UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA

Corso di Laurea Magistrale in Ingegneria Elettronica

Classificazione di immagini fish-eye mediante reti neurali implementabili su microcontrollore

Complexity bounded classification of fish-eye distorted objects with micro-controllers

Relatore:

Chiar.mo Prof. Claudio TURCHETTI

Correlatori:

Dott.ssa Laura FALASCHETTI

Ing. Danilo PAU

Tesi di Laurea di:

Alessandro CARRA

A.A. 2020 / 2021

Indice

Capitolo 1

Introduzione

Dal 2019, il tiny machine learning [10] si è imposto ovunque come una tecnologica innovativa implementata ed è stata dominante in molte applicazioni IoT. Un aspetto interessante, affrontato in questo lavoro, è legato al benessere degli animali da laboratorio. La loro salute potrebbe essere preservata acquisendo e classificando alcune immagini per monitorare la presenza di oggetti nelle loro gabbie. Ad esempio, le attività dei roditori, come bere e mangiare, possono essere indicatori del loro stato di salute. In tale contesto, **µCageNet** è stata sviluppata per classificare la presenza o l'assenza della gabbia, **µBottleNet** per classificare la presenza o l'assenza della bottiglia d'acqua e **µFoodNet** per classificare il livello del cibo nella mangiatoia. Tali **reti neurali convoluzionali (CNN)** hanno raggiunto una precisione del 99,8%. Dal momento che non fossero disponibili dataset a supporto di questo studio, ne abbiamo creati tre, ciascuno composto da una porzione dell'inquadaratura della gabbia. In ognuno, le immagini comprendono e sono focalizzate su un oggetto di interesse; in particolare il **Cage dataset** sullo spigolo della gabbia, il **Bottle dataset** sulla bottiglia dell'acqua e il **Food dataset** sul contenitore del cibo. Tutti e tre sono stati necessari per l'addestramento delle CNN. L'intero processo, dall'acquisizione delle immagini all'esecuzione delle inferenze, è stato eseguito sul **microcontrolle (MCU) STM32H7** o sul **STM32L4** (ultra-low-power con consumo pari a 120 µA/MHz). Le reti µCageNet, µBottleNet e µFoodNet sono state progettate per adattarsi alle risorse limitate delle MCUs. Particolare attenzione è stata fornita all'occupazione di memoria del dispositivo per garantire un valore complessivo di **RAM** necessaria pari a 34.92 kB. Per valutare e testare sul campo le prestazioni di questi modelli, è stata sviluppata un'interfaccia

utente grafica (GUI), in grado di mostrare i risultati di delle predizioni su MCU. Inoltre, è stato verificato che le reti neurali convoluzionali sviluppate hanno raggiunto e superato la precisione ottenuta da un'architettura più complessa.

1.1 Obiettivo

Il progetto di cui tratta la seguente tesi è stato svolto in collaborazione con l'Università Politecnica delle Marche, con la STMicroelectronics (sede di Agrate Brianza (MI)), una fra le maggiori società di semiconduttori a livello mondiale e con la Tecniplast, un'azienda italiana leader mondiale nel settore dello stabulario. Entrambe riconosciute per la loro capacità di innovazione tecnologica. L'obiettivo del progetto è stato quello di sviluppare, progettare e validare tre reti neurali convoluzionali (CNN) in grado di classificare immagini, che presentano distorsione dovuta a lenti fisheye, implementandole su microcontrollore. Il primo passo è stato quello di fornire supporto per la creazione di tre dataset necessari per la progettazione e l'addestramento delle CNN. Ciascuno è composto da porzioni di fotografie a colori o in scala di grigio che rappresentano particolari di una gabbbia per topolini cavia. Gli scatti sono caratterizzati da una distorsione dovuta alla lente fish-eye di cui il comune sensore d'immagine è dotato.

1.2 Claims

L'autore desidera affermare che non ci sono ratti o altri animali coinvolti, filmati o danneggiati nella realizzazione di questo progetto, in qualsiasi momento e per qualsiasi attività nel lavoro qui descritto. La fotocamera utilizzata per acquisire le immagini inserite nel dataset, non hanno catturato nessun animale. Il sensore utilizzato non emette alcun tipo di raggi ed è un sensore d'immagine passivo convenzionale e quindi non invasivo.

1.3 Motivazione del progetto

La misura dell'attività dei singoli topolini nelle loro gabbie fornisce informazioni utili sia per il benessere dell'animale che per scopi scientifici. Gli ambiti

sono molteplici e possono variare da semplici ricerche, fino alla scoperta e allo sviluppo di medicinali, come ad esempio:

- la capacità di individuare e valutare gli effetti tossicologici del candidato;
- la caratterizzazione degli agenti che causano disordini al sistema nervoso centrale (CNS);
- il comportamento degli animali geneticamente modificati;
- studi delle aritmie cardiache.

Come descritto da Ulman *et al.* [37], tutti gli animali soddisfano il loro senso di fame e sete, mangiando e bevendo. Per secoli i biologi si sono interessati a scoprire e comprendere il sistema che costringe e controlla cosa e quando mangiare o bere. Un sistema che in modo autonomo monitora in modo continuo le condizioni della gabbia e il comportamento dell'animale è una necessità per la scienza, per il benessere dell'animale e per facilitare le operazioni. In figura 1.1 è rappresentato un esempio di una gabbia che può essere utilizzata come abitazioni per roditori. si può osservare che la gabbia è dotata di un apposito contenitore a griglia per il cibo e una bottiglia per l'acqua.

Figura 1.1: Gabbia per roditori [7]

Una challenge sponsorizzata da NC3R ha portato ad una collaborazione positiva tra la scienza e l'industria per sviluppare un sistema di monitoraggio per una gabbia domestica. Questa tecnologia potrebbe essere utilizzata con più specie animali, tra cui i roditori che costituiscono il 75% di tutti gli animali utilizzati per gli studi. Per risolvere questa sfida, i team hanno proposto soluzioni innovative per monitorare in modo automatizzato il comportamento, le attività e la temperatura delle cavie 24 ore su 24, 7 giorni su 7. La valutazione visiva rimane di fondamentale importanza per monitorare i bisogni di base dei roditori in modo da potergli fornire la giusta quantità di cibo e acqua. Anche questo compito può essere svolto tramite l'elaborazione delle immagini basata su cloud, come nelle applicazioni per la challenge. Tuttavia, se ciò venisse svolto localmente, nelle vicinanze della gabbia, faciliterebbe una migliore scalabilità, risparmio di larghezza di banda e privacy. In questo senso le tiny machine learning (TinyML) hanno l'obiettivo di permettere l'utilizzo sia di algoritmi classici che di Artificial Neural Network (ANN) su microcontrollori con risorse limitate.

1.4 Descrizione dei requisiti

L'intero processo, dalla acquisizione delle immagini, all'esecuzione dell'inferenza, all'elaborazione dei risultati e infine alla creazione dell'interfaccia utente, è eseguito su microcontrollore (MCU) a basso costo, come STM32H7 o STM32L4. Gli algoritmi da sviluppare devono rispettare i limiti e le risorse messe a dispozione da questi MCU. Ad esempio, l'STM32H743 è un ARM Cortex M7 Core con 2 MByte di memoria Flash, 1 MByte di SRAM e clock di sistema a 480 MHz. L'STM32L4 è un ARM Cortex M4 Core con 1 MByte di memoria Flash, 128 KByte di SRAM e clock di sistema a 80 MHz. Il sistema non ha requisiti in termini di tempo dato che i roditori mangiano e bevono lentamente. Le immagini presentano una distorsione fish-eye ma non dovranno essere usate tecniche correttive. Maggiore attenzione è stata fornita all'occupazione di memoria su chip, in particolare alla RAM, in modo da lasciar spazio alle altre attività. Le rete neurali dovranno essere:

- **µCageNet** addestrata con il **Cage Dataset** per la rilevazione della presenza della gabbia;

- **µBottleNet** addestrata con il **Bottle Dataset** per la rilevazione della presenza della bottiglia di acqua;
- **µFoodNet** addestrata con il **Food Dataset** per la classificazione del livello di cibo presente nell'apposito contenitore.

1.5 Stato dell'arte

Le telecamere fish-eye sono ampiamente utilizzate in varie attività di visione artificiale, inclusa la guida autonoma [11], la video sorveglianza [38], [41], il riconoscimento dei volti [42] e la navigazione autonoma per i robot [27]. La caratteristica principale delle lenti fish-eye che ne ha facilitato lo sviluppo è l'ampio campo visivo. Inoltre, sono utilizzate per fotografare oggetti a brevi distanze [11]. In questa pubblicazione, gli autori hanno discusso l'implementazione di una deep neural network (DNN) che classifica gli oggetti in movimento vicini ad un'automobile. L'algoritmo utilizza le quattro viste, grazie a videocamere fish-eye fissate ai lati del veicolo. L'articolo [17] descrive una CNN per la segmentazione semantica proponendo un approccio chiamato Overlapping Pyramid Pooling (OPP).

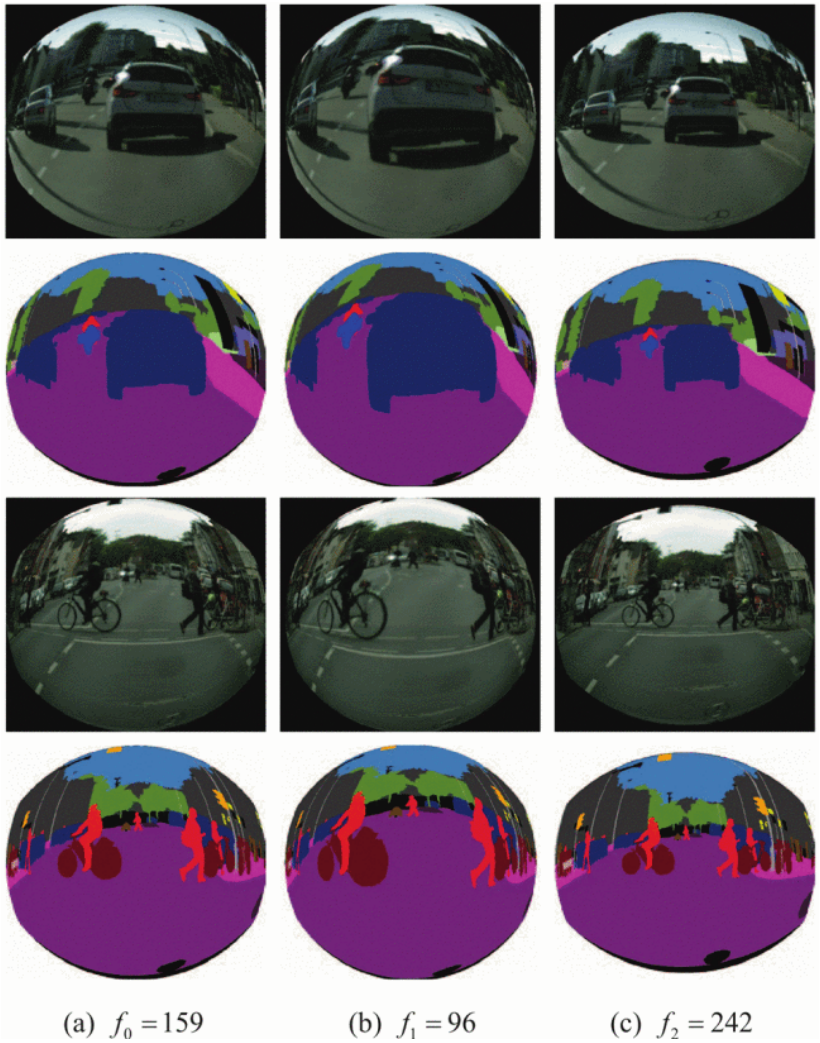


Figura 1.2: Risultati sul fisheye image dataset [17]

Nell'articolo di ricerca [16] gli autori hanno generato un dataset "fish-eye" a partire dalle immagini non distorte del dataset "Cityscapes". Inoltre, in [17] hanno introdotto la zoom augmentation, un'innovativo metodo di data augmentation in grado di aumentare le capacità di generalizzare della NN. In [20], gli autori hanno migliorato l'algoritmo YOLOv3 [25] in modo da aumentare l'accuratezza per il rilevamento di oggetti e ridurre i tempi d'inferenza. Le lenti fish-eye sono utilizzate anche nell'ambito della videosorveglianza e del riconoscimento facciale. Nella pubblicazione [38], è stato verificato che nel caso in cui il background è fisso, le informazioni che si possono estrarre, sono utili per rilevare la presenza umana nella scena. In particolare, le immagini sono processate in modo da ottenere una maschera dello sfondo in scala di grigi. Gli autori, in [43] e [21] hanno proposto due deep neural network (DNN) in grado di rimuovere la distorsione da immagini prima di

utilizzare un ulteriore algoritmo per la classificazione, l'object detection o l'image segmentation. Questi metodi non sempre sono essenziali, come in [42], dove viene descritto un rilevatore di volti, che non richiede l'utilizzo di image rectification. In questo modo è possibile aumentare la velocità mantenendo elevati i valori di accuracy.

Figura 1.3: Esempio immagine fisheye con face detection [42]

Nessuno di questi studi ha esplorato l'opportunità di sviluppare e/o eseguire i loro algoritmi su MCU. YOLO V3 [25] e altre NN addestrate in questi articoli hanno milioni di parametri. Senza un adeguato ridimensionamento e una riduzione di parametri, queste reti difficilmente potranno essere compatibili con le risorse della memoria integrata di un microcontrollore. I modelli di rete neurali sono approssimatori universali di funzioni che possono adattarsi a ogni tipo di dato. Come sostenuto in [36], per dimezzare l'error rate, ci si deve aspettare un aumento di circa 500 volte la complessità computazionale. Ciò risulta in contrasto con il pensiero environment-friendly sulla riduzione del consumo di energia. Uno studio sull'energia e sulle politiche per il deep learning sono discusse in [34]. Uno dei possibili metodi potrebbe essere quello di ridurre l'onere computazionale concentrando il focus sull'implementazione di reti neurali di piccole dimensioni. Questo approccio è stato utilizzato in questo progetto per abbassare i costi di implementazione. Sviluppare NNs accurate e veloci in grado di essere utilizzate su MCU non sempre è un operazione semplice. Ogni singolo caso richiede una progettazione adeguata in modo da mantenere elevata l'accuracy e raggiungere il minimo fabbisogno in termini di costo computazione, consumo di potenza e occupazione di memoria.

1.6 Benessere animale

"Il benessere è uno stato di salute completo, sia fisico che mentale, in cui l'animale è in armonia con il suo ambiente" (Hughes, 1976).

Il benessere di qualsiasi animale è una condizione intrinseca, il soggetto che riesce ad adattarsi all'ambiente si trova in uno stato di benessere, viceversa si troverà in una condizione di stress. Poiché tutti gli animali hanno avuto un percorso evolutivo e ogni specie si è adattata ad un particolare habitat, ogni definizione del benessere deve tener conto dell'ambiente, della fisiologia e del comportamento specifico dell'animale. Nel 1965 Brambell pubblicò un report relativo al benessere animale, che enunciò il principio delle cinque libertà per la tutela del benessere animale:

1. libertà dalla fame, dalla sete e dalla cattiva nutrizione;
2. libertà dai disagi ambientali;
3. libertà dalle malattie e dalle ferite;
4. libertà di poter manifestare le caratteristiche comportamentali specie-specifiche;
5. libertà dalla paura e dallo stress.

Come si può leggere dalle Linee guida dell'IZSVe per la gestione di mammiferi, volatili e specie ittiche utilizzati a fini scientifici [40], il principio delle cinque libertà è oggi completamente superato dalla necessità di riconoscere ogni animale come essere senziente. In quanto tale, è da considerare dotato di capacità psichiche che danno origine a bisogni che vanno ben oltre l'espletamento delle necessità fisiologiche. Si riconosce all'animale il diritto di avere delle aspettative sull'ambiente che lo circonda, di agire su di esso per soddisfare i propri bisogni, di scegliere secondo motivazioni che vanno oltre il bisogno fisiologico, ma sono frutto dell'insieme delle esperienze vissute dal soggetto. Si ammette l'importanza, per il benessere animale, delle emozioni positive che derivano dal raggiungimento di un obiettivo desiderato, dal contatto sociale intra ed interspecifico, dall'apprendimento, da una prevedibilità ambientale che non deve mai essere assoluta, ma riservare elementi di incertezza che stimolano l'animale nello sviluppo di abilità e di metodi di risoluzione di problemi più o meno complessi. Si riconosce, infine, il valore

intrinseco di ogni soggetto (in sé e per sé), che deve indurre ad una riflessione costante sui bisogni dell'individuo e non solo dell'insieme degli animali che costituiscono il gruppo sperimentale.

1.7 L'ambiente di stabulazione e l'alimentazione

L'ambiente in cui gli animali sono alloggiati deve rispondere alle loro esigenze fisiche, fisiologiche ed etologiche in termini di tipologia di stabulazione, illuminazione, rumorosità, ventilazione e qualità dell'aria presente nel microambiente. In particolare [40]:

- l'ambiente deve essere privo di insetti e parassiti e facile da pulire e disinfettare;
- nell'ambiente deve essere presente una zona adibita al riposo, dotata di lettiera o di altro materiale morbido, asciutto e pulito;
- le superfici su cui gli animali si muovono devono essere il più asciutte e pulite possibili;
- le dimensioni dell'alloggiamento, l'altezza e lo spazio a disposizione al suolo per il movimento e per il riposo del singolo animale devono corrispondere almeno a quanto previsto dalla normativa vigente;
- tutte le strutture devono garantire la massima sicurezza, sia per gli animali sia per gli operatori.

I parametri ambientali, come temperatura, umidità e ventilazione devono essere costantemente controllati per garantire il mantenimento delle condizioni ottimali di stabulazione. Viceversa, l'animale metterebbe in atto risposte fisiologiche o comportamentali volte a compensare le condizioni ambientali. Il controllo della temperatura deve mantenere gli animali in uno stato di benessere termico, in una zona di termo neutralità caratterizzata da una temperatura critica inferiore e una superiore. L'umidità relativa deve rimanere nell'intorno del 50% con una variabilità del 15%. Se fosse troppo bassa causerebbe patologie ed un'eccessiva perdita di calore o se troppo elevata favorirebbe lo sviluppo di ammoniaca nei locali e nelle gabbie.

La ventilazione serve a garantire un'elevata qualità dell'aria, bassa polverosità e contribuisce a mantenere l'ambiente a temperatura ed umidità il più possibile costanti. Questi e altri parametri devono essere controllati secondo le normative vigenti. L'alimentazione svolge un ruolo fondamentale per la salute degli animali. Ad esempio nel caso dei mangimi per roditori se necessario andranno autoclavati in piccole quantità che sono consumate nell'arco di una settimana. L'accesso all'alimento deve avvenire facilmente e senza che si crei competizione per la risorsa tra i soggetti stabulati in gruppo. L'alimento non si deve contaminare con feci o urina e si deve conservare in ottime condizioni fino a che non viene consumato. L'acqua potabile, proveniente dagli acquedotti comunali nel caso in cui risponde ai criteri di legge per i roditori, se richiesto viene autoclavata direttamente nelle bottiglie. Quest'ultime sono sostituite e non riempite ex-novo, onde evitare contaminazioni crociate.

Capitolo 2

Convolutional Neural Network

Sin da quando il computer è stato inventato, ogni scienziato si è chiesto se fosse fatto per imparare. Nel 1997 Mitchell, nel suo libro [23], affermava che se fossimo riusciti a capire come programmare una macchina in modo che impari e migliori automaticamente con l'esperienza, l'impatto sarebbe stato incredibile. L'autore si immaginava applicazioni nei campi della medicina, per scoprire cure per nuove malattie, delle abitazioni civili, per ottimizzare i costi dell'energia o per la creazione di assistenti personali che possano evidenziare news dai notiziari. A distanza di anni, alcune di queste possibili applicazioni fanno parte della nostra quotidianità. Svariate operazioni, grazie all'intelligenza artificiale (AI), sono state facilitate o in certe condizioni l'onere è stato annullato. Verranno ora introdotti i principi del Machine Learning (ML), del Deep Learning (DL), dell'Intelligenza Artificiale (AI), delle Reti Neurali (NN) e delle Reti Neurali Convoluzionali (CNN).

2.1 Introduzione al machine learning

Con il termine Machine learning si indica un insieme di metodi con cui si allena l'intelligenza artificiale in modo da poter svolgere delle attività non programmate. Consiste nell'apprendere dall'esperienza pregressa proprio come l'intelligenza umana, migliorando attraverso gli errori commessi. Il Deep Learning è un insieme di tecniche necessarie per la realizzazione del ML e può essere considerato una sottoclasse dello stesso. Il DL è un metodo statistico per estrarre informazioni da un set di dati utilizzando reti neurali. Il termine Deep deriva dal fatto che le reti neurali possono essere composte

da svariati strati nascosti di neuroni, che le rendono appunto profonde e somiglianti, a livello strutturale, al cervello umano.

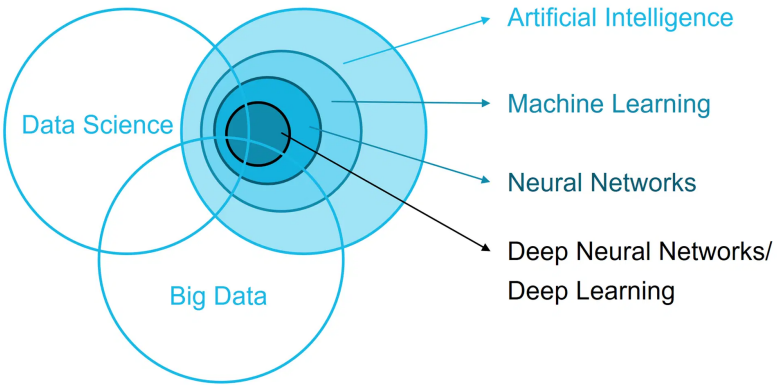


Figura 2.1: Diagramma intelligenza artificiale [8]

Lo scopo principale degli algoritmi di DL è quello di costruire un modello di ottimizzazione che, tramite un processo iterativo, minimizzi o massimizzi una funzione chiamata loss function. L'apprendimento automatico avviene tramite una prima fase di allenamento, nella quale viene acquisita una base di conoscenza. Gli algoritmi di apprendimento automatico sono suddivisibili in due macro-categorie in base al tipo di feedback su cui si basa il sistema di apprendimento artificiale:

- gli **algoritmi di apprendimento supervisionato**, richiedono una base di conoscenza acquisita dalla macchina, attraverso dati annotati similari a quelli che verranno predetti;
- gli **algoritmi di apprendimento non supervisionato**, a differenza del precedente non vengono utilizzati dati etichettati ma l'algoritmo esplorerà la struttura dei dati e definirà il proprio modello.

Si parla di apprendimento automatico poiché alla macchina vengono forniti degli esempi, sotto forma di una coppia di dati composti dal dato originale e dal risultato atteso. Al centro del diagramma di figura 2.1 vi è il sottoinsieme che comprende i Big data, la Data Science e le Deep Neural Network (DNN). L'enorme mole di dati necessaria e il lavoro oneroso per l'etichettatura, svolgono un ruolo fondamentale e necessario per il corretto addestramento di una rete neurale.

2.2 Neuroni e reti neurali

Le reti neurali artificiali sono dei sistemi di elaborazione dell'informazione, il cui funzionamento trae ispirazione dai sistemi nervosi biologici. Una rete neurale artificiale (Artificial Neural Network ANN) possiede molte semplici unità di elaborazione differentemente connesse tra di loro (Figura 2.2).

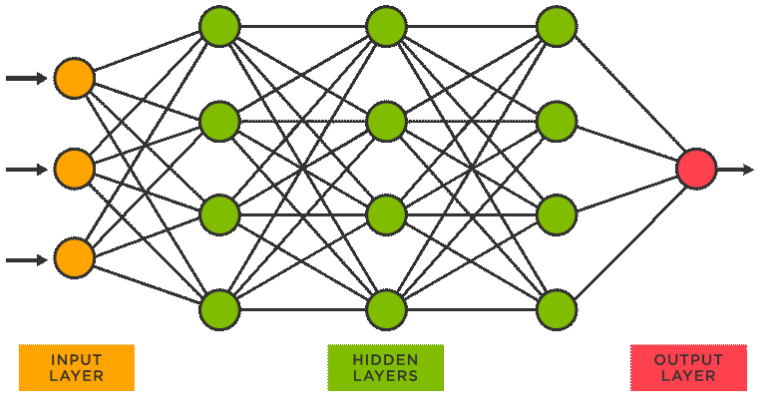


Figura 2.2: Esempio di una rete neurale [9]

Alcune ricevono informazioni dall'ambiente, altre emettono risposte nell'ambiente e altre comunicano solamente con le unità all'interno della rete. Esse sono definite rispettivamente:

- unità di ingresso (input);
- unità di uscita (output);
- unità nascoste (hidden).

Ciascuna unità intende simulare il ruolo di un neurone nelle reti neurali biologiche: per questo motivo esse vengono anche definite impropriamente neuroni oppure nodi. Ciascuna unità svolge un'operazione molto semplice che consiste nell'attivarsi se la quantità totale di segnale che riceve supera la propria soglia di attivazione. Se ciò avviene, essa emette un segnale che viene trasmesso lungo i canali di comunicazione fino alle altre unità a cui essa è connessa. Ciascun punto di connessione agisce come un filtro che trasforma il messaggio ricevuto in un segnale inibitorio o eccitatorio. Aumentandone o diminuendone nel contempo l'intensità, a seconda delle proprie caratteristiche individuali, questi punti di connessione simulano le sinapsi biologiche. Inoltre, poiché il loro ruolo consiste nel "pesare" l'intensità

dei segnali trasmessi, essi vengono definiti anche con il nome di pesi. In pratica l'operazione svolta da una sinapsi è semplicemente il prodotto tra il valore del segnale ricevuto e il valore della sinapsi stessa (che può essere un numero qualsiasi appartenente all'insieme dei numeri reali). Formalmente, il segnale di risposta emesso da un nodo n_i è uguale a:

$$n_i = \Phi(\sum_i w_{i,j} n_i - v_i) \tag{2.1}$$

ovvero a una funzione Φ della somma dei prodotti dei segnali d'ingresso n_j (provenienti da altri nodi o dall'ambiente esterno).

Le reti neurali artificiali presentano alcune caratteristiche che si rivelano interessanti in molti campi di ricerca e domini di applicazione. Benché molte di queste caratteristiche varino da modello a modello, ve ne sono alcune sufficientemente generali:

- robustezza, una rete neurale è resistente al rumore;
- flessibilità, un modello neurale può essere impiegato per un gran numero di finalità diverse e non ha bisogno di conoscere le proprietà specifiche del dominio di applicazione;
- generalizzazione, una rete neurale che è stata addestrata su un numero limitato di esempi è in grado di produrre una risposta adeguata a dei pattern d'ingresso che non ha mai visto in ingresso;
- recupero in base al contenuto, le reti neurali artificiali sono in grado di recuperare le proprie memorie in base al contenuto partendo da dati incompleti o corrotti da rumore.

2.3 La convoluzione

Si può notare come il numero di parametri di una NN composta da strati fully connected aumenta velocemente incrementando le dimensioni. Ad esempio, aggiungendo un ulteriore layer di dimensione $size_l$, i parametri aumenteranno in funzione di $(size_l+1)\cdot size_l$. Ipotizzando un numero di unità pari a 1000, i weight e i bias aumenteranno di 1 milione. Questo problema è evidente utilizzando le immagini, dove ogni pixel rappresenta una feature. Le Convolutional Networks [14], conosciute anche come Convolutional Neural

Network, sono una speciale tipologia di reti neurali per il processing di dati sotto forma di matrici. Ad esempio, si potrebbe utilizzare una serie temporale di dati, che può essere pensata come una serie di dati monodimensionali (1-D) con campioni ad intervalli regolari di tempo oppure delle immagini composte da una griglia 2-D di pixel. Le CNN prendono il nome dall'operazione matematica chiamata Convoluzione, che deve essere utilizzata almeno in uno degli strati di cui è composta. Nella sua forma più generale, la convoluzione è un'operazione in due funzioni con un argomento reale:

$$s(t) = \int x(a)w(t-a)da \tag{2.2}$$

dove $x(t)$ è il segnale temporale preso in esame e $w(a)$ è il filtro. Viene tipicamente denotata con un asterisco $*$.

$$s(t) = (x * w)(t) \tag{2.3}$$

Nella terminologia utilizzata per le CNN, il primo termine nella 2.3, x viene chiamato **input** mentre il secondo w, **kernel**. Nel caso in cui il segnale venga discretizzato nel tempo, l'equazione 2.3 diventa:

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \sum \tag{2.4}$$

Solitamente, nelle applicazioni Machine Learning, l'input e il kernel sono array multidimensionali rispettivamente di dati e di parametri che vengono adattati in fase di addestramento. Come in figura 2.3, nel caso si utilizzassero immagini, cioè matrici 2-D o 3-D (luma only o RGB), il kernel che si vorrà utilizzare potrà essere a sua volta bidimensionale o tridimensionale.

$$s(t) = (I * K)(i,j) = \sum_m \sum_m I(m,n)K(i-m, j-m) \tag{2.5}$$

E dato che la convoluzione gode della proprietà commutativa:

$$s(t) = (K * I)(i,j) = \sum_m \sum_m I(i-m, j-m)K(m,n) \tag{2.6}$$

L'equazione 2.6 è la formula che viene maggiormente utilizzata dato che il kernel risulta di dimensioni inferiori rispetto all'input. La convoluzione si basa su tre principi in grado di migliorare un sistema AI, l'**interazione**

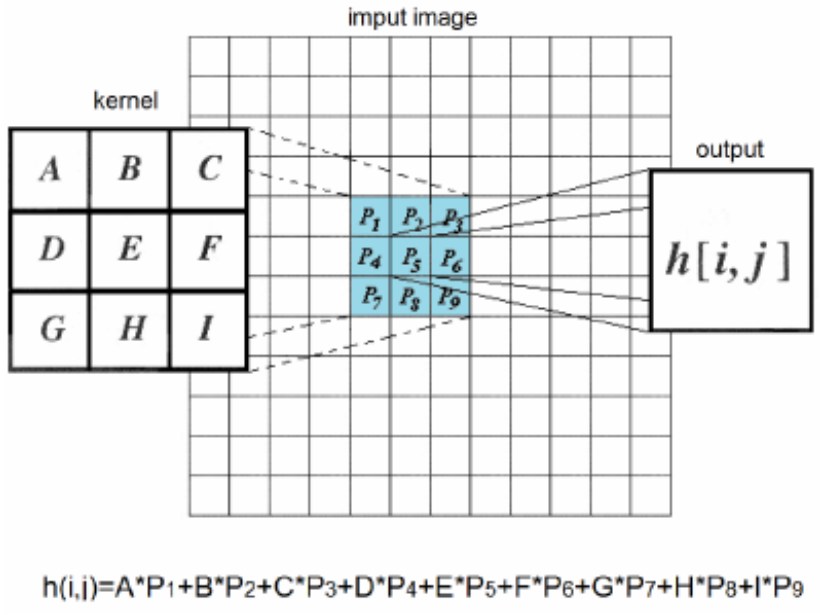
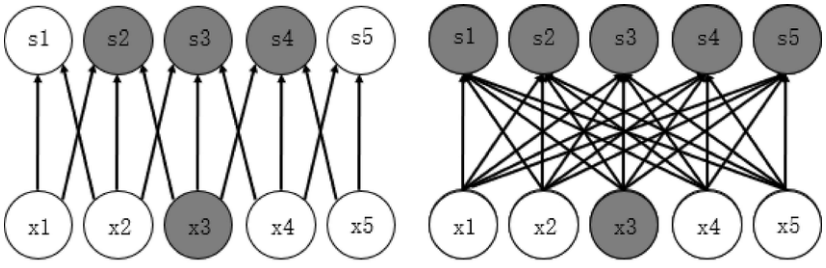


Figura 2.3: Processo convoluzione con kernel 3×3 [22]

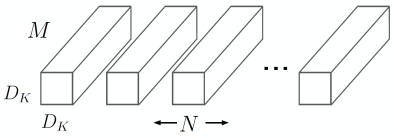
sparsa (sparse interactions), la **condivisione dei parametri** (parameters sharing) e l'**invarianza alle traslazioni**. I layer delle reti neurali tradizionali utilizzano moltiplicazioni tra matrici di parametri con un'ulteriore parametro che descrive l'interazione tra le unità d'input e quelle d'output. Ciò significa che ogni unità interagisce con tutte le altre. Viceversa le CNN, avendo solitamente un kernel più piccolo rispetto alle immagini con milioni di pixel, sono in grado di riconoscere piccoli dettagli, features significative in sole centinaia o migliaia di pixel.



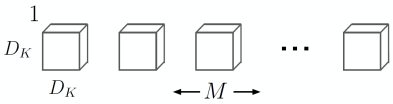
(a) sparse connectivity **(b)** dense connectivity

Figura 2.4: Sparse connectivity e dense connectivity (fully connected) [22]

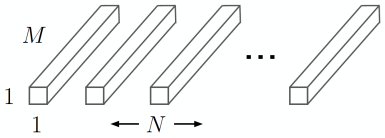
Inoltre, ciò significa che è necessario memorizzare solamente pochi parametri, riducendo la memoria richiesta dal modello, migliorando l'efficenza statistica. Il secondo principio, la **condivisione dei parametri**, è riferito all'utilizzo degli stessi parametri per più di una funzione nel modello. Nelle NN tradizionali, ogni elemento della matrice dei pesi viene utilizzato una



(a) Filtri convoluzione standard



(b) Filtri convoluzione depth-wise



(c) Filtri convoluzione point-wise

Figura 2.5: Differenza tra filtri per la convoluzione standard e per la convoluzione deptwise separable [19]

sola volta per il calcolo dell'output. Viceversa, in uno strato convolutivo il valore del peso nel kernel, viene applicato per tutto l'input. Per **invarianza alle traslazioni** si intende la proprietà per cui se l'input cambia, l'output varierà alla stesso modo.

2.4 La convoluzione separabile

L'articolo di ricerca [19], dal titolo "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications" introduce una classe di reti neurali efficenti basate sul layer *depthwise separable filters*. L'operazione alla base di questo strato è la **convoluzione separabile** e consiste nel fattorizzare una convoluzione standard, in una **depthwise** e in una 1×1 chiamata **pointwise**. Quest'ultima poi utilizza una convoluzione 1×1 per combinare gli output della depthwise. Questa scissione riduce drasticamente il costo computazionale e la dimensione del modello. Ipotizzando che un layer convoluzionale standard riceva in ingresso un tensore con shape (D_P, D_F, M) e il kernel K abbia shape (D_P, D_F, M, N), dove M è il numero di canali in ingresso e N il numero di canali in uscita, il suo costo computazionale risulterà:

$$Computational\_cost = D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \tag{2.7}$$

Mentre per un convoluzione separabile con il medesimo ingresso si dovranno sommare i due contributi. Rispettivamente quello per la depthwise convolution, con kernel di dimensioni (D_K, D_K, M) e quello per la point-

wise convolution con kernel di dimensioni $(1, 1, M, N)$ ottenendo un costo computazionale pari a:

$$Computational\_cost = D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \quad (2.8)$$

La riduzione del costo computazionale può essere calcolata:

$$Ratio = D_K \frac{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F}{\cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2} \quad (2.9)$$

2.5 Activation, pooling, dropout e flatten layers

Un layer di una CNN può essere scomposto in tre fasi:

1. operazione matematica della convoluzione
2. Attivazione lineare o non lineare
3. Pooling layer

Nel paragrafo 2.3 e 2.4 è stata introdotta la prima. L'**attivazione** è una funzione che viene utilizzata per calcolare l'output di un nodo o della rete. Può essere lineare o non lineare. Le più utilizzate sono:

- **Linear o Identity activation**, corrisponde alla funzione di una retta nel piano:
$$f(x) = ax + c \quad (2.10)$$
- **Sigmoid**, è utilizzata con CNN per la classificazione binaria e restituisce valori compresi da 0 e 1:
$$sigmoid(x) = \frac{1}{(1 + exp(-x))} \quad (2.11)$$
- **Softmax**, è utilizzata con CNN per la classificazione con più di due classi e restituisce valori compresi da 0 e 1:
$$softmax(x) = \frac{exp(x)}{\sum(exp(x))} \quad (2.12)$$

- **Tangente iperbolica (Tanh)**, è una funzione iperbolica, non lineare e restituisce valori compresi tra -1 e 1.

$$f(x) = tanh(x) = \frac{sinh(x)}{cosh(x)} \tag{2.13}$$

- **ReLU (Rectified Linear Unit)**, è la funzione di attivazione non lineare maggiormente utilizzata e restituisce il valore massimo tra zero e il valore in ingresso.

$$ReLU(x) = max(0, x) \tag{2.14}$$

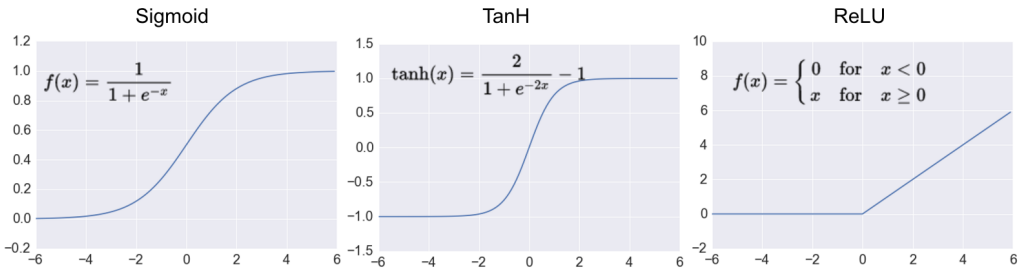


Figura 2.6: Grafici delle funzioni di attivazione [13]

L'ultima fase, il **Pooling**, consiste nel modificare l'output e la sua dimensione in modi diversi. Ad esempio il **Max Pooling** mantiene il valore massimo in output da una sottomatrice e l'**Average Pooling** calcola la media dei valori contenuti in una funzione e la riporta per ogni canale dell'output. Entrambe le operazioni sono implementate in keras e i parametri sono:

- pool_size, la dimensione della finestra;
- strides, numero degli shift da effettuare dopo ogni calcolo;
- padding, dimensione della matrice di output che può essere modificata per ottenere dimensioni uguali a quelle dell'input.

Oltre al pooling, dopo la convoluzione è possibile introdurre altri tipi di operazioni sottoforma di layer. Ad esempio uno strato **Dropout** viene utilizzato per ovviare al problema dell'overfitting, impostando in modo random alcune unità di input a zero con una determinata frequenza in fase di addestramento. Un altro layer utilizzato in questo progetto è quello con

il compito di appiattire i dati in ingresso per ottenere un array 1-D. Viene chiamato **Flatten** layer.

2.6 Loss function ed overfitting

Come descritto nella sezione 2.1, gli algoritmi di apprendimento supervisionato necessitano di una funzione che indichi la qualità delle predizioni effettuate con un modello. Le **Loss function** o **funzioni di perdita** sono scelte dall'utente in fase di progettazione in base al compito della NN e ai dati in input e output. Vi sono diverse Loss che misurano il distacco tra i valori reali e quelli predetti. La selezione farà variare il risultato ottenuto, penalizzando alcuni errori e modificando ciò che il modello apprende. In questa fase nasce il rischio di apprendere valori che vanno ad interpolare le features estratte rischiando di cadere nella tendenza di memorizzare il training set. Questa condizione appena evidenziata è detta **overfitting**.

Capitolo 3

Introduzione al progetto

Come descritto nella sezione 1.4, l'intero processo, dall'acquisizione delle immagini, all'esecuzione dell'inferenza, all'elaborazione dei risultati e infine l'interfaccia utente è eseguito su microcontrollore (MCU). A tale scopo è stato progettato e realizzato un printed circuit board (PCB) dotato di una MCU prodotta dalla STMicroelectronics. In particolare, STM32H7 è un microcontrollore High-performance basato sul core Arm Cortex-M7 32-bit RISC @480MHz. Questo microcontrollore è stato scelto poichè è uno dei più performanti tra quelli prodotti dalla STMicroelectronics. Nei capitoli seguenti verranno discusse le diverse fasi e sviluppi riguardanti il progetto di tesi.

3.1 Step progettuali

In prima battuta ho contribuito alla creazione di tre dataset necessari per lo sviluppo, l'addestramento e la verifica di tre reti neurali convoluzionali. Il lavoro non comprende la scrittura del firmware per l'intero processo. Una volta verificato che i risultati soddisfino i requisiti imposti, è stato generato il codice sorgente C utilizzando il tool X-CUBE-AI, disponibile pubblicamente nel STM32CubeIde. STM32CubeIde e STM32CubeMX sono due applicazioni per PC, Mac o Linux ideali per scrivere codice firmware e per utilizzare le librerie e i tool sviluppati da ST. Oltre a verificare dai report generati che la memory footprint del MCU rispetti i riquisiti, sono state misurate le performance grazie ad un codice Python scritto ad hoc tramite immagini dai dataset. Durante il progetto sono state sviluppate 5 versioni di dataset e di conseguenza 5 versioni per ogni CNN al fine di perfezionare

il sistema. L'ultimo passo è stato quello di sviluppare un'interfaccia grafica utente per PC o Linux, in linguaggio Python. La suddetta Graphical User Interface (GUI) è in grado di favorire il testing e la validation di reti neurali implementate su microcontrollore. Ad esempio nel caso di reti per classificazione di immagini è possibile utilizzare dati dal PC, scatti dalla webcam o immagini salvate da altri dispositivi.

3.2 Software

Il codice è stato scritto in linguaggio Python sfruttando le potenzialità dei notebook nell'ambiente di sviluppo Jupyter-Lab. In particolare sono stati creati 4 file notebook:

- **Dataset_process.ipynb**;
- **NN_cage_unico.ipynb**;
- **NN_bottle_unico.ipynb**;
- **NN_food_unico.ipynb**.

Il primo contiene tutte le istruzioni per il processamento delle immagini e la creazione dei dataset. Verrà illustrato nel dettaglio nella sezione 4.4. Ciascuno dei restanti è composto dalle celle responsabili dell'intero processo di creazione, addestramento e verifica delle reti neurali convolizionali. I particolari sono riportati nella sezione 5.4 e 8.5.

Capitolo 4

Dataset

Durante questo progetto di tesi ho avuto la possibilità di collaborare alla creazione di tre dataset fondamentali per l'addestramento e il testing delle reti sviluppate. Dalle ricerche effettuate non sono disponibili dataset utili ai fini del progetto. Nello stato dell'arte non vi è uno studio con la stessa tipologia di immagini o organizzato secondo le classi richieste da specifiche progettuali. Dunque è stato deciso di creare un dataset di immagini catturate tramite il prototipo realizzato. É stato utilizzato un comune sensore fotografico dotato di un obiettivo fish-eye [24]. Le immagini ottenute presentano distorsione dovuta alla lente fortemente grandangolare. Nonostante ciò è stato deciso di non utilizzare algoritmi di rettificazione per non gravare sul microcontrollore. Il sensore è in grado di catturare immagini con risoluzione VGA 640x480 Pixel ma per mantenere basso l'utilizzo della memoria le loro dimensioni sono state ridotte. Ciascun scatto viene ritagliato per ottenere 3 porzioni rispettivamente con il focus sulla gabbia, sulla bottiglia per l'acqua e sul contenitore del cibo. Questo procedimento è stato svolto sia nella prima fase per creare i dataset e che ciclicamente in locale su MCU per classificare le immagini.

4.1 Cage, Bottle e Food Dataset

Ogni dataset pone l'attenzione sulla gabbia o su un oggetto contenuto in essa. Infatti il **Cage Dataset**, è composto dalle porzioni di immagini corrispondente ad un angolo della gabbia, il **Bottle Dataset**, a quelle sulla bottiglia dell'acqua e il **Food Dataset**, a quelle sul contenitore del cibo. L'insieme di fotografie sono state etichettate con 2, 3 o 4 label, in particolare:

- **Cage Dataset**, dimensioni 64x64 pixel:
    0. Cage-off, gabbia assente;
    1. Cage-on, gabbia presente.
- **Bottle Dataset**, dimensioni 50x50 pixel:
    0. Bottle-off, bottiglia dell'acqua assente;
    1. Bottle-on, bottiglia dell'acqua presente.
- **Food Dataset**, dimensioni 102x256 pixel (102x200 pixel nella versione 5):
    0. Food-empty, livello cibo nel contenitore minimo;
    1. Food-low, livello cibo nel contenitore basso;
    2. Food-high, livello cibo nel contenitore alto;
    3. Food-full, livello cibo nel contenitore massimo (per V1-V4).

La label "*Food-full*" per il dataset Food è stata eliminata nella versione 5 poichè risultata irrilevante ai fini scientifici. Non è stato possibile suddividere il dataset Bottle in livelli come il dataset Food, dato che compare nell'inquadratura solamente la parte terminale della bottiglia e non tutto il corpo.

4.2 Changelog e versioni

Non essendo presenti lavori o sviluppi pregressi nello stato dell'arte, lo sviluppo del dataset è consistito nella creazione di 5 loro versioni, ciascuna necessaria al raggiungimento dell'obiettivo e a soddisfare i requisiti stringenti. In figura 4.1 si possono osservare le specifiche e le caratteristiche per ogni versione dei dataset Cage, Bottle e Food sotto forma di un pseudo Changelog. I dataset a partire dalla versione 2 sono stati resi bilanciati per uniformare l'accuracy sulle classi. Per ridurre al minimo la memoria occupata e ridurre il costo computazionale per l'elaborazione e la classificazione, si è passato da immagini a colori ad altre in scala di grigi, Luma only. Gli scatti, a partire dalla versione 3, sono stati salvati in formato RAW 8 bit poichè è stata riscontrata una grande perdita di precisione. É stato constatato che ciò è dovuto al diverso formato con cui le immagini sono state salvate per i

dataset, in formato JPG, e processate per la predizione su microcontrollore, in formato RAW. Dalla versione 4 è stato aggiunto il dataset Cage e dalla quinta versione le classi del dataset Food sono state ridotte a 3 eliminando la label Food-full.

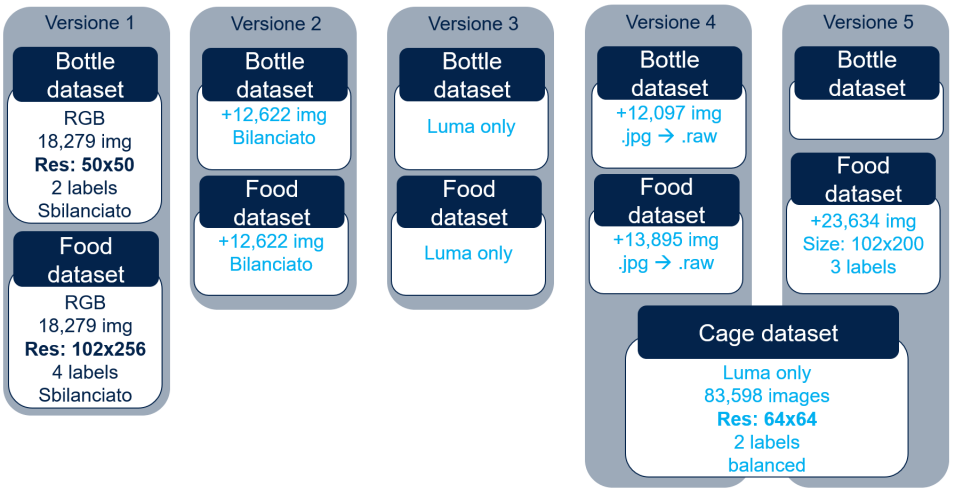


Figura 4.1: Pseudo Changelog versioni dataset

Con il termine bilanciato vengono indicati i dataset con la medesima quantità di immagini per tutte le classi, viceversa per il termine sbilanciato. Nelle tabelle 4.1, 4.2, 4.3 viene riportato il numero di campioni che compongono ogni versione. Si può notare che con l'avanzare delle versioni i dataset sono stati bilanciati e di volta in volta il numero delle immagini è aumentato. Ciò ha garantito un miglioramento dei risultati e una migliore generalizzazione dato che la posizione e l'orientamento delle gabbie causa variazioni di luminosità.

Tabella 4.1: Cage Dataset, dati sul numero di campioni per ogni versione

n° of images	V4 RAW	V5 RAW
Cage OFF	41799	41799
Cage ON	41799	41799

Tabella 4.2: Bottle Dataset, dati sul numero di campioni per ogni versione

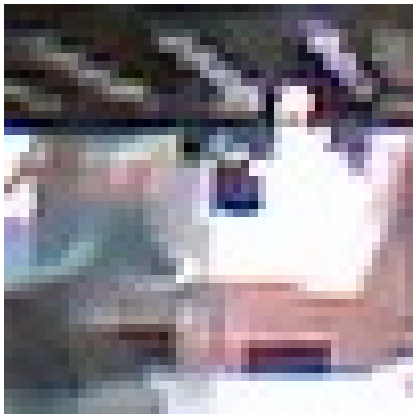
n° of images	V1 Unbalanced RGB	V2 Balanced RGB	V3 Luma only	V4 RAW	V5 RAW
Bottle OFF	9183	15461	15461	21499	21499
Bottle ON	9096	15440	15440	21499	21499

Tabella 4.3: Food Dataset, dati sul numero di campioni per ogni versione

n° of images	V1 Unbalanced RGB	V2 Balanced RGB	V3 Luma only	V4 RAW	V5 RAW
Food EMPTY	1450	7728	7728	11199	23225
Food LOW	7420	7712	7712	11199	22807
Food HIGH	7733	7733	7733	11199	22398
Food FULL	1676	7728	7728	11199	-

4.3 Immagini

Nella corrente sezione vengono riportate delle immagini estratte dai 3 dataset in varie versioni. In figura 4.2 e 4.3 si possono osservare le fotografie a colori rispettivamente per il dataset Bottle e Food entrambi nella versione 2. In figura 4.4, 4.5 e 4.6 si possono osservare le fotografie in scala di grigio rispettivamente per il dataset Cage, Bottle e Food, tutte nella versione 5. Non sono state salvate immagini a colori per il dataset Cage dato che è subentrato a partire dalla versione 4, già in scala di grigi. Si può osservare come non sia facile riconoscere ad occhio nudo la presenza o meno della gabbia e della bottiglia d'acqua dalle immagini riportate.



(a) Bottle-off

(b) Bottle-on

Figura 4.2: Immagini dal dataset Bottle, versione 2

(a) Food-empty

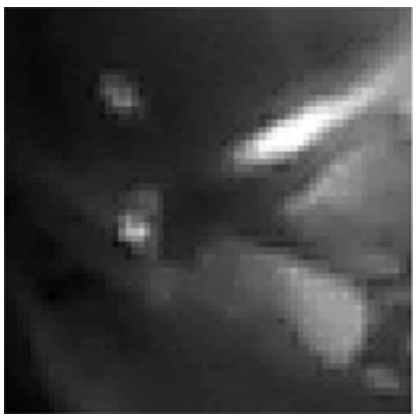
(b) Food-low

(c) Food-high

(d) Food-full

Figura 4.3: Immagini dal dataset Food, versione 2

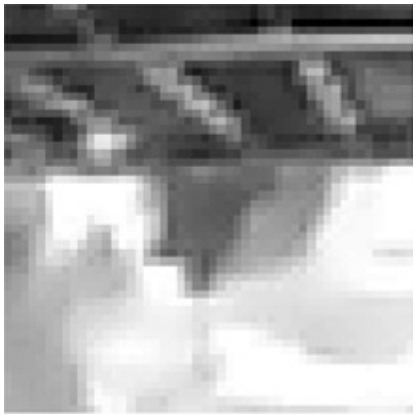
(a) Cage-off RAW in scala di grigi



(b) Cage-on RAW in scala di grigi

Figura 4.4: Immagini dal dataset Cage, versione 5

(a) Bottle-off RAW in scala di grigi



(b) Bottle-on RAW in scala di grigi

Figura 4.5: Immagini dal dataset Bottle, versione 5

(a) Food-empty RAW in scala di grigi

(b) Food-low RAW in scala di grigi

(c) Food-high RAW in scala di grigi

Figura 4.6: Immagini dal dataset Food, versione 5

4.4 Software

Come introdotto nella sezione 3.2, il codice relativo alla creazione dei dataset Cage, Bottle e Food è contenuto in un notebook in linguaggio Python. Di seguito verranno illustrati i passaggi principali e alcune porzioni di codice. L'algoritmo consiste nel trasformare le immagini in formato jpg o raw, in due file numpy [3] contenenti i campioni e le loro classi associate. Ciò dovrà essere svolto per ogni dataset. Il codice è stato scritto in modo da facilitare l'utilizzo e le possibili modifiche che verranno apportate in futuro. In tal senso, come riportato nella pozione di codice 4.1, i principali parametri sono modificabili nella prima cella.

Source Code 4.1: Porzione di codice relativa ai paramentri per la creazione della Bottle dataset

```
#chose wich dataset will be processed
data_bottle = False
data_food = True
data_cage = True
...
dataset_version_bottle = 5

list_input_bottle = ['bottle-off','bottle-on']
folders_bottle = ['bottle-off','bottle-on']

if dataset_version_bottle == 1: # dataset unbalanced
        PATH_MAIN =
        ↪ "./dataset/Dataset1notbalanced/st-cage-dataset/images-cropped/"
        data_file_path_bottle = PATH_MAIN +
        ↪ "mouse_data_label_bottle.txt"
        PATH_BOTTLE = PATH_MAIN + "bottle/"
        PATH_BOTTLE_X_NPY = PATH_MAIN + "x_bottle_v1.npy"
        PATH_BOTTLE_Y_NPY = PATH_MAIN + "y_bottle_v1.npy"
        PATH_BOTTLE_CSV = PATH_MAIN +
        ↪ "y_bottle_v1.csv"
        data_type='jpg'
```

Se in futuro dovesse essere creato un ulteriore versione di dataset, andranno solamente aggiunti i percorsi relativi alle immagini e ai files. Come si può osservare dalla porzione di codice 4.3, il parametro *data_bottle*, è utilizzato come condizione per l'assegnazione dei path e per la creazione o meno del dataset . La funzione *create_classes* genera due dizionari Python contenenti le classi del dataset, che in questo caso coincide con il Bottle. Con la terza riga di codice verrà generato un file di testo contenente tutti i percorsi delle immagini valide, che saranno processate ed inserite nei due array numpy X, Y grazie alla funzione *create_dataset_array*. Per non dover ricreare i dataset, processo che impiega del tempo dato il grande ammontare di campioni, vengono salvati e successivamente utilizzati i soli file "*.npy". L'utente potrà osservare la quantità e le dimensioni delle immagini elaborate.

Source Code 4.2: Porzione di codice relativa alle funzioni per la creazione della Bottle dataset

```
if data_bottle:
        classes_dict_bottle, count_dict_bottle =
        ↪ create_classes(list_input_bottle,folders_bottle)
        data_list_file(data_file_path_bottle,PATH_BOTTLE,data_type)
        X,Y =
        ↪ create_dataset_array(classes_dict_bottle,data_file_path_bottle,fold

        np.save(PATH_BOTTLE_X_NPY,X)
        np.save(PATH_BOTTLE_Y_NPY,Y)

        print("Dataset version
        ↪ bottle:",dataset_version_bottle)
        print(classes_dict_bottle)
        print("X.shape",X.shape)
        print("Y.shape",Y.shape)
```

Il seguente Notebook comprende anche la porzione di codice relativa alla creazione e al salvataggio del file CSV necessario per svolgere i test tramite X-CUBE-AI [33] o la GUI descritta nella sezione 7.1. I campioni sono processati in modo da formare un array monodimensionale, con la classe corrispondente come ultimo valore.

Source Code 4.3: Porzione di codice relativa alla creazione del file CSV per il testing della rete con GUI

```
if data_bottle:
        X = np.load(PATH_BOTTLE_X_NPY)
        Y = np.load(PATH_BOTTLE_Y_NPY)

        # For data splitting use the sckit lib
        x_train, x_test, y_train, y_test = train_test_split(X,
        ↪ Y, test_size=0.2, random_state=42)

        num_sample = 50
```

```
9           sample = np.random.randint(0, len(x_test), num_sample)
10
11          print(sample.max())
12          print(x_test.shape)
13
14          if os.path.exists(PATH_BOTTLE_CSV):
15          os.remove(PATH_BOTTLE_CSV)
16          for i in range(0,len(sample)):
17                  data = np.asarray(x_test[sample[i]])/255
18                  #print("sample[i]",sample[i],"     i",i)
19                  #print("data.shape", data.shape)
20                  data.flatten()
21                  output = y_test[sample[i]]
22                  data=np.append(data,output)
23                  #print("data.shape", data.shape)
24                  data = np.reshape(data, (1,data.size))
25                  #print("data.shape", data.shape)
26                  with open(PATH_BOTTLE_CSV, 'ab') as f:
27                  np.savetxt(f, data, delimiter=",")
```

Capitolo 5

Reti neurali

L'obiettivo del progetto è quello di riuscire a classificare immagini che rappresentano gabbie o oggetti al loro interno, nonostante siano caratterizzate da distorsione fish-eye. In questo senso sono state progettate, sviluppate e testate tre reti neurali, **µCageNet**, **µBottleNet** e **µFoodNet**. Ciascuna è stata addestrata utilizzando i dataset creati e descritti nella sezione 4.1, rispettivamente **Cage**, **Bottle** o **Food** dataset. Le reti neurali di tipo convoluzionale descritte nella sezione 2.3 sono largamente utilizzate nel mondo della computer vision per le ottime performance con modelli con complessità e dimensione limitate. Sono state studiate per raggiungere un'accuracy elevata mantenendo minima l'impronta del modello. Ciò è stato possibile utilizzando tipologie di layer efficenti e sviluppando dataset di immagini di dimensioni ridotte. Le versioni delle CNN sono state create di pari passo con i dataset, sfruttandoli in fase di training e testing. Dall'aggiornamento numero 4, è stata introdotta la CNN **µCageNet**. Come si può osservare dal diagramma di figura 5.1 l'esecuzione delle reti **µBottleNet** e **µFoodNet** è subordinata al rilevamento della gabbia. In questo modo sono state evitate predizioni errate nel caso in cui la gabbia non fosse presente.

5.1 Topologie

I modelli delle tre reti neurali sono stati generati utilizzando le application programming interface (API) di TensorFlow (TF) [31], una piattaforma open source end-to-end per l'apprendimento automatico. Essa dispone di un ecosistema completo e flessibile di strumenti, librerie e risorse della community che consente ai ricercatori di promuovere lo stato dell'arte del

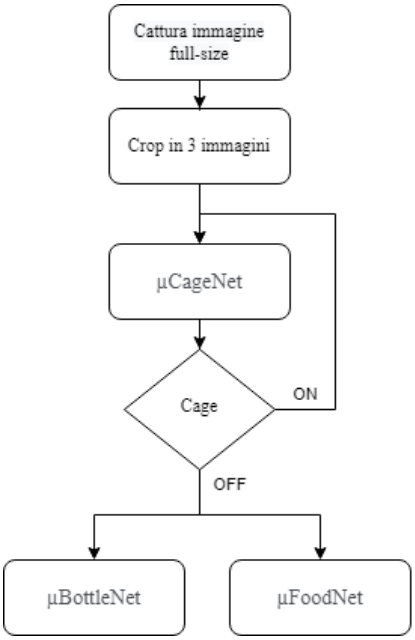


Figura 5.1: Diagramma di flusso algoritmo reti neurali

machine learning (ML) e agli sviluppatori di creare e distribuire facilmente applicazioni basate su ML. Si possono creare e addestrare modelli ML utilizzando le deep learning API, di alto livello messe disposizione da Keras [28]. Un Modello Keras è un insieme di layer con funzionalità per il training e l'inferenza. Ciascuno può essere composto da layer concatenati o collegati l'un l'altro fino a formare un grafo, anche molto complesso. In questo caso è stato generato un modello sequenziale, cioè ogni nodo ha un solo ingresso e una sola uscita. I dati in input e output sono di tipo Floating point 32-bit (FP32) e sono normalizzati tra 0 e 1 prima di essere utilizzati dalla rete neurale. Come si può osservare dalla figura 5.2, le tre reti sono caratterizzate dalla stessa topologia. Il motivo di questa somiglianza è dato dai possibili sviluppi futuri e dalla volontà di effettuare le predizioni di tutte e 3 le tipologie d'immagine utilizzando un solo modello di CNN. Si differenziano solamente per lo shape in ingresso ed uscita e per l'ultima attivazione.

- µCageNet: input shape (64,64,1), output shape (2), Sigmoid Activation;
- µBottleNet: input shape (50,50,1), output shape (2), Sigmoid Activation;
- µFoodNet: input shape (102,200,1), output shape (3), Softmax Activation.

Lo shape in ingresso corrisponde a quello delle immagini di ciascun dataset. La dimensione del vettore in uscita dalla rete, generato dall'ultimo layer dense e dall'attivazione, è uguale al numero delle classi del dataset con cui è stata addestrata la CNN. L'attivazione Sigmoid è stata utilizzata per µCageNet e µBottleNet, reti per la classificazione binaria, mentre la Softmax, per µFoodNet addestata con il dataset Food, di 3 classi. Per le motivazioni argomentate nelle sezioni 2.3 e 2.4 l'operazione che caratterizza le reti in questione è la convoluzione depthwise, utilizzata nel layer Keras *SeparableConv2D*. É composto da una prima DeptWise spatial convolution seguita da una convoluzione PointWise. Nelle topologie sono stati inseriti layer Keras di tipo *MaxPooling2D*, *Dropout*, *Flatten* e *Dense*. Il loro funzionamento e le operazioni sono state descritte nel capitolo 2.5.

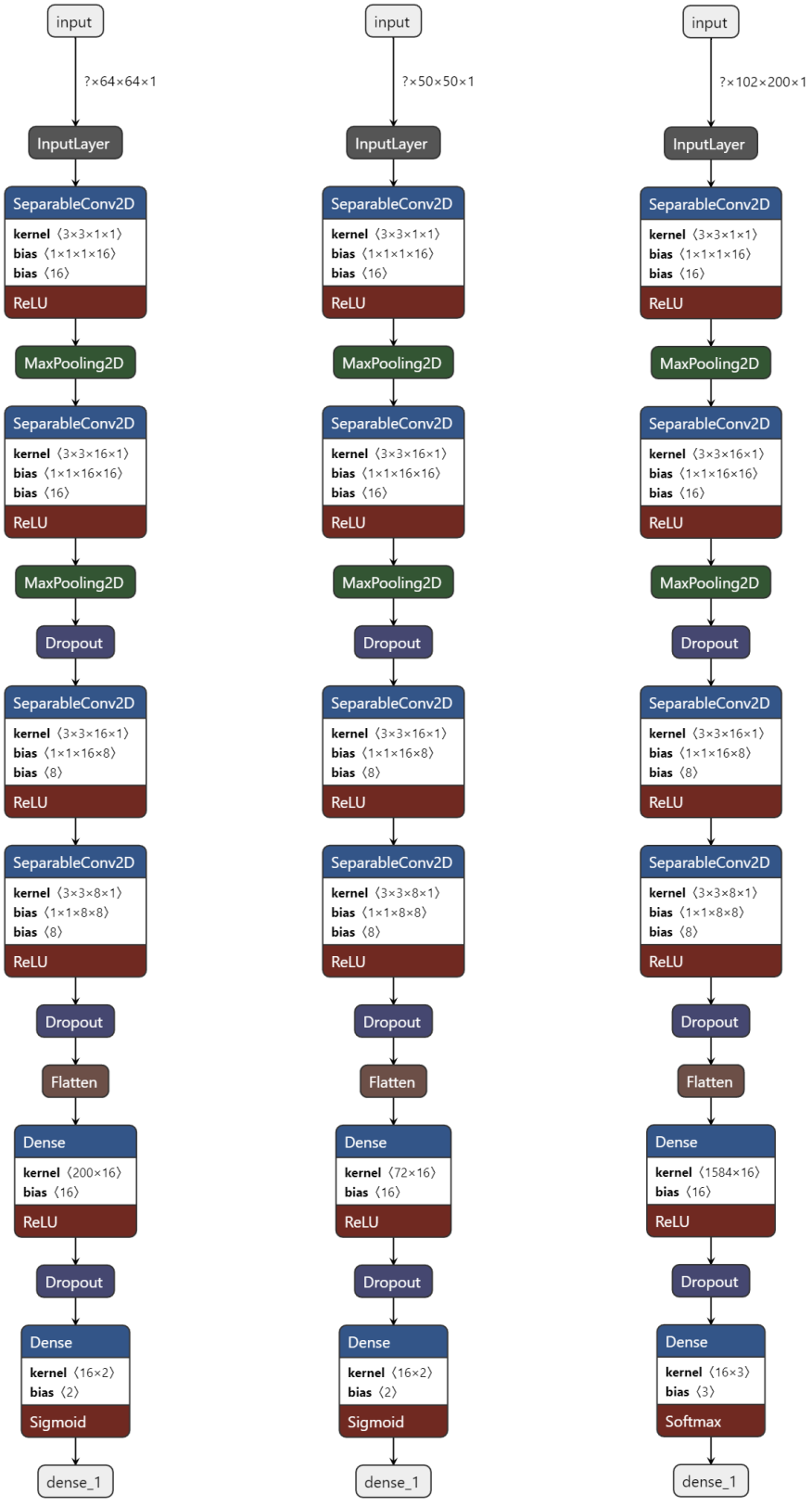
Maggiori dettagli sul numero dei parametri e l'output shape dei singoli layer sono riportati in tabella 5.1. Si può osservare come la µFoodNet ha un numero di parametri totali maggiore alle altre 2 CNN. Ciò è dovuto al primo layer Dense e alla dimensione delle immagini in ingresso.

5.2 Quantizzazione con procedura TFLite

In questo progetto è stata fornita maggiore attenzione all'occupazione di memoria su chip, in particolare alla RAM. Il modello CNN keras utilizza dati di tipo FP32. Per ridurre le dimensioni del modello e migliorare la latenza della CPU, è possibile utilizare la procedura post-training quantization di TensorFlow Lite [32]. Con i nostri modelli ciò non ha inficiato sull'accuratezza delle CNN. Ci sono diverse opzioni di quantizzazione e possono essere scelte dall'utente per raggiungere un giusto compromesso tra efficienza e degrado delle predizioni. Per ridurre il più possibile la memoria necessaria è stata scelta la massima compressione quantizzando con numeri interi 8bit (INT8). A tal proposito, dato che le immagini scattate ed elaborate dal MCU sono di tipo unsigned integer 8-bit (UINT8), si è preferito impostare come tipo di dato in ingresso quest'ultimo. Per questo tipo di quantizzazione è necessario stimare l'intervallo, cioè il valore massimo e minimo di tutti i tensori FP32. A differenza dei tensori costanti come pesi e bias, i tensori variabili come l'input del modello, le attivazioni (output degli strati intermedi) e l'output del modello possono essere calibrati solamente eseguendo alcuni cicli di inferenza con un dataset rappresentativo.

Tabella 5.1: Network topology

	µCageNet		µBottleNet		µFoodNet	
Layer	Out Shape	N° Param.	Out Shape	N° Param.	Out Shape	N° Param.
Separable convolution 2D	32, 32, 16	41	25, 25, 16	41	51, 100, 16	41
Max pooling 2D	16, 16, 16	0	12, 12, 16	0	25, 50, 16	0
Separable convolution 2D	14, 14, 16	416	10, 10, 16	416	23, 48, 16	416
Max pooling 2D	7, 7, 16	0	5, 5, 16	0	11, 24, 16	0
Dropout	7, 7, 16	0	5, 5, 16	0	11, 24, 16	0
Separable convolution 2D	5, 5, 8	280	3, 3, 8	280	9, 22, 8	280
Separable convolution 2D	5, 5, 8	144	3, 3, 8	144	9, 22, 8	144
Dropout	5, 5, 8	0	3, 3, 8	0	9, 22, 8	0
Flatten	200	0	72	0	1584	0
Dense	16	3,216	16	1,168	16	25,360
Dropout	16	0	16	0	16	0
Dense	2	34	2	34	3	51
Total:		4,131		2,083		26,292



(a) µCageNet (b) µBottleNet (c) µFoodNet

Figura 5.2: Analisi delle reti neurali con Netron [29]

Per la conversione del modello è stato scelto il tipo di dato UINT8 per input e output e INT8 per i tensori restanti. I valori in virgola mobile sono approssimati utilizzando la seguente formula:

$$valore\_FP32 = (valore\_INT8 - zero\_point)Xscale \tag{5.1}$$

I valori quantizzati saranno compresi nell'intervallo -127, 128.

5.3 Ulteriori modelli esaminati

Prima di ottenere i modelli delle reti neurali illustrati nella sezione 5.1 ne sono stati generati svariati. Verranno ora illustrati i risultati e i valori delle analisi ottenuti modificando la dimensione dei kernel dei layer Separable convolution 2D. In tabella 5.2 si possono osservare le dimensioni dei tensori di output per ogni layer e il numero dei paramentri, utilizzando kernel 3X3, 5X5 e 7X7. I parametri dei layer convoluzionali aumentano con l'aumentare delle dimensioni, ma il numero totale diminuisce. Ciò è causato dal rimpicciolimento del tensore in ingresso al primo layer Dense, quello con il maggior numero di parametri per tutte e 3 le reti. I valori di accuracy ottenuti raggiungono il 99% per tutte le reti con iperparametri differenti. In accordo con i valori di tabella 5.2 e 5.3, si può affermare che a parità di accuratezza, il miglior rapporto precisione/complessità computazionale e memoria necessaria lo ottiene il modello CNN con kernel di dimensione 3X3. Infatti questa architettura è quella utilizzata per la µCageNet, µBottleNet e µFoodNet. Sono state omesse le analisi per la µCageNet e µBottleNet per sinteticità, dato che le variazioni ottenute sono proporzionali alla dimensione del tensore in ingresso.

Tabella 5.2: Differenze tra i diversi modelli

µFoodNet						
Conv. Kernel size:	3X3		5X5		7X7	
Layer	Out Shape	N° Param.	Out Shape	N° Param.	Out Shape	N° Param.
Separable convolution 2D	51, 100, 16	41	51, 100, 16	57	51, 100, 16	81
Max pooling 2D	25, 50, 16	0	25, 50, 16	0	25, 50, 16	0
Separable convolution 2D	23, 48, 16	416	21, 46, 16	672	19, 44, 16	1,056
Max pooling 2D	11, 24, 16	0	10, 23, 16	0	9, 22, 16	0
Dropout	11, 24, 16	0	10, 23, 16	0	9, 22, 16	0
Separable convolution 2D	9, 22, 8	280	6, 19, 8	536	3, 16, 8	620
Separable convolution 2D	9, 22, 8	144	6, 19, 8	272	3, 16, 8	464
Dropout	9, 22, 8	0	6, 19, 8	0	3, 16, 8	0
Flatten	1584	0	912	0	384	0
Dense	16	25,360	16	14,608	16	6,160
Dropout	16	0	16	0	16	0
Dense	3	51	2	51	3	51
Total:		26,292		16,196		8,732

Per completezza, in tabella 5.3, sono riportati i risultati delle analisi effettuate grazie a X-CUBE-AI (nella sezione 6.1 verranno fornite spiegazioni al riguardo).

Tabella 5.3: Differenze tra i diversi modelli

µFoodNet			
Conv. Kernel size:	3X3	5X5	7X7
RAM [kB]	44.89	44.89	45.12
ROM [kB]	26.04	16.18	8.89
MACC	813,137	1,083,409	1,406,433

5.4 Software

Le reti neurali convoluzionali µCageNet, µBottleNet e µFoodNet sono state create utilizzando tre differenti notebook python. La porzione di codice 5.1 introduce le librerie utilizzate, tra cui numpy, SciKit Learn, TensorFlow, Keras, VisualKeras e Counter.

Source Code 5.1: Porzione di codice relativa agli import delle librerie

```
import os
# Libary for manipulation of multi-dimensional data
import numpy as np

# scikit-learn Library for machine learning and associated facilities
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, plot_confusion_matrix, ConfusionMatrixDisplay, roc_curve, roc_auc_score, f1_score
from sklearn.utils.class_weight import compute_class_weight
from sklearn import preprocessing
from sklearn.model_selection import KFold

import matplotlib.pyplot as plt

# import TensorFlow 2.5 Deep learning libary
```

```
15 import tensorflow as tf
16 from tensorflow import keras
17 from tensorflow.keras.callbacks import ModelCheckpoint
18 from tensorflow.keras.callbacks import EarlyStopping
19
20 #import tool to visualize architectures of neural network
21 import visualkeras
22
23 #import tool to count the number of the elements of the
   ↪  classes
24 from collections import Counter
25
26 print(tf.__version__)
```

Con la porzione di codice 5.2 sono stati processati i campioni del dataset, suddividendoli in un 80% per train set e il restante 20% nel test set. Inoltre i valori sono stati normalizzati a 0 e 1 per migliorare le qualità della rete in fase d'addestramento.

Source Code 5.2: Porzione di codice relativa alla suddivisione del dataset in train set e test set e successiva normalizzazione

```
1  # For data splitting use the sckit lib
2  x_train, x_test, y_train, y_test = train_test_split(X, Y,
   ↪  test_size=0.2, random_state=42)
3
4  #Image counter for each class
5  print("Y",Counter(Y))
6  print("Y train", Counter(y_train))
7  print("Y test", Counter(y_test))
8
9  # Convert class vectors to binary class matrices.
10 y_train = keras.utils.to_categorical(y_train, num_classes)
11 y_test = keras.utils.to_categorical(y_test, num_classes)
12
13 x_train = x_train.astype('float32')
14 x_test = x_test.astype('float32')
```

```
print(x_train.max())

x_train /= 255
x_test /= 255

print(x_train.max())
print(x_test.max())
```

Nella porzione di codice 5.3 è riportata la cella relativa alla quantizzazione della rete con la procedura post-training quantization di TensorFlow Lite. In ordine viene creato un representative dataset con i campioni del train set, vengono impostati i parametri relativi alla quantizzazione come l'optimizer e il tipo di dato in input e output alla rete. Infine il modello Keras FP32 viene quantizzato grazie alla funzione "converter.convert()".

Source Code 5.3: Porzione di codice relativa alla quantizzazione con la procedura post training di TensorFlowLite

```
model = keras.models.load_model(model_path_5x5)
# Convert using integer-only quantization
# Now you have an integer quantized model that uses integer data for
# the model's input and output tensors, so it's compatible with integer-only hardware

def representative_data_gen():
for input_value in tf.data.Dataset.from_tensor_slices(x_train).batch(1).take(100):
yield [input_value]

converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT] #converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
converter.representative_dataset = representative_data_gen
# Ensure that if any ops can't be quantized, the converter throws an error
```

```
converter.target_spec.supported_ops =
    [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
# Set the input and output tensors to uint8 (APIs added in
    r2.3)
converter.inference_input_type = tf.uint8
converter.inference_output_type = tf.uint8
tflite_model_quantInt = converter.convert()

interpreter =
    tf.lite.Interpreter(model_content=tflite_model_quantInt)
input_type = interpreter.get_input_details()[0]['dtype']
print('input: ', input_type)
output_type = interpreter.get_output_details()[0]['dtype']
print('output: ', output_type)
```

Capitolo 6

Implementazione su microcontrollore

Nei capitoli precedenti è stato illustrato come il modello della rete FP32 è stato sviluppato in linguaggio Python con le API di Tensorflow e Keras e come è stato quantizzato mediante la procedura post-training quantization di TensorFlow Lite. In questo capitolo verrà esposto come sono state implementate le reti su microcontrollore e come è stato generato il codice C.

6.1 X-CUBE-AI

I modelli delle reti neurali generati non possono essere inseriti nel MCU senza prima scrivere codice ad-hoc. La STMicroelectronics mette a disposizione pubblicamente il tool di sviluppo X-CUBE-AI [33], un pacchetto di espansione che fa parte dell'ecosistema STM32Cube.AI. Estende le capacità di STM32CubeMX con la conversione automatica di algoritmo di intelligenza artificiale pre-addestrati. La procedura che può essere svolta interamente con il programma per PC STM32CubeIde, consiste nel generare codice C per un determinato microcontrollore selezionato dall'utente. Accetta modelli Keras, TFLite e ONNX, un formato open per rappresentare algoritmi di Machine Learning. In figura 6.1 è possibile osservare un progetto per microcontrollore STM32H743. Dalla parte centrale è possibile gestire i diversi modelli di rete, potendoli aggiungere e osservarne i risultati della analisi. Il tool permette di selezionare il modello salvato su file che dapprima va analizzato per stabilire la compatibilità con il microcontrollore selezionato. Successivamente sarà possibile procedere con la validazione su desktop e sul microcontrollore pro-

grammato con dati random o da file comma-separated values (CSV). Inoltre, come si può osservare da figura 6.2 è possibile visualizzare la struttura del modello con la funzione *Show graph*.

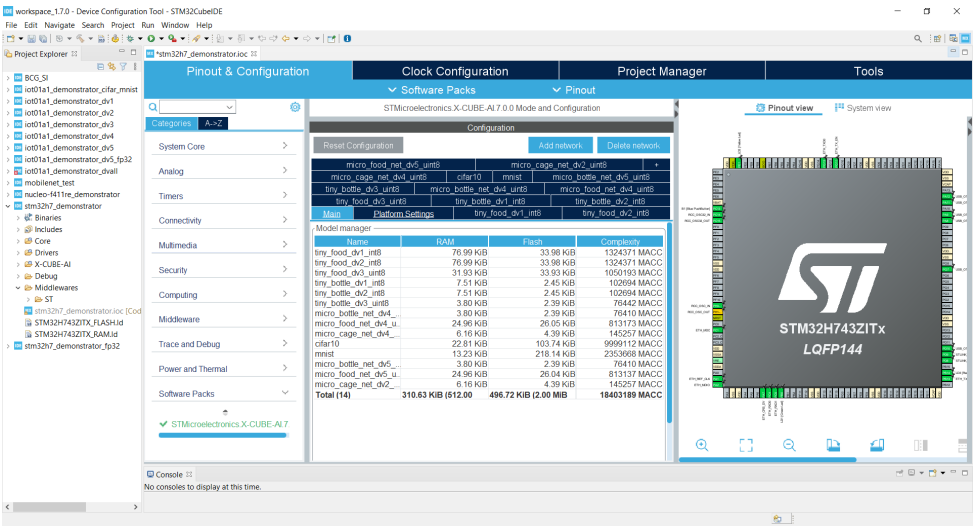


Figura 6.1: Finestra STM32CubeIDE con file di configurazione, X-CUBE-AI e reti d'esempio

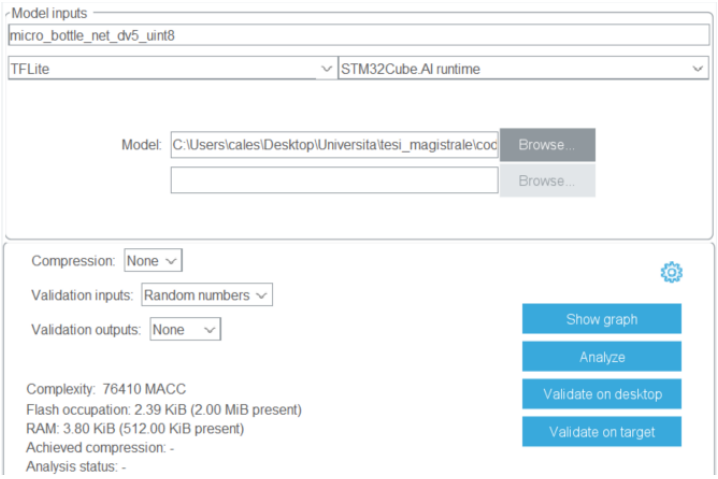
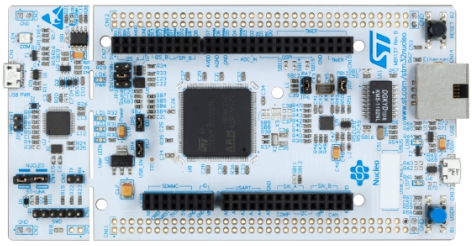


Figura 6.2: Possibili operazioni e parametri per modelli di rete

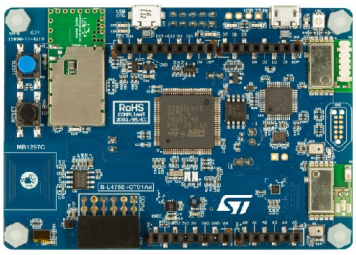
6.2 Microcontrollori e board

Il microcontrollore presente nel prototipo realizzato è il STM32H7, un microcontrollore High-performance basato sul core Arm Cortex-M7 32-bit RISC @480MHz. Il Cortex-M7 presenta una floating point unit (FPU) che supporta ARM double precision, un set completo di istruzioni DSP e una memoria protetta per applicazioni security. La MCU è dotata di memorie ad

alta velocità con una capacità massima di 2 MBytes suddivisa in due banchi e 1 MByte di RAM. Il tutto è coadiuvato da numerosi input/output (I/Os) e da altre periferiche come bus, connessioni secondo diversi standard, tre analog to digital converter (ADC), due digital to analog converter (DAC), un low power RTC, diverse tipologie di timer e un pulse width modulation (PWM) per il controllo dei motori. Per il corrente progetto di tesi non è stato utilizzato il prototipo ma bensì la board Nucleo-H743ZI2 basata sul microcontrollore STM32H743ZI2. Per confermare i risultati ottenuti in termini di occupazione di memoria RAM, Flash e di MACC è stato utilizzata la board Discovery Kit IoT Node dotata del microcontrollore ultra-low power STM32L475 @80MHz, basato sull'architettura Cortex-M4 a 32-bit. La baord si presta ad applicazioni IoT dato che è dotata di connettività WiFi , NFC, Bluetooth Low Energy e banda Sub-GHz (915MHz). Inoltre sono presenti un giroscopio, un accellerometro, un magnetometro e altri sensori tra cui quelli di pressione, prossimità, temperatura e umidità. É possibile programmare le board, collegate al pc con un cavo micro usb, utilizzando il programma *STM32 ST-LINK Utility* e il codice generato grazie a STM32CubeIDE.



(a) STM32H743ZI2 [1]



(b) Discovery board IoT node [2]

Figura 6.3: Board utilizzate

6.3 Analisi

Come illustrato nella sezione 6.1 il tool permette l'analisi dei modelli di rete che si desidera implementare su microcontrollore. L'analisi e la validazione su MCU genera un report, dal quale si possono osservare diversi valori calcolati e misurati tra cui RAM, Flash, MACC e tempo di inferenza. Questi parametri sono stati il focus di questa tesi, in quanto gran parte del lavoro è

stato quello di ridurli il più possibile senza degradare le performance degli algoritmi. In tal senso, sono riportati in tabella 6.1 e 6.2, i valori di MACC, ROM, RAM e i tempi medi per un'inferenza. Si può osservare come questi parametri sono diminuiti in seguito alla post-training quantization procedure e per ogni versione di dataset creata ed utilizzata per l'addestramento. I valori di RAM e flash dipendono rispettivamente dalle dimensioni dei tensori d'attivazione e dai parametri della rete come pesi e bias.

Tabella 6.1: Valori MACC, ROM, RAM estratti dall'analisi della CNN µFoodNet con i Dataset V1-V2

µFoodNet on STM32H743					
	Dataset V1 Unbalanced			Dataset V2 Balanced	
In/Out:	Float32	int8	$\frac{Keras}{TFLite}$	int8	$\frac{Keras}{TFLite}$
MACC:	114.467	102.694	1,11	102.694	1,11
ROM: [kB]	8,84	2,45	3,61	2,45	3,61
RAM: [kB]	30,312	7,51	4,04	7,51	4,04
Inference: [ms]	3,533	1,962	1,80	1,962	1,80

Tabella 6.2: Valori MACC, ROM, RAM estratti dall'analisi della CNN µFoodNet con i Dataset V3-V5

µFoodNet on STM32H743						
	Dataset V3 Luma Only		Dataset V4 RAW Luma Only		Dataset V5 RAW Luma Only	
In/Out:	uint8	$\frac{Keras}{TFLite}$	uint8	$\frac{Keras}{TFLite}$	uint8	$\frac{Keras}{TFLite}$
MACC:	71.442	1,60	76.410	1,50	76.410	1,50
ROM: [kB]	2,39	3,70	2,39	3,70	2,39	3,70
RAM: [kB]	3,80	7,98	3,80	7,98	3,80	7,98
Inference: [ms]	1,72	2,06	1,69	2,09	1,72	2,05

Tabella 6.3: Somma valori MACC, ROM, RAM estratti dall'analisi delle CNN µCageNet, µBottleNet and µFoodNet

	Totale CNN:
In/Out:	uint8
MACC:	1034840
ROM: [kB]	32,83
RAM: [kB]	34,92
Inference: [ms]	19,021

In tabella 6.4 sono riportati i tempi di inferenza medi in millisecondi utilizzando il microcontrolore STM32H7 e STM32L4. Si può notare come ciascuna versione ha portato ad una diminuzione delle latenze e come il MCU STM32L4 impiega circa 10 volte il tempo del MCU STM32H7. Nel caso in cui non si necessiti di esecuzioni Real-Time ma di una maggiore efficenza energetica è stato dimostrato che il microcontrollore Ultra Low-Power è in grado di processare le reti neurali µCageNet, µBottleNet e µFoodNet.

Tabella 6.4: Tempo di inferenza in ms per le CNN µCageNet, µBottleNet e µFoodNet.
Legenda MCU:
-L4 STM32L4 @80MHz
-H7 STM32H7 @480MHz

Dataset	µBottleNet		µFoodNet		µCageNet		Totale	
MCU→	H7	L4	H7	L4	H7	L4	H7	L4
V1 Un-balanced	1,969	20,38	20,94	206,54			22,91	226,93
V2 Balanced	1,97	20,38	20,93	206,54			22,909	226,93
V3 Luma Only	1,733	17,23	18,47	173,41			20,21	190,65
V4 RAW Luma Only	1,731	17,52	14,433	139,39	2,96	29,03	19,12	185,96
V5 RAW Luma Only	1,723	17,24	14,353	135,94	2,94	28,46	19,02	181,63

Capitolo 7

Dimostratore live

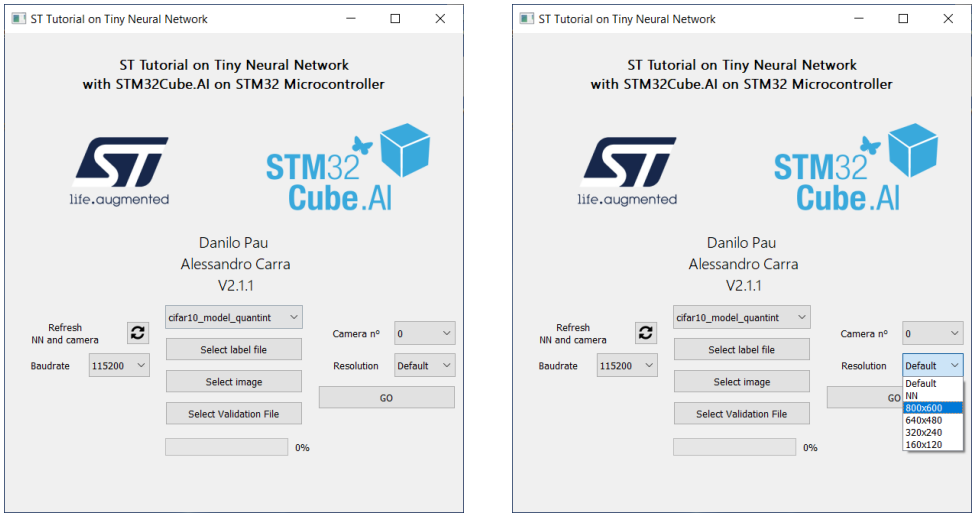
I modelli di rete neurale creati saranno utilizzati in tempo reale per la classificazione di immagini implementata su microcontrollore. Si possono effettuare validazioni e test con i campioni dal dataset ma non sempre l'accuratezza viene mantenuta con nuove immagini. Ciò si può verificare con dei field-test utilizzando un codice ad-hoc per ogni rete.

7.1 GUI

In questo progetto di tesi ho avuto l'opportunità di modificare e perfezionare un'interfaccia grafica che permette di testare le rete neurali presenti su MCU STM32 con file o immagini scattate da webcam su un PC, mediante il collegamento della board ad un computer con sistema operativo Windows o Linux. In questo modo è stato possibile verificare con facilità l'accuratezza delle reti progettate con nuove immagini scattate. L'interfaccia si presta ad una varietà di modelli NN per la classificazione. L'utente potrà scegliere ed utilizzare una delle reti tra quelle rilevate attraverso il menù a tendina centrale mostrato in figura 7.1a. I parametri che si possono impostare sono il baud rate e la risoluzione della webcam (Figura 7.1b). Con i vari button presenti nell'interfaccia è possibile scegliere il tipo di dato e il file da inviare alla board per l'inferenza:

- file CSV, contenente i campioni e i valori GroundTruth per la validazione;
- file Immagine, che dovrà essere compatibile con le librerie utilizzate, per poter predirre la classe;

- scatto da Webcam, per predirre immagini Live in modalità "Single-shot" o in "Loop".



(a) GUI Main Window

(b) GUI con menù selezione risoluzione Webcam

Figura 7.1: Graphical User Interface con menù a tendina

Per facilitare la comprensione dei risutati da parte dell'utente è stata inserita la possibilità di selezionare un file di testo contenente un elenco di elementi "Key-Value" in modo da affiancare una label ad ogni classe. A seconda del tipo di dato e dell'elaborazione scelta dall'utente, verrà generata la finestra dei risultati in modalità "Test" o "Validation". In figura 7.2 si possono osservare i risultati delle predizioni con rete µFoodNet. A sinistra vengono elencati le classi predette, su sfondo verde nel caso di esito positivo o su sfondo rosso nel caso di esito negativo. Nella parte destra si possono osservare la confusion matrix e i valori di accuracy per ogni label. In questo modo si è potuto verificare che l'accuracy della rete rimane costante anche con immagini non presenti nel dataset.

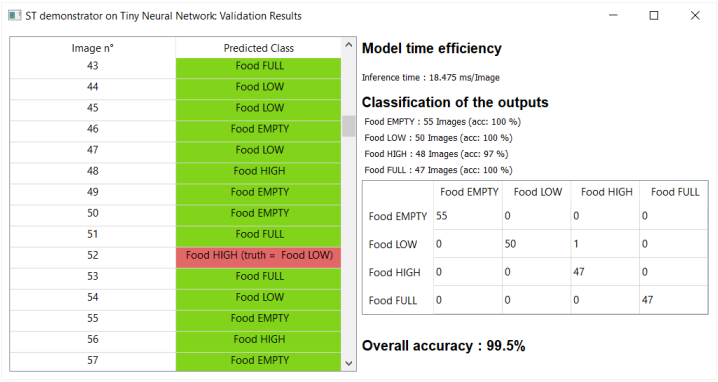


Figura 7.2: Results window con predizioni µFoodNet

La libreria alla base di questa interfaccia è stata sviluppata dalla ST-Microelectronics. Essa è in grado di rilevare le reti disponibili, estrarre le loro caratteristiche, instaurare la comunicazione tra PC e board e gestire le inferenze con i tensori di input.

Sono state utilizzate altre librerie, tra cui:

- **Numpy** [3], per la creazione degli array, operazioni varie, resize e reshape dati;
- **Time**, per calcolare i tempi di elaborazione e comunicazione;
- **OpenCV** [5], per gestire le immagini e le periferiche di acquisizione video.
- **PYQt5** [6], per realizzare l'interfaccia utente.

Grazie alla libreria PYQt5 è stata creata una QMainWindow con loghi, caselle di testo, bottoni, menù a tendina e altri Widgets.

Capitolo 8

Risultati

8.1 Addestramento rete neurale

Le tre reti neurali, **µCageNet**, **µBottleNet** e **µFoodNet**, sono state addestrate rispettivamente con i dataset **Cage**, **Bottle** e **Food**. Il dataset è stato suddiviso in un 80% per il train set (di cui un 30% per la validation) e il restante 20% per il test set. I valori dei pixel di ogni immagine sono stati normalizzati tra 0 e 1. Per la prima versione di dataset è stata utilizzato il parametro "class_weight" della funzione *model.fit()*, in modo da bilanciare le classi con un minor numero di immagini. Il numero di epoche massimo è stato fissato a 100, ma come si può vedere dal grafico di figura 8.1, la loss function decresce rapidamente già dalle prime epoche.

Source Code 8.1: Porzione di codice per il training delle NN

```
...
x_train, x_test, y_train, y_test = train_test_split(X, Y,
    test_size=0.2, random_state=42)
...
my_callbacks = [
EarlyStopping(monitor='val_loss', mode='min', verbose=1,
    patience=20)
#ModelCheckpoint(model_name, monitor='val_accuracy',
    mode='max', verbose=1, save_best_only=True)
]

history=model.fit(x_train, y_train,
```

```
10              batch_size=batch_size,
11              epochs=epochs,
12              validation_split=0.3,
13              shuffle=True,
14              callbacks = my_callbacks)
```

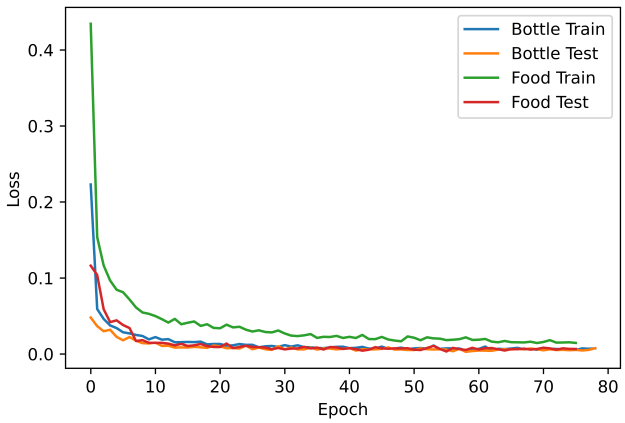


Figura 8.1: Andamento della loss in fase di training

8.2 Test reti neurali convoluzionali

Dopo aver addestrato i vari modelli di CNN, per ogni versione e tipologia, sono state effettuate delle predizioni sul test set utilizzando una metodologia ben precisa. Dato che per la prima versione del dataset le classi risultavano sbilanciate è stato deciso di calcolare delle metriche specifiche:

- l'Area under the receiver operating characteristic curve (ROC AUC);
- la precision;
- la recall;
- l'F1-score;
- l'accuracy;
- la confusion matrix.

Nelle figure 8.2, 8.3 e 8.4 sono riportati i valori dell'output della cella per il calcolo delle metriche. Si può notare come la quasi totalità delle immagini è

stata riconosciuta correttamente e che le metriche hanno valori superiori al 99%. L'utilizzo di questo schema per valutare le performance è risultato di fondamentale importanza date le numerose versioni che sono state sviluppate e quindi dei risultati che sono stati comparati.

Dataset V5 RAW Luma Only	
X.shape	(83598, 64, 64)
Y.shape	(83598,)
{'cage-off': 0, 'cage-on': 1}	
Y Counter	{0: 41799, 1: 41799}
Y train Counter	{1: 33497, 0: 33381}
Y test Counter	{0: 8418, 1: 8302}

Dataset V5 RAW Luma Only

µCageNet TFLite uint8 Results:

Area Under the Receiver Operating Characteristic Curve (ROC AUC):	0,9997

Classification report

	precision	recall	f1-score	support
cage-off	0,9995	1,0000	0,9997	8418
cage-on	1,0000	0,9995	0,9997	8302
accuracy			0,9998	16720
macro avg	0,9998	0,9998	0,9998	16720
weighted avg	0,9998	0,9998	0,9998	16720

Confusion Matrix:

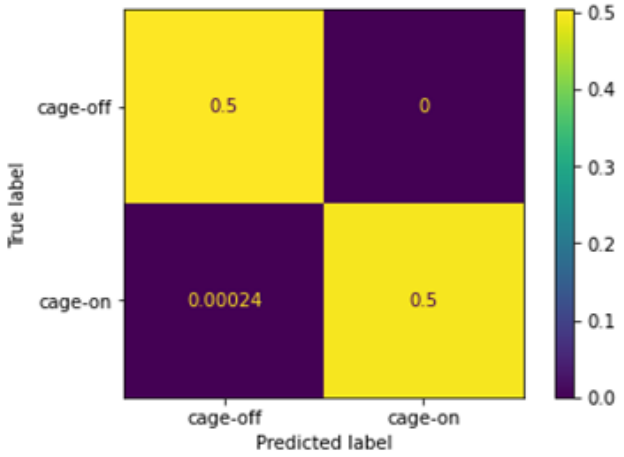


Figura 8.2: Risultati ottenuti con la rete µCageNet versione 5

Dataset V5 RAW Luma Only	
X.shape	(42998, 50, 50)
Y.shape	(30901,)
{'bottle-off': 0, 'bottle-on': 1}	
Y Counter	{0: 21499, 1: 21499}
Y train Counter	{1: 17219, 0: 17179}
Y test Counter	{0: 4320, 1: 4280}

Dataset V5 RAW Luma Only

µBottleNet TFLite uint8 Results:

Area Under the Receiver Operating Characteristic Curve (ROC AUC):	0,9984

Classification report

	precision	recall	f1-score	support
bottle-off	0,9988	0,9979	0,9984	4320
bottle-on	0,9979	0,9988	0,9984	4280
accuracy			0,9984	8600
macro avg	0,9984	0,9984	0,9984	8600
weighted avg	0,9984	0,9984	0,9984	8600

Confusion Matrix:

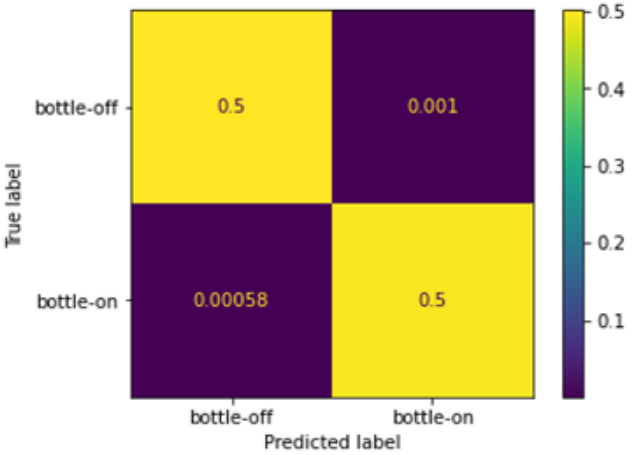


Figura 8.3: Risultati ottenuti con la rete µBottleNet versione 5

Dataset V5 RAW Luma Only	
X.shape	(68430, 102, 200)
Y.shape	(68430,)
{'food-empty': 0, 'food-low': 1, 'food-high': 2}	
Y Counter	{0: 23225, 1: 22807, 2: 22398}
Y train Counter	{0: 18603, 1: 18261, 2: 17880}
Y test Counter	{0: 4622, 1: 4546, 2: 4518}

Dataset V5 RAW Luma Only

µFoodNet TFLite uint8 Results:

Area Under the Receiver Operating Characteristic Curve (ROC AUC):	0,9999

Classification report

	precision	recall	f1-score	support
food-empty	1,0000	1,0000	1,0000	4622
food-low	1,0000	0,9997	0,9998	4546
food-high	0,9997	1,0000	0,9998	4518
accuracy			0,9999	13686
macro avg	0,9999	0,9999	0,9999	13686
weighted avg	0,9999	0,9999	0,9999	13686

Confusion Matrix:

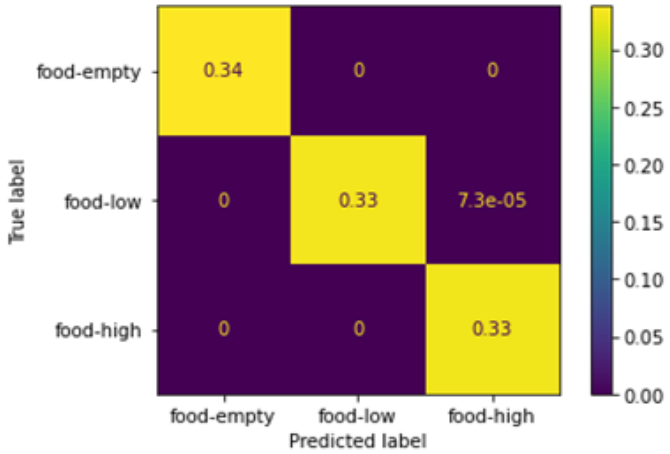


Figura 8.4: Risultati ottenuti con la rete µFoodNet versione 5

8.3 K-Fold Cross validation

La K-Fold Cross Validation è una procedura utilizzata per stimare le abilità del modello su nuovi campioni. La quinta versione delle reti neurali progettate rispetta le specifiche progettuali e questa procedura è stata applicata solo per quest'ultima versione. É stato scelto un valore di K uguale a 5 in modo da mantenere un rapporto 1 a 5 per il test set e per il train set come in fase d'addestramento. Al termine di ogni fold sono stati salvati il valore di Loss e Accuracy e solo successivamente sono stati calcolati i loro valori medi e la loro deviazione standard. Le porzioni di testo a seguire sono i risultati della K-Fold Cross validation per le reti **µCageNet**, **µBottleNet** e **µFoodNet** versione 5. Tutte hanno raggiunto valori di accuracy superiore al 99% con una bassa deviazione standard.

Source Code 8.2: µCageNet K-Fold cross valiadation K=5

```
------------------------------------------------------------------------
Score per fold
------------------------------------------------------------------------
> Fold 1 - Loss: 6.5804386395e-05 - Accuracy: 100.0%
------------------------------------------------------------------------
> Fold 2 - Loss: 0.00212531443685 - Accuracy: 99.9222517013%
------------------------------------------------------------------------
> Fold 3 - Loss: 0.00043212802847 - Accuracy: 99.9880373477%
------------------------------------------------------------------------
> Fold 4 - Loss: 0.00017345877131 - Accuracy: 99.994021654%
------------------------------------------------------------------------
> Fold 5 - Loss: 1.2822108146e-05 - Accuracy: 100.0%
------------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 99.98086214065552 (+- 0.029639012493380853)
> Loss: 0.0005619055462375399
------------------------------------------------------------------------
```

Source Code 8.3: µBottleNet K-Fold cross valiadation K=5

```
------------------------------------------------------------------------
Score per fold
```

```
------------------------------------------------------------------
Fold 1 - Loss 0.00811260379850 - Accuracy 99.8139560222%
------------------------------------------------------------------
Fold 2 - Loss 0.01429016701877 - Accuracy 99.5232582092%
------------------------------------------------------------------
Fold 3 - Loss 0.00827706884592 - Accuracy 99.7558116912%
------------------------------------------------------------------
Fold 4 - Loss 0.00724746054038 - Accuracy 99.8139321804%
------------------------------------------------------------------
Fold 5 - Loss 0.0068168770521  - Accuracy 99.7441589832%
------------------------------------------------------------------
Average scores for all folds
Accuracy 99.7302234172821 (+- 0.10742612649602429)
Loss 0.008948835451155901
```

Source Code 8.4: µFoodNet K-Fold cross valiadation K=5

```
------------------------------------------------------------------
Score per fold
------------------------------------------------------------------
> Fold 1 - Loss: 1.2337032785e-05 - Accuracy: 100.0%
------------------------------------------------------------------
> Fold 2 - Loss: 0.00093760358868 - Accuracy: 99.9853849411%
------------------------------------------------------------------
> Fold 3 - Loss: 0.00103799090720 - Accuracy: 99.9853849411%
------------------------------------------------------------------
> Fold 4 - Loss: 8.6751097114e-06 - Accuracy: 100.0%
------------------------------------------------------------------
> Fold 5 - Loss: 3.9196343393e-10 - Accuracy: 100.0%
------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 99.99415397644043 (+- 0.007159887372618142)
> Loss: 0.000399321406069858
------------------------------------------------------------------
```

8.4 Comparazione con MobileNet V2

La MobileNet V2 [26] è l'architettura che ha migliorato lo stato dell'arte per quanto riguarda le performance dei modelli mobile. É molto simile alla MobileNet V1 [19], tranne per il fatto che utilizza inverted residual blocks come collo di bottiglia. Ha un numero di parametri drasticamente inferiore rispetto al MobileNet originale. Per confrontare le performance e le caratteristiche della MobileNet V2, sono stati addestrati due modelli con i datase Bottle e Food versione 3. In tabella 8.1 sono riportati i valori dell'accuracy e delle analisi dei modelli, ottenute con il tool X-CUBE-AI. Si può notare che il modello MobileNet V2:

- addestrato con il Dataset Food, ha raggiunto un'accuracy inferiore alla µFoodNet del -17%;
- ha un numero di campioni 65 volte superiore;
- impiega circa 15 volte in più per ogni epoca durante l'addestramento;
- per l'elevato numero di parametri e la dimensione dei tensori dei layer interni, il modello non è compatibile con tutti i microcontrollori.

Dunque, non sempre si ottengono risultati migliori con reti di dimensioni maggiori e con un elevato numero di parametri.

Tabella 8.1: Analisi e comparativa CNN con MobileNet V2 addestrata con dataset bottle e food

	V3 luma only	Accuracy	Time per epochs [s]	Parameters	Parameters density
MobileNet V2	Bottle Dataset	100%	120	2.262M	91.5
	Food Dataset	82%	720	2.259M	92.5
µBottleNet	Bottle Dataset	99%	8	2,133	0.08
µFoodNet	Food Dataset	99%	40	34,423	1.39

8.5 Software

Nella porzione di codice 8.5 sono state calcolate varie metriche, illustrate nella sezione 8.2, per confrontare la precisione delle reti neurali grazie alle libreria SciKit Learn.

Source Code 8.5: Porzione di codice relativa alla quantizzazione con la procedura post training di TensorFlowLite

```
if keras_model_predict:
        print(model_name,"results:\n")
else:
        print(tflite_model_name,"results:\n")

print("Number of test images:",y_pred.shape[0])

y_pred_cat = y_pred
y_test_arg = np.argmax(y_test,axis=1)
y_pred_cat_arg = np.argmax(y_pred,axis=1)
target_names = labels_food
labels = [0,1]

#ROC AUC
y_test_arg_bin = preprocessing.label_binarize(y_test_arg,
    classes=np.arange(0,len(labels_food),1))
y_pred_cat_arg_bin =
    preprocessing.label_binarize(y_pred_cat_arg,
    classes=np.arange(0,len(labels_food),1))
auc = roc_auc_score(y_test_arg_bin, y_pred_cat_arg_bin)
print("Area Under the Receiver Operating Characteristic Curve
    (ROC AUC):",auc)

#Confusion matrix
print("\n \t \t Classification report
    \n",classification_report(y_test_arg, y_pred_cat_arg,
    target_names = labels_food, digits = 5  ))
cm = confusion_matrix(y_test_arg, y_pred_cat_arg,
    labels=np.arange(0,len(labels_food),1), normalize = 'all')
```

```
23 #print(cm)
24 disp = ConfusionMatrixDisplay(confusion_matrix=cm,
   ↪  display_labels=target_names)
25 disp.plot()
```

Capitolo 9

Conclusioni

Basandomi sui risultati discussi nel capitolo 8, in particolare quelli riportati nelle tabelle di figura 8.3, 8.3 e 8.3 e sugli output della cella per la K-Fold Cross Validation, riportati nella porzione di codice 8.2, 8.3 e 8.4, le reti neurali µCageNet, µBottleNet e µFoodNet hanno raggiunto un valore medio di accuratezza del 99.9% senza l'utilizzo di algoritmi di rettificazione. I risultati migliori sono stati ottenuti addestrando le reti neurali convoluzionali, utilizzando la versione numero 5 dei dataset Cage, Bottle e Food descritti nel capitolo 4 e quantizzando i modelli Keras FP32 con la procedura post-training quantization di TensorFlowLite a 8-bit. Sono stati rispettati i requisiti presentati nella sezione 1.4. Le somme dei valori ottenuti dalle analisi dell'implementazione delle CNN su MCU, grazie al tool X-CUBE-AI, sono di:

- MACC: 1034840
- RAM: 32.83 kB
- ROM: 34.92kB

La somma dei tempi medi di inferenza delle reti µCageNet, µBottleNet e µFoodNet su microcontrollore è di 19.02ms su STM32H7 e di 181.63ms su STM32L4. Inoltre, è stato dimostrato come modelli a complessità limitata possono ottenere valori di accuracy simili o migliori di uno più grande e complicato, utilizzando meno risorse e mantendendo la possibilità di essere implementato su MCU con risore limitate.

In sintesi, i risultati prinicipali di questo lavoro sono:

- i dataset Cage, Bootle e Food sono stati creati poichè non presenti nello stato dell'arte;
- entrambi i microcontrollori, STM32H7 e STM32L4, hanno abbastanza memoria e potenza di calcolo per essere utilizzati in questo progetto;
- non è stato utilizzato nessun algoritmo di image rectification.

Sviluppi futuri comprendo:

- lo sviluppo di una sola rete in grado di predire le immagini dei dataset Cage, Bootle e Food;
- l'utilizzo del prototipo finale per aumentare le dimensioni dei dataset e per svolgere ulteriori field-test.

Bibliografia

[1] URL: https://www.st.com/en/evaluation-tools/nucleo-h743zi.html.

[2] URL: https://os.mbed.com/blog/entry/ST-on-Mbed/.

[3] URL: https://numpy.org/.

[4] URL: https://scikit-learn.org/stable/.

[5] URL: https://opencv.org/.

[6] URL: https://pypi.org/project/PyQt5/.

[7] URL: https://www.tecniplast.it/uk/product/em500.html.

[8] URL: https://www.fokus.fraunhofer.de/en/fame/workingareas/ai.

[9] URL: https://www.tibco.com/it/reference-center/what-is-a-neural-network.

[10] URL: https://www.tinyml.org/.

[11] Iljoo Baek et al. *Real-time Detection, Tracking, and Classification of Moving and Stationary Objects using Multiple Fisheye Images*. 2018. arXiv: 1803.06077 [cs.CV].

[12] Indranil Balki et al. «Sample-Size Determination Methodologies for Machine Learning in Medical Imaging Research: A Systematic Review». In: *Canadian Association of Radiologists Journal* 70 (set. 2019). DOI: 10.1016/j.carj.2019.06.002.

[13] Akanksh Basavaraju et al. «A Machine Learning Approach to Road Surface Anomaly Assessment Using Smartphone Sensors». In: *IEEE Sensors Journal* PP (nov. 2019), pp. 1–1. DOI: 10.1109/JSEN.2019.2952857.

[14] Y. Bengio e Yann Lecun. «Convolutional Networks for Images, Speech, and Time-Series». In: (nov. 1997).

[15] Simone Bianco et al. «Benchmark Analysis of Representative Deep Neural Network Architectures». In: *CoRR* abs/1810.00736 (2018). arXiv: 1810.00736.

[16] Marius Cordts et al. «The Cityscapes Dataset for Semantic Urban Scene Understanding». In: *CoRR* abs/1604.01685 (2016). arXiv: 1604.01685.

[17] Liuyuan Deng et al. «CNN based semantic segmentation for urban traffic scenes using fisheye camera». In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 231–236. DOI: 10.1109/IVS.2017.7995725.

[18] Albert Einstein. «Zur Elektrodynamik bewegter Körper. (German) [On the electrodynamics of moving bodies]». In: *Annalen der Physik* 322.10 (1905), pp. 891–921. DOI: http://dx.doi.org/10.1002/andp.19053221004.

[19] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. DOI: 10.48550/ARXIV.1704.04861. URL: https://arxiv.org/abs/1704.04861.

[20] Xiaodong Lei et al. «Fisheye Image Object Detection Based on an Improved YOLOv3 Algorithm». In: *2020 Chinese Automation Congress (CAC)*. 2020, pp. 5801–5805. DOI: 10.1109/CAC51589.2020.9326859.

[21] Jing Li et al. «Fisheye image rectification for efficient large-scale stereo». In: *2016 3rd International Conference on Systems and Informatics (ICSAI)*. 2016, pp. 881–885. DOI: 10.1109/ICSAI.2016.7811075.

[22] Junjie Li e Ding Liu. «Information Bottleneck Theory on Convolutional Neural Networks». In: *Neural Processing Letters* 53 (apr. 2021), pp. 1–16. DOI: 10.1007/s11063-021-10445-6.

[23] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997. ISBN: 9780071154673. URL: https://books.google.it/books?id=EoYBngEACAAJ.

[24] *OV7725 CMOS VGA (640x480) Image Sensor, Datasheet*. 1.4. 2007.

[25] Joseph Redmon e Ali Farhadi. «YOLOv3: An Incremental Improvement». In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767.

[26] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV].

[27] S. Shah e J. Aggarwal. *Mobile robot navigation and scene modeling using stereo fish-eye lens system*. 1997.

[28] *Sito ufficiale Keras*. URL: https://keras.io/.

[29] *Sito ufficiale Netron*. URL: https://netron.app/.

[30] *Sito ufficiale STM32CubeIDE*. URL: https://www.st.com/en/development-tools/stm32cubeide.html.

[31] *Sito ufficiale TensorFlow*. URL: https://www.tensorflow.org/.

[32] *Sito ufficiale TensorFlow Lite*. URL: https://www.tensorflow.org/lite.

[33] *Sito ufficiale tool X-CUBE-AI*. URL: https://www.st.com/en/embedded-software/x-cube-ai.html.

[34] Emma Strubell, Ananya Ganesh e Andrew McCallum. «Energy and Policy Considerations for Deep Learning in NLP». In: *CoRR* abs/1906.02243 (2019). arXiv: 1906.02243.

[35] Ozan Tezcan M. *Deep learning algorithms for background subtraction and people detection*. 2021.

[36] Neil C. Thompson et al. «Deep Learning's Diminishing Returns: The Cost of Improvement is Becoming Unsustainable». In: *IEEE Spectrum* 58.10 (2021), pp. 50–55. DOI: 10.1109/MSPEC.2021.9563954.

[37] E.A. Ulman, Douglas Compton e J. Kochaneck. «Measuring food and water intake in rats and mice». In: *ALN Mag* (gen. 2008), pp. 17–20.

[38] Nguyen Van Tuan, Thanh Binh Nguyen e Sun-Tae Chung. «ConvNets and AGMM based real-time human detection under fisheye camera for embedded surveillance». In: *2016 International Conference on Information and Communication Technology Convergence (ICTC)*. 2016, pp. 840–845. DOI: 10.1109/ICTC.2016.7763311.

[39] Franck Vandewiele, Cina Motamed e Tarek Yahiaoui. «Visibility management for object tracking in the context of a fisheye camera network». In: *2012 Sixth International Conference on Distributed Smart Cameras (ICDSC)*. 2012, pp. 1–6.

[40] Istituto Zooprofilattivo Sperimentale dekke Venezie. «Linee guida dell'IZSVe per la gestione di mammiferi, volatili e specie ittiche utilizzati a fini scientifici». In: (2019 (3.0)).

[41] Tsaipei Wang et al. «Mask-RCNN Based People Detection Using A Top-View Fisheye Camera». In: *2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI)*. 2019, pp. 1–4. DOI: 10.1109/TAAI48200.2019.8959887.

[42] Cheng-Yun Yang e Homer H. Chen. «Efficient Face Detection in the Fisheye Image Domain». In: *IEEE Transactions on Image Processing* 30 (2021), pp. 5641–5651. DOI: 10.1109/TIP.2021.3087400.

[43] Xiaoqing Yin et al. *FishEyeRecNet: A Multi-Context Collaborative Deep Network for Fisheye Image Rectification*. 2018. arXiv: 1804.04784 [cs.CV].