processa, cercando associazioni tra le parole catturate e quelle del dizionario utilizzato, aggirando problematiche quali il rumore di sottofondo, voci sovrapposte, ecc. I risultati di questa prima elaborazione possono poi essere inviati ad un server remoto per elaborazioni suc-

cessive (vedi 3.1). Durante la fase di progettazione del sistema è stato importante tenere a mente questi dettagli di contesto, che saranno più volte richiamati, col fine di creare un sistema capace di rispondere correttamente agli input del mondo reale.

3.1 Architettura del sistema

L'architettura del sistema creato prevede componenti software e hardware, in particolare la componente hardware del sistema è proprio il microcontoller Arduino Portenta, con la scheda di elaborazione e il microfono. Deve essere poi elaborato il software da eseguire nella scheda e questo deve svolgere due compiti principali:

- Elaborare i dati catturati dal microfono della scheda Portenta;
- Prendere in input questi dati ed eseguire la classificazione.

Queste due compiti sono essenziali ed agnostici rispetto alla specifica tecnologia. Inoltre, la componente che svolge questi task, quindi il modello di inferenza, deve essere caricata ed eseguita sul microcontroller, previa quindi preparazione dell'ambiente Arduino. Essendo le parole definite a priori, il loro riconoscimento automatico per la successiva classificazione (attraverso le tecnologie riportate nel capitolo precedente, ed in particolare le reti neurali nell'ambito del TinyMl) ricade nell'ambito dell'apprendimento supervisionato. Rispetto a questa tecnologia scelta è stato definito l'insieme delle componenti software necessarie. Come illustrato in Fig. 3.2, si possono distinguere:

- Componenti Dati: Queste sono le componenti che rappresentano i dati del sistema e la loro organizzazione. Ci sono tre set differenti che, come è risaputo dall'utilizzo delle reti neurali, saranno utilizzati i primi due, validation set e training set dalla componente di training e il test set sul modello di inferenza finale;
- Componenti Training: Questo è il cuore del sistema, e si occupa di coordinare le differenti parti necessarie al training ed eseguirlo. È presente la parte di elaborazione dati audio che esegue operazioni preliminari sui dati come ridimensionamento della finestra, e una componente più specifica che esegue elaborazioni audio mirate invece all'estrazione delle features, come MFCC che dallo studio della letteratura risulta essere adatta per la voce umana. L'altra componente riguardante il training è invece la definizione della rete neurale vera e propria: qui saranno definiti i parametri che dovranno essere allenati per riconoscere i dati e classificarli. Infine il modello di inferenza che è rappresentato dai suoi parametri ed è il risultato del training;
- Componenti Hardware: La componente hardware dovrà essere equipaggiata invece con un software in grado di preparare l'ambiente della scheda Portenta e richiamare il modello per eseguire le inferenze, che quindi dovrà essere caricato a disposizione dello sketch come libreria. E' qui che dovranno essere implementate eventuali comunicazioni ad esempio per trasmettere i risultati ad un server.

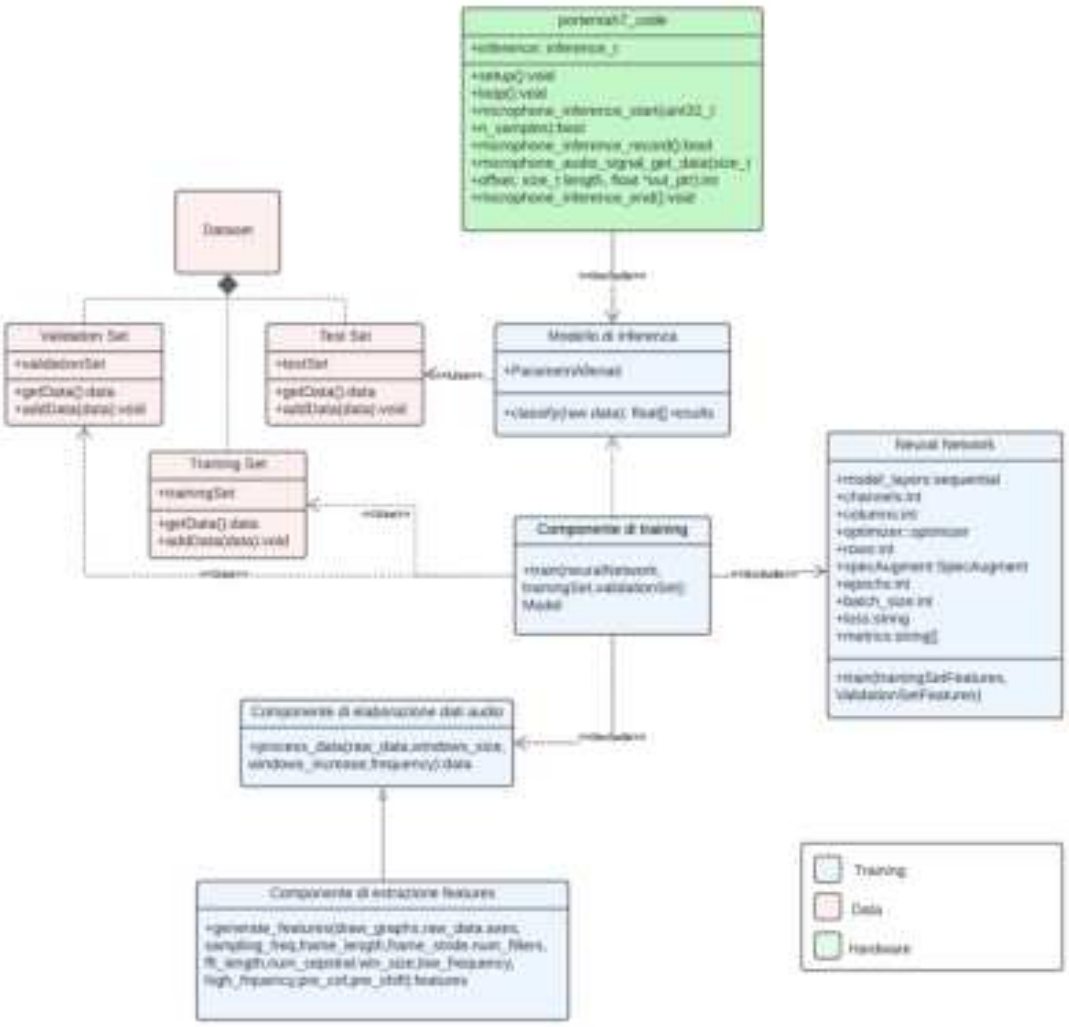


Figura 3.2: *Diagramma UML dell'architettura progettuale*

Nonostante l'obiettivo finale del progetto sia dotare il microcontroller di un modello di inferenza ben funzionante, tutte le altre componenti continueranno a essere attive, ad esempio aggiungendo sempre nuovi dati al dataset e reiterando il processo di training.

3.2 Modellazione e processo di training

Sono stati modellati quindi i passaggi necessari per ottenere il sistema. La fase di sviluppo prevederà una fase in cui si insegna al software a riconoscere i dati, quindi saranno raccolti i dati e a ognuno si metterà una etichetta, il software vedendoli e associandoli alle etichette dovrà imparare a riconoscerli, quindi poi in una fase di testing saranno

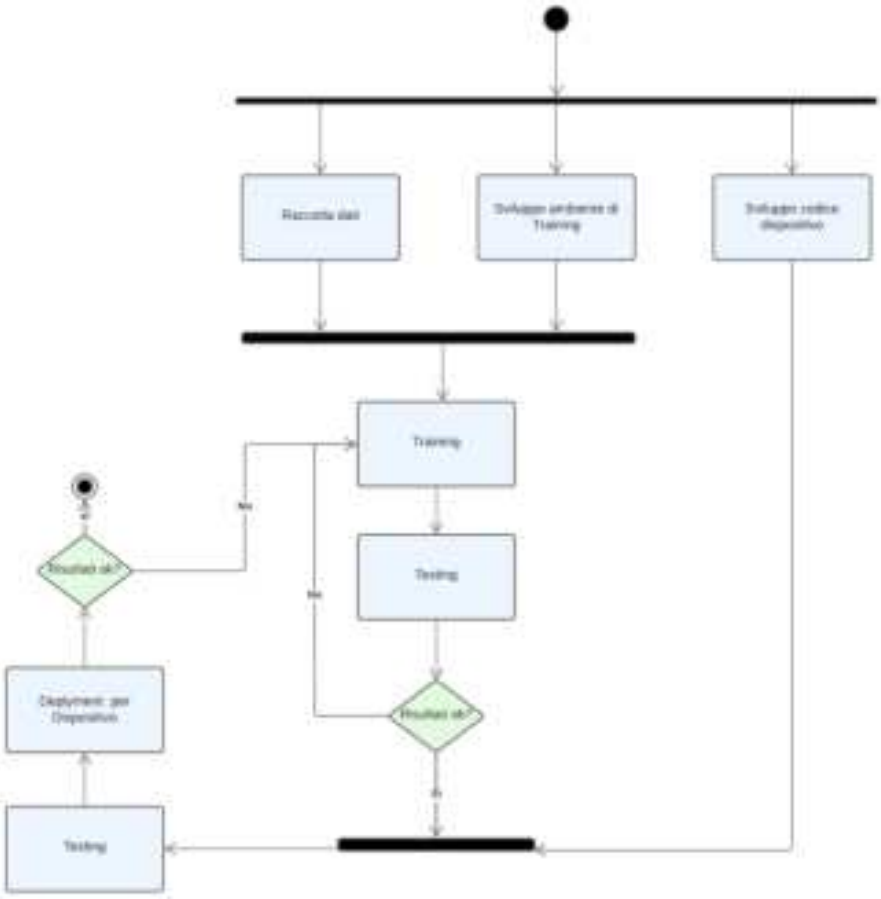


Figura 3.3: *Macroattività per lo sviluppo del sistema*

dati nuovi dati e si vedrà quanto il software lavora bene nella loro etichettazione. Il diagramma riportato in figura 3.3 riporta queste attività. Si è dato rilievo anche ai passaggi preliminari e conclusivi del processo di training per contestualizzarli. Si può partire dunque dalla preparazione dell'ambiente di sviluppo della scheda Portenta andando in parallelo a raccogliere i dati audio per il dataset. E' utile tenere a mente questo processo anche prima dell'implementazione delle componenti. Si vede quindi, andando nello specifico del processo di training, come strutturalo per il caso di studio, infatti, per quanto riguarda i modelli nell'ambito delle reti neurali oltre a definire una buona architettura di training è importante elaborare la struttura del processo di allenamento e vedere come le varie componenti vengono usate/interagiscono tra di loro. In figura 3.4 è possibile vedere la strutturazione di tale processo, quindi è come prendere l'attività di training della figura 3.3 entrando nel dettaglio delle sue sotto-attività. Il diagramma utilizzato è un diagramma UML applicato alle azioni del software necessarie per avere lo sviluppo. Questo definisce le fasi che devono essere implementate ed eseguite per attivare il training e ottenere il modello di inferenza. Quindi è come se la componente

modello del diagramma architetturale in 3.2 venga implementata automaticamente dalle componenti di training quando vengono eseguite. Ogni attività nel diagramma è facil-

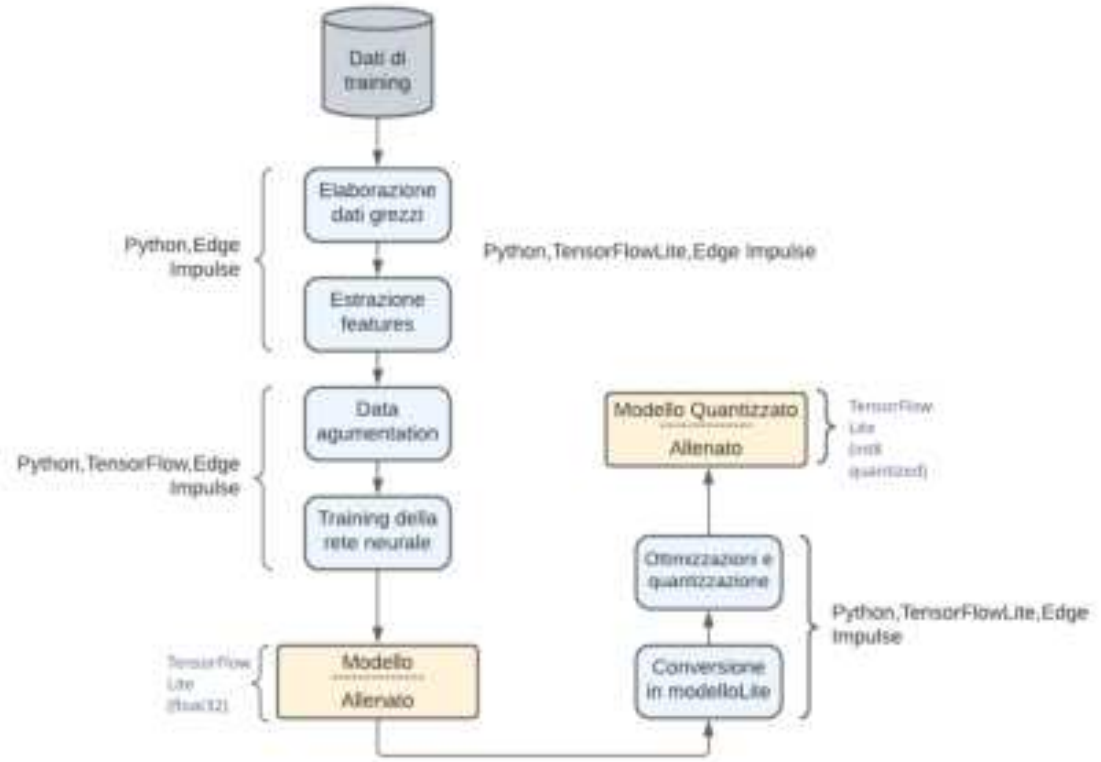


Figura 3.4: *Diagramma delle attività di ottenimento del modello*

mente associabile alle componenti descritte precedentemente, particolare nota va alla fase di **Data Augmentation** che sarà implementata nella componente di training e, da come è noto dal capitolo 2, è utile per aumentare la qualità e la quantità dei dati. Inoltre, anche se non viene indicata nel diagramma, è importante la fase di testing che andrà ad essere eseguita in diversi step dello sviluppo, la prima volta quando si ottiene la prima versione del modello, la seconda volta con il modello quantizzato, sopratutto per testare le ottimizzazioni e quanto differisce dal modello precedente, infine quando si testerà il modello nella scheda fisica, durante la quale saranno fatti test di simulazione dell'ambiente reale. Durante questo lavoro di progettazione sono state scelte anche le tecnologie e i tool da utilizzare in fase di implementazione. Si possono vedere sotto forma di nota nel diagramma delle attività 3.4. Edge Impulse, il software introdotto e dettagliatamente descritto nel capitolo precedente, è stato utilizzato nelle fasi cruciali dello sviluppo grazie a tutti i servizi che mette a disposizione, inoltre sono state inserite tecnologie come TensorFlow per il training e TensorFlowLite per la conversione e Python nella rete neurale come anche nei flussi di elaborazione dati. C++ invece è stato associato con Arduino Ide allo sketch Arduino. Ovviamente il modello allenato che sarà

sviluppato dovrà possedere una struttura tale che gli permetterà di eseguire operazioni molto simili a quelle fatte durante l'allenamento. Il modello per eseguire le inferenze su un certo dato audio dovrà eseguire alcune operazioni di elaborazioni dati e classificazione identiche al training, quindi tutti i parametri usati per il dataset. Queste operazioni e i parametri verranno automaticamente incorporate da TensorFlow e successivamente da Edge impulse quando vengono eseguiti rispettivamente, il training il deoployment per la scheda. Alla fine si otterrà una libreria che contiene il modello in modo automatico, la cui architettura verrà analizzata solo in alcuni aspetti, per il resto sarà considerata come una black box.

3.3 Strutturazione del Dataset

Merita una certa attenzione anche la strutturazione del dataset per il quale si è scelto, ancor prima di iniziare l'implementazione, di strutturarlo in due versioni:

- La prima versione contiene una parte di dataset dedicata alle parole del dizionario e le rispettive etichette, più una parte di dataset etichettata con "altro" per indicare tutto ciò che non fa parte delle parole, come rumori,silenzio e tutte le altre parole che possono essere pronunciate oltre a quelle note;
- Un'altra versione invece in cui oltre il dataset oltre a contenere le etichette del dizionario ne contiene altre due, una per indicare le altre parole, oltre a quelle note, una per ogni altro audio che non rappresenta parole.

Questa scelta è stata effettuata per scegliere poi in fase di implementazione e testing quella che funziona meglio, poichè come si vedrà molte scelte implementative dipendono da risultati empirici, quindi non è possibile sapere a priori cosa è meglio fare, ma eseguire i test o vedere progetti già esistenti per scegliere poi quale funziona meglio. Sapendo di avere a che fare con le reti neurali è noto che per riconoscere degli oggetti specifici queste devono conoscere anche il mondo circostante, cioè sapere oltre a cos'è anche cosa "non" è l'oggetto desiderato. La struttura vera e propria del dataset e la sua suddivisione è stata invece già vista nel diagramma 3.2.

3.4 Design dell'impulso

Tutto il cuore dell'architettura del sistema, per la scelta di usare il software Edge Impulse, deve essere modellato nella logica del suo ambiente. Ricordando il concetto di impulso descritto si va quindi ad associare, quando possibile, ogni componente ad un blocco dell'impulso.

Nella struttura di partenza in figura 3.5 il primo blocco si occupa di ridimensionare i dati, il secondo di elaborarli per estrarre le features e il terzo della classificazione con l'uso interno di Keras. L'ultimo blocco rappresenta il numero di label in output.

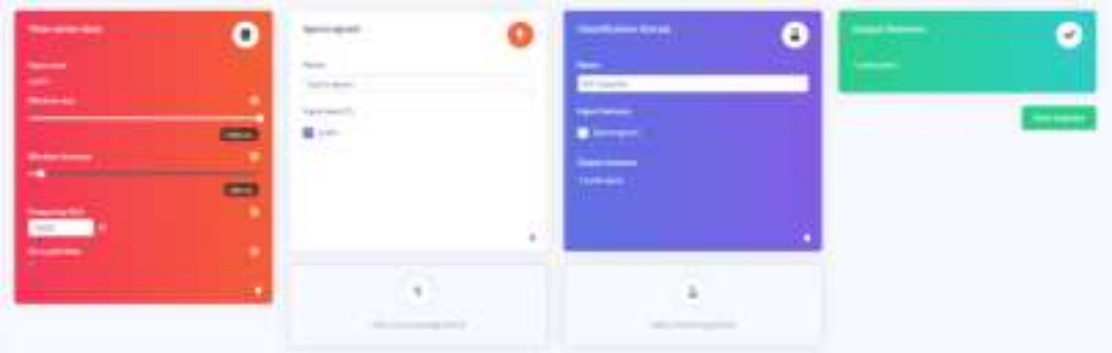


Figura 3.5: *Design di partenza dell'impulso*

3.5 Design Arduino sketch

La creazione dello sketch che deve essere usato sulla scheda deve includere alcune caratteristiche che sono essenziali al funzionamento del sistema. Si pensi che la creazione del modello fa ottenere soltanto l'oggetto che esegue l'inferenza, dopodichè questo deve essere usato per riconoscere delle parole. Possono venire in mente tante problematiche a riguardo [38]. Si pensi ad esempio a come registrare le parole ed eseguire le inferenze. Ad esempio se la finestra di inferenza è di 1 secondo, e vengono registrati sample di 1 secondo in sequenza, potrebbe succedere che la parola da riconoscere si trovi tra un sample e l'altro, trovandosi a metà. In quel caso è difficile che si riconosca quella parola. Alcuni di questi problemi sono stati risolti direttamente da Edge Impulse nel deployment come libreria Arduino di un modello. Qui sono già inserite le funzioni sullo slicing, ad esempio, che dividendo i sample in più parti fanno in modo di eseguire l'inferenza sul sample diverse volte. Ad esempio su sample di 1 secondo con slice di 0.25, per ogni slice viene eseguita l'inferenza in ognuna delle quattro posizioni in cui si può trovare, infine viene eseguita una media. Un altro problema fondamentale è che non si vuole perdere nessun istante durante le registrazioni, però se vengono eseguite le azioni in sequenza, cioè registrazione e poi inferenza, durante l'inferenza si perderebbe tempo di registrazione che potrebbe contenere dati importanti. Questo problematica deve essere affrontata direttamente nello sketch Arduino. Una soluzione è dunque di seguire il diagramma delle attività in figura 3.6 eseguendo le due fasi, di registrazione e inferenza, in parallelo. Nel capitolo successivo saranno visti i dettagli implementativi.

Figura 3.6: *Diagramma attività per lo sketch Arduino*

Capitolo 4

Implementazione del sistema

Il capitolo è dedicato all'implementazione del sistema. Dopo aver visto l'architettura che il sistema dovrebbe avere ed aver scandito le attività necessarie al suo sviluppo, si vede ora come implementare le idee del capitolo precedente e sopratutto come farlo con i software introdotti nei capitoli precedenti, cercando di sfruttare al meglio le loro potenzialità.

4.1 Raccolta dati

Basandosi sui diagrammi del capitolo precedente, sopratutto sui diagrammi delle attività, si è immediatamente provveduto ad effettuare la raccolta dati. Questa è una fase importante alla quale è stato dedicato molto tempo e che influisce largamente sui risultati finali perché, come anticipato, la rete deve apprendere da ciò che le viene proposto. Se proponessimo ad esempio dei dati audio che non generalizzino bene sulla situazione specifica allora la rete apprenderebbe solo quei dettagli, e quando ci sarà qualcosa di diverso non sarà in grado di riconoscerlo. Il lavoro di raccolta dati innanzitutto è stato eseguito nell'ambiente di Edge Impulse. Qui c'è la possibilità di collegare diversi dispositivi. Si ricorda che i dati sono audio, quindi ogni dispositivo deve possedere il sensore per raccogliere dati audio, come il vision sheld della Portenta. Sono stati catturati quindi dati con:

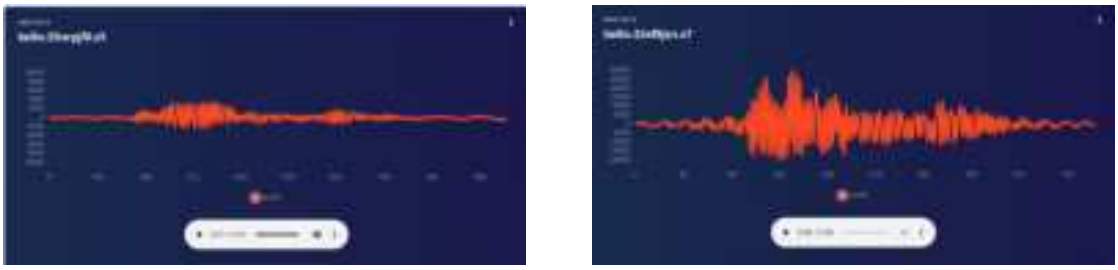
- Smartphone;
- PC;
- Portenta.

L'idea è di avere più dati possibili per poi, come si vedrà, nella fase di Testing, provare tante combinazioni o ad esempio capire se è meglio utilizzare solamente dati con lo stesso dispositivo con cui si eseguiranno le inferenze. Inizialmente si sono fatte le prove iniziali creando un dataset di sole due parole, analizzando e scegliendo la quantità ideale di dati

Figura 4.1: *Dispositivi usati per la raccolta dati*

che devono essere raccolti, la durata dei sample e altri dettagli simili, tenendo conto che sarà nella fase di testing in cui si deciderà sulla struttura finale. Per quanto riguarda PC e smartphone, il loro collegamento alla piattaforma avviene tramite l'interfaccia grafica di Edge Impulse. Per quanto riguarda la Portenta, ci sono diversi modi per connetterla al software e si è scelto di utilizzare la Command Line di Edge Impulse, installando il firmware di Edge Impulse sulla Portenta ed eseguendo il demone Edge-Impulse-Daemon per la configurazione del dispositivo e l'associazione al progetto. Una volta fatto, si è passati alla raccolta dati vera e propria con una lunghezza dei sample di 1 secondo e ad una frequenza di 16.000 Hz, ma anche per questo sono stati eseguiti successivi test riguardo al compromesso da selezionare. Sono stati rispettati alcuni criteri per introdurre particolare varietà nei dati, i più importanti:

- Coinvolgere il numero più alto di persone possibile per eseguire le registrazioni, sia uomini che donne. Questo, dato il contesto multiculturale in cui è stato eseguito il lavoro di tesi, è stato un obiettivo largamente raggiunto. Persone di diverso sesso, nazionalità, quindi con differenti accenti e toni di voce nel pronunciare le parole, hanno reso molto vario il dataset e contribuito alla realizzazione;
- Eseguire le registrazioni in diversi ambienti, con rumore, come ad esempio in strada o con la televisione accesa, o ambienti silenziosi come in casa o in un laboratorio;
- Eseguire le registrazioni a diverse distanze, anche molto vicino o molto lontano, in particolare nelle situazioni in cui era possibile con distanze superiori anche ai 5 metri, per comprendere meglio fin dove possono spingersi le potenzialità della scheda;
- Eseguire le registrazioni pronunciando le parole con differenti velocità o in modo differente;
- Rendere le registrazioni verosimili il più possibile, ad esempio pronunciando le parole all'interno di frasi a senso compiuto, così da contestualizzare l'intonazione della parola.



(a) *(b)*

Figura 4.2: *Due sample differenti della parola bello con pattern riconoscibile*

Tutti questi accorgimenti consentono di insegnare al sistema il pattern generale per ogni parola, prescindendo dunque da fattori come vicinanza, rumore di sottofondo e altre variabili. Notare in figura 4.2 come già la stessa parola pronunciata da due persone differenti produce un segnale molto differente, ma con un pattern comune chiaramente riconoscibile. Come era già stato anticipato nel capitolo precedente, sono state inizialmente create due versioni, una in cui l'etichetta altro comprende tutto il resto e una in cui viene suddivisa in "rumore" e "altre parole". Per quanto riguarda il rumore sono state eseguite registrazioni in Università, in locali aperti al pubblico di vario tipo(bar,pub ecc..), in strada, con musica di sottofondo, con la televisione e creando poi differenti tipi di rumore come il battito delle mani, il movimenti di oggetti di varia natura e così via. Invece per quanto riguarda le registrazioni di altre parole queste sono state eseguite da differenti persone, pronunciando liste di parole singole scelte casualmente o pronunciando interi discorsi in modo fluido, il tutto sempre seguendo i criteri elencati per le parole del dizionario. Importanti considerazioni da fare durante questa fase riguardano lo split dei dati: ricordando che si hanno diversi set di dati, ogni sample audio raccolto deve collocarsi in uno dei set. Essendoci solamente criteri di divisione consigliati e non esistendo il modo migliore per selezionarli si è partiti con una suddivisione tra training set e test set molto sbilanciata, ponendo quasi tutti gli audio nel set di training, questo perché l'idea è stata di rieseguire la suddivisione durante la fase di testing andando ad aggiungere o ribilanciare i due set di dati fino a raggiungere la configurazione migliore. In genere è consigliato una range che va dal 10 % al 30 % per il test set e il resto per il dataset. Inoltre va considerata anche la suddivisione tra training set e validation set che è stata inizialmente settata al 20 % del training set. Alla fine del processo di raccolta dati sono stati acquisiti:

- 1 ora e 33 minuti di dati attraverso la scheda portenta per un totale di 5260 samples;
- 10 minuti attraverso il microfono PC;
- 5 minuti attraverso il microfono di uno smartphone.

Come si può osservare è stata approfondita maggiormente la raccolta di dati con la Portenta, questo perché i primi test eseguiti hanno dimostrato un funzionamento migliore

per modelli che usano solo dati raccolti con il dispositivo usato per le inferenze. Andando ad approfondire le tracce audio si nota un andamento del segnale digitale nettamente differente quando viene registrata la pronuncia della stessa parola con strumenti diversi.

4.2 Implementazione delle componenti

4.2.1 Componente di elaborazione dati audio

Per l'implementazione della componente di elaborazione dati audio è necessario implementare le funzioni di elaborazione dei dati audio. Questa avrà una parte quasi sempre uguale per elaborare i dati grezzi e un'altra parte più specifica che è incaricata di estrarre le features. Per l'implementazione nell'ambiente Edge Impulse è stato dunque scelto un blocco dell'impulso *Times Series Data* in cui è stata settata inizialmente una finestra di 1 secondo per ogni registrazione, un incremento di 1/2 secondo massimo per dati audio che superano la grandezza prefissata, la frequenza di 16.000 Hz per le registrazioni, la stessa che è stata settata durante le registrazioni vere e proprie, e il padding attivo con bit a 0 nel caso di dati audio più piccoli di 1 secondo. Per quanto riguarda l'estrazione

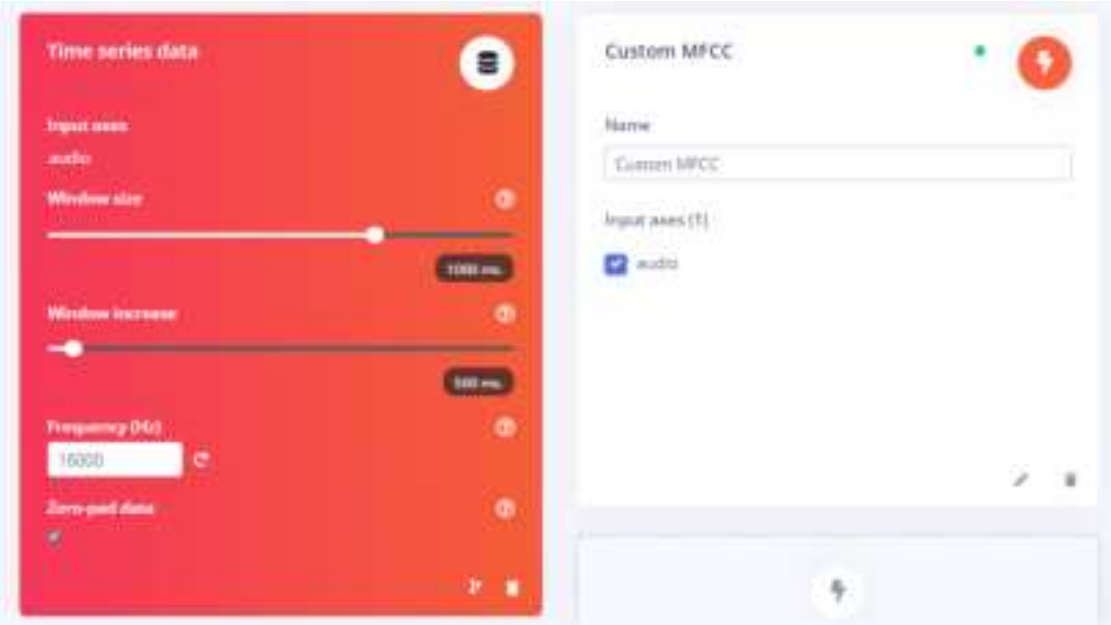


Figura 4.3: *Componente di elaborazione audio*

delle features, analizzando le tecnologie disponibili e i risultati raggiunti in altri progetti visionati, si è scelto di usare *MFCC*(Mels frequency Cepstral coefficient), ma sono stati provati anche altri tipi di blocchi. Si è anche scelto per di non usare il blocco predefinito, ma di clonarne quello esistente per poterlo personalizzare [39]. Questo perchè poter modificare i parametri e le funzioni direttamente dal codice Python da più libertà nella configurazione, è cosi possibile aggiungere parametri o funzioni in questa fase o modi-

ficare quelle esistenti. Si è andato a clonare quindi il repository Git di Edge Impulse contenente il blocco MFCC , dopodichè è stato reso disponibile come server Python con un tunnel Ngrok per essere raggiungibile da Edge Impulse. In figura 4.4 si vede una delle configurazioni utilizzate per il blocco *MFCC*.

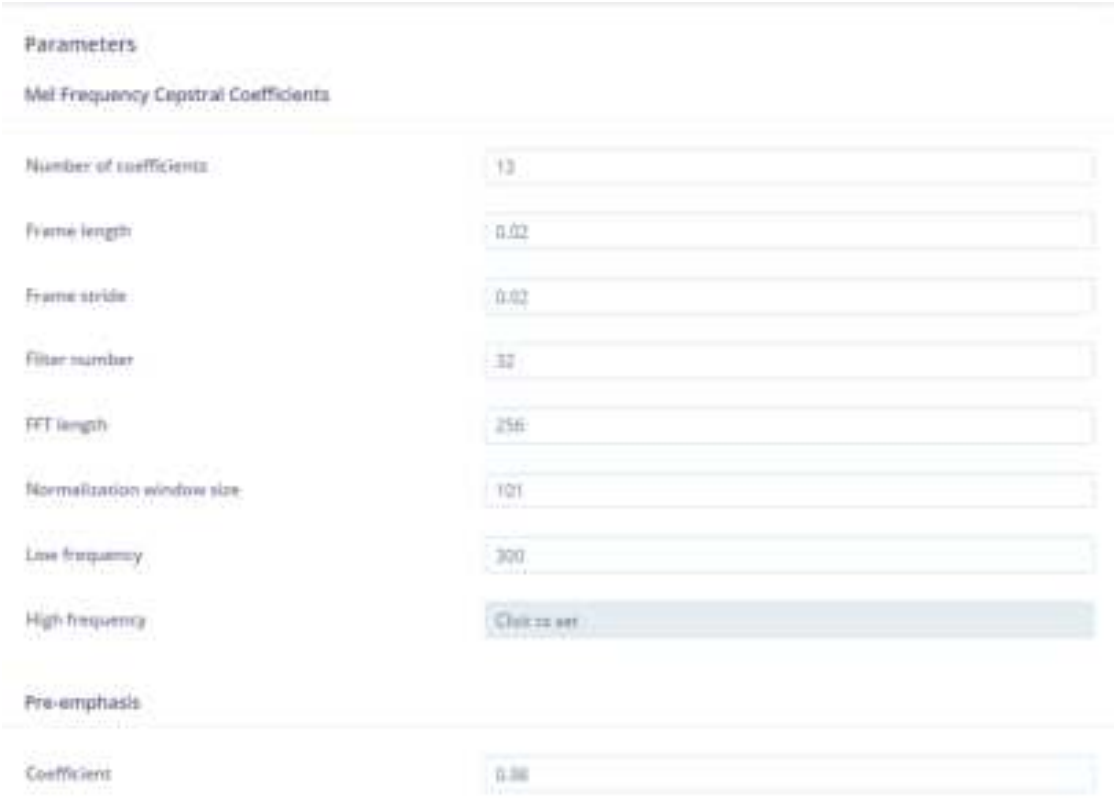


Figura 4.4: *Esempio di configurazione del blocco Custom MFCC*

4.2.2 Implementazione della rete neurale

La componente che contiene la Neural Network rappresenta il fulcro dell'applicazione, è questa che si occupa della classificazione ed è quindi stata implementata con le dovute accortezze per provare ad ottenere un buon modello di inferenza. La preparazione è stata eseguita con la scelta di un blocco *Classification* di Edge Impulse che rende disponibile un ambiente per la creazione della rete neurale con Keras. Una delle implementazioni della rete neurale è visibile in tabella 4.1. Questa è stata sviluppata partendo da una rete neurale di base che è stata elaborata visionando reti neurali per problemi similari e studiandone i meccanismi [40]. Fondamentale poi testare le varie reti per incontrare la configurazione migliore, tutte le prove verranno descritte successivamente, compresi i test che hanno portato alla versione finale. Questo particolare è molto importante poiché molte volte alcune scelte o cambi nell'implementazione funzionano senza rendere visibile

il perché e per questo i valori e l'architettura migliore devono essere scelti empiricamente durante la fase di testing. Sono state realizzate inizialmente due reti neurali di tipo convoluzionale, una con topologia 1D e l'altra 2D, cercando di elaborare in parallelo sia una che l'altra e scegliere poi quella con i migliori risultati. In generale è risaputo che non esiste una rete neurale migliore di altre, ma solo quella più adatta ad un problema o ad un dataset. I parametri come numero di neuroni o di livelli, così come la funziona di attivazione devono essere semplicemente testati nel maggiore numero di combinazioni possibili, però sempre seguendo dei criteri di progettazione. Si è partiti da una rete CNN(convolutional neural network) semplice e man mano è stata aggiunta complessità. In generale sono stati seguiti questi criteri, molto diffusi tra i progettisti di reti neurali e logicamente comprensibili:

- Creare una rete con livelli nascosti con un ordine di dimensione simile all'input e tutti della stessa dimensione, sulla base del fatto che non vi è alcun motivo particolare per variare le dimensioni;
- Iniziare in modo semplice e aumentare la complessità per vedere cosa fa migliorare le prestazioni e cosa invece no;
- Provare a variare le profondità della rete quando sembra si sia raggiunto il massimo risultato dai dati di input;
- Provare ad aggiungere un po' di dropout il più delle volte migliora nettamente le prestazioni (l'aggiunta di dropout può migliorare la generalizzazione, ma può anche aumentare le dimensioni, il numero dei livelli e i tempi di addestramento richiesti poichè ovviamente fa perdere informazioni).

In generale per strutturare la rete iniziale sono stati seguiti i criteri generali di design delle reti neurali, principi di base presenti in [40] e [41] per le reti convoluzionali. In particolare si fa presente che è stata usata una rete convoluzionale per la particolare capacità che ha di riconoscere le features, nella prima parte della rete convoluzionale, e di classificarle negli ultimi strati completamente collegati. Un importante criterio è quello che riguarda il rapporto tra il numero di livelli e l'overfitting per il quale sono state provate le combinazioni per trovare la configurazione migliore, così come per il numero di neuroni. In generale quest'ultimo è stato inizializzato attenendosi ai seguenti criteri:

- Un numero di neuroni nascosti dovrebbe essere compreso tra la taglia dell'input e quella del livello di output;
- Un numero di neuroni nascosti dovrebbe essere pari a 2/3 della taglia del livello di input più quello di output;
- Un numero di neuroni nascosti dovrebbe essere minore del doppio della taglia dell'input.

Mentre per il numero di livelli nascosti:

$$Nt = \frac{Ns}{\alpha * (Ni + No)}$$

Nt=numero di livelli nascosti
Ns=numero di sample
Ni=neuroni in input
No=neuroni in output
α=fattore arbitrario tra 2 e 10

Questi criteri sono stati utilizzati come punto di partenza ma non sono stati acquisiti come verità assolute, infatti non hanno valide dimostrazioni del loro corretto funzionamento [42] e in più in altri progetti [43] hanno funzionato meglio altri tipi di configurazione. In generale infatti è noto che uno o due livelli sono sufficienti per risolvere il maggior numero di problemi non lineari. In generale l'approccio è stato dunque di attenersi ai criteri noti ma non usarli come vincoli, quindi eseguire il maggior numero possibile di prove. In più una rete convoluzionale 2D potrà essere strutturata sicuramente con meno livelli data la sua complessità maggiore. Oltre a questi criteri si è fatta attenzione anche ad altri aspetti durante la costruzione della rete, ad esempio ai dati in input. La rete potrebbe funzionare perfettamente con alcuni dati, ma, essendo che il dataset può essere sempre migliorato, quando si vanno ad aggiungere altri dati le prestazioni possono peggiorare, quindi ad esempio è necessario aumentare i livelli, il tempo di training o i neuroni.

	Filters number	Kernel size	Padding	Pool size	Activation	Strides	Rate
Conv1D	8	3	'same'		'relu'		
MaxPooling1D			'same'	2		2	
Dropout							0.25
Conv1D	16	3	'same'		'relu'		
MaxPooling1D			'same'	2		2	
Dropout							0.25
Flatten							
Dense	13				'softmax'		

Tabella 4.1: *Configurazione per la rete neurale*

La rete neurale è stata poi implementata inizialmente utilizzando l'interfaccia grafica di EdgeImpulse, successivamente per una personalizzazione più efficace si è dovuto passare allo sviluppo in codice Python con l'aiuto del framework TensoFlow(ad esempio in modalità grafica non è possibile variare i parametri dell'ottimizatore, aggiungere livelli di BatchNormalization e personalizzare tanti altri importanti dettagli). Per non creare confusione viene riportata nel dettaglio la creazione della rete in codice Python. Prima della rete neurale vanno eseguite delle operazioni preliminari, nel caso della rete neurale descritta è stato innanzitutto creato il modello sequenziale con la funzione "model = Sequential()" della libreria TensoFlow. Di seguito vengono indicati i livelli utilizzati per l'implementazione. Primo fra tutti un livello che esegue un'azione di **DataAug-**

mentation con il parametro *standard deviation of the noise distribution*(stddev) pari 0.45. Questo rappresenta un parametro di rumore che servirà mitigare anche l'overfitting ma anche a creare più varietà nei dati. Dopodiché è stato inserito un reshape dei dati input per prepararli alla rete neurale ridimensionandoli a $[[input\_length/13], [13]]$ cioè lunghezza input diviso 13 come grandezza dei vettori e 13 canali. Questo perché nel blocco MFCC era stato inserito come numero di coefficienti 13, quindi si era andato a creare un uno spettrogramma di 13 righe e facendo in questo modo la rete neurale userà gli stessi valori dei filtri per i vari vettori che contengono a frequenze diverse le informazioni audio in funzione del tempo. Questo ridimensionamento sarà ovviamente diverso nel caso della rete neurale CNN in due dimensioni. A questo punto sono stati inseriti i livelli descritti nella tabella 4.1. Quindi una struttura di cinque stage più i livelli finali dove vengono appiattiti i dati(attraverso il metodo Flatten) e passati in un ultimo livello "Dense" quindi completamente collegato che avrà tredici neuroni, quindi uno per ogni classe da riconoscere, infine il livello di output. Per quanto riguarda i 2 stage interni sono composti tutti e due da 3 livelli in cui variano poi solamente i parametri. Si può vedere una rete neurale sviluppata in ambiente Python nella figura 4.5. Si è arrivati a

```
# model architecture
model = Sequential()

model.add(tf.keras.layers.GaussianNoise(stddev=0.45))

model.add(Reshape((int(input_length / 13), 13), input_shape
    =(input_length, )))

model.add(Conv1D(8, kernel_size=3, padding='same',
    activation='relu'))
model.add(MaxPooling1D(pool_size=2, strides=2, padding
    ='same'))
model.add(Dropout(0.25))

model.add(Conv1D(16, kernel_size=3, padding='same',
    activation='relu'))
model.add(MaxPooling1D(pool_size=2, strides=2, padding
    ='same'))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(classes, name='y_pred', activation
    ='softmax'))
```

Figura 4.5: *Implementazione della rete neurale in python*

questa configurazione analizzando molte altre reti prese come esempio ,che funzionano bene su problemi simili, e poi adattandola al caso specifico, sempre tenendo a mente che questa rete non può esplodere dimensionalmente poichè se fosse troppo pesante il modello poi violerebbe la memoria massima consentita dalla Portenta. Ogni livello oltre a comprendere i livelli di convoluzione, pooling e activation, hanno anche un livello di dropout, quest'ultimo per i vantaggi descritti nel capitolo precedente, sopratutto per il contenimento dell'overfitting, ma sopratutto per i risultati quasi "magici" derivanti dall'inserimento di questo livello. Successivamente alla definizione del modello sono stati

scelti e configurati importanti parametri e aspetti della rete. Innanzitutto l'ottimizzatore ADAM(ADAptive Moment estimation), il più utilizzato nelle reti neurali, che ha funzionato molto bene anche in questo caso. I parametri passati sono un learning rate dal valore di 0.005 che è sembrato quello ideale dai primi risultati dei test e gli altri due parametri che sono quasi sempre standard. Un processo di Data augmentation di cui è stata presa l'implementazione di Edge Impulse della descrizione teorica in [44]. A questo dopo lo studio dei parametri sono stati forniti i livelli necessari di augmentation. In particolare valori di *Mask time bands* e il valore di *Mask frequency bands* settati inizialmente al livello minimo. Viene usata come funzione di perdita la *categorical crossentropy* dato il tipo di problema, quindi una classificazione non binaria e *one-hot-encoded*. La gradenzza del batch di allenamento è 32 e le epoche 100.

```
23  sa = SpecAugment(spectrogram_shape=[int(input_length / 13),
        13], mF_num_freq_masks=0, F_freq_mask_max_consecutive=0
        , mT_num_time_masks=1, T_time_mask_max_consecutive=1,
        enable_time_warp=False, W_time_warp_max_distance=6,
        mask_with_mean=False)
24  train_dataset = train_dataset.map(sa.mapper(),
        num_parallel_calls=tf.data.AUTOTUNE)
25
26  EPOCHS = args.epochs or 100
27  LEARNING_RATE = args.learning_rate or 0.005
28
29  BATCH_SIZE = 32
30  train_dataset = train_dataset.batch(BATCH_SIZE,
        drop_remainder=False)
31  validation_dataset = validation_dataset.batch(BATCH_SIZE,
        drop_remainder=False)
32
33  opt = Adam(learning_rate=LEARNING_RATE, beta_1=0.9, beta_2
        =0.999)
34  callbacks.append(BatchLoggerCallback(BATCH_SIZE,
        train_sample_count, epochs=EPOCHS))
35
36  model.compile(loss='categorical_crossentropy', optimizer
        =opt, metrics=['accuracy'])
37  model.fit(train_dataset, epochs=EPOCHS, validation_data
        =validation_dataset, verbose=2, callbacks=callbacks)
38  disable_per_channel_quantization = False
```

Figura 4.6: *Setting dei parametri e creazione del modello in python*

Nella figura 4.7 è stata rappresentata la struttura di base della rete neurale implementata, in particolare si possono osservare graficamente i dati e il modo in cui vengono processati. Per il modo in cui sono stati gestiti i parametri i dati vengono suddivisi in 13 canali con un vettore di 50 elementi per ognuno su cui vengono eseguite le operazioni della rete convoluzionale ottenendo i dati in output in un vettore di 3 elementi.

4.3 Preparazione dei dati e Training

Il training è la fase successiva all'implementazione, come si sa queste due fasi potrebbero alternarsi continuamente finché non si trova una configurazione ben funzionante. Viene

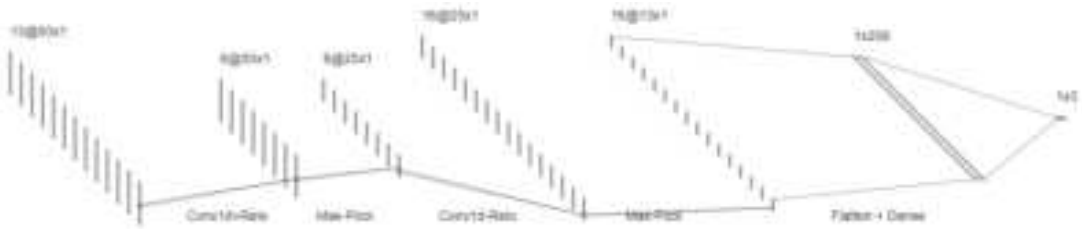


Figura 4.7: *Rappresentazione della rete neurale creata*

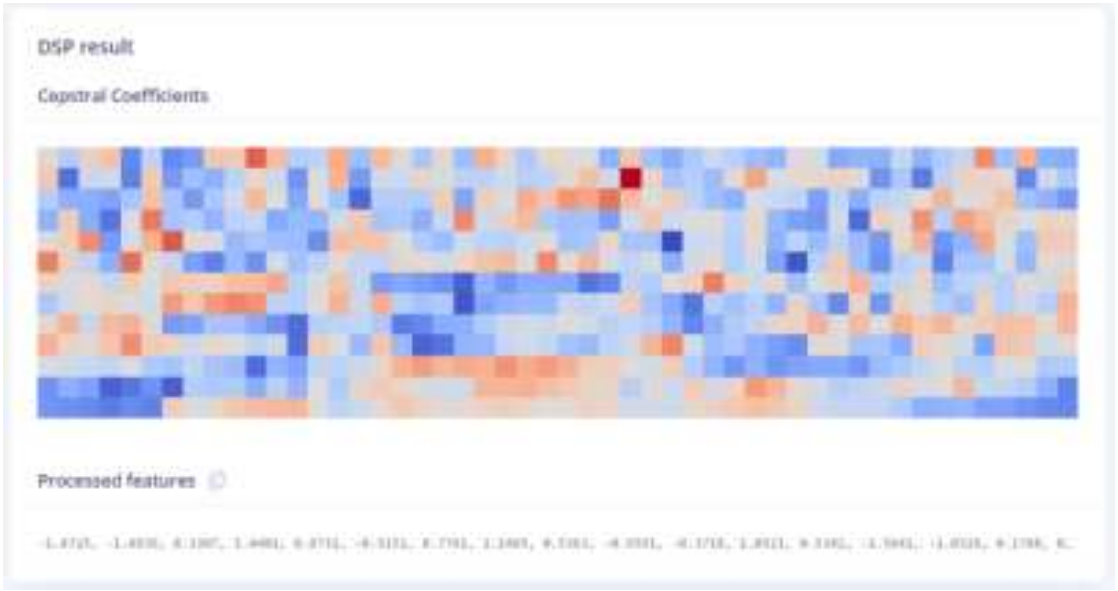
qui descritto come sono state messe insieme le varie componenti per eseguire il training.

4.3.1 Analisi ed estrazione features

Prima dell'allenamento vero e proprio vengono eseguite delle operazioni sui dati come descritto nel capitolo 3. In questa parte viene sfruttata la componente MFCC di cui è stata descritta l'implementazione in questo capitolo 4.2.1. In questa fase vengono dati in input i dati grezzi, o le features grezze, e si ricavano le features elaborate. Edge Impulse permette attraverso un'interfaccia grafica di richiamare il processo di estrazione features. Nella figura 4.8 si può vedere un confronto tra un sample di esempio, prima e dopo essere stato processato. Il risultato è nella forma di spettrogramma e non più di onda sonora, le features grezze ed elaborate in confronto sono molto differenti e il processo porta le 16.000 iniziali(si ricorda che la frequenza è di 16 kHz e la registrazione di un secondo) a diventare 650(il numero dipende dai parametri scelti), rendendole più significative per il processo che si vuole eseguire 2.2.6.

Dopo l'esecuzione del blocco MFCC si possono analizzare i risultati anche con lo strumento di Edge Impulse *Features Explorer* ed analizzarne le performance per il device target. Sono in 4.9 le performance del blocco elaborato, quindi il tempo stimato da Edge Impulse per l'esecuzione di tale blocco sul device . Bisogna calibrare i parametri per evitare di rendere il modello da caricare sul device troppo pesante, sia in termini di memoria che velocità. La memoria occupata e il tempo di elaborazione dovranno essere sommati a quelli della rete neurale e dello sketch di preparazione. Attraverso lo strumento *Data Explorer*, selezionando come criterio il *pre-processing block*, si può eseguire un analisi approfondita dei dati in un ambiente bi-dimensionale in cui la multi-dimensionalità delle features viene ridotta da Edge Impulse attraverso un algoritmo di ridimensionamento.Si osservano i risultati di tale operazione in figura 4.10. Qui sono stati analizzati eventuali **outlier**, eseguendo pulizia dati, rielaborandoli, rieseguendo lo split, o analizzando i dettagli che risultano maggiormente presenti nelle features. Nella rappresentazione grafica iniziano ad apparire già i vari **clusters**, quindi, ricordando che la rete neurale può classificare bene i dati sopratutto se in questa fase si riesce a distinguere le varie classi, si può concludere che il blocco descritto funziona bene con il tipo di dati utilizzati.

(a)



(b)

Figura 4.8: *Dati grezzi ed elaborazione MFCC dello stesso sample*

Figura 4.9: *Performance di elaborazione blocco MFCC*

4.3.2 Training

Nella fase di training vera e propria viene utilizzata la rete neurale descritta in precedenza. Si inserisce l'output del blocco MFCC come input della rete neurale e viene eseguito l'allenamento. L'output dell'esecuzione è visibile nell'interfaccia del software e ci si basa

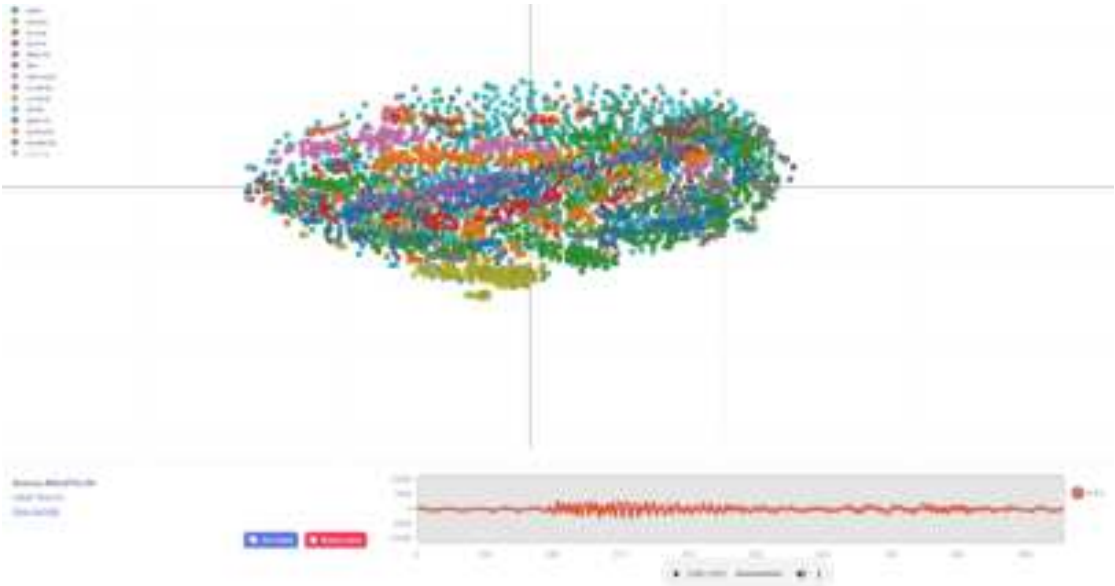


Figura 4.10: *Data explorer con blocco MFCC*

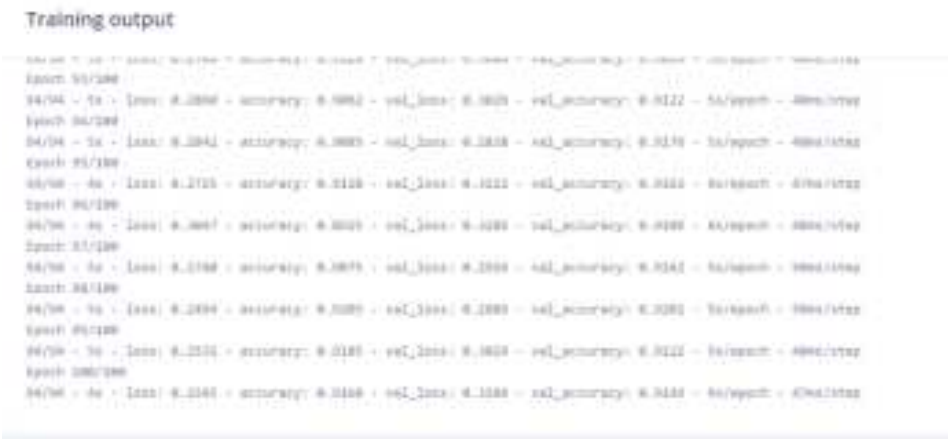


Figura 4.11: *Output del training*

su questo per analizzare l'andamento del training prima dell'analisi dei risultati. Questo perchè è durante questa fase che sono stati valutati importanti fattori come l'overfitting. Nell'output sono stati analizzati l'andamento dei risultati sul training set e validation set attraverso l'output in figura 4.11. Si può apprezzare una buona media di accuratezza e di perdita nelle ultime iterazioni. In questo caso è stato fermato l'allenamento quando dopo circa 20 epoche i risultati sui diversi set di dati si equiparavano. Ciò vuol dire che la rete riesce a generalizzare bene anche sui dati del validation set ma senza esserci distacco rispetto al training set e quindi senza essere arrivati ad un overfitting.

4.4 Deployment per scheda Portenta

Una volta ottenuto il modello si può continuare con l'esecuzione di test prestazionali o andare avanti convertendo il modello in un modello **??**. In particolare sono stati eseguiti i dovuti test sia sul modello standard che su quello ottimizzato con divario di prestazioni molto ridotto tra il modello ottimizzato e originale ma con un guadagno significativo in termini di memoria. E' stato portato avanti il lavoro creando una libreria

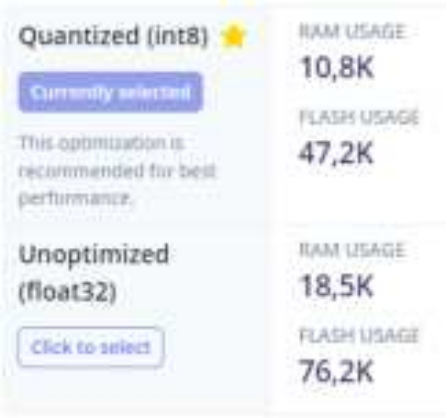


Figura 4.12: *Differenza di memoria stimata da Edge Impulse tra il modello originale e ottimizzato*

del modello in C++ e creando lo sketch che dovrà importare questa libreria. Le azioni di conversione del modello lite e creazione della libreria sono già implementate in Edge Impulse. Una volta ottenuto il modello è stato eseguita la conversione dello stesso, attraverso Edge Impulse, come libreria C++ . A questo punto per poter eseguire il passo finale è necessario utilizzare uno sketch che includendo la libreria suddetta implementa le operazioni descritte nel capitolo 3.5 . La creazione dello sketch è una delle parti più delicate e complesse del sistema e che ne determinano il funzionamento. Lo scopo dello sketch è di eseguire le inferenze in parallelo alle registrazioni, senza perdere dati audio mentre si deducono i risultati, il tutto ragionando nella logica della libreria implementata, quindi richiamandone le sue funzioni. Uno dei parametri più importanti di cui tenere conto è sicuramente la grandezza degli slice. Per ogni finestra di registrazione infatti la libreria farà inferenze unendo il numero di slice consecutivi indicato e creando una finestra di un secondo. All'inferenza successiva poi si scarta l'ultimo slice in testa e se ne aggiunge un altro in coda, quindi con logica FIFO. Da tenere in considerazione che:

- Un numero di slice piccolo per finestra(quindi slice di grandi dimensioni) potrebbe far perdere qualità alle inferenze ma migliorare la complessità computazionale;
- Un numero di slice grande, invece, aumenta il numero di inferenze che si eseguono per una certa quantità fissata di dati, ma le inferenze hanno meno possibilità di perdere una parola pronunciata. Questo aumenta la complessità computazionale e può anche portare ad un overrun dei buffer.

Figura 4.13: *Metodo di inizializzazione dei buffer e del microfono*

Valutando vantaggi e svantaggi è stata selezionata inizialmente una grandezza degli slice di 250 ms, quindi quattro slice che scorrono nella finestra di inferenza. Importante sarà la fase di testing per valutare il numero ottimale. In Arduino Ide l'implementazione dello sketch è eseguita comprendendo diversi metodi e, chiaramente, i metodi di *setup()* e *loop()*. Per implementare l'architettura vista vengono usati due buffer che si alternano iterativamente per le registrazioni e le inferenze, così da poter eseguire in parallelo inferenze e registrazioni. E' importantissimo quindi tenere a mente che la gestione del tempismo è ottimale, quindi quando vengono sovrascritti dati delle registrazioni in uno dei due buffer bisogna essere sicuri che l'inferenza su quei dati è già stata eseguita. Ven-

Figura 4.14: *Metodo di callback passato all'interfaccia PDM*

gono quindi inizializzati, nella funzione in figura 4.13, i due buffer con una grandezza pari alla grandezza dello slice scelto, dopodichè ,utilizzando l'interfaccia *PDM* della libreria Arduino, viene inizializzato il microfono. Poi viene settata la funzione di callback da richiamare quando ci sono dati disponibili dal microfono. Questa funzione, visibile in figura 4.14, li legge e li mette nel buffer attivo, dopodichè setta un parametro(*buf_ready*) a 1, che avvisa che i dati sono pronti e un altro parametro per cambiare il buffer in uso. A questo punto il processo che richiama l'inferenza legge *buf_ready*,lo cambia di nuovo a 0 e inizia l'inferenza sul buffer pieno. Qui bisogna stare molto attenti perchè quando l'inferenza finisce questo processo controlla di nuovo *buf_ready*, se è diverso da 0 vuol dire che l'altro processo, che esegue le registrazioni, aveva già cambiato il parametro e quindi aveva iniziato a registrare sui dati che erano in uso per l'inferenza creando eventuali problemi ed errori di classificazione. Quindi la soluzione in questo caso è diminuire il nu-

```
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;
    }
```

Figura 4.15: *Particolare del metodo loop()*

mero slice e di conseguenza aumentare la grandezza di ogni di ognuno di essi, così che il processo di registrazione ritarda ed evita di sovrapporsi a quello che richiama l'inferenza. In figura 4.15 si può vedere un particolare del metodo di loop nel quale viene controllato attraverso *microphone_inference_record()* la variabile *buf_ready*, questa deve essere trovata a 0, poi si aspetta che il processo di registrazione finisce di riempire uno dei buffer e quindi di mettere la variabile a 1. A questo punto il processo di inferenza rimette subito la variabile a 0 e chiama *run_classifier_continuous()* della libreria per eseguire l'inferenza passando un oggetto *signal*, attraverso il puntatore alla funzione in figura 4.16 , che può accedere al buffer corrente, poi vengono stampati i risultati, dopodiché viene rieseguito il loop.

Figura 4.16: *Funzione con riferimento al buffer corrente*

Capitolo 5

Sperimentazione del sistema ed analisi dei risultati

5.1 Ricerca della configurazione migliore

La parte forse più importante per lo sviluppo di un sistema basato su reti neurali è l'esecuzione di molteplici prove al fine di trovare la configurazione migliore. Molte volte, infatti, in questo ambito si ottengono risultati migliori provando varie configurazioni di cui non sempre si riesce a intuire il significato logico. In questo sottocapitolo vengono usati gli strumenti messi a disposizione da Edge Impulse per testare le prestazioni di un impulso creato e i concetti di valutazione di reti neurali più utilizzati, tra cui le **matrici di confusione** e l'**F1 score**. Verranno mostrate di seguito alcune prove eseguite su tutti i blocchi di processamento e quindi il percorso che ha portato alla configurazione finale. Il dataset è stato inizialmente ridotto a due sole parole per velocizzare le prove nelle varie configurazioni. I risultati del modello creato e quantizzato rispetto a questi dati sono visibili in figura 5.1 e 5.2 nell'interfaccia di Edge Impulse, una rappresentante i risultati sul validation set e l'altra sul test set. Possono essere visionate le matrici di confusione, e i vari punteggi, tra cui l'F1 e le prestazioni computazionali di velocità e memoria. Si nota un margine di errore abbastanza significativo nei risultati del test set ma ancora più ridotto nel validation set. Nella figura 5.3 può essere vista invece una rappresentazione in 2 dimensioni dei dati nel training set distribuiti nello spazio in base ai risultati della rete neurale. Si può vedere qui che i risultati sono per lo più corretti ed è messa in evidenza la suddivisione nei tre cluster.

Questa configurazione ha dato buoni risultati, ma bisogna tenere conto che è stata testata su sole due parole. Di seguito si riportano le prove più significative che sono state eseguite e i loro risultati. Sono state prima eseguite tutte le prove che è possibile fare sul dataset ridotto, dopodiché sono stati usati questi risultati per lavorare sul dataset completo.

Figura 5.1: *Risultati iniziali della rete neurale sul validation set*

Figura 5.2: *Risultati iniziali della rete neurale sul test set*

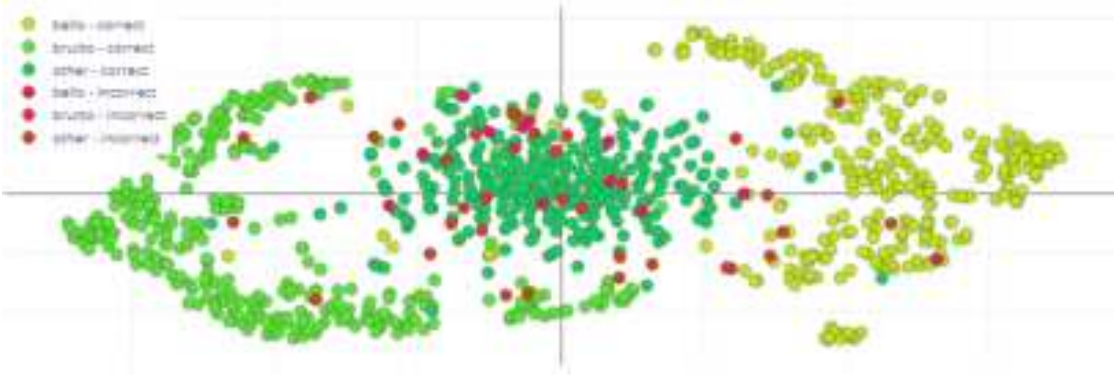


Figura 5.3: *Un'immagine del Feature explorer di Edge Impulse sul dataset iniziale*

5.1.1 Prove sulla classificazione di due parole

In questa sezione vengono riportati i risultati ottenuti con l'utilizzo di un dataset ridotto a due parole, questi non sono sicuramente i risultati definitivi, ma hanno permesso una flessibilità maggiore ed un tempo di computazione dei modelli ridotto, caratteristiche importanti per testare i parametri in un ampio range di valori.

Dataset

La prova più significativa riguardante il dataset è stato lo split tra il rumore e tutte le altre parole per creare un'etichetta per ognuno dei due insiemi, ciò ha portato dei risultati significativi sul set di training e di validation ma non sul test set. Si tenga presente che il distacco tra risultati del training set e test set non è dovuto ad overfitting date le considerazioni in 4.3.2. In particolare sono stati riportati i valori seguenti:

- Validation set accuracy: 91.1%;
- Loss:0.43;
- Test set:72.77%.

Blocco di elaborazione dati

Nella tabella 5.1 è illustrata una parte dei risultati ottenuti sulle prove del blocco MFCC. In particolare si vedono le prime e le ultime otto prove della classifica, ordinate in ordine decrescente in base al punteggio ricevuto e a seconda dei parametri di elaborazione dati audio del blocco MFCC numerati come segue:

- **1**:Cepstral coefficients;
- **2**:Frame length;
- **3**:Frame stride;
- **4**:Filter number;
- **5**:Pre-emphasis coefficient;
- **6**:FFT length;
- **7**:Normalization window size;
- **8**:Low frequency;
- **9**:High frequency;
- **10**:Test set accuracy.

. In generale le prove sul blocco MFCC hanno dimostrato che i valori di alcuni parametri funzionano meglio in alcune combinazioni particolari mentre per altri si riesce a notare un trend di miglioramento quando si trovano in un determinato range di valori. Dopo l'analisi dei vari parametri sono stati studiati esempi di molteplici prove similari come [45] [46]. In particolare con la prova numero uno è stato raggiunto il picco di accuratezza con un valore di 89.20% che, considerando la rete neurale triviale usata, equivale a un risultato importante, anche tenendo a mente che il risultato iniziale con i parametri proposti da Edge impulse mostrava un 82.63% di precisione. Questa combinazione di

#	1	2	3	4	5	6	7	8	9	10
1	13	0.02	0.02	32	0.98	1024	101	300	8000	89.20%
2	13	0.02	0.02	32	0.98	512	101	300	8000	87.79%
3	13	0.02	0.03	16	0.98	512	101	300	8000	87.32%
4	13	0.02	0.03	16	0.98	256	101	300	8000	85.92%
5	13	0.03	0.03	32	0.98	256	101	300	8000	84.98%
6	13	0.02	0.03	32	0.98	256	101	300	8000	84.98%
7	13	0.03	0.03	16	0.98	256	101	300	8000	84.98%
8	13	0.03	0.02	32	0.98	256	101	300	8000	84.51%
50	12	0.03	0.03	32	0.95	256	101	300	8000	81.22%
51	13	0.01	0.02	32	0.98	256	101	300	8000	81.22%
52	13	0.01	0.01	32	0.98	256	101	300	8000	81.22%
53	13	0.02	0.02	32	0.98	256	101	100	6000	79.34%
54	13	0.02	0.02	32	0.98	256	201	300	8000	78.87%
55	13	0.02	0.01	32	0.98	256	101	300	8000	79.81%
56	8	0.02	0.02	32	0.98	256	101	300	8000	79.81%
57	13	0.02	0.02	32	0.96	256	101	300	8000	79.81%

Tabella 5.1: *Risultati dei primi e ultimi otto modelli ordinati in base all'accuratezza in funzione dei parametri della lista 5.1.1*

parametri si differenzia poi da quella di partenza solo per il valore *FFT length* quattro volte superiore, da 256 a 1024, si consideri che è bene tenerlo sempre come un numero multiplo di 2[47]. In particolare questo è quindi il parametro che funziona meglio per migliorare le prestazioni del modello,infatti anche la seconda e la terza prova nella classifica hanno questo valore molto alto. Di contro è però quello che peggiora maggiormente la complessità computazionale. Da notare che il valore di alcuni parametri modifica la forma del risultato di questo blocco e in particolare la dimensionalità dei dati, di conseguenza questa deve essere modificata anche in ingresso alla rete neurale. Sono stati usati i risultati di queste prove per lavorare poi sul dataset completo.

Rete neurale

Si passa alle prove eseguite sulla rete neurale nel caso del dizionario di due parole. Ovviamente la configurazione di rete ottenuta non può essere usata allo stesso modo quando poi si passerà al dataset completo poichè la quantità di dati in input alla rete è uno dei fattori più importanti per la sua strutturazione. Questo passaggio serve però a farsi un idea di ciò che migliora o peggiora i risultati della rete neurale. Si consideri che viene utilizzato il blocco MFCC che nel passaggio precedente ha dato i migliori risultati e che la struttura nella rete neurale viene in base a precedenti considerazioni vista come una serie di stage in numero variabile in cui per ogni stage vi sono il livello convoluzionale, quello di Pooling, di BatchNormalization e Dropout variabili. In figura 5.2 si può vedere una sintesi dei parametri utilizzati per le prove e il loro significato.

Filters	Il numero di filtri che vengono applicati per ogni livello convoluzionale
Kernel size	La grandezza di ogni filtro(un filtro con kernel=5 avrà ad esempio 25 valori)
Pool	E' la grandezza del filtro di pooling che quindi determina la riduzione dei dati e il modo in cui vengono messe in risalto le informazioni
Stride	Rappresenta il numero passi con cui ogni filtro di pooling si sposta
Dropout	La percentuale di collegamenti da scartare su quel livello
BatchNormalization	Applica una trasformazione che mantiene la media di output vicina a 0 e la deviazione standard vicino a 1
Learning rate	Indica alla CNN quanto i valori devono spostarsi nella direzione opposta al gradiente in ogni passo
Epoche	Il numero di iterazioni dell'algoritmo di apprendimento completo(un passo forward ed uno backward)

Tabella 5.2: *Parametri utilizzati per le prove sulla rete neurale*

Prova a

- Tipologia di rete: Convoluzionale 1D;
- Dataset: ridotto a 3 classi;
- Numero di prove: 150.

Per questa prima prova sono state usate 150 combinazioni dei parametri che si è scelto di testare sulla struttura della rete convoluzionale a una dimensione descritta in precedenza. I risultati sono nella tabella 5.3. Si consideri inoltre che la tabella mostra solamente i cambiamenti rispetto ai parametri in 5.2, la struttura di base che era stata scelta è rimasta uguale, quindi non vengono considerati ad esempio gli ultimi due livelli della rete, Flatten e Dense, i primi di ridimensionamento e rumore e altri parametri della rete. La tabella mostra i risultati ottenuti solo in una parte, infatti vengono mostrate le otto combinazioni che hanno raggiunto il miglior punteggio e le otto con il punteggio peggiore. In questo caso le colonne nella tabella saranno associate ai parametri e numerate come segue:

- **1**:Filters, Kernel size primo livello convoluzionale;
- **2**:Pool, stride del primo livello di Pooling;
- **3**:Filters, Kernel size del secondo livello convoluzionale;
- **4**:Pool , stride, del secondo livello di Pooling;
- **5**:Filters, Kernel size del terzo livello convoluzionale;
- **6**:Pool, stride del terzo livello di Pooling;

- **7**:Filters, Kernel size del quarto livello convoluzionale;
- **8**:Pool, stride del quarto livello di Pooling;
- **9**: Valore di dropout per ogni stage(quando per ogni stage non è uguale viene indicato esplicitamente per ogni stage);
- **10**: Batch normalization in ogni stage;
- **11**: Learning rate;
- **12**: Epoche;
- **13**: Test set accuracy.

#	1	2	3	4	5	6	7	8	9	10	11	12	13
1	32,3	2,2	32,3	2,2	32,3	2,2	32,3	2,2	0.2	si	0.005	80	96.24%
2	32,3	2,2	32,3	2,2	32,3	2,2	64,3	2,2	0.2	si	0.005	80	94.37%
3	32,3	2,2	32,3	2,2	32,3	2,2	32,3	2,2	0.22	si	0.005	80	94.79%
4	32,3	2,2	32,3	2,2	32,3	2,2	32,3	2,2	0.18	si	0.005	80	93.43%
5	32,3	2,2	32,3	2,2	32,3	2,2			0.4	si	0.005	80	93.43%
6	16,5	4,2	32,5	4,2	48,3	4,2			0.2	si	0.005	80	93.43%
7	32,3	2,2	32,3	2,2	32,3	2,2	48,3	2,2	0.2	si	0.005	80	92.96%
8	32,3	2,2	32,3	2,2	32,3	2,2			0.25	no	0.005	80	92.96%
143	16,3	2,2	32,3	2,2					0.25	no	0.005	80	88.73%
144	16,5	2,2	32,3	2,2					0.2	si	0.005	80	88.26%
145	16,3	2,2	32,3	2,2					0.1	si	0.005	80	88.26%
146	16,3	9,2	16,3	2,2					0.25	no	0.005	80	88.26%
147	32,3	2,2	32,3	2,2					0.25	no	0.005	80	88.26%
148	16,3	2,2	16,3	2,2					0.25	si	0.005	80	87.79%
149	16,5	2,2	16,5	2,2					0.25	si	0.005	80	87.32%
150	64,3	2,2	128,3	2,2					0.2	si	0.005	80	82.16%

Tabella 5.3: *Prova a: Risultati dei primi e ultimi otto modelli ordinati in base all'accuratezza in funzione dei parametri della lista 5.1.1*

Dunque sono state eseguite queste prime prove con lo scopo di trovare un trend di crescita dei risultati della rete neurale. Come si può notare da un'analisi attenta della tabella alcuni parametri o funzioni danno risultati totalmente diversi quando applicati a una rete di piccole dimensioni rispetto ad una rete più grande. Ad esempio l'aggiunta del livello di batch normalization peggiorava inizialmente i risultati, migliorandoli invece nettamente quando la rete è divenuta via via più grande. Inoltre diversi studi hanno dimostrato che velocizza l'apprendimento della rete [48], qui si suggerisce di inserire il livello di BatchNormalization tra il livello di convoluzione e attivazione, ma altri studi [49] hanno riportato che è meglio inserirlo dopo l'attivazione, che è il modo in cui è stato implementato nel sistema proposto. Il dropout funzionava bene inizialmente con valori nel range tra 0.25 e 4, mentre con la crescita della rete i risultati migliori sono stati ottenuti con un dropout di 0.2. Le prove eseguite provando ad usare differenti

funzioni di attivazione avanzate non hanno portato ai risultati sperati ma solamente ad un peggioramento. I parametri dei livelli di pooling funzionano bene nei valori standard di 2 per la pool size e 2 per il parametro stride. Per il resto si è notato un trend positivo, come ci si aspettava, andando ad aggiungere neuroni o interi livelli, quindi appesantendo la rete, anche se una volta raggiunta una certa complessità la rete peggiora in termini di prestazioni. Nelle prove eseguite è stato raggiunto il picco dei risultati nella prova numero 1 con un 96.24% di accuratezza che rappresenta una risultato molto alto e quasi ideale. Nella figura 5.4 si può vedere questo picco, ed in particolare una raffigurazione di un campione dei risultati. Si nota come, una volta raggiunto un certo numero di parametri di allenamento ed aver quindi reso la rete complessa, i risultati iniziano a peggiorare, quindi può essere ad esempio un problema di overfitting in cui la rete è troppo complessa per i dati. In particolare si nota che un numero basso di livelli fa aumentare la velocità con cui viene raggiunto il picco.

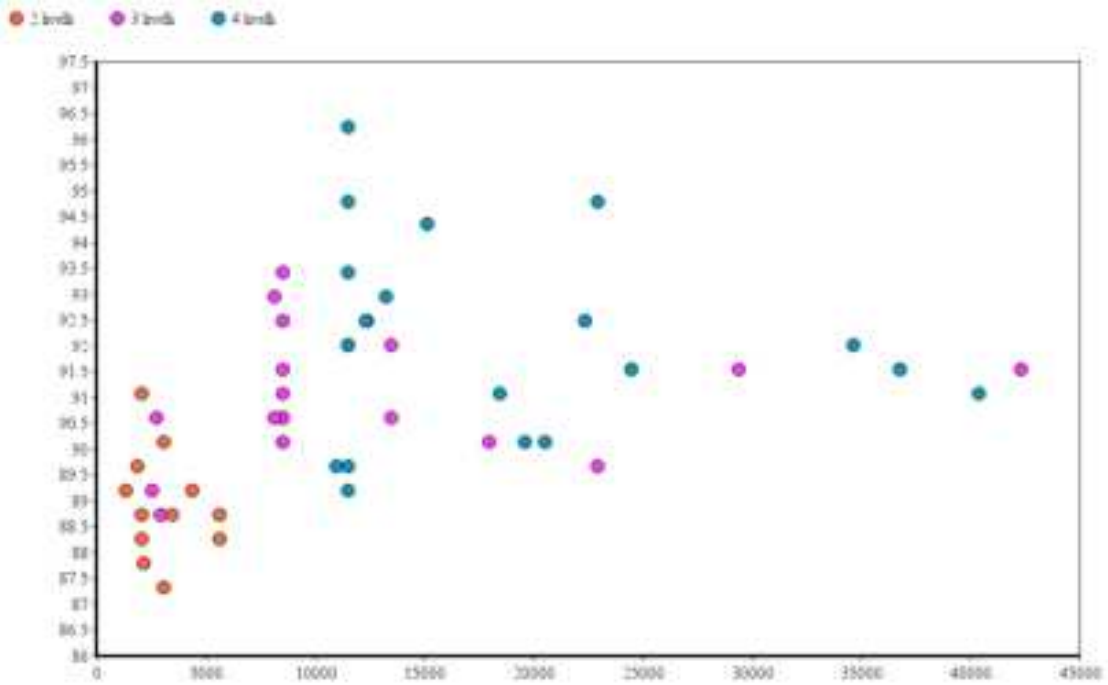


Figura 5.4: *Test set accuracy sui risultati della prova a in funzione del numero di parametri*

Prova b

- tipologia di rete: Convoluzionale 2D;
- Dataset: ridotto a 3 classi;
- Numero di prove: 150.

Successivamente, usando lo stesso dataset, sono state eseguite ulteriori prove, questa volta utilizzando una rete convoluzionale 2D. Questo con lo scopo di vedere quale delle due funziona meglio in questo tipo di contesto. Nella tabella **??**sono raccolti i risultati ottenuti sostituendo i livelli convoluzionali 1D con quelli 2D così come i livelli di maxpooling 1D con la versione in 2D. I parametri testati sono gli stessi della prova a e numerati in tabella allo stesso modo.

#	1	2	3	4	5	6	7	8	9	10	11	12	13
1	16,3,	2,2	32,2,	2,2	64,2,	2,2	128,3,	2,2	0.05	no	0.0025	20	91.08%
2	16,3,	2,2	32,2,	2,2	64,2,	2,2	128,3,	2,2	0.1	no	0.0025	30	91.08%
3	16,3,	2,2	32,3,	2,2	64,3,	2,2	128,3,	2,2	0.05,0.05,0.05,0.5	si	0.0025	20	90.61%
4	16,3,	2,2	32,2,	2,2	64,2,	2,2	128,3,	2,2	0.12	no	0.0025	30	90.14%
5	32,3,	2,2	32,2,	2,2	128,2,	2,2			0.05	no	0.0025	20	90.14%
6	32,3,	2,2	64,2,	2,2	128,2,	2,2			0.04	no	0.0025	20	90.14%
7	32,3,	2,2	64,2,	2,2	128,2,	2,2			0.05	no	0.0025	20	90.14%
8	32,3,	2,2	128,2,	2,2	128,2,	2,2			0.05	no	0.0025	20	89.67%
143	64,3,	2,2	128,3,	2,2					0.6	no	0.0025	70	84.51%
144	128,3,	2,2	128,3,	2,2					0.6	no	0.0025	70	84.51%
145	16,3,	2,2	32,2,	2,2					0.3	no	0.0025	40	84.04%
146	8,3,	2,2	16,2,	2,2					0.2	no	0.0025	40	83.57%
147	16,3,	2,2	32,2,	2,2					0	no	0.0025	40	83.10%
148	16,3,	2,2	32,3,	2,2					0.5	no	0.0025	70	81.22%
149	64,3,	2,2	128,2,	2,2					0.7	si	0.0025	70	78.87%
150	8,3,	2,2	16,3,	2,2					0.5	no	0.0025	70	77.46%

Tabella 5.4: *Prova b: Risultati dei primi e ultimi otto modelli ordinati in base all'accuratezza in funzione dei parametri della lista 5.1.1*

I risultati ottenuti in questa prova sono peggiori rispetto a quando si è fatto uso della rete convoluzionale 1D. La combinazione che ha raggiunto il risultato migliore ha comunque un valore con sei punti percentuali in meno rispetto alla combinazione migliore della prova a. Anche il risultato peggiore con una percentuale del 77.46% di accuratezza è inferiore al precedente risultato peggiore. Anche in questo caso si è notato un trend di miglioramento all'aumento del numero di parametri che diventa un peggioramento quando il numero diventa eccessivamente alto. I livelli di BatchNormalization, inaspettatamente, non hanno funzionato bene come nel caso della rete convoluzionale 1D, il dropout ha funzionato meglio quando settato ad un valore molto basso, come la combinazione della prova 1 che infatti ha un dropout pari a 0.005, così come le combinazioni che hanno raggiunto i punteggi migliori. Una particolarità da notare riguarda la struttura della rete, in questo caso una struttura lineare come nel caso della prova 1 non offre buoni risultati, infatti le combinazioni migliori hanno valori crescenti di neuroni man mano che si aggiungono livelli. Altro particolare importante riguarda il numero di epoche che è stato ridotto a 20, era prevedibile che un valore più basso funzionava meglio nelle reti convoluzionali 2D.

5.1.2 Prove sul dataset completo

Dopo aver raggiunto dei risultati soddisfacenti con il modello di classificazione di due parole si è passati all'implementazione del modello di classificazione del dizionario com-

pleto. Per il blocco MFCC sono stati utilizzati i risultati delle prove precedenti, poichè questo lavora sulla singola parola e sulle caratteristiche che vengono estratte, quindi un dataset più esteso non influisce su questo blocco. Invece per la rete neurale si è partiti dalle configurazioni migliori ottenute in precedenza, usando le conoscenza acquisite durante il precedente allenamento "più semplice" e andando ad aggiungere le opportune modifiche. Si consideri sopratutto che va sicuramente aggiunta complessità per processare un dataset che passa da 15 minuti a più di 1 ora e 30 minuti.

Rete neurale

Come nel caso precedente verranno analizzati i risultati per quanto riguarda una prima struttura convoluzionale 1D ed una seconda 2D. Essendo che ora la ricerca ha l'obiettivo di ottenere il modello che sarà utilizzato per comporre il sistema finale, deve essere posta particolare attenzione alla complessità computazionale.

Prova c

- Tipologia: Convoluzionale 1D;
- Dataset: tutte le classi;
- Numero prove:100.

Questa volta, data la complessità più elevata della struttura i parametri su cui è stata eseguita la ricerca sono in numero maggiore, quindi è allo stesso tempo cambiata la numerazione nelle tabelle. In particolare è identica a quella eseguita in precedenza fino al numero 8, di seguito la numerazione dal numero 9 al 17, quindi con i nuovi parametri che vengono aggiunti alla ricerca:

- **9**:Filters, Kernel size del quinto livello convoluzionale;
- **10**:Pool, stride del quinto livello di Pooling;
- **11**:Filters, Kernel size del sesto livello convoluzionale;
- **12**:Pool, stride del sesto livello di Pooling;
- **13**: Valore di dropout per ogni stage(quando per ogni stage non è uguale viene indicato esplicitamente per ogni stage);
- **14**: Batch normalization in ogni stage;
- **15**: Learning rate;
- **16**: Epoche;
- **17**: Test set accuracy.

#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	96,3,	2,2	96,3,	2,2	96,3,	2,2	96,3,	2,2	96,3,	2,2	96,3,	2,2	0.15	si	0.002	100	93.00%
2	92,3,	2,2	96,3,	2,2	96,3,	2,2	96,3,	2,2	96,3,	2,2	96,3,	2,2	0.15	si	0.005	80	92.90%
3	98,3,	2,2	98,3,	2,2	98,3,	2,2	98,3,	2,2	98,3,	2,2			0.3	si	0.005	80	92.90%
4	96,3,	2,2	96,3,	2,2	96,3,	2,2	96,3,	2,2	96,3,	2,2			0.25	si	0.002	100	92.90%
5	98,3,	2,2	98,3,	2,2	98,3,	2,2	98,3,	2,2					0.25	si	0.005	100	92.57%
6	96,3,	2,2	96,3,	2,2	96,3,	2,2	96,3,	2,2	96,3,	2,2			0.15	si	0.002	100	91.82%
7	98,3,	2,2	98,3,	2,2	98,3,	2,2	98,3,	2,2	98,3,	2,2	0.3			si	0.005	80	91.60%
8	64,3,	2,2	64,3,	2,2	64,3,	2,2	64,3,	2,2			0.25			si	0.005	80	91.50%
93	128,3,	2,2	160,3,	2,2	192,3,	2,2							0.35	si	0.005	80	88.05%
94	16,3,	2,2	32,3,	2,2	64,3,	2,2	128,3,	2,2	256,3,	2,2			0.35	si	0.005	80	88.05%
95	64,3,	2,2	128,3,	2,2	256,3,	2,2							0.45	si	0.005	80	87.62%
96	128,3,	2,2	256,3,	2,2	512,3,	2,2							0.45	si	0.005	80	87.84%
97	128,3,	2,2	256,3,	2,2									0.2	si	0.005	50	85.04%
98	512,3,	2,2	512,3,	2,2									0.45	si	0.005	80	84.82%
99	8,3,	2,2	16,3,	2,2	32,3,	2,2	64,3,	2,2	128,3,	2,2			0.35	si	0.005	80	81.59%
100	64,3,	2,2	64,3,	2,2	64,3,	2,2	64,3,	2,2					0.25	no	0.005	80	80.95%

Tabella 5.5: *Prova c: Risultati dei primi e ultimi otto modelli ordinati in base all'accuratezza in funzione dei parametri della lista 5.1.2*

I risultati della prova c sono molto importanti per quanto riguarda il sistema poichè da questo punto in poi è stato utilizzato il dizionario completo. Si è raggiunta una precisione molto elevata, del 93.00%, ma essendo il dataset completo la complessità del problema era molto più alta della prova a. La rete convoluzionale 1D ha dimostrato di funzionare molto bene per questo problema e per questo dataset. C'è stato, come si aspettava, il bisogno di complicare la rete neurale, e in particolare si è partiti da una rete semplice, cioè quella con i risultati migliori nella prova a andando via via ad aggiungere complessità4.2.2. La struttura che ha ottenuto il miglior risultato ha, come nella prova a, un numero uguale di neuroni per i vari livelli, i livelli di batch normalization hanno funzionato molto bene e per quanto riguarda il dropout ha funzionato meglio, come nella prova a, un valore inversamente proporzionale al numero di parametri. Di nuovo non ci sono stati miglioramenti apprezzabili modificando i parametri dei livelli di pooling. Data la qualità dei risultati ottenuti si prende in considerazione di usare queste reti, in particolare le prime classificate, nel sistema finale. Su questo aspetto verranno eseguite ulteriori analisi, infatti si ricorda che le prove descritte non prendono in considerazione la complessità dei modelli.

Prova d

- Tipologia: Convoluzionale 2D;
- Dataset: Tutte le classi;
- Numero prove : 100.

Le prove con la rete convoluzionale in due dimensioni, anche nel caso dataset completo, non hanno ottenuto buoni risultati, o quantomeno sono risultati di qualità inferiore alla rete convoluzionale 1D. La tabella, di nuovo, ha la stessa struttura della precedente a differenza che i livelli di convoluzione e di pooling arrivano alla colonna 10. Questa volta non si è riusciti a raggiungere un punteggio che arriva al 90% e la maggior parte dei risultati si trovano in un range tra il 79% e l'85%. Il dropout ha funzionato bene quando

è stato settato ad un valore basso, tra lo 0.005 e lo 0.1 . Si notano delle differenze anche con i risultati delle reti 2D della prova b, in particolare si nota che hanno funzionato meglio le prove con i livelli di batch normalization e quelle che mantenevano nei diversi stage un numero uguale di neuroni.

#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	64,3,	2,2	64,3,	2,2	64,3,	2,2	64,3,	2,2			0.1	si	0.002	30	87.30%
2	32,3,	2,2	32,3,	2,2	64,3,	2,2	64,3,	2,2			0.05	si	0.002	30	86.98%
3	32,3,	2,2	32,3,	2,2	32,3,	2,2	32,3,	2,2			0.05	si	0.002	30	86.98%
4	64,3,	2,2	64,3,	2,2	64,3,	2,2	64,3,	2,2			0.05	si	0.002	30	86.01%
5	64,3,	2,2	64,3,	2,2	64,3,	2,2	64,3,	2,2			0.1	si	0.002	50	85.90%
6	64,3,	2,2	64,3,	2,2	64,3,	2,2	64,3,	2,2			0.1	si	0.003	30	85.15%
7	64,3,	2,2	64,3,	2,2	64,3,	2,2	64,3,	2,2			0.1	si	0.001	30	85.15%
8	32,3,	2,2	32,3,	2,2	64,3,	2,2	64,3,	2,2			0.1	si	0.002	30	85.15%
93	32,3	2,2	32,3	2,2	32,3	2,2	32,3	2,2			0.1	si	0.002	30	80.73%
94	32,3,	2,2	32,3,	2,2	32,3,	2,2	32,3,	2,2	32,2,	2,2	0.05	no	0.002	30	80.73%
95	32,3,	2,2	32,3,	2,2	32,3,	2,2	32,3,	2,2	32,2,	2,2	0.06	no	0.002	30	80.73%
96	8,3,	2,2	16,3,	2,2	32,3,	2,2					0.2	si	0.002	70	80.72%
97	32,3	2,2	128,3,	2,2	256,3	2,2					0.1	si	0.002	30	79.33%
98	64,3,	2,2	64,3,	2,2	128,3,	2,2	128,3,	2,2			0.3	si	0.002	70	78.58%
99	32,3,	2,2	64,3,	2,2	128,3,	2,2					0.05,0.05,0.5,0.5	si	0.002	30	76.75%
100	16,3,	2,2	16,3,	2,2	32,3,	2,2	64,3,	2,2			0.35	si	0.002	70	76.43%

Tabella 5.6: *Prova d: Risultati dei primi e ultimi otto modelli ordinati in base all'accuratezza in funzione dei parametri della lista 5.1.2*

5.1.3 Scelta del modello

Dopo aver eseguito le varie prove si è passati all'analisi dei migliori modelli ottenuti e alla loro comparazione. Da qui i modelli verranno nominati con il numero in classifica e la lettera che indica di quale prova fanno parte.

Comparazione modello originale e ottimizzato

Prendendo le prove c e d della sezione5.1.2, cioè quelle sul dataset completo, sono state prese le combinazioni con i punteggi più alti che fanno tutte parte della prova c, cioè con rete convoluzionale 1D. E' stato però anche analizzato il modello 1d, quello con il punteggio più alto della prova d. In figura 5.5 per ogni modello si può visualizzare il modo in cui vengono clusterizzati i dati delle varie classi. Una divisione netta tra le varie classi può essere sinonimo di un modello di qualità. Questo come già detto è utile anche a trovare eventuali outlier nei dati. La tabella 5.7 viene usata per eseguire una prima analisi su Training accuracy, Loss, Test accuracy e numero di parametri. I modelli questa volta sono rappresentati dalle colonne. Come modelli destinati alla comparazione si è scelto di prenderne sei, non solo in base ai criteri della classifica. Infatti sono stati selezionati i primi tre, il quinto e l'ottavo modello della prova c, questi ultimi date le ottime prestazioni e il numero ridotto di neuroni e livelli. In più è stato usato il modello 1 della prova d(in cui si usavano modelli 2D), per confrontarlo con quelli a 1 dimensione per controllare approfonditamente l'utilità di questo tipo di reti nel problema proposto.

Riguardo la tabella 5.7 si possono osservare alcuni tra i parametri più importanti che caratterizzano i vari modelli e quindi confrontarli tra loro. Il parametro più importante è sicuramente la test accuracy, che è anche il parametro di classificazione nelle prove precedenti. Ma in più qua ci si può fare un'idea della pesantezza del modello osservando

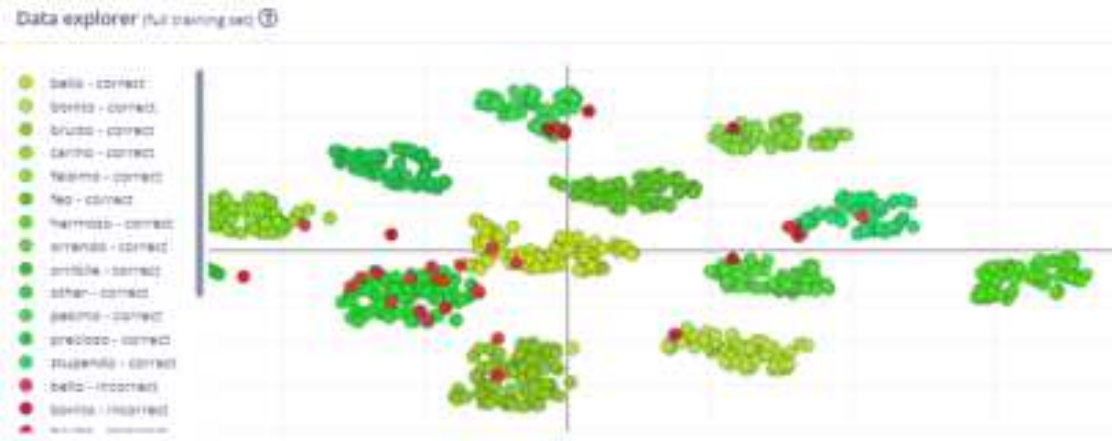


Figura 5.5: *Data explorer di Edge impulse sui risultati del training set con il modello 1c*

	1c	**1d**	**2c**	**3c**	**5c**	**8c**
Training accuracy	94.5%	90.1%	93.6%	94.2%	94.7%	93.1%
Loss	0.43	0.32	0.48	0.33	0.51	0.32
Test accuracy	93.00%	87.30%	92.90%	92.90%	92.57%	91.50%
Numero di parametri	109861	114765	146125	119245	97327	43981

Tabella 5.7: *Comparazione dei modelli che hanno ottenuto il punteggio più alto sul dataset completo*

il numero di parametri che, tra questi analizzati, raggiunge il picco nel modello 2c con 146125 parametri. Nella tabella 5.8 è possibile osservare l'accuratezza dei vari modelli per ogni classe, insieme all'F1 score, di cui si ricorda che il calcolo è:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Un problema comune in questo caso è il punteggio riguardante la classe other che raggiunge livelli non alti quanto le altre classi. Il valore più apprezzabile raggiunto è per il modello 1c con un F1 score di 0.82.

Per quanto riguarda la bontà del modello, in termini di qualità della classificazione, il modello 1c prevale su tutti. Però l'altra analisi importante che deve essere eseguita, ricordando che ci troviamo nel contesto di device portatili, quindi con capacità e memoria limitata, riguarda la complessità spaziale e temporale del modello. Il numero di parametri visto precedentemente fornisce già un indicazione importante. Per quanto riguarda la complessità temporale sono state eseguite le analisi attraverso le stime ottenute dalla piattaforma di Edge Impulse, in più sono state utilizzate le funzioni messe a disposizioni dalla libreria Arduino precedentemente creata per calcolare i tempi effettivi delle

	1c	1d	2c	3c	5c	8c
Bello	90.2%	93.4%	95.1%	91.8%	94.3%	90.2%
	0.93	0.94	0.94	0.94	0.93	0.93
Bonito	100%	95.4%%	96.9%	98.5%	92.3%	98.5%
	0.95	0.89	0.98	0.98	0.94	0.98
Brutto	92.9%	86.7%	93.9%	96.9%	95.9%	92.9%
	0.95	0.91	0.96	0.97	0.97	0.94
Carino	93.4%	86.9%	93.4%	96.7%	95.%	91.8%
	0.94	0.92	0.94	0.97	0.92	0.93
Feisimo	89.8%	91.5%	88.1%	89.8%	91.5%	89.8%
	0.91	0.84	0.92	0.93	0.93	0.91
Feo	96.7%	96.7%	93.4%	96.7%	95.1%	91.8%
	0.98	0.94	0.97	0.96	0.97	0.95
Hermoso	98.4%	95.2%	100%	96.8%	98.4%	98.4%
	0.99	0.95	0.99	0.97	0.98	0.98
Orrendo	96.7%	88.3%	98.3%	98.3%	93.3%	95%
	0.98	0.92	0.99	0.98	0.96	0.97
Orribile	90.3%	83.9%	98.4%	95.2%	96.8%	91.9%
	0.93	0.89	0.98	0.97	0.98	0.96
Other	80.4%	57.6%	77.2%	76.1%	76.1%	69.9%
	0.82	0.68	0.81	0.82	0.80	0.78
Pesimo	93.3%	83.3%	90%	90%	91.7%	90%
	0.93	0.86%	0.92	0.92	0.93	0.92
Precioso	100%	83.1%	94.9%	96.6%	98.3%	96.6%
	0.99	0.90	0.97	0.97	0.98	0.97
Stupendo	95.6%	80.9%	91.2%	92.6%	89.7%	88.2%
	0.95	0.87	0.95	0.95	0.94	0.94

Tabella 5.8: *Accuratezza ed F1 score delle diverse classi in base alla classificazione per ogni modello candidato*

inferenze nel dispositivo. Per quanto riguarda i tempi stimati è possibile visualizzarli in figura 5.6 a confronto. Il modello 1d, con rete convoluzionale 2D offre sicuramente le prestazioni peggiori in questi termini, superando nettamente i tempi di inferenza degli altri modelli. I modelli 5c e 8c invece ,ricordando che l'accuratezza delle classificazioni era comunque elevata, offrono dei tempi di inferenza molto ridotti rispetto agli altri, ed è questo un dato da tenere fortemente in considerazione. Infatti come si vedrà più avanti la progettazione dello sketch Arduino introduce alcune problematiche. Anche se ad esempio il tempo di inferenza del modello 1c potrebbe sembrare accettabile non è detto che lo sia e potrebbe esserci il bisogno di usare un modello con un'accuratezza minore. Questi dettagli verranno approfonditi più avanti.

Nel grafico 5.7 invece vengono messi a confronto i dati riguardanti la memoria occupata sia da dati e variabili, cioè la memoria ram in blu, che dai dati occupati dal programma stesso, nella memoria flash in azzurro. In questo caso i modelli 1c e 2c fanno un uso della memoria flash addirittura maggiore del modello 1d, anche se questo supera nettamente tutti gli altri per quanto riguarda la memoria ram. In ogni caso ,date le caratteristiche di memoria della scheda Portenta, la memoria utilizzata da questi modelli non dovrebbe superarne le capacità. A questo punto l'unico problema potrebbe essere

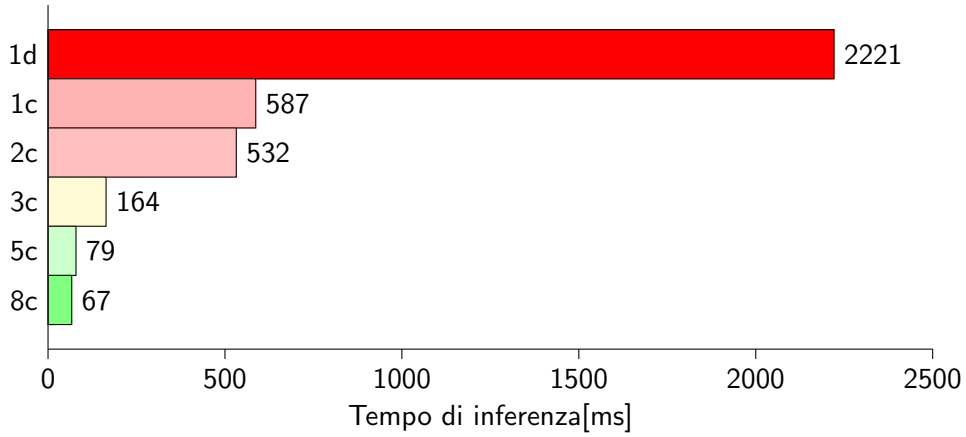


Figura 5.6: *Comparazione tempo di inferenza(stimato) per i modelli candidati*

creato dal tempo di inferenza.

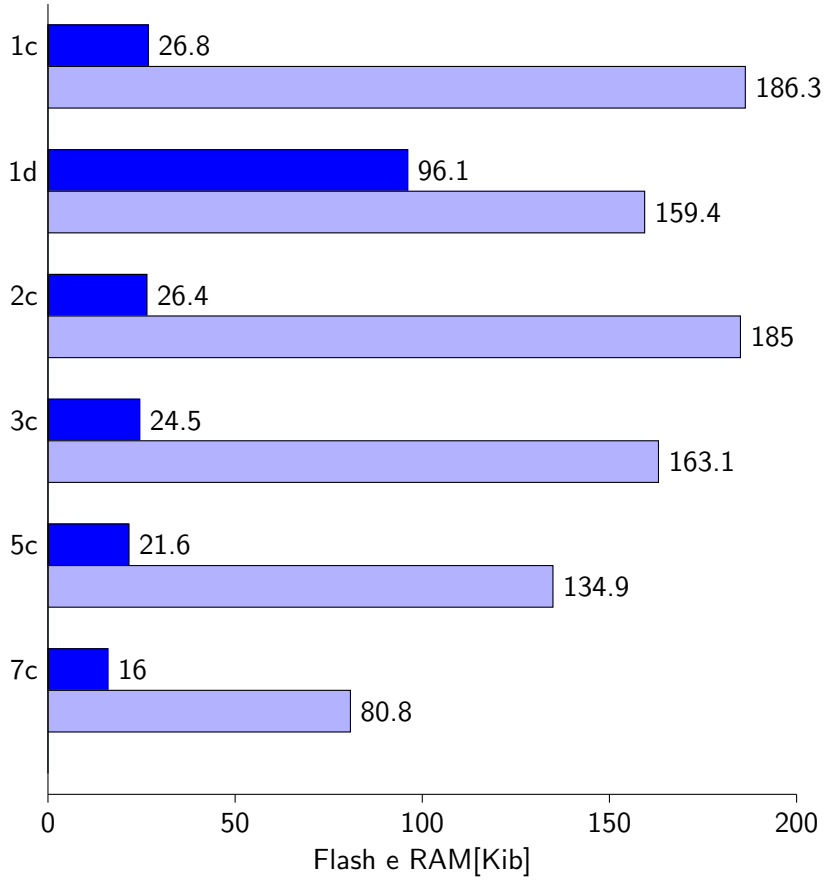


Figura 5.7: *Comparazione flash e RAM dei differenti modelli candidati*

5.2 On-device Testing

Figura 5.8: *Portenta vision shield connessa alla Portenta h7*

In questa sezione verranno illustrate le prove più importanti eseguite per testare il sistema finale, quindi le prove direttamente con il device mobile, cioè la scheda Portenta. Ricordando il capitolo 3, lo scopo è quello di riconoscere l'insieme di parole prefissate in contesti come quello di un museo o di un parco naturale, quindi zone particolarmente rumorose e con suoni molto variabili, in più deve riconoscere la parole di una persona qualsiasi, accenti diversi e modi di parlare. La rete era stata addestrata proprio usando un dataset nel quale si provava a simulare questa varietà dei dati, quindi ora si andrà a vedere quanto il modello è stato ben addestrato.

A questo scopo vengono suddivise le prove in più tipologie, in particolare verranno eseguite prove:

- semplici;
- in strada/all'aperto;
- a distanza dal microfono;
- con televisione in sottofondo;
- con differenti persone;
- di parole in frasi a senso compiuto.

Queste dovrebbero generalizzare bene i casi reali in cui il sistema dovrebbe funzionare(musei e parchi naturali). E' stato utilizzato dunque lo sketch di cui ne è stata mostrata l'implementazione, aggiungendo una parte per il conteggio dei risultati per ogni classe, in particolare prendendo il risultato dall'oggetto di tipo *ei_impulse_result_t*, ed il valore delle classificazioni, viene contato un nuovo risultato per una classe ogni volta che la classificazione supera il punteggio di 0.6. Questo per avere un punteggio non molto alto, utile a non complicare di molto la classificazione, e non troppo basso per ridurre il

numero di falsi positivi. Il caricamento sulla scheda dello sketch e la cattura dell'output della scheda sono stati eseguiti attraverso Arduino Ide.

	Disturbo	Voce	Distanza	Tentativi/parola	Accuratezza
a	nessuno	1	<50cm	100	83.75%
b	televisione	1	<50cm	100	74.08%
c	nessuno	1	<3m	100	83.33%
d	nessuno	1	<6m	100	81.58%
e	parole all'interno di frasi	1	<50cm	50	58.03%
f	nessuno	2	<50cm	100	72%
g	nessuno	3	<50cm	100	72.83%
h	all'aperto/in strada	1	<50cm	100	63.66%
i	musica	1	<50cm	100	72.5%

Tabella 5.9: *Sintesi dei risultati per i test on-device eseguiti*

Osservando questa panoramica generale dei vari testi eseguiti si può vedere come il modello e la sua configurazione hanno dato risultati discreti anche nei test sul device. L'accuratezza è parecchio minore rispetto alle prove che erano state eseguite in ambiente virtuale, ma era questo un risultato da aspettarsi principalmente per due motivi. Il primo è dato dal fatto che i test sono stati eseguiti per creare delle condizioni di disturbo per il sistema che fossero il più possibile verosimili, creando situazioni di diversa natura anche differenti da quelli inseriti nel test set. Il secondo motivo riguarda ovviamente il live testing e quindi il fatto che nonostante lo slicing, descritto nella sezione dello sketch arduino 4.4, è come se eseguissimo l'inferenza sulla finestra di 1 secondo ogni volta che passa tempo pari alla grandezza dello slice(nel caso descritto 250 ms). Si intuisce quindi che comunque i dati sensibili possono sempre trovarsi dislocati tra uno slice e l'altro e quindi influire sulla qualità dell'inferenza. A conferma di queste considerazioni si può vedere che senza nessun disturbo l'accuratezza raggiunge comunque l'83.75%, circa 10% in meno rispetto ai risultati sul test set, ma raggiunge il 58% circa nel caso di parole all'interno di frasi per le quali è evidente che altre parole che si inseriscono nella finestra disturbano il modello.

Guardando i risultati da un altro punto di vista, in tabella 5.10, si può vedere l'accuratezza misurata per ogni parola nei diversi casi delle prove descritte. Questo è servito a capire eventuali problemi di classificazione che dipendono dalla parola stessa.

	Test a	**Test b**	**Test c**	**Test d**	**Test e**	**Test f**	**Test g**	**Test h**	**Test i**
Bello	93%	84%	82%	91%	62%	85%	90%	64%	76%
Bonito	84%	68%	85%	61%	40%	56%	69%	83%	74%
Brutto	88%	63%	82%	82%	58%	72%	56%	69%	55%
Carino	81%	93%	76%	92%	76%	93%	80%	65%	86%
Feisimo	86%	94%	84%	79%	60%	67%	77%	38%	65%
Feo	86%	65%	93%	82%	54%	82%	97%	79%	78%
Hermoso	90%	82%	93%	57%	54%	69%	70%	70%	74%
Orrendo	79%	66%	81%	86%	76%	61%	66%	45%	78%
Orribile	88%	73%	81%	81%	59%	73%	61%	59%	76%
Pesimo	81%	59%	91%	86%	52%	67%	63%	65%	58%
Precioso	71%	69%	77%	88%	58%	66%	73%	79%	81%
Stupendo	78%	73%	81%	94%	48%	73%	72%	48%	69%
Totale	83.75%	74.08%	83.33%	81.58%	58.03%	72%	72.83%	63.66%	72.5%

Tabella 5.10: *Accuratezza del sistema su ogni classe per ogni test eseguito*

Capitolo 6

Conclusioni e Sviluppi futuri

In questo lavoro di tesi è stata sviluppata una soluzione basata su SA e TinyML per la valutazione dell'apprezzamento del pubblico nei confronti di entità di interesse a loro esposte. La soluzione proposta è basata sulla classificazione di parole pronunciate in un determinato raggio d'azione attraverso algoritmi di apprendimento automatico per sistemi di edge intelligence, con il supporto del software di Edge Impulse. Questo tipo di approccio innovativo consente di superare i limiti imposti dagli strumenti tradizionali di valutazione che dipendono da interazioni esplicite, e quindi non sempre spontanee, con il cliente. Il sistema progettato è stato sviluppato attraverso la creazione di un modello di inferenza e il suo caricamento sulla scheda Portenta, sono stati quindi eseguiti numerosi test per verificare la robustezza del sistema. I risultati ottenuti sono apprezzabili: il modello ha raggiunto sul test set il 93.00% di accuratezza, mentre nei live test sul dispositivo ha raggiunto valori pari all'83.75% In questi ultimi però vi è un livello di complessità maggiore, che dipende strettamente dall'algoritmo con cui vengono acquisiti i dati attraverso il microfono. In conclusione, il lavoro proposto conferma che l'utilizzo di tinyML per sviluppare sistemi di SA è una soluzione percorribile e promettente per soddisfare le esigenza di un mercato sempre più esigente e attento alle opinioni dell'utente: in particolare, Edge Impulse si conferma essere un software all'avanguardia in grado di semplificare l'approccio al tinyML e renderlo implementabile anche su sistemi con limitate risorse computazionali.

Sviluppi futuri

Il lavoro di questa tesi rappresenta un punto di partenza e pone le basi per nuove ricerche in questo ambito. Un primo sviluppo riguarda sicuramente il rafforzamento del dataset, in termini di una maggiore quantità di dati e di una migliore qualità e varietà degli stessi. In secondo luogo, in futuro si potrebbe inviare i risultati delle analisi sul microcontrollore ad un server, ad esempio tramite il protocollo LoRa, per eseguire ulteriori operazioni sui dati. Infatti, nel sistema sviluppato è stato utilizzato l'apprendimento automatico solamente nella fase di raccolta dati e non durante l'analisi vera e propria,

mantenendo invece l'uso di un classico approccio basato su dizionario. Uno sviluppo possibile sarebbe dunque quello di utilizzare il ML integrando nell'analisi word-based anche altre caratteristiche, come tono di voce, espressioni facciali o altro, eseguendo le computazioni complesse sul server. Questo sicuramente renderebbe i risultati basati sul dizionario più affidabili e completi. Un altro modo di migliorare i risultati sarebbe quello di perfezionare anche le componenti fuori dall'apprendimento automatico come lo sketch di registrazione eseguito sul dispositivo Arduino, poichè questo ha un'influenza decisiva sull'output del sistema.

Bibliografia

[1] Walaa Medhat, Ahmed Hassan e Hoda Korashy. «Sentiment analysis algorithms and applications: A survey». In: *Ain Shams Engineering Journal* 5.4 (2014), pp. 1093–1113. ISSN: 2090-4479. DOI: https://doi.org/10.1016/j.asej.2014.04.011. URL: https://www.sciencedirect.com/science/article/pii/S2090447914000550.

[2] Liu Zhang Wang. «Deep Learning for Sentiment Analysis : A Survey Article Type: Overview». In: *Wires* ().

[3] ain ali riaz nouren kamran hayat rehman. «Sentiment Analysis Using Deep Learning Techniques: A Review». In: *(IJACSA) International Journal of Advanced Computer Science and Applications* (2017).

[4] Jyoti Islam e Yanqing Zhang. «Visual Sentiment Analysis for Social Images Using Transfer Learning Approach». In: ott. 2016, pp. 124–130. DOI: 10.1109/BDCloud-SocialCom-SustainCom.2016.29.

[5] Eric Chu e Deb Roy. «Audio-Visual Sentiment Analysis for Learning Emotional Arcs in Movies». In: *2017 IEEE International Conference on Data Mining (ICDM)*. 2017, pp. 829–834. DOI: 10.1109/ICDM.2017.100.

[6] V Viswanatha et al. *Implementation of Tiny Machine Learning Models on Arduino 33 – BLE for Gesture and Speech Recognition*. EasyChair Preprint no. 8495. EasyChair, 2022.

[7] Arnab Neelim Mazumder e Tinoosh Mohsenin. «A Fast Network Exploration Strategy to Profile Low Energy Consumption for Keyword Spotting». In: *CoRR* abs/2202.02361 (2022). arXiv: 2202.02361. URL: https://arxiv.org/abs/2202.02361.

[8] Lakshmish Kaushik, Abhijeet Sangwan e John Hansen. «Automatic Audio Sentiment Extraction Using Keyword Spotting». In: set. 2015. DOI: 10.21437/Interspeech.2015-571.

[9] Boden. *Artificial Intelligence*. OUP Oxford, 2018.

[10] Earl B. Hunt. *Artificial Intelligence*. Accademic Press, 1975.

[11] Murphy Naqa. «What is Machine Learning?» In: (2015).

[12] Yap Yan Siang, Mohd. Ridzuan Ahamd e Mastura Shafinaz Zainal Abidin. «Anomaly Detection Based on Tiny Machine Learning: A Review». In: *Open International Journal of Informatics* 9.Special Issue 2 (nov. 2021), pp. 67–78. URL: https://oiji.utm.my/index.php/oiji/article/view/148.

[13] Sridhar Gopinath et al. «Compiling KB-Sized Machine Learning Models to Tiny IoT Devices». In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2019. Phoenix, AZ, USA: Association for Computing Machinery, 2019, pp. 79–95. ISBN: 9781450367127. DOI: 10.1145/3314221.3314597. URL: https://doi.org/10.1145/3314221.3314597.

[14] Diederik P. Kingma e Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

[15] Keiron O'Shea e Ryan Nash. «An introduction to convolutional neural networks». In: *arXiv preprint arXiv:1511.08458* (2015).

[16] URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[17] URL: https://www.ibm.com/cloud/learn/convolutional-neural-networks.

[18] URL: https://www.mathworks.com/help/audio/ref/mfcc.html.

[19] URL: https://docs.edgeimpulse.com/docs/edge-impulse-studio/processing-blocks/audio-mfcc.

[20] URL: https://vitolavecchia.altervista.org/mel-frequency-cepstral-coefficient-mfcc-guidebook/.

[21] URL: https://medium.com/analytics-vidhya/data-augmentation-in-deep-learning-3d7a539f7a28.

[22] URL: https://docs.aws.amazon.com/it_it/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html.

[23] URL: https://towardsdatascience.com/a-look-at-precision-recall-and-f1-score-36b5fd0dd3ec.

[24] Francisco Costa. «TinyML models — what happens behind the scenes». In: (set. 2021). URL: https://medium.com/marionete/tinyml-models-whats-happening-behind-the-scenes-5e61d1555be9.

[25] URL: https://learning.edx.org/.

[26] URL: https://towardsdatascience.com/model-compression-via-pruning-ac9b730a7c7b.

[27] Nitish Srivastava et al. «Dropout: A Simple Way to Prevent Neural Networks from Overfitting». In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[28] Song Han, Huizi Mao e William J. Dally. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. 2015. DOI: 10.48550/ARXIV.1510.00149. URL: https://arxiv.org/abs/1510.00149.

[29] Samuil Stoychev e Hatice Gunes. «The Effect of Model Compression on Fairness in Facial Expression Recognition». In: (gen. 2022).

[30] Martin Abadi et al. «TensorFlow: A system for large-scale machine learning». In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283. URL: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

[31] «TensorFlow Lite». In: (2022). URL: https://www.tensorflow.org/lite/guide.

[32] Vittoriomazzia. «Intro to TensorFlow Lite». In: (2019). URL: https://vittoriomazzia.com/tensorflow-lite/.

[33] «Edge Impulse». In: (). URL: https://www.edge-ai-vision.com/companies/edge-impulse/.

[34] «Adding sight to your sensors». In: (). URL: https://docs.edgeimpulse.com/docs/tutorials/image-classification.

[35] «Portenta H7». In: (). URL: https://store.arduino.cc/products/.

[36] POOJA B BASANT A NAMITA M. «Sentiment analysis using common-sense and context information.» In: (2015).

[37] Esha Tyagi e Arvind Sharma. «Sentiment Analysis of Product Reviews using Support Vector Machine Learning Algorithm». In: *Indian Journal of Science and Technology* 10 (giu. 2017), pp. 1–9. DOI: 10.17485/ijst/2017/v10i35/118965.

[38] *EdgeImpulse developers. Continuous audio sampling.* https://docs.edgeimpulse.com/docs/tutorials/continuous-audio-sampling. (Visitato il 22/05/2022).

[39] *EdgeImpulse developers. Building custom processing blocks.* https://docs.edgeimpulse.com/docs/edge-impulse-studio/processing-blocks/custom-blocks. (Visitato il 23/06/2022).

[40] Hagan Demuth Beale Dejesus. *Neural network design*.

[41] Saad Albawi, Tareq Abed Mohammed e Saad Al-Zawi. «Understanding of a convolutional neural network». In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

[42] Saurabh Karsoliya. «Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture». In: *International Journal of Engineering Trends and Technology* 3.6 (2012), pp. 714–717.

[43] Muhammad Uzair e Noreen Jamil. «Effects of Hidden Layers on the Efficiency of Neural networks». In: *2020 IEEE 23rd International Multitopic Conference (INMIC)*. 2020, pp. 1–6. DOI: 10.1109/INMIC50486.2020.9318195.

[44] Yu Zhang Daniel S Park William Chan. ≪SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition≫. In: *Proc. Interspeech 2019, 2613-2617* (3 Dec 2019).

[45] Sanjana Singh e Wenyao Xu. ≪Robust Detection of Parkinson's Disease Using Harvested Smartphone Voice Data: A Telemedicine Approach≫. In: *Telemedicine and e-Health* 26 (apr. 2019). DOI: 10.1089/tmj.2018.0271.

[46] Abdelmajid Mansour, Gafar Zen Alabdeen Salh e Hozayfa Alabdeen. ≪Voice recognition Using back propagation algorithm in neural networks≫. In: *International Journal of Computer Trends and Technology* 23 (mag. 2015), pp. 132–139. DOI: 10.14445/22312803/IJCTT-V23P128.

[47] *EdgeImpulse developers. Spectrogram.* https://docs.edgeimpulse.com/docs/edge-impulse-studio/processing-blocks/spectrogram. (Visitato il 23/06/2022).

[48] Sergey Ioffe e Christian Szegedy. ≪Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift≫. In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[49] Dmytro Mishkin, Nikolay Sergievskiy e Jiri Matas. ≪Systematic evaluation of convolution neural network advances on the Imagenet≫. In: *Computer Vision and Image Understanding* (2017). ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2017.05.007. URL: http://www.sciencedirect.com/science/article/pii/S1077314217300814.