ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA (DISI)

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

TESI DI LAUREA

in

Computer Vision and Image Processing M

SVILUPPO DI UNA RETE NEURALE PER LA CLASSIFICAZIONE DI IMMAGINI SU PIATTAFORMA RISC-V

CANDIDATO
Dario Grandi

RELATORE:
Chiar.mo Prof. Luigi Di Stefano

Anno Accademico 2021/22

Sessione III

Sommario

Introduzione 1

Capitolo 1 Machine Learning e Reti Neurali..... 3

1.1 Panoramica..... 3

1.2 Tipi di reti e convoluzione.....8

1.3 Reti Convoluzionali a basso consumo..... 14

1.4 Algoritmi di Ottimizzazione.....16

1.5 MobileNet 1000..... 19

Capitolo 2 Architettura Risc-V..... 23

2.1 Panoramica Risc/Cisc 23

2.2 Architettura Risc-V..... …25

Capitolo 3 Analisi e scelta dell''hardware..... 28

3.1 Fase di analisi dei requisiti..... 28

3.2 Camera CMOS utilizzata..... 30

3.2 ST7789V-LCD..... 33

3.3 Kendryte K210 e KPU..... 36

3.4 Linguaggi e piattaforme utilizzate..... 37

3.5 Codice di verifica dell'Hardware..... 40

Capitolo 4 Analisi e progettazione della rete..... 43

4.1 Tecnologie utilizzate..... 43

4.2 Transfer Learning..... 46

4.3 Definizione del modello..... 48

Capitolo 5 Sistema di cattura delle immagini..... 51

4.1 Componenti Hardware..... 51

4.2 Logica Operativa..... 52

4.3 Implementazione..... 53

Capitolo 6 Creazione del Dataset……..……..... 56
6.1 Fase di campionamento..... 56
6.2 Elaborazione delle immagini 58

Capitolo 7 Training e test del modello sulla scheda……..... 65
7.1 Creazione Cnn in Keras 65
7.2 Training della rete..... 66
7.3 Deploymente e test del modello sulla scheda..... 70

Capitolo 8 Conclusioni e Sviluppi Futuri……..... 73
9.2 Bibliografia..... 74

0-Introduzione

Al giorno d'oggi, l'avanzamento tecnologico, soprattutto nel mondo dello sviluppo software e hardware, ha favorito la nascita di sistemi complessi, in grado di rispondere a sfide che sarebbero sembrate impossibili solo pochi anni prima. Una di queste è sicuramente la capacità dei calcolatori di imparare autonomamente e prendere delle decisioni a partire da un insieme di dati.
Se da un lato c'è l'esigenza di rendere i processi sempre più intelligenti dall'altro, c'è una maggiore consapevolezza di doverne aumentare la sostenibilità, mantenendo i consumi bassi e riducendo gli impatti ambientali.

L'obbiettivo di questa tesi è stato sviluppare una rete neurale convoluzionale per la classificazione di immagini a colori su un dispositivo embedded a basso consumo con architettura Risc di ultima generazione.
L'idea è quella di creare un dispositivo di riconoscimento visivo in grado di determinare il tipo di terreno sottostante, sono state scelte 5 categorie come: stradale, sterrato, ghiaioso, sintetico e pedonale.
Il modello creato è stato sviluppato a partire da un modello pre-esistente ed è stato ampliato e modificato per adattarlo allo scopo della tesi.
Successivamente è stata eseguita la fase di addestramento ed è stato verificato il suo corretto funzionamento direttamente sulla scheda, provandolo su casistiche reali.
Il Data set utilizzato per la fase di apprendimento del modello è stato appositamente creato da zero ed è stato necessario effettuare le catalogazioni manualmente.
Per la catalogazione delle imamgini è stato creato un sistema di cattura hardware/software autonomo e portabile.
Il sistema sviluppato è stato alla fine testato su immagini reali per verificarne la qualità.
Al fine di illustrare al meglio il lavoro svolto, questa tesi è stata strutturata nella seguente maniera.
Nel primo capitolo viene data una panoramica sul Machine learning, verrà spiegato cosa sono le reti neurali, il livello di convoluzione e ne verranno descritte le principali tipologie e tecniche.

Nel capitolo 2 verrà descritto il concetto di CISC e RISC arrivando poi a definire il concetto di Risc-V.
A seguire nel capitorlo 3 verranno analizzate le componenti che saranno utilizzate per la creazione del sistema, verrà descritta la tecnologia e l'implementazione di alcune sue parti.
Nel capitolo quattro descriveremo come sia stato realizzato il modello per la classificazione, forniremo una panoramica sulla tecnica del transfer learning e verrà descritto come sarà strutturata la rete che sarà utilizzata per la classificazione.
Arriveremo poi al capitolo 5 analizzando i requisiti per la creazione del sistema di cattura, verranno individuate le componenti e sarà implementata la logica di funzionamento.
Nel sesto capitolo è illustrata la fase di campionamento per la composizione del dataset, verranno descritte anche le tecniche per la sua creazione e elaborazione.
Nel capitolo 7 verrà eseguito il training del modello sulla piattaforma Google Colab utilizzando le API specifiche di Tensorflow e ne verrà analizzato il risultato.

Infine nel caitolo 8 verrano scritte alcune considerazioni finali mostrando gli aspetti positivi e negativi e i futuri sviluppi che è possibile apportare sia a livello hardware che software al sistema.

1-Machine Learning e Reti Neurali

1.1 Panoramica

Al giorno d'oggi il campo dell'intelligenza artificiale è in continua evoluzione, sono innumerevoli gli studi relativi a questo settore e tra questi emerge il machine learning detto anche apprendimento automatico.
Il Machine Learning riguarda lo studio, la costruzione e l'implementazione di algoritmi e strutture che permettono ai sistemi di calcolo di evolvere e fare previsioni in modo automatico a partire da un insieme di dati in ingresso, costruendo modelli previsionali accurati.
Mediante una serie di metodi possono svolgere delle attività non programmate e possono apprendere dall'esperienza pregressa come esattamente fa l'intelligenza umana, correggendosi e migliorandosi attraverso gli errori commessi e prendendo decisioni autonome.
In questo contesto possiamo classificare le reti neurali, ovvero particolari modelli computazionali composti da "neuroni" artificiali, sono ispirate vagamente a una rete neurale biologica semplificata.

In termini generici, una rete neurale artificiale è un processore distribuito, ispirato ai principi di funzionamento del sistema nervoso degli organismi evoluti, costituito dalla interconnessione di unità computazionali elementari (neuroni), la caratteristica principale è che la conoscenza viene acquisita dall'ambiente attraverso un processo adattativo di apprendimento ed è immagazzinata nei parametri della rete, in particolare, nei pesi associati alle connessioni.
I neuroni, che si possono vedere come nodi di una rete orientata provvisti di capacità di elaborazione, ricevono in ingresso una combinazione dei segnali provenienti dall'esterno o dalle altre unità e ne effettuano una trasformazione tramite una funzione, tipicamente non lineare, detta funzione di attivazione. L'uscita di ciascun neurone viene poi inviata agli altri nodi oppure direttamente all'uscita della rete, attraverso connessioni orientate e pesate.

Nella struttura più semplice, basata sul modello proposto nel 1943 da McCulloch e Pitts [CUL43] e rielaborato successivamente da Rosenblatt [ROS58], gli ingressi sono

moltiplicati per dei pesi e la somma algebrica pesata viene confrontata con un valore di soglia, il neurone fornisce l'uscita 1 se la somma pesata degli ingressi è maggiore del valore di soglia, e l'uscita -1 (oppure 0) altrimenti.
La figura in basso mostra come è composto a livello grafico.

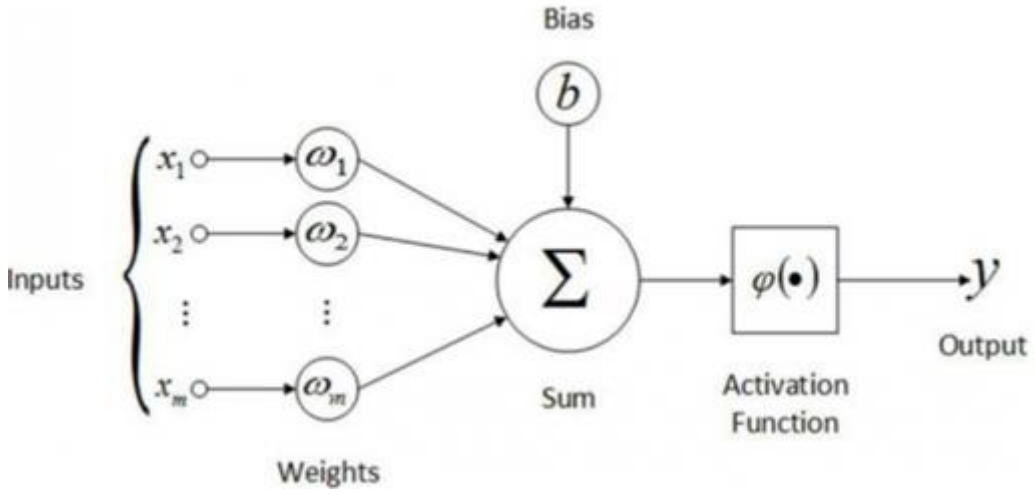

Figura 1.1: percettrone [MAY19]

Come possiamo vedere in figura (Fig.1.1), la funzione di attivazione determina il valore del segnale in uscita in base alla somma dei valori in ingresso moltiplicati per i relativi pesi, mappando i valori risultanti tra 0 e 1 o -1 e 1 ecc. (a seconda della funzione).

L'aspetto fondamentale delle funzioni di attivazione oltre a determinare il valore dell'output, è anche quello di poter dare non linearità.
Sebbene ci siano problemi nella vita che sono di natura lineare, per esempio, se si sta cercando di capire il costo di un certo numero di mele e si conosce il costo di una singola mela e ipotizzando che non ci sono sconti per le vendite all'ingrosso, allora l'equazione per calcolare il prezzo di qualsiasi numero di questi prodotti è un'equazione lineare.
Altri problemi invece non sono così semplici, come il prezzo di una casa.
Il numero di fattori che entrano in gioco, come le dimensioni, la posizione, il periodo dell'anno in cui si cerca di vendere, il numero di stanze, il cortile, il quartiere e così via, rende il tutto un operazione più complessa. Molti dei problemi più interessanti e difficili del nostro tempo sono di natura non lineare. La principale attrattiva delle reti neurali è la loro capacità di risolvere questi tipi di problemi.

Generalmente una funzione di attivazione ha 3 proprietà:

Non linearità - questa è la proprietà cruciale della funzione di attivazione. Grazie ad

essa, la rete neurale può essere utilizzata per risolvere problemi non lineari.

Continuamente differenziabile - È una proprietà desiderabile per supportare metodi di ottimizzazione basati sul gradiente.

Monotona: aiuta la rete neurale a convergere più facilmente in un modello più preciso.

Funzine attivazione Lineare:

La funzione di attivazione lineare, spesso chiamata funzione di attivazione dell'identità , è proporzionale all'input (Fig.1.2). L'intervallo della funzione di attivazione lineare sarà (da -∞ a ∞). La funzione di attivazione lineare somma semplicemente il totale ponderato degli input e restituisce il risultato (f(x)=x):

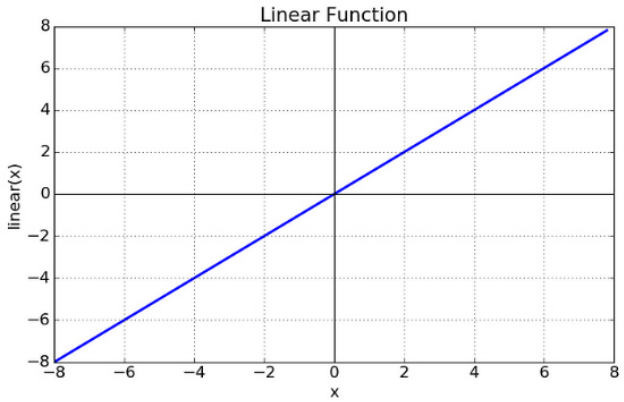

Figura 1.2: funzione lineare [SAG17]

Non lineare

Per far sì che la rete rappresenti problemi più complessi, sono necessarie funzioni di attivazione non lineari.

Non lineare significa che l'output non può essere riprodotto da una combinazione lineare degli input, il che non è la stessa cosa di un output che si presenta come una linea retta (Fig.1.3).

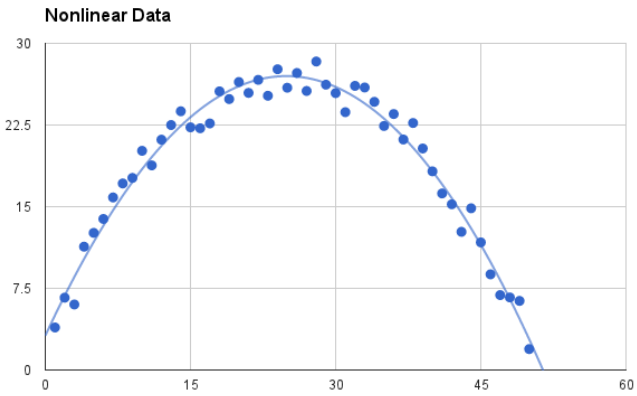

Figura 1.3: non-linearità [SAG17]

Funzione di attivazione sigmoide

Sigmoid accetta un numero come input e restituisce un numero compreso tra 0 e 1. È semplice da usare e ha tutte le qualità desiderabili delle funzioni di attivazione: non linearità, differenziazione continua, monotonia e un intervallo di output (Fig.1.4).

Questa funzione di attivazione è utilizzata principalmente nei problemi di classificazione binaria. Questa funzione sigmoidea fornisce la probabilità dell'esistenza di una classe particolare.

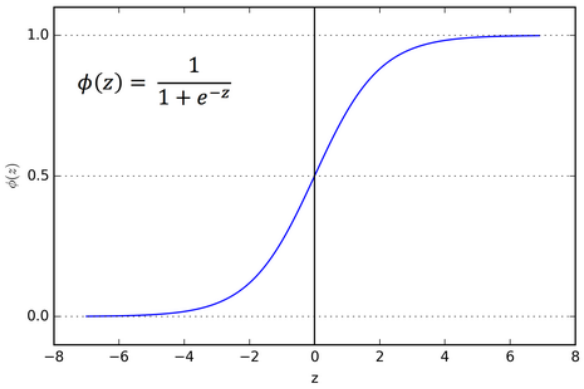

Figura 1.4: sigmoid function [SAG17]

Tanh o tangente iperbolica

Hyperbolic Tangent Function, è una funzione iperbolica, indicata con tanh(x), è definita come rapporto tra il seno iperbolico ed il coseno iperbolico ed è mappata in un intervallo fra -1 e 1 (Fig.1.5).

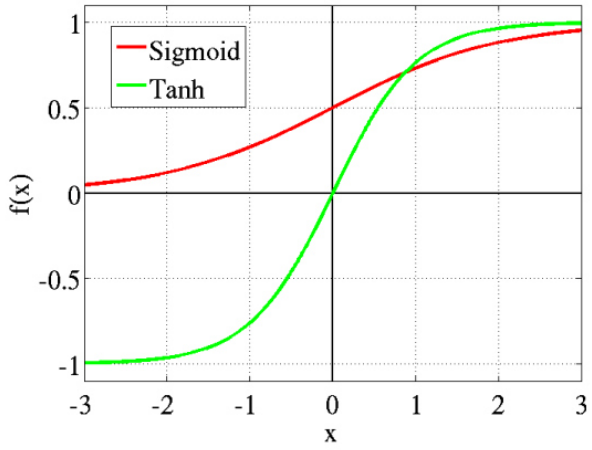

Figura 1.5: Tanh function [SAG17]

ReLU (Rectified Linear Unit) Activation Function

La ReLU è la funzione di attivazione più utilizzata al mondo in questo momento, poiché viene impiegata in quasi tutte le reti neurali convoluzionali o di deep learning (Fig.1.6).

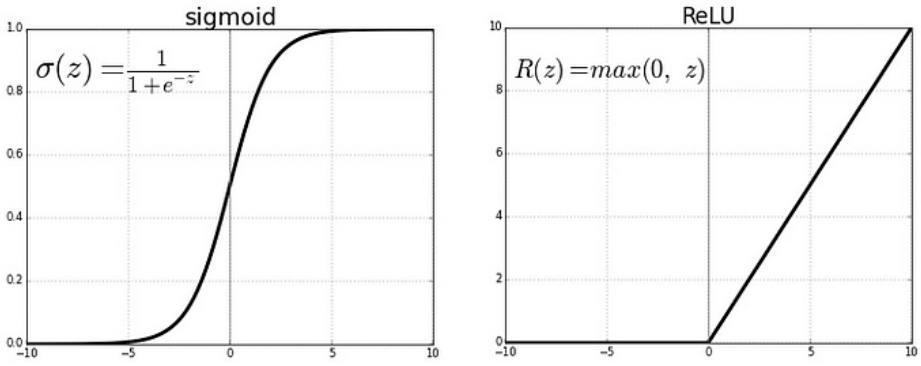

Figura 1.6: sigmoide e relu a confronto [SAG17]

SoftMax

La funzione softmax produce un vettore di valori che possono essere interpretati come probabilità di appartenenza alla classe (Fig.1.7).

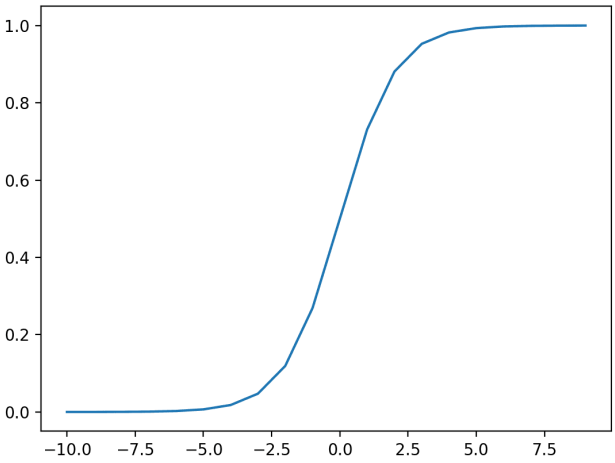

Figura 1.7: SoftMax [JAS21]

È correlata alla funzione argmax, che dà come risultato 0 per tutte le opzioni e 1 per l'opzione scelta. Softmax è una versione più "morbida" di argmax, che consente di ottenere un risultato simile a una funzione winner-take-all.

1.2 Tipi di reti e convoluzione

Un aspetto importante per differenziare le reti neurali è sicuramente il modo in cui i segnali viaggiano all'interno degli strati.
Le due tipologie di reti principali sono

- **Feed_forward Neural Network:**
  Permettono ai segnali di viaggiare in maniera unidirezionale, non ci sono feedback (loop); cioè, l'uscita di uno strato non influisce sullo stesso strato (backpropagation). Queste reti non hanno memoria degli input avvenuti a tempi precedenti, per cui l'output è determinato solamente dall'attuale input.
- **Recurrent Neural Network:**
  A differenza delle Feed-forward queste reti possono avere segnali che viaggiano in entrambe le direzioni, introducendo dei loop nella rete. Queste reti sono

potenti e possono diventare estremamente complicate. I calcoli derivati da un input precedente vengono reimmessi nella rete, il che le conferisce una sorta di memoria. Le RNN sono dinamiche: il loro "stato" cambia continuamente fino a raggiungere un punto di equilibrio. Rimangono al punto di equilibrio finché l'input non cambia e occorre trovare un nuovo equilibrio.

Nell'immagine in Fig. 1.8 possiamo vedere lo schema di esempio delle due tipologie.

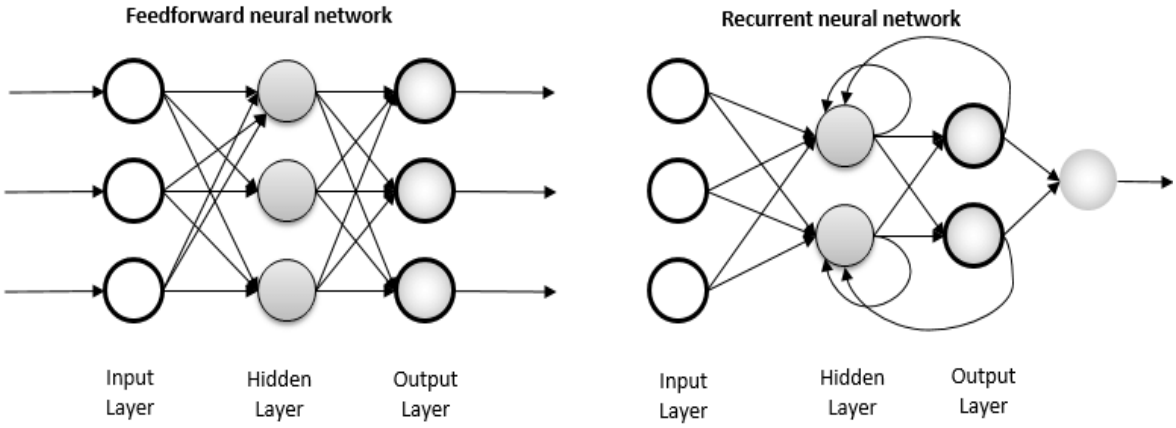

Figura 1.8: Feedforward e Recurrent Neural Network [ENG17]

Se da un lato le RNN hanno un potenziale di calcolo maggiore avendo memoria interna dall'altro abbiamo ritardi più elevati e velocità minore rispetto alle reti feed-forward, ovviamente la potenza di calcolo impiegata è superiore e necessita di risorse adeguate, l'indipendenza del neurone sullo stesso layer puo esser un fattore determinante per il problema che si sta affrontando quindi la scelta del tipo di rete dipende dal problema, le RNN sono tipicamente usate per traduzioni di linguaggi, conversioni voce-testo, controlli robotici etc..le reti feed_forward invece sono usate per il riconoscimento di Pattern, Riconoscimento della voce e del testo etc..

Ora ci focalizziamo unicamente sulle reti feed-forward, tendenzialmente quando una rete ha piu di un layer nascosto viene comunemente chiamata Deep Neural Network, questo proprio perché la profondità sta per il numero di layer che sono presenti, più livelli sono più è profonda la rete.
Queste reti permettono oggi di sfidare le prestazioni degli esseri umani raggiungendo risultati inimmaginabili fino a qualche decina di anni fa.

L'approccio "profondo" è emerso nel 2006 con i primi discorsi su reti a più livelli da parte di Geoffrey Hinton [GEO06], uno tra i ricercatori più influenti nello sviluppo dell'apprendimento profondo, nel 2009 ha registrato una nuova ondata di popolarità con la prima reale applicazione di successo: il riconoscimento vocale.

Queste due tappe sono state fondamentali per l'avanzare delle Deep Neural Network all'interno dell'Hype Cycle di Gartner [JAC10] verso il "peak of inflated expectations" [GAR22] ma anche altri 3 fattori hanno contribuito al loro sviluppo.

- Aumento dei dati. Grazie ai big data ora il software di deep learning può migliorare notevolmente il suo livello di apprendimento permettendo di ottenere prestazioni di gran lunga più soddisfacenti rispetto a quelle di altri algoritmi.
- Aumento delle performance dei computer che, grazie allo sviluppo di sistemi di calcolo parallelo altamente performanti basati su GPU (Graphics Processing Unit), hanno consentito di ottenere risultati migliori e allo stesso tempo hanno ridotto notevolmente i tempi di calcolo.
- Miglioramento dell'addestramento: grazie all'applicazione di metodi sempre più efficaci le reti neurali hanno aumentato la qualità delle loro performance permettendo ai ricercatori di ottenere risultati significativi.

All'interno dell'insieme delle reti neurali profonde troviamo una struttura denominata Rete Neurale Convoluzionale(Convolutional Neural Network), come viene specificato dal nome questa prende spunto da una operazione chiamata Convoluzione.
Le Reti Neurali Convoluzionali sono particolarmente utili per individuare pattern nelle immagini per il riconoscimento di oggetti, volti e scene. Inoltre, possono essere efficaci per la classificazione di dati non immagine come dati audio, serie storiche e segnali.

Le applicazioni che richiedono il riconoscimento di oggetti e la visione artificiale, come i veicoli a guida autonoma e le applicazioni di riconoscimento facciale, si basano proprio su queste reti.

Tipicamente a seguire lo strato di convoluzione vi sono altre operazioni connesse come:

- Livello di pooling
- Livello completamente connesso (FC, Fully-Connected)

Il livello convoluzionale è il primo livello di una rete convoluzionale. Ad ogni livello, la complessità della CNN aumenta così come la porzione dell'immagine che viene identificata. I primi livelli si concentrano su funzioni semplici, ad esempio i colori e i contorni. Mentre i dati dell'immagine avanzano attraverso i livelli della CNN, vengono riconosciuti elementi o forme più grandi fino a quando, infine, non viene identificato l'oggetto.

Soffermandoci un attimo sul concetto di Convoluzione possiamo dire che l'operazione di convoluzione, dal punto di vista matematico, consiste nel "moltiplicare" tra di loro due matrici opportunamente traslate per calcolare la risultante, di cui una rappresenta l'immagine oggetto di analisi e l'altra il filtro che viene applicato.

Si possono applicare filtri specifici per estrarre un set di informazioni, ad esempio nel riconoscimento di un'immagine si potrebbero estrarre i bordi di sinistra con il primo filtro, quelli di destra con il secondo, gli spigoli con il terzo etc. … fino a raggiungere il livello di accuratezza ottimale.

Nel processo di convoluzione abbiamo tre elementi fondamentali:

- L'immagine di input: una matrice di pixel rappresentativa dell'immagine. Ad esempio, sia 32×32 per visualizzare un numero, una lettera o un'icona.
- La feature detector o Kernel: una matrice che agisce da filtro attraverso l'operazione di convoluzione. Ad esempio 3×3 o 7×7.
- La feature map: la matrice risultante dalla convoluzione tra immagine di input e il Kernel.

Al termine di questo processo si ottiene la feature map, una matrice più piccola rispetto a quella relativa all'immagine di input.
Per esempio in figura 1.9, filtrando un'immagine 32×32 con un kernel 7×7, otterremo una matrice risultante di dimensione 26×26.
Nel processo di convoluzione si perdono alcune informazioni sull'immagine ossia quelle non contenute nella feature detector o kernel prescelta; tuttavia, quelle informazioni saranno contenute in altre future map.

Infatti, utilizzando più filtri di convoluzione si otterranno molteplici feature map, ciascuna delle quali memorizzerà feature distintive della nostra immagine, ad esempio il contorno destro, gli spigoli, i colori più scuri etc.

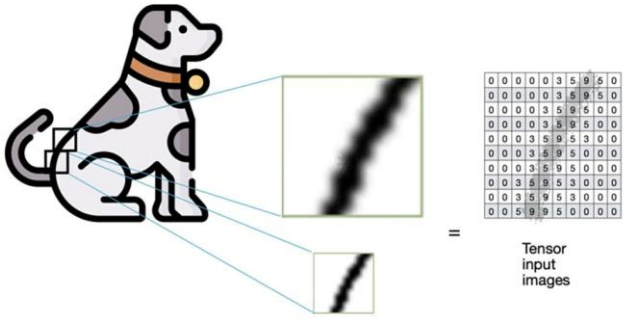

Figura 1.9: immagine di input [HUM18]

In figura l'immagine del cane viene codificata ottenendo la matrice di input.

In realtà all'algoritmo CNN verranno passate in input una serie di immagini di training, quindi un vettore di matrici chiamato in gergo anche Tensor (tensore).

Come detto, se alla matrice di input rappresentativa dell'immagine, vengono applicati diversi filtri con più operazioni di convoluzione si otterranno una serie di feature map.
Le feature map ottenute sono a loro volta "sotto-campionate" attraverso un processo che prende il nome di pooling.

Pooling è la tecnica che consente di ridurre la complessità considerando solo una parte dei dati.
Si tratta di un processo di dimensional reduction che consente di semplificare la complessità di una CNN e permettono di accelerare la crescita del recepptive field.

Le principali due tecniche di pooling sono: max pooling e average pooling.
In pratica si effettuano operazioni di calcolo del massimo o del valor medio su un subset di caselle della matrice in input.

Il processo di convoluzione e quello di sotto campionamento tramite pooling, sono entrambi ripetuti, applicando tra l'altro funzioni di attivazione non lineari alle varie feature map, ovvero la Rectified Linear Unit (RELU).

Dopo N processi di convoluzione – Pooling – RELU, occorrerà trasformare le pooled feature map in un vettore a una dimensione, utilizzando il processo chiamato Flattening (appiattimento).

Sarà così possibile fornire in input il vettore risultante ad una rete neurale fully connected, dove ogni nodo risulterà connesso con tutti gli altri: questo è in pratica l'hidden level della nostra CNN che si occupa della predizione e classificazione dell'immagine.

Al termine del processo si giungerà il livello di output, dove verranno applicate due tipologie di funzioni di attivazione per completare la classificazione dell'immagine: la funzione sigmoidea in caso di classificazione binaria, oppure la sua variante multi-dimensionale chiamata softmax, adatta alla classificazione multiclasse.

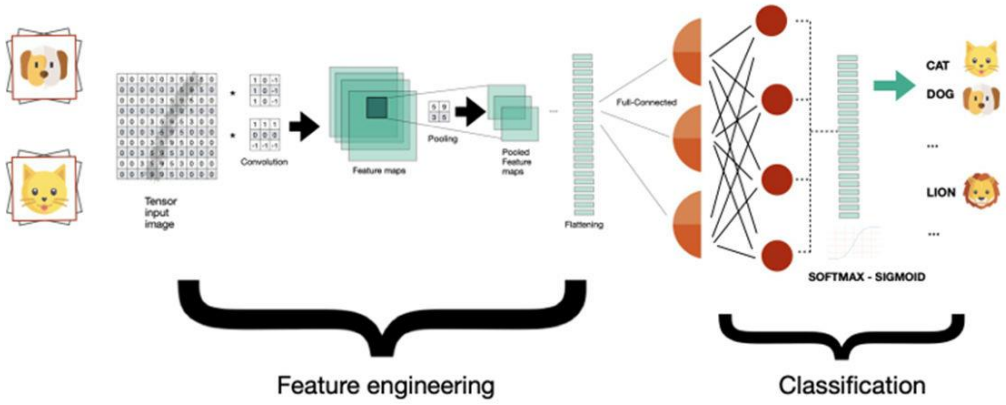

Figura 1.10: suddivisione della rete[HUM18]

Analizzando la figura sopra riportata (Fig.1.10), possiamo notare che la rete CNN è suddivisa in due processi:

-il primo racchiude i livelli iniziali dove vengono preparate le feature utilizzando i processi di convoluzione, pooling e flattening. L'obiettivo è quello di estrarre le caratteristiche dell'immagine che siano presentabili al processo successivo, la rete neurale di classificazione.

-Il secondo processo è costituito da una rete neurale tradizionale, nel nostro caso completamente connessa per classificare al meglio l'immagine in base alle feature ricevute in input.

1.3 Reti Convoluzionali a basso consumo

Le reti neurali convoluzionali tradizionali sono computazionalmente potenti e richiedono prestazioni di calcolo estremamente elevate se si vuole avere un risultato ottimale.

Considerando tutte le reti neurali convoluzionali di grandi dimensioni, come le ResNet, le VGG e simili, viene da chiedersi come sia possibile rendere tutte queste reti più piccole con meno parametri, mantenendo lo stesso livello di accuratezza o addirittura migliorando la generalizzazione del modello utilizzando una quantità inferiore di parametri.
Per questo sono state progettate in passato delle varianti di reti convoluzionali che possano risolvere tale problema.

DepthWise Convolution
Questa è un tipo di convoluzione in cui si applica un singolo filtro convoluzionale per ogni canale di ingresso, Nella normale convoluzione 2D eseguita su più canali di ingresso, il filtro è profondo quanto l'ingresso e ci permette di mescolare liberamente i canali per generare ogni elemento in uscita. Al contrario, le convoluzioni in profondità mantengono ogni canale separato.
La convoluzione quindi viene applicata a un singolo canale alla volta, a differenza delle CNN standard in cui viene eseguita per tutti gli M canali. Quindi, in questo caso i filtri/kernel saranno di dimensioni Dk x Dk x 1. Dal momento che ci sono M canali nei dati di ingresso sarebbero necessari M filtri di questo tipo. L'output risultante sarà di dimensioni Dp x Dp x M.

Nella figura sottostante (Fig.1.11) possiamo vederne una rappresentazione grafica:

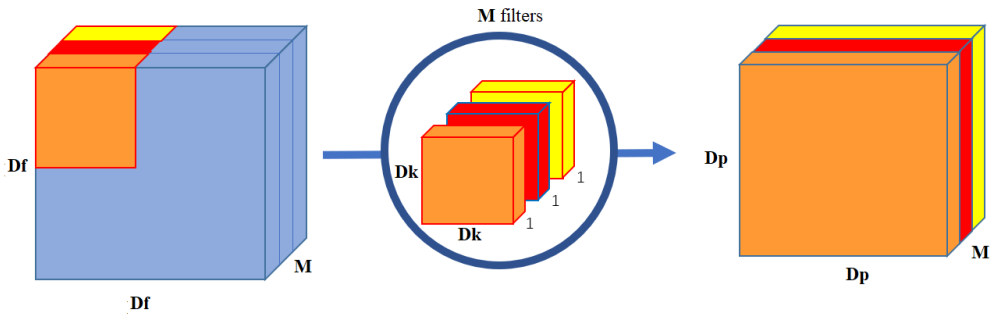

Figura 1.11: Operazione DepthWise [DIE22]

Per quello che riguarda il costo di questa operazione possiamo dire che una singola operazione di convoluzione richiede Dk*Dk moltiplicazioni.

Poiché il filtro viene slittato di Dp x Dp volte su tutti i canali M, il numero totale di moltiplicazioni è uguale a M * Dp * Dp * Dk * Dk.

Quindi per l'operazione di convoluzione in profondità avremo un totale di moltiplicazioni pari a M * Dk2 * Dp2.

Depth-wise Separable Convolution

Questo tipo di convoluzione deriva dalla depthwise convolution ma con l'eccezione che viene applicata un ulteriore operazione, chiamata Point wise Convolution.

Nell'operazione point-wise, viene applicata un'operazione di convoluzione 1×1 sui canali M. Quindi la dimensione del filtro per questa operazione sarà 1 x 1 x M.. Se utilizziamo N filtri di questo tipo, la dimensione di uscita diventa Dp x Dp x N (Fig.1.12).

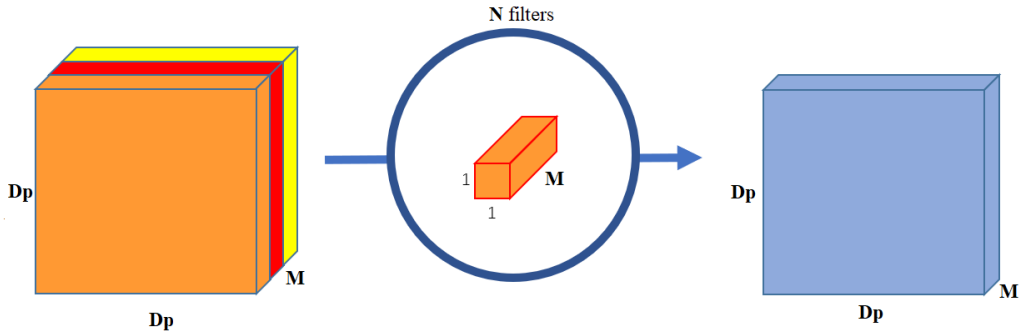

Figura 1.12: Separable DepthWise [DIE22]

Una singola operazione di convoluzione richiede 1*M moltiplicazioni, il numero totale di moltiplicazioni è uguale a M * Dp * Dp * (num. di filtri).

Quindi per la point wise convolution operation il numero totale di moltiplicazioni è M x Dp2 x N.

Sommando le due operazioni separate per definire la Depthweise Separable Convolution avremo che Totale moltiplicazioni = M * Dk2 * Dp2 + M * Dp2 * N = M * Dp2 * (Dk2 + n)
Ovvero: M * Dp2 * (Dk2 + n)

Riassumendo in figura sotto possiamo schematizzare le diverse operazioni di convoluzioni che poi vengono utilizzate per creare la Depthweise Separable Convolution.

$$\mathrm{Conv}(W,y)_{(i,j)} = \sum_{k,l,m}^{K,L,M} W_{(k,l,m)} \cdot y_{(i+k,j+l,m)}$$

$$\mathrm{PointwiseConv}(W,y)_{(i,j)} = \sum_{m}^{M} W_m \cdot y_{(i,j,m)}$$

$$\mathrm{DepthwiseConv}(W,y)_{(i,j)} = \sum_{k,l}^{K,L} W_{(k,l)} \odot y_{(i+k,j+l)}$$

$$\mathrm{SepConv}(W_p,W_d,y)_{(i,j)} = \mathrm{PointwiseConv}_{(i,j)}(W_p, \mathrm{DepthwiseConv}_{(i,j)}(W_d,y))$$

Figura 1.13: ripi di convoluzione [ATU18]

In conclusione , le DepthWise Convolution sono ideali quando si ottimizza il modello per ottenere dimensioni ridotte o una maggiore velocità, compromettendo in minima parte la precisione. Non è consigliabile usarle quando si ottimizza il modello unicamente per la precisione. Molti algoritmi di deep learning sono costruiti utilizzando le convoluzioni separabili come saranno citate nel sottoparagrafo successivo.

1.4 Algoritmi di Ottimizzazione

Nell'ambito dell'apprendimento profondo esiste il concetto di Loss Function, che ci dice quanto il modello stia funzionando male in quell'istante.

Questa funzione verrà utilizzata per addestrare la rete, sarà necessario prendere la perdita e cercare di ridurla al minimo, più bassa è più significa che il nostro modello funzionerà meglio. Il processo di minimizzazione (o massimizzazione) di qualsiasi espressione matematica si chiama ottimizzazione.
Gli ottimizzatori sono algoritmi o metodi utilizzati per modificare gli attributi della rete neurale, come i pesi e il tasso di apprendimento, per ridurre le perdite. Gli ottimizzatori vengono utilizzati per risolvere i problemi di ottimizzazione minimizzando la Loss function.

Gradient Descent

La discesa del gradiente è un potente algoritmo di ottimizzazione utilizzato per minimizzare la funzione di perdita in un modello di apprendimento automatico. È una scelta popolare per una varietà di algoritmi, tra cui la regressione lineare, la regressione logistica e le reti neurali.

Sara utilizzato questo concetto per capire meglio cos'è, come funziona e le diverse varianti dell'algoritmo, progettate per affrontare sfide diverse e fornire ottimizzazioni per casi d'uso diversi.

La discesa del gradiente è un algoritmo di ottimizzazione iterativa utilizzato per minimizzare la funzione di costo di un modello di apprendimento automatico. L'idea è di muoversi nella direzione della discesa più ripida della funzione di costo per raggiungere il minimo globale o un minimo locale. Ecco le fasi dell'algoritmo di discesa del gradiente (Fig.1.14):

1. Inizializzare i parametri del modello con valori casuali.
2. Calcolare il gradiente della funzione di costo rispetto a ciascun parametro.
3. Aggiornare i parametri sottraendo una frazione del gradiente da ciascun parametro. Questa frazione è chiamata tasso di apprendimento, che determina la dimensione del passo dell'algoritmo.

Si dovranno ripetere le fasi 2 e 3 fino alla convergenza, che si ottiene quando la funzione di costo smette di migliorare o raggiunge una soglia predeterminata.

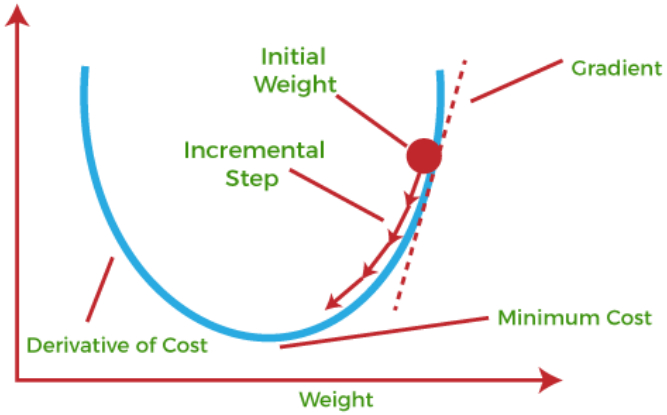

Figura 1.14: gradient descent [JAV19]

Esistono diverse varianti dell'algoritmo di discesa del gradiente, che si differenziano per il modo in cui calcolano gli aggiornamenti dei parametri:

Batch Gradient Descent: In questa variante, l'intero set di dati di allenamento viene utilizzato per calcolare il gradiente e aggiornare i parametri. Questa procedura può essere lenta per i dataset di grandi dimensioni.

Stochastic Gradient Descent (SGD): In questa variante, viene utilizzato un solo esempio di addestramento casuale per calcolare il gradiente e aggiornare i parametri. Può essere più veloce del Batch Gradient Descent, ma gli aggiornamenti possono essere rumorosi.

Mini-Batch Gradient Descent: In questa variante, un piccolo sottoinsieme del set di dati di addestramento viene utilizzato per calcolare il gradiente e aggiornare i parametri. Si tratta di un compromesso tra Batch Gradient Descent e SGD, in quanto è più veloce di Batch Gradient Descent e meno rumoroso di SGD.

Discesa del gradiente basata sul momento: In questa variante, gli aggiornamenti dei parametri si basano sul gradiente corrente e sugli aggiornamenti precedenti. Questo aiuta l'algoritmo a superare i minimi locali e ad accelerare la convergenza.

Adagrad: In questa variante, il tasso di apprendimento viene scalato in modo dinamico per ogni parametro in base alle informazioni storiche sul gradiente. Ciò consente di ottenere aggiornamenti maggiori per i parametri poco frequenti e aggiornamenti minori per i parametri frequenti.

RMSprop: In questa variante, il tasso di apprendimento viene scalato in modo adattivo per ogni parametro in base alla media mobile del gradiente quadratico. Questo aiuta l'algoritmo a convergere più velocemente in presenza di gradienti rumorosi.

Adam: in questa variante, che sarà quella utilizzata poi anche nel nostro caso, il tasso di apprendimento viene scalato in modo adattivo per ogni parametro in base alla media mobile del gradiente e del gradiente quadratico. Combina i vantaggi di Momentum-based Gradient Descent, Adagrad e RMSprop ed è uno degli algoritmi di ottimizzazione più popolari per il deep learning.

1.5 MobileNet1000

La rete MobileNetwork (Fig.1.15) è una rete neurale profonda molto leggera proposta da Google per un uso per smartphone ma più genericamente per dispositivi embedded a basso consumo. La sua caratteristica principale è che usa una Depthwise Separable Convolution invece che una normale convoluzione, riducendo così la quantità di calcoli e migliorando l'efficienza computazionale della rete.
Questa rete è stata pensata per classificare 1000 classi ma puo essere impegata per svolgere ulteriori compiti, è possibile ampliarla o estrarne anche solo una parte di layer a seconda dell'obbiettivo.

Comparando i risultati ottenuti sullo Standford DataSet, il modello a rete neurale convoluzionale Inception V3 ottiene un'accuratezza dell'84%, il MobileNet più grande ottiene un'accuratezza dell'83,3%. Ma se guardiamo al numero di parametri di ciascuna architettura del modello, mentre Inception V3 ha 23,2 milioni di parametri, MobileNet ha solo 3,3 milioni di parametri. Inoltre, è possibile realizzare versioni MobileNet ancora più piccole e veloci semplicemente utilizzando un moltiplicatore. Per questo motivo è un modello molto utilizzato su dispositivi mobili e integrati.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figura 1.15: struttura MobileNet1000 [AND17]

Sebbene l'architettura MobileNet di base sia già piccola e computazionalmente poco costosa, ha due diversi iperparametri globali per ridurre ulteriormente il costo computazionale.
Uno è il moltiplicatore di larghezza e l'altro è il moltiplicatore di risoluzione.

Moltiplicatore di larghezza:
Per ridurre ulteriormente il costo computazionale, hanno introdotto un semplice parametro chiamato moltiplicatore di larghezza, noto anche come α.
Per ogni strato, il moltiplicatore di larghezza α viene moltiplicato per i canali di ingresso e di uscita (N e M) al fine di restringere la rete.

Quindi il costo computazionale con il moltiplicatore di larghezza diventa.

$$D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F$$

Qui α varia da 0 a 1, con valori tipici di [1, 0,75, 0,5 e 0,25]. Quando α = 1, si chiama MobileNet di base e α < 1, si chiama MobileNet ridotto. Il moltiplicatore di larghezza ha l'effetto di ridurre il costo computazionale di α^2.

Moltiplicatore di risoluzione

Il secondo parametro per ridurre efficacemente il costo computazionale è noto anche come ρ.

Per un dato livello, il moltiplicatore di risoluzione ρ viene moltiplicato per la mappa di caratteristiche in ingresso. Ora possiamo esprimere il costo computazionale applicando il moltiplicatore di larghezza e il moltiplicatore di risoluzione come:

$$D_K \cdot D_K \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F$$

Facendo un confronto con altri modelli simili al MobileNet come GoogleNet e VGG 16, si può notare che MobileNet dal punto di vista del numero di parametri e precisione ha il predominio sugli altri (Fig.1.16).

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Figura 1.16: comparazione con altri modelli[AND17]

L'immagine seguente mostra la differenza tra il modello separabile in profondità e il modello di convoluzione standard (Fig.1.17).

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

Figura 1.17: differenze con full convolution[AND17]

È evidente che l'accuratezza di ImageNet è inferiore di appena lo 0,1% rispetto al modello di convoluzione standard, ma con un numero molto inferiore di aggiunte e parametri.

2-Architettura Risc-V

2.1 Panoramica Risc/Cisc

In generale, i processori possono essere classificati in base alla loro ISA (Instruction Set Architecture), ovvero un modello astratto che definisce le istruzioni supportate ed i modi di operare del processore.
L'ISA può essere considerata come un'interfaccia tra il processore ed il software: un software realizzato per un determinato ISA, potrà essere eseguito da un qualsiasi processore che rispetta tale ISA. Nel caso invece, lo stesso software, dovesse essere eseguito da processori con ISA diverso, sarebbe necessario convertirlo per la differente architettura. Per risolvere questo problema, solitamente il software viene scritto con linguaggi di più alto livello, e, attraverso un processo di compilazione viene trasformato in un file binario eseguibile.
Nel corso degli anni sono state definite innumerevoli ISA, le più popolari sono classificabili in due grandi categorie:

CISC:
CISC è l'acronimo di Complex Instruction Set Computer e dispone di un'ampia collezione di istruzioni che vanno da semplici a molto complesse e specializzate a livello di linguaggio assembly, che richiedono molto tempo per esser eseguite. L'approccio CISC cerca di minimizzare il numero di istruzioni per programma, ma al costo di aumentare il numero di cicli per istruzione.
L'intento è stato quello di creare istruzioni complesse direttamente nell'hardware, poichè il livello hardware è sempre più veloce rispetto a logiche di più alto livello. Tuttavia, i chip CISC sono relativamente più lenti rispetto ai chip RISC anche se utilizzano poche istruzioni. Esempi di processori ISA sono VAX, AMD, Intel x86 e System/360.

RISC:
Una Cpu basata su RISC ha un insieme ridotto di istruzioni, sono state pensate per eseguire operazioni semplici e richiedono un solo ciclo di clock per esser completate.
L'architettura Risc esegue le operazioni sui dati immagazzinati nei registri.

Il fatto che siano utilizzati i registri per le operazioni sui dati è un notevole vantaggio a livello di performance, infatti questi sono molto più vicini alla CPU piuttosto che alla memoria principale.
Strutture e metodi di pipeline sono molto efficaci su queste architetture e permettono l'esecuzione di più istruzioni contemporaneamente.
Proprio questo è un punto di forza importante di questa architettura.

Altre tipologie di architetture che sono meno popolari sono riportate in seguito:

- **Minimal instruction set computers (MISC)**
  Architettura per microprocessori basata su un numero minimale di istruzioni. A volte è inteso come Mono Instruction Set Computer, cioè un processore che ha un'unica istruzione, che però è solitamente noto come "one instruction set computer".

- **Explicitly parallel instruction computing (EPIC)**
  L'obbiettivo dei EPIC è lo sviluppo di processori in grado di eseguire nativamente e in modo efficace codice parallelo senza dover utilizzare complesse strutture hardware sul processore ma invece demandando la maggior parte dei problemi di parallelizzazione al compilatore.
  Questo permette un miglioramento delle prestazioni senza innalzare eccessivamente la frequenza di funzionamento dei processori evitando consumi eccessivi e problemi di dissipazione.

- **Very long instruction word (VLIW)**
  Queste sono basate sul parallelismo intrinseco presente nelle istruzioni, queste CPU sono dotate di più unità di calcolo indipendenti (es. moltiplicatori) per dare la possibilità alla CPU di eseguire più calcoli contemporaneamente, come per esempio moltiplicazioni.
  L'idea è di evitare di sovraccaricare il processore togliendogli il compito di ottimizzare al massimo il parallelismo delle istruzioni per affidarlo al compilatore.

- **Zero instruction set computer (ZISC)**
  Questa è un'architettura e una tecnologia di compilazione per la progettazione di processori e acceleratori hardware personalizzati altamente efficienti, consente

al compilatore di avere un controllo a basso livello delle risorse hardware, questa architettura è basata su due idee fondamentali: Corrispondenza di pattern e assenza di microistruzioni.

Tenendo in considerazione l'evoluzione storica di queste due architetture possiamo notare che i computer degli anni cinquanta erano RISC, a partire dagli anni sessanta crebbero come CISC, e macchine di qualunque classe: mainframe, mini, personal, etc.. si arricchirono di centinaia di istruzioni hardware. Negli anni ottanta si cominciò a valutare la loro effettiva utilità e si scoprì che moltissime istruzioni non venivano mai utilizzate. Perciò negli anni novanta iniziarono a ridiffondersi i RISC, più compatti e adatti a semplici scopi.
Attualmente la tecnologia è fortemente orientata verso soluzioni RISC, questo per via dell'esigenza odierna nell'utilizzo sempre più crescente di dispositivi embedded e riduzione dei consumi e costi.

2.2 Risc-V e architettura

RISC-V è un ISA relativamente recente e viene attivamente promossa come concorrente di molte architetture che richiedono il pagamento di una licenza.

RISC sta per "Reduced Instruction Set Computer" (già discusso nel sottoparagrafo precedente), mentre "V" è il numero romano 5. Perciò RISC-V è la quinta generazione di una famiglia di core per computer e si pronuncia "Risk Five". A differenza della maggior parte dei progetti ISA, RISC-V viene fornito gratuitamente con una licenza open-source.

A lvello di storia sappiamo che Intel ha sviluppato l'architettura x86/x64 negli anni '70 e da allora è il cavallo di battaglia del settore, mentre l'architettura ARM è stata sviluppata alla fine degli anni '80 inizialmente per l'uso su sistemi a basso consumo.
ARM è l'architettura RISC più conosciuta al mondo ed è l'architettura CPU dominante per microcontrollori, microprocessori e sistemi mobili. Attualmente sta aumentando la sua presenza anche nei sistemi HPC.
.

Mentre x86/x64 può guardare ARM e RISC-V da una posizione inifluente , ARM invece potrebbe sentire la competizione di RISC-V.

Dalla sua nascita nel 2010, RISC-V è stata ben accolta dall'industria dei microprocessori e la sua adozione è cresciuta costantemente sia nell'hardware che nel software. Nella seguente immagine possiamo vedere un grafico (Fig.2.1):

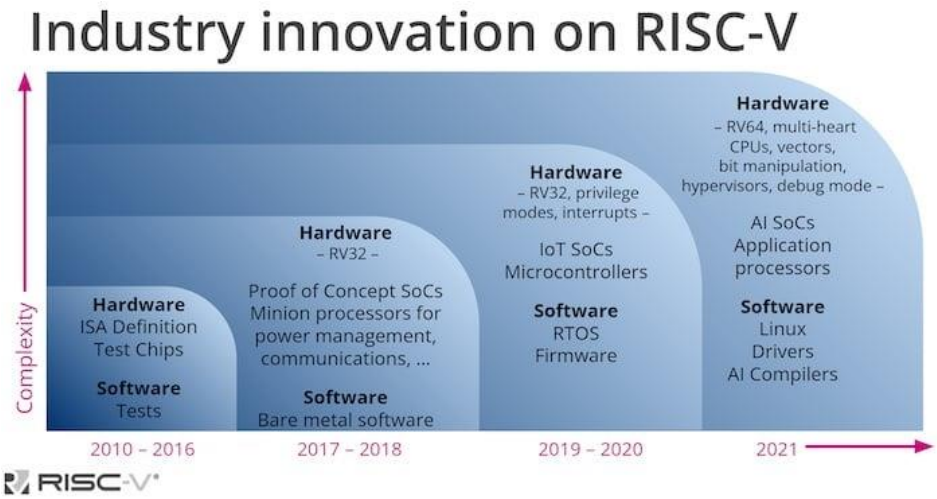

Figura 2.1: evoluzione Risc-V [EDU22]

La base dell'architettura RISC-V comune a tutte le implementazioni è minimale, in quanto contiene soltanto 47 istruzioni, rispetto alle oltre 1500 di un sistema x86 e alle oltre 500 dell'architettura Arm. Utilizzando un semplice meccanismo di load/store, tutte le operazioni sono eseguite sui registri interni, mentre esistono istruzioni dedicate per eseguire il trasferimento tra registri e memoria. Queste scelte progettuali riflettono la natura originale di RISC-V, ovvero la sua scalabilità e adattabilità (tramite varianti ed estensioni) a diversi tipi di applicazioni.

SiFive, uno tra i principali produttori di sistemi RISC-V con oltre un milione di core venduti, ha recentemente annunciato un'implementazione dell'architettura RV32E che utilizza soltanto 13.500 gate. In modo del tutto analogo, IZERO-RISCY, un core sviluppato come parte della piattaforma PULP per il calcolo ad alta efficienza energetica, ha un'implementazione di pipeline a due stadi che impiega 11.600 gate. La specifica RISC-V consente inoltre di definire microcontrollori semplici che non richiedono molta logica attorno al core.

Un altro aspetto dell'architettura RISC-V molto importante per i sistemi embedded riguarda la sicurezza. La protezione dagli attacchi perpetrati via rete è una funzionalità non sempre disponibile su ogni tipo di core, specialmente se si tratta di applicazioni IoT. Sotto questo punto di vista, RISC-V è unico, in quanto include nativamente meccanismi per la sicurezza informatica, come il blocco del buffer overflow e protezione nei confronti degli attacchi via rete che sfruttano le vulnerabilità del codice. Aggiungendo delle estensioni specifiche a un core RISC-V, come filtri FIR (Finite Impulse Response), calcolo del CRC32 (controllo di ridondanza ciclica a 32 bit) e algoritmi 3DES (Triple Data Encryption Standard), le funzionalità per la sicurezza possono essere notevolmente aumentate.

Grazie al suo approccio open source, RISC-V offre una flessibilità quasi illimitata, essendo un'architettura priva di barriere o restrizioni. I creatori del progetto hanno raggiunto questo obbiettivo attraverso il concetto basilare di estensione. Ciò significa che l'ISA, l'elemento costitutivo più piccolo, può essere esteso per soddisfare i requisiti di ogni implementazione hardware, piccola o grande che sia. È così possibile impiegare la medesima architettura hardware in applicazioni anche molto differenti tra loro, come PC industriali, dispositivi indossabili, sistemi embedded, nodi IoT, applicazioni automotive o persino aerospaziali.

RISC-V non è la prima architettura aperta e libera per processori, ma è stata la più riuscita fino ad oggi. Data la sua eredità, la flessibilità, l'approccio aperto, l'interesse del mondo accademico e l'ampio sostegno dell'industria, è una tecnologia che prenderà sempre più piede nella futura generazione.

3-Analisi e scelta dell'hardware

3.1 Fase di analisi dei requisiti

In questa fase si è proceduto ad analizzare le tecnologie disponibili sul mercato che potessero soddisfare i requisiti computazionali per questa attività.
Sono state individuate delle componenti fondamentali per poter creare il nostro sistema da zero e verranno elencate in basso:

- Soc basata su Architettura RISC
- Modulo camera a basso consumo con capacità di calcolo adeguate
- Acceleratore per reti neurali a supporto della CPU(HWCE)
- Tool di supporto allo sviluppo e SDK (toolchain, Tensorflow..)
- Possibilità di utilizzare implementazioni non proprietarie
- Costo contenuto

Come architettura di riferimento ci focalizzeremo su RiscV, dopo un attenta fase di ricerca sono state messe a confronto le principali schede in commercio.
Molte delle schede analizzate sono state prese dal sito ufficiale RiscV International.
Il risultato della comparazione è descritto dalla tabella mostrata in seguito (Fig.3.1):

	Sipeed Maixduino Kit for RISC-V AI + IoT	SparkFun RED-V RedBoard and RED-V Thing using SiFive RISC-V FE310 SoC	GAPuino Development Boar(	LoFive RISC-V SoC Evaluation Kit	Kendryte K210 SoC, KD233 Development Board	Maix Bit AI Development Board RISC-V K210 IoT
Specifiche	(https://www.distrelec.it/Web/Downloads/ t/ds/11	https://www.elektor.com/amfile/file/download/file/2359/product	https://gwt-website-files.s3.eu-central-1.amazonaws.com/GA	https://groupgets.com/manufacturers/qwertv-e	https://media.digikey.com/pdf/Data%20Sheets/GroupGets%	https://raw.githubusercontent.com/Mav-DFRobot/DFRobot/master/D
PREZZO	~ 35 €	~ 35 €	100,00€	35(NA)	$49.99	15-31€
requenza di clock	400MHz-500MHz	256MHz(default) 320MHz(Max)	175 MHz (Cluster) internal clock, Up to 250 MHz	320 MHz	320Mhz	400Mhz up to 800Mhz
numero di core	2	1	MultiCore (8/1)	1	2	2
numero di bit	64	32	32	32	64	64
set di istruzioni	RISC	RISC	RISC	RISC	RISC	RISC
Processore ReteNeurale	FPU, FFT accelerator, audio accelerator (APU), capable CNN neural network accelerator (KPU), and	FPU,Hardware accelerator ,Hardware Multiply/Divide, Flexible Clock Generation with on-chip oscillators and PLLs	FPU,ardware accelerator engine(HWCE) Dynamic & Voltage Frequency Supply control of controller and cluster domains	FPU,Hardware multiply and divide, Flexible clocking options including internal PLL, free-running ring	KPU Convolutional Neural Network (CNN) hardware accelerator, APU audio hardware accelerator	ardware accelerator (KPU),FPU,Neural network accelerator
Memoria	16MiB Flash, support micro SDXC expansion storage (max 128GB)	16 KB Instruction Cache, 16 KB Data Scratchpad	Quad SPI Flash 256 Mbits, HyperBus combo DRAM/Flash 512 Mbits Flash + 64 Mbits DRAM	8 kB OTP program memory 8 kB mask ROM 16 kB instruction cache 16 kB data SRAM	128Mb of SPI NOR Flash and a micro SD card slot for storage	6MB general purpose +2MB for AI,Flash 16MB
Operating Voltage	6-12V (support 4.8V ~ 5.2V)	3.3 V and 1.8 V	3.3V/5V	5V via pin 1 on header; Operating Voltage: 3.3 V	1.8V/3.3V	from 0.8V~1.2V
Power Input	USB Type-C, DC connector 6-12V input;Provide 5V 1.2A output	5 V USB or 7-15 VDC Jack	USB port, DC power supply (7V-15V)	1.8 V and 3.3 V	5V via USB type-C	4.8 to 5.2 V
SVILUPPO	support TensorFlow Lite,FreeRtos & Standard SDK ,MicroPython	SiFive Freedom E SDK,debug module The RED-V requires Freedom	fully automated toolchain from Tensorflow, GAP SDK	SDK LoFive Freedom-E	TensorFlow/Keras/Darknet,standalone SDKs	standalone SDK, FreeRTOS SDK base on C/C++,UART and JTAG interface

Figura 3.1: comparazione schede Risc-V

Analizzando bene le caratteristiche di queste schede possiamo notare che tutte soddisfano i requisiti richiesti, per questo progetto verrà considerato come fattore significativo la disponibilità fisica, il prezzo, la semplicità di utilizzo e la documentazione associata.

Ad oggi queste schede non hanno una domanda cosi elevata ed è difficile trovarle sul mercato nazionale.

La scelta è ricaduta su MaixDuino, la motivazione è che oltre ad avere un ottima KPU(K210) con un consumo energetico < 1 W è dotata di una community attiva, è possibile utlizzare diversi tipi di tool a riga di comando o visuali come Arduino IDE,VBC, MaixPy IDE etc.. ed è provvista di Kit con sensore camera incluso e idoneo al progetto, la potenza è adeguata avendo 2 core da 64 bit.

Una possibile altra scelta interessante sarebbe stata Gapuino ma in questo contesto si è scelto di virare su una tecnologia più semplice visto che la gestione del multicore su quella scheda risulta più complicata.

In basso viene raffigurato uno schema delle componenti della scheda (Fig. 3.2):

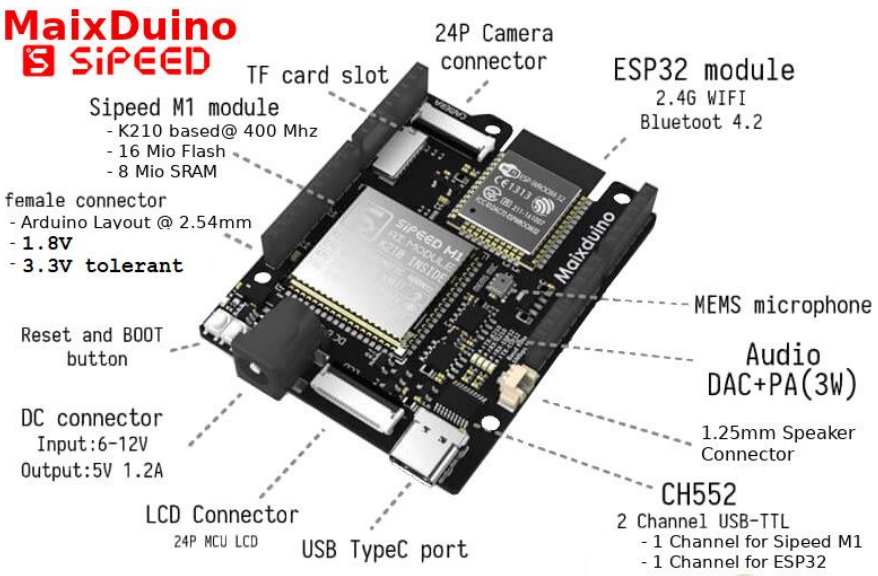

Figura 3.2: componenti Maixduino [SIP19]

Maixduino è dotata di MCU(Micro-Controller Unit) Kendryte K210 M1.

È dotata di un acceleratore hardware di reti neurali (KPU) in grado di eseguire operazioni di rete neurale convoluzionale ad alte prestazioni.

Questa scheda è tra le più popolari della lista e permette di avere tutte le risorse di nostro interesse.

3.2 Camera CMOS utilizzata

All'interno del kit della scheda Maixduino è compreso un modulo sensore CMOS, il modulo telecamera OV2640 - 2,0 MP, basato sul sensore a colori OV2640 di OmniVision. Il sensore ha un formato ottico da 1/4,0″ che, combinato con la dimensione dei pixel di 2,2 µm x 2,2 µm, produce un'immagine da 2,0 MP 1600 x 1200 UXGA. Si tratta di un sensore rolling shutter, che cattura un'intera colonna (o riga) simultaneamente e compila l'immagine con una scansione trasversale (o verso il basso). Rilasciato nel 2005 e dismesso nel 2009, è ancora oggi molto utilizzato. Sebbene questo sensore d'immagine rimanga popolare, non mancano le criticità e difetti.

In basso un immagine raffigurante il sensore (Fig.3.3):

Figura 3.3: sensore OV2640 [OMN06]

Rispetto alle più recenti fotocamere, l'OV2640 non è affatto sofisticato in termini di formato del sensore, risoluzione e qualità dell'immagine.
Ma rimane comunque un buon sensore d'immagine.

L'OV2640 è dotato di un ISP on-chip in grado di eseguire l'esposizione automatica e il bilanciamento automatico del bianco.

L'interfaccia del sensore è DVP (Fig 3.4), standard per la porta video digitale, una sorta di interfaccia parallela di sincronizzazione della telecamera con dati a 8 bit, segnale di sincronizzazione orizzontale e verticale e un Time pixel associato.

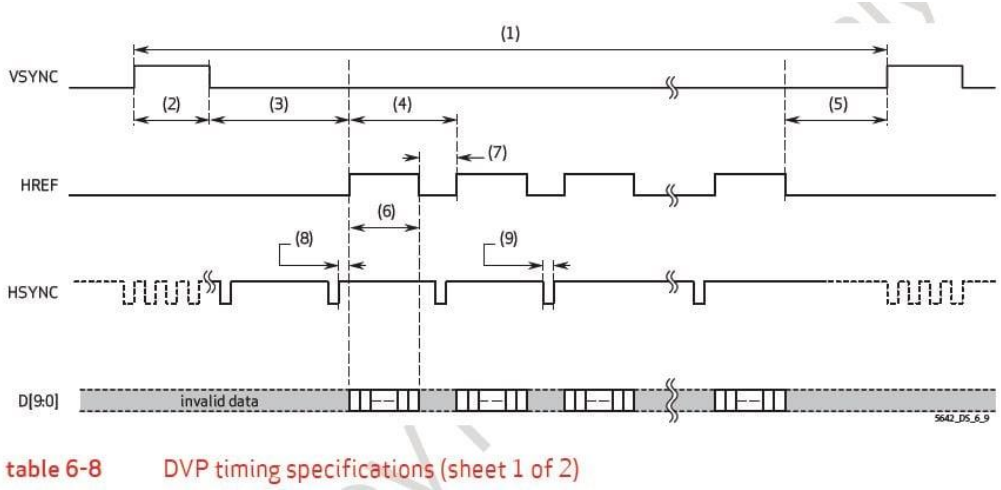

Figura 3.4: diagramma temporale DVP [OMN06]

È molto adatto alla maggior parte dei microcontrollori ARM e RISC di fascia bassa, come STM32 e ESP32, mentre l'interfaccia MIPI è supportata solo dai processori di fascia alta.

La caratteristica più importante di OV2640 è l'encoder JPEG hardware, che scarica la potenza di elaborazione e riduce l'utilizzo della memoria da parte dei microcontrollori. Una tipica immagine in formato RGB565/YUV 1600×1200 occupa 3,66 MB di spazio nella RAM, mentre un'immagine in formato JPEG di pari qualità occupa solo 150 KB, con un rapporto di compressione quasi x25.

Con una risoluzione inferiore o un rapporto di compressione più elevato, le dimensioni dell'immagine JPEG in uscita saranno ancora più ridotte e potranno essere facilmente archiviate ed elaborate nella memoria RAM interna del microcontrollore.

Colori Digitali

Le immagini digitali sono costituite da pixel e se vogliamo memorizzare o trasmettere un'immagine, dobbiamo fornire informazioni su ogni singolo pixel. Per le immagini monocromatiche si tratta semplicemente di un singolo bit di informazione, dove un esempio importante sono le immagini sui display LCD come quelli dei vecchi GameBoy. Ciò tuttavia comporta la limitazione che possiamo solo indicare se un pixel è ON o OFF (ad esempio bianco o nero), ma nient'altro. Se vogliamo avere immagini colorate, abbiamo bisogno anche di informazioni sul colore. La profondità del colore definisce quanti bit vengono utilizzati per rappresentare un colore. Più bit

(informazioni) ci sono, più colori possiamo visualizzare. Esistono diversi modi per rappresentare un colore, ma praticamente tutti i display utilizzano RGB, dove un colore è rappresentato dai suoi componenti Rosso Verde e Blu.

RGB888

Una rappresentazione comune delle informazioni sui colori è il formato RGB888 (24 bit/3 byte). Definisce 8 bit/1 byte di informazioni per ogni colore primario (rosso, verde, blu) che, se sommati, danno come risultato il colore desiderato. Di solito il codice colore è rappresentato in cifre esadecimali (poiché 1 byte corrisponde esattamente a 2 cifre).

#ffff00 0xffff00

Nella tabella sottostante si può vedere com'è memorizzato (Fig.3.5).

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
r7	r6	r5	r4	r3	r2	r1	r0	g7	g6	g5	g4	g3	g2	g1	g0	b7	b6	b5	b4	b3	b2	b1	b0

Figura 3.5: schema RGB 24bit [BAR20]

Nella prima riga viene indicato il numero del bit e nella seconda riga il colore che rappresenta quel bit e quanto è "significativo". I bit a sinistra sono più significativi e quindi hanno un impatto maggiore sul colore.

RGB565

Soprattutto gli schermi (economici) utilizzati con dispositivi embedded non forniscono una profondità di colore a 24 bit. Inoltre, la memorizzazione e/o la trasmissione di 3 byte per pixel consuma molta memoria e crea latenza. RGB565 richiede solo 16 (5+6+5) bit/2 byte ed è comunemente utilizzato con schermi integrati.

Fornisce 5 bit per il Rosso e il Blu e 6 bit per il Verde.

Fornire 5 bit per 2 colori e 6 bit per un altro sembra asimmetrico, ma archiviare e trasmettere qualcosa che non può essere ben impacchettato in byte sarebbe complicato, infatti 16bit sono necessari per una gestione corretta. Si può notare però che poiché abbiamo meno bit (informazioni) disponibili, possiamo rappresentare meno colori. Mentre RGB888 fornisce 2^24=16 777 216 colori, RGB565 fornisce solo 2^16=65 536

colori. Il motivo per cui il verde ha più informazioni è che gli occhi umani possono distinguere meglio le sfumature di verde rispetto ad es. sfumature blu (Fig.3.6).

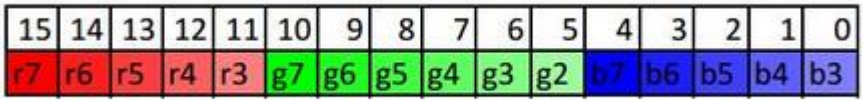

Figura 3.6: schema RGB 16bit [BAR20]

In conclusione, l'interfaccia intuitiva della telecamera, la qualità dell'immagine, la genstione a consumo ridotto e il potente ISP on-chip con codifica JPEG rendono il sensore di immagine OV2640 popolare per le applicazioni di telecamere IoT.

3.3 ST7789V-LCD

L'ST7789V (Fig.3.7) è un controller/driver a chip singolo per TFT-LCD di tipo grafico a 262K colori. Questo chip è in grado di collegarsi direttamente a un microprocessore esterno e accetta un'interfaccia parallela a 8 bit/9 bit/16 bit/18 bit. I dati del display possono essere memorizzati nella RAM di 240x320x18 bit. È in grado di eseguire operazioni di lettura/scrittura sulla RAM senza un clock operativo esterno per ridurre al minimo il consumo energetico. Inoltre, grazie al circuito di alimentazione integrato necessario per pilotare i cristalli liquidi, è possibile realizzare un sistema di visualizzazione con il minor numero di componenti.

Figura 3.7: sensore ST7789V [SIT13]

Riassumeremo ora alcune delle caratteristiche.

- risoluzione del Display: 240*RGB(H)*320(V)
- Grandezza memoria frame :240x320x18-bit=1,382,400 bits
- Circuiti di uscita del driver LCD:
  Output sorgente: 240 Canali RGB
  Output del gate: 320 Canali
- Colori del Display(Color Mode):
  Full Color: 262K, RGB=(666) massimo, modalità Idle off
  Riduzione del colore: 8-colori, RGB=(111), modalità Idle ON
- Formato colore dei pixel programmabile (profondità colore) per vari formati di ingresso dei dati del display
  -12-bit/pixel: RGB= (444)
  - 16-bit/pixel: RGB= (565)
  - 18-bit/pixel: RGB= (666)
- Interfaccia MCU
  Interfaccia MCU parallela 8080-series (8-bit,9-bit, 16-bit & 18-bit
  Interfaccia RGB 7/16/18 (VSYNC,HSYNC,DOTCLK,ENABLE,DB[17:0])

Serial Peripheral Interface (Interfaccia SPI)

Interfaccia VSYNC

- Caratteristiche Display

  CABC per il risparmio del consumo di corrente

  Miglioramento del colore

- Circuiti integrati su chip

  Convertitore DC/DC

  Generazione VCOM regolabile

  Memoria non volatile (NV) per memorizzare l'impostazione del registro iniziale e valore di fabbrica di Default(ID modulo,Versione del modulo, ecc.)

  Controllo del timing

  4 curve gamma preimpostate con impostazione separata della gamma RGB

- Memoria NV integrata per l'impostazione iniziale del registro LCD

  8-bits per impostazione ID1

  8-bits per impostazione ID2

  8-bits per impostazione ID3

  6-bits per la regolazione offset VCOM

- Sistema di alimentazione su chip

  voltaggio sorgente (VAP(GVDD) to VAN(GVCL)): +6.4~-4.6V

  livello VCOM: GND

  Gate driver livello ALTO (VGH - AGND): +12.2V ~ +14.97V

  Gate driver livello BASSO(VGL to AGND): -12.5V ~ -7.16V

  Intervallo di tensione regolabile per la compensazione del feed through: 0,1V~1,675V

- Layout ottimizzato per l'assemblaggio COG
- Temperatura in cui può operare : da -30°C a +85°C
- Consumo di energia ridotto

3.4 Kendryte K210 e KPU

Il modulo di cui è composto Maixduino è Sipeed M1 ed è il primo modulo RISC-V 64 AI basato sul sistema KPU K210.

Kendryte K210 è un “system on a chip” che fornisce un accelleratore per reti neurali e altre funzioni per eseguire tecniche di visione artificiale su piattarforma Risc.

La parte principale risiende nella KPU e nella APU.

La KPU è un processore di rete neurale general purpose, in grado di realizzare calcoli di rete neurale convoluzionale a basso consumo energetico, di ottenere le dimensioni, le coordinate e i tipi di oggetti rilevati di volta in volta e di rilevare e classificare volti o oggetti.
KPU ha le seguenti caratteristiche:

- Non esiste un limite diretto al numero di livelli di rete e ogni livello di parametri di rete neurale convoluzionale può essere configurato separatamente, incluso il numero di canali di input e output, larghezza di riga di input e output e altezza di colonna
- Supporta due kernel di convoluzione 1x1 e 3x3
- Supporta qualsiasi funzione di attivazione
- Quando si lavora in tempo reale, la dimensione massima dei parametri della rete neurale supportata va da 5,5 MiB a 5,9 MiB

Kendryte K210 (Fig. 3.8) include due core di CPU RISC-V a 64 bit, ciascuno con FPU integrata e indipendente, dotata di un acceleratore per la trasformata di Fourier veloce (FFT), integra gli acceleratori degli algoritmi AES e SHA256 per fornire agli utenti le funzioni di sicurezza di base, provvista di SRAM ad alte prestazioni e basso consumo e di un potente DMA per un throughput di dati superiore.
Dispone di un'ampia gamma di unità periferiche: DVP, JTAG, OTP, FPIOA, GPIO, UART, SPI, RTC, I2S, I2C, WDT, Timer e PWM, per un gran numero di scenari applicativi.

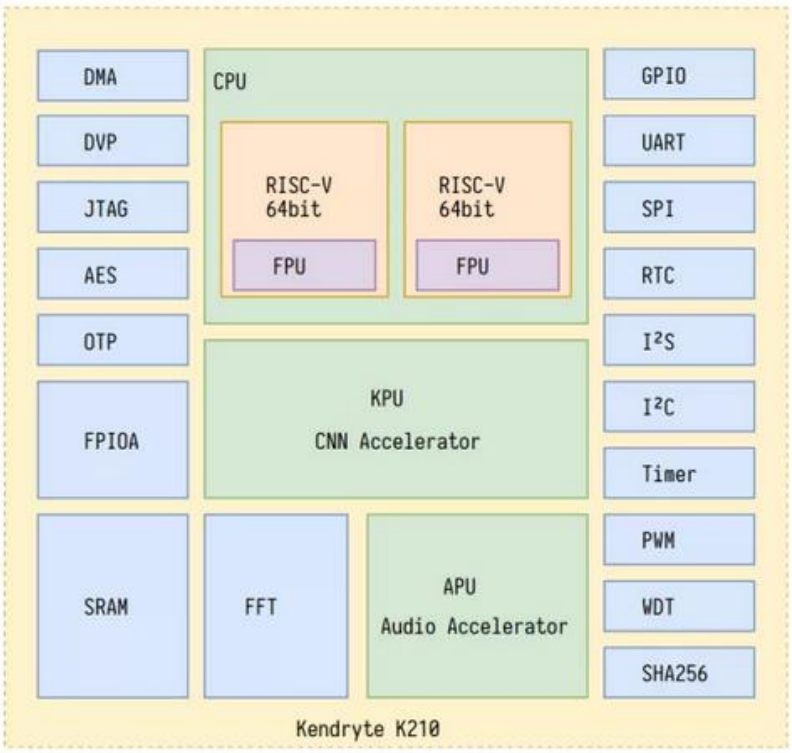

Figura 3.8: Kendryte K210[MIC20]

3.5 Linguaggi e piattaforme utilizzate

Maixduino ha la possibilità di esser programamta con il linguaggio Micropython, un derivato del python con moduli ridotti proprio per essere adattati all'architettura sosttostante , un'altra possibilità è utilizzare il linguaggio C/C++.

Per questo progetto è importante fare anche analisi su quale sia il linguaggio più adatto e del perché utilizzarlo, tenendo in considerazione anche del livello di difficoltà che comporta.

Forniremo ora una breve comparazione tra le due tipologie (Fig.3.9):

Parametri di comparazione	**C++**	**Python**
Compilazione	Compilato	Interpretato
Utilizzo	Non semplice da scrivere	facile e intuibile da scrivere

Natura del linguaggio	Staticamente tipizzato	Dinamicamente tipizzato
Portabilità	Non portabile	Portabile
Garbage collection	Non supporta la Garbage Collection.	Supporta la Garbage Collection.
Tipi di dati	I tipi di dati legati ai nomi vengono controllati in fase di compilazione.	Legati a valori, controllati in fase di esecuzione.
Funzioni	Restrizioni sul tipo di parametri o di valore di ritorno.	Nessuna restrizione sul tipo di parametri o sul valore di ritorno.
Efficienza	Difficile da mantenere.	Facile da mantenere.
Velocità di esecuzione	Veloce	Lento
Performance	Alte performance	Basse performance
Popolarità	Più popolare per le applicazioni embedded o aziendali.	Il più popolare per l'apprendimento automatico.
Semplicità e usabilità	Linguaggio non semplice	Semplice e utilizzato per l'apprendimento automatico o per applicazioni web.

Figura 3.9: comparazione dei linguaggi

La scelta in base alle caratteristiche principali elencate è ricaduta a favore del C/C++, ci affideremo a un linguaggio di più basso livello optando per la velocità esecutiva e consumo ridotto.
Tutto questo sacrificando la parte di difficoltà implementativa e la facilità di scrittura.

Per il tool di sviluppo abbiamo diverse possibilità come PlatformIO IDE (VSCode), Arduino IDE, Kendrite, linea di comando e altre ancora ma per questo progetto sarà utilizzato Arduino IDE per via della semplicità e popolarità.

Arduino IDE

Arduino IDE è derivato dall'IDE creato per il linguaggio di programmazione Processing e per il progetto Wiring. È stata creata da Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino, e David Mellis.
È multipiattaforma e basata su Electron, include un editor di testo dotato di syntax highlighting, controllo delle parentesi e l'indentazione automatica. L'editor è in grado di compilare e caricare sulla scheda Arduino il programma eseguibile un maniera semplice. In questo caso non c'è necessità di creare dei Makefile o eseguire programmi da riga di comando.
L'ambiente è fornito con librerie C/C++, grazie alla documentazione fornita dal Team Arduino, la scrittura di nuove librerie è alla portata di tutti coloro che hanno dimestichezza con il linguaggio.

Dal giorno 15 settembre 2022 Arduino ha rilasciato la versione stabile di Arduino IDE 2.0.

Arduino IDE 2.0 è stata rilasciato nella primavera 2021 in versione Beta e grazie alle segnalazioni dei vari utenti i dipendenti di Arduino hanno avuto modo di risolvere i vari bug presenti e implementare varie feature in modo tale da migliorarne l'user experience.

La versione Arduino IDE 2.0 offre un editor moderno e tempi di compilazioni più rapidi. Questo IDE si basa sul framework Eclipse Theia, che è un progetto open source basato sulla stessa architettura di VS Code. Il front-end è scritto in TypeScript, mentre la maggior parte del back-end è scritto in Golang.

3.6 Codice di verifica dell'hardware

Per poter sviluppare ed eseguire il codice sulla nostra scheda avremo bisogno di predisporre la piattaforma con i driver specifici di Maixduino.
Fortunatamente Arduino IDE ci da la possibilità di installare automaticamente moltissimi tipi di Board all'intenro del nostro ambiente e installarne i relativi Driver.

Tra le Board configurabili sull'IDE però non avremo di default la possibilità di scegliere Maixduino, per poterlo fare utilizzeremo un parametro sulle impostazioni dell'IDE che ci permetterà di specificare un URL esterno in modo tale da poter attingere a librerie personalizzate in rete.
Nel nostro caso indicheremo in un parametro chiamato "Additional boards manager URLs", uno specifico indirizzo che sarà poi utilizzato come fonte per lo scaricamento e l'utilizzo dei driver associati.
Sipeed infatti ha già provveduto alla realizzaione del porting per Arduino della MaixBoard (K210), e la troveremo disponibile online su GitHub.

Dopo un attenta analisi è stato effettuato un test per verificarne il corretto funzionamento.
Come codice di verifiche sarà utilizzato il progrmma di esempio "selfie.ino" presente all'interno della libreria mostrato nell'immagine sotto (Fig. 3.10):

```
#include <Sipeed_OV2640.h>
#include <Sipeed_ST7789.h>

SPIClass spi_(SPI0); // MUST be SPI0 for Maix series on board LCD
Sipeed_ST7789 lcd(320, 240, spi_);
Sipeed_OV2640 camera(FRAMESIZE_QVGA, PIXFORMAT_RGB565);

void setup()
{
    Serial.begin(115200);
    lcd.begin(15000000, COLOR_RED);
    if(!camera.begin())
      Serial.printf("camera init fail\n");
    else
      Serial.printf("camera init success\n");
    camera.run(true);
}
void loop()
{
  uint8_t*img = camera.snapshot();
  if(img == nullptr || img==0)
    Serial.printf("snap fail\n");
  else
    lcd.drawImage(0, 0, camera.width(), camera.height(), (uint16_t*)img);
}
```

Figura 3.10: codice di esempio di cattura

Il settaggio della camera come si vede dal codice proposto è stato impostato con un formato QVGA(320x240) e un PixelFormat RGB565.
Il formato QVGA è un buon formato da utilizzare considerando che il nostro target size è 224x224.
Eseguendo il codice mostrato però si sono evidenziate delle problematiche e l'incapacità di poter effettuare una cattura di un immagine tramite il senosre OV2640 portando in eccezione il programma.
Tramite il log dell'esecuzione e un'attenta ricerca è stato constatato che questa problematica era nota e che vi fosse la possibilità di aggirarla parzialmente utilizzando una libreria personalizzata presente in rete chiamata `Maixduino_GC0328`

Questa libreria risolve il problema della cattura dell'immagine ma nel contempo ha presenti meno funzionalità rispetto alla versione originale Sipeed_OV2640.
Questo tipo di discrepanza sarà sicuramente colmata in futuro ma attualmente ci troviamo in queste condizioni nel momento del suo utilizzo.

Possiamo verificare il corretto funzionamento dell'acquisizione grafica ottenuta visualizzando su schermo LCD il risultato dei pixel dell'immagine.

Una volta determinato il corretto funzionamento di cattura dell'immagine sarà importante anche verificare come questa debba esser memorizzata sulla scheda Maixduino.
La memoria interna presente sulla scheda non è adeguata per la memorizzazione delle immagini, questo perché è di limitata capacità e ci permetterebbe di salvare soltanto qualche immagine ma non un numero considerevole.
Per poter risolvere il problema dello spazio possiamo utilizzare logiche di memorizzazione su SD card di cui Maixduino è dotata.
Prima di procedere al test su scheda è stata fatta una fase di ricerca per poter determinare quali Card fossero compatibili con la SoC in oggetto.
Sono stati presi dei test effettuati con molteplici memorie appartenenti a brand e capacità differenti e sono state elencate nella tabella in basso:

Brand	Storage	Type	Class	Format	Test Results
Kingston	8G	HC	Class4	FAT32	OK
Kingston	16G	HC	Class10	FAT32	OK
Kingston	32G	HC	Class10	FAT32	NO
Kingston	64G	XC	Class10	exFAT	OK
SanDisk	16G	HC	Class10	FAT32	OK
SanDisk	32G	HC	Class10	FAT32	OK
SanDisk	64G	XC	Class10	/	NO
SanDisk	128G	XC	Class10	/	NO
XIAKE	16G	HC	Class10	FAT32	OK(purple)
XIAKE	32G	HC	Class10	FAT32	OK
XIAKE	64G	XC	Class10	/	NO
TURYE	32G	HC	Class10	/	NO

Figura 3.11: tabella SDcard supportate

4-Analisi e progettazione della rete

4.1 tecnologie utilizzate

Per creare efficacemente modelli efficenti è necessario creare una rete neurale e poterla addestrare adeguatamente in tempi ragionevoli, l'esecuzione del codice deve avvenire in un sistema di calcolo adeguato, piuttosto che eseguire il training su dispositivi personalizzati nel nostro caso è stato pensato di avvalersi di una piattaforma Cloud chiamata Google Colab.

Google Colab è un servizio cloud, offerto da Google in maniera gratuita. Si basa sull'ambiente Jupyter Notebook ed è destinato alla formazione e alla ricerca nell'apprendimento automatico.
Tramite Google Colab è possibile eseguire codici, specialmente in python e bash direttamente sul Cloud.

Come accennato in precedenza, una volta individuata la piattaforma di sviluppo, bisogna decidere quali pacchetti installarvi al suo interno, in base ovviamente all'obbiettivo da raggiungere.
Sono molte le librerie utilizzate in questo progetto, citeremo solo le più significative:

TensorFlow

Questa è una piattaforma Open Source per Python, che consente lo sviluppo di sistemi di machine learning. Esiste anche una versione Web, chiamata Tensorflow.js.
Tensorflow è una parola composta da Tensor e Flow
Tensor:
I tensori sono la naturale generalizzazione del concetto di vettore. Consideriamo uno spazio vettoriale ad N coordinate, sul quale possiamo effettuare operazioni come rotazioni e traslazioni, che definiamo genericamente trasformazioni; in tali operazioni la distanza tra due punti arbitrari rimane inalterata. Sotto tali trasformazioni invece le N componenti che definiscono un vettore non rimangono inalterate ma si trasformano in un modo ben preciso. In generale, possiamo affermare che un generico tensore è una

grandezza le cui componenti si trasformano come il prodotto delle componenti di un certo numero di vettori.

Flow:

Indica il flusso che viene seguito per il calcolo e rapresenta la quantità scalare ottenuta tramite l'integrale di superficie del prodotto scalare, calcolato in ogni punto della superficie.

Tensorflow si configura quindi come un framework per la definizione e il calcolo di operazioni che coinvolgono tensori, rappresentazioni generiche di vettori e matrici a dimensioni maggiori.

Internamente, Tensorflow rappresenta i tensori come array n-dimensionali di datatypes base (int, string, etc..), che per il momento non consideriamo.

TensorFlow ha due caratteristiche fondamentali:

- Facile costruzione dei modelli: la costruzione e l'addestramento di modelli è facile ed intuitiva. Usando API di alto livello come Keras, con un'esecuzione sistematica e facile da assimilare, l'interazione tra utente e modello è veloce consentendo un debug rapido ed intelligente.
- Produzioni di modelli in qualsiasi ambiente: è possibile costruire e distribuire facilmente i diversi modelli tra i vari cloud, tra i browser o su diversi dispositivi, indipendentemente dal linguaggio utilizzato.

OpenCV

Questa è una libreria software multipiattaforma (Windows, Mac OS, Unix, Android) nell'ambito della computer vision. Il linguaggio nativo di tale libreria è il C++, ma esistono estensioni per Python, Java e Mathlab. OpenCV è accessibile e modificabile da tutti grazie alla licenza BSD (Berkley Software Distribution). Chi apporta variazioni a un programma protetto da questa licenza, può ridistribuirlo senza l'obbligo di indicare tutte le modifiche apportate al codice sorgente.

Sono circa 500 le funzioni messe a disposizione da OpenCV, e sono tutte finalizzate all'elaborazione di immagini, al tracking e all'object detection. L'immagine può venire rappresentata in tre maniere diverse:

- Bianco e Nero, forma di rappresentazione visiva che non utilizza il colore, il termine fa riferimento al solo uso del bianco e del nero.
- Scala di Grigi, nota come acromatica, sono una gamma di tonalità monocromatiche (grigie), che vanno dal bianco puro sull'estremità più chiara al nero puro sull'estremità opposta. La scala di grigi contiene solo informazioni sulla luminanza (luminosità) e nessuna informazione sul colore; ecco perché la luminanza massima è bianca e la luminanza zero è nera; ogni cosa in mezzo è una sfumatura di grigio.
- A Colori, in quest'ultimo caso OpenCV di default utilizza lo spazio di colori BGR, una permutazione dei tre classici canali RGB.

Python Imaging Library

È una libreria aggiuntiva gratuita e open source per il linguaggio di programmazione Python che aggiunge il supporto per l'apertura, la manipolazione e il salvataggio di diversi formati di file immagine.

Keras

Keras è la libreria di alto livello definita su TensorFlow. Fornisce un tipo di API di apprendimento sci-kit scritto in Python per la creazione di reti neurali.
Si può utilizzare "Keras" per sviluppare rapidamente una rete neurale senza preoccuparsi delle caratteristiche matematiche dell'algebra tensoriale, dei metodi di ottimizzazione e delle tecniche numeriche.

L'idea chiave in ritardo rispetto allo sviluppo di Keras è quella di semplificare le indagini mediante la prototipazione rapida. La capacità di passare da un'idea a un risultato con il minimo ritardo è la chiave per una buona ricerca.
Ogni organizzazione ha sempre cercato di incorporare il Deep Learning in un modo o nell'altro, Keras offre un'API semplificata e permette di creare applicazioni di Deep Learning con il minimo sforzo.

NumPy

Libreria in Python usata per lavorare con gli array. In generale è molto utilizzata quando si ha a che fare con il campo dell'algebra lineare, di vettori e di matrici.

E’ stata creata nel 2005 da Travis Oliphant, è un progetto open source e il suo nome sta per Numerical Python.
In Python spesso utilizziamo le liste quando vogliamo avere degli array, ma le liste sono un tipo di dato molto lento e che quindi rallentano tutti i processi.

NumPy usa invece degli oggetti specifici, che si chiamano ndarray, che sono fino a 50 volte più veloci delle liste.
Il motivo per cui gli array di NumPy sono più veloci delle liste è perchè sono salvati in uno spazio di memoria che è di più facile accesso, e questo permette quindi manipolazioni molto più facili.
Questa libreria è stata scritta parzialmente in Python, ma molte delle sua parti che richiedono computazioni veloci sono state scritte in C e C++.

Keras
Keras è una libreria per reti neurali di alto livello, scritta in Python e in grado di funzionare sopra TensorFlow o Theano. È stata sviluppata con l'obiettivo di consentire una sperimentazione rapida. Passare dall'idea al risultato con il minor ritardo possibile è fondamentale per fare una buona ricerca.
Keras permette una prototipazione facile e veloce (grazie alla modularità totale, al minimalismo e all'estensibilità), supporta sia reti convoluzionali che reti ricorrenti o combinazioni delle due.
Supporta schemi di connettività arbitrari (compreso l'addestramento multi-input e multi-output), funziona senza problemi su CPU e GPU.

4.2 Transfer Learning

L'addestramento dei modelli richiede molti dati, che non sono sempre disponibili. È qui che entra in gioco il transfer learning, che sfrutta i modelli precedentemente addestrati.

Il transfer learning è una tecnica basata sull’apprendimento automatico che si focalizza nella memorizzazione di relazioni tra input e output, pattern e modelli, acquisite durante la fase di training su un problema non correlato con quello preso in esame, per poi metterle in relazione con quest’ultimo.
L’obiettivo è quello di riutilizzare o trasferire informazioni da learned tasks, precedentemente appresi, per l’apprendimento di nuovi. Tutto questo ha portato un

significativo incremento del numero e della tipologia delle reti utilizzate nell'ambito del machine learning. Questa tecnica inoltre cerca di migliorare l'efficienza del singolo campione del dataset.

È possibile utilizzare questa tecnica di apprendimento su problemi molto spesso di natura predittiva.

Il Transfer Learning è sostanzialmente un'ottimizzazione, una scorciatoia per risparmiare tempo e in caso ottenere prestazioni migliori.

E' una tecnica da provare solo in alcuni casi; se è possibile identificare nel dataset a disposizione una qualche correlazione con il dataset ampio di partenza su cui il modello è già stato pre-allenato, oppure se è disponibile già un modello pre-addestrato che è possibile utilizzare come punto di partenza.

In basso sarà mostrata un immagine di esempio (Fig.4.1):

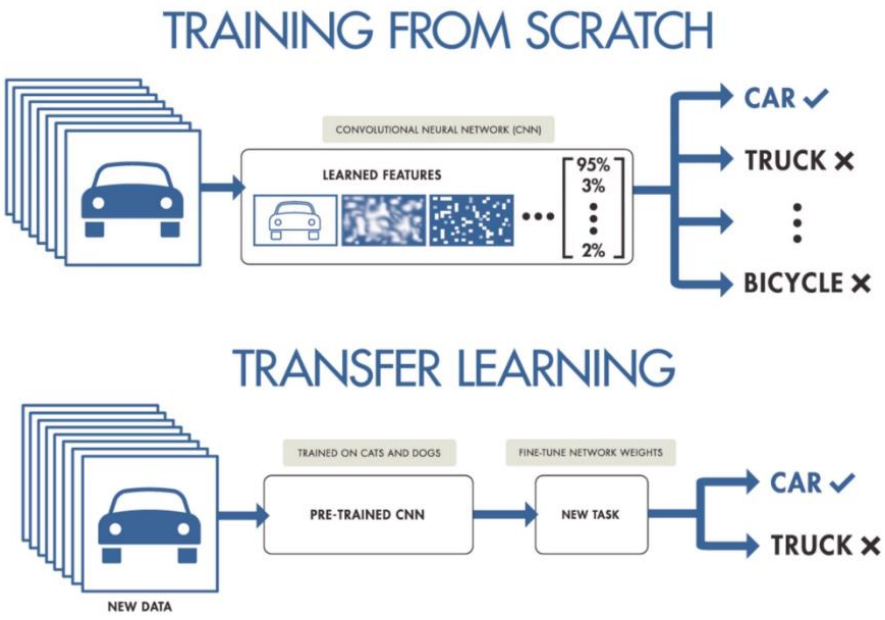

Figura 4.1: schema sui tipi di training [HAR20]

Gli steps per poter applicare tipicamente il trasfer learning sono i seguenti:

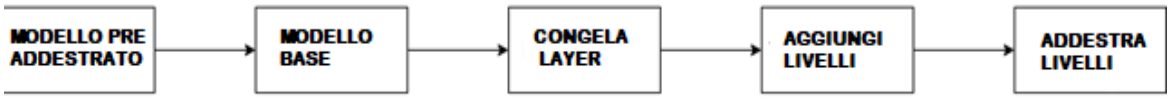

Figura 4.2: steps transfer learning

Ottenere il modello pre-addestrato

Il primo passo è ottenere il modello pre-addestrato che si vuole utilizzare per il problema in oggetto.

Crea un modello di base

Di solito, il primo passaggio consiste nell'istanziare il modello di base utilizzando una delle architetture come ResNet o Xception o MobileNet. Si può anche opzionalmente scaricare i pesi pre-addestrati. Si può anche non scaricare i pesi ma utilizzare direttamente l'architettura per addestrare il modello da zero.

Congelamento dei livelli

Il congelamento dei livelli del modello pre-addestrato è fondamentale. Questo perché non si vuole che i pesi in quei livelli vengano reinizializzati. Se lo sono, allora si perderà tutto l'apprendimento che ha già avuto luogo. Questo non sarà diverso dall'addestrare il modello da zero.

Aggiungi nuovi livelli addestrabili

Il passaggio successivo consiste nell'aggiungere nuovi livelli addestrabili che trasformeranno le vecchie funzionalità in previsioni sul nuovo set di dati. Questo è importante perché il modello pre-addestrato viene caricato senza il livello di output finale.

Addestramento dei nuovi livelli sul set di dati

Qui verrà svolta la fase di training dei soli livelli aggiunti, tipicamente sono inseriti una serie di strati densi in cui lo strato finale corrisponde con il numero delle classi da etichettare.

4.3 Definizione del modello

Per la creazione del nostro modello si è scelto di utilizzare un modello pre-addestrato. Avvalendoci del Transfer learning possiamo modificare la rete per adattarla meglio alle nostre esigenze.

Il codice iniziale che useremo in Python su Colab sara quello usato per scaricare le librerie contenenti il modello preaddestrato del MobileNetV1 (Fig.4.3).

```
# Using MobileNetv1
base_model=MobileNet(input_shape=(IMAGE_SIZE, IMAGE_SIZE, 3), alpha = ALPHA,
                     depth_multiplier = 1, dropout = 0.001, include_top = False,
                     weights = "imagenet", classes = 1000, backend=keras.backend,
                     layers=keras.layers,models=keras.models,utils=keras.utils)
```

Figura 4.3: codice per lo scaricamento della mobile net

Keras fornisce la possibilità di utilizzare le API functional, queste possono gestire modelli di tipologia non lineare, livelli condivisi e persino ingressi o uscite multiple.
In questo caso immagazzineremo il modello nella variabile, inizializzeremo l'architettura MobileNet indicando i parametri come dimensione dell'immagine, tipi di pesi, numeri di classi e altri parametri relativi ai livelli.

Una volta ottenuta la struttura possiamo aggiungere 2 strati nascosti da 100 e 50 nodi. È possibile aggiungere altri strati o nodi, ma questo aumenterà le dimensioni del modello e potrebbe non rientrare nella memoria di Maixpy.

Per farlo è stata implementata una funzione che permette di indicare il dropout, gli strati nascosti e il numero di livelli di uscita (Fig.4.4).

```
def build_finetune_model(base_model, dropout, fc_layers, num_classes):
    for layer in base_model.layers:
        layer.trainable = False

    x = base_model.output
    x = GlobalAveragePooling2D()(x)

    for fc in fc_layers:
        x = Dense(fc, activation='relu')(x)
        x = Dropout(dropout)(x)

    predictions = Dense(num_classes, activation='softmax')(x)
    finetune_model = Model(inputs=base_model.input, outputs=predictions)

    return finetune_model
```

Figura 4.4: codice per la creazione dei livelli aggiuntivi

Tramite la funzionalità “trainable” disattiviamo i livelli di apprendimento della rete preaddestrata.
Procederemo quindi ad aggiungere un GlobalAveragePooling2D più un Dense e Dropout layer, utilizzando una funzione softmax e restituendo il modello come ritorno alla funzione.

A questo punto possiamo utilizzare la funzione passandogli i parametri della rete appena scaricata.
Alla funzione verrà passato il parametro “num_classes”, questo ci permetterà di inserire nel codice del fine tuning aggiuntivo lo strato di Dense finale con il numero delle classi che nel nostro caso sono 5 (Fig.4.5).

```
FC_LAYERS = [100, 50]
dropout = 0.5

finetune_model = build_finetune_model(base_model,
                                      dropout=dropout,
                                      fc_layers=FC_LAYERS,
                                      num_classes=5)
```

Figura 4.5: applicazone della funzione “build_finetune_model”

Come risultato otterremo una rete composta da 92 livelli complessivi, di cui gli ultimi 6 sono quelli relativi alla nostra personalizzazione (Fig. 4.6).

```
82 conv_dw_13_bn
83 conv_dw_13_relu
84 conv_pw_13
85 conv_pw_13_bn
86 conv_pw_13_relu
87 global_average_pooling2d
88 dense
89 dropout
90 dense_1
91 dropout_1
92 dense_2
```

Figura 4.6: log dei livelli aggiunti

5-Sistema per la cattura delle immagini

Per la cattura delle immagini che comporranno poi il dataset avremo bisogno di creare un dispositivo di cattura che sia autonomo e facilmente portabile.
Questo agevolerà la fase di acquisizione per poterci poi concentrare in un secondo momento sulla parte relativa ai dati estratti.
Per creare questo sistema inanzitutto ci serviranno delle componenti che riassumeremo nel sottoparagrafo successivo.
In seguito descriveremo le componenti hardware utilizzate, la modalità operativa del dispositivo e l'implementazione del codice risultante.

5.1 Componenti hardware

Sono state individuate alcune componenti hardware necessarie per poter effettuare la creazione del sistema di cattura e sono:

1-BreadBoard Classica

Rappresenta un mezzo molto comodo e nello stesso tempo potente per realizzare montaggi di circuiti elettronici senza saldature.

2-Adattatore 9V jack 2.1mm

Connettore 9V con Jack 2.1mm adatto alla scehda Maixduino.

3-Batteria 9V

Classica Batteria da 9V che è possibile reperire in commercio.

4-Sistema per fissaggio camera OV2640

Bloccaggio fisso/mobile dell sensore CMOS in modo da poter avere un verso di puntamento di default per l'immagine.

5-Micro SD card

Utilizzata per contenere i dati ottenuti dal sensore CMOS.

Qui rappresenteremo il circuito complessivo realizzato (Fig.5.1).

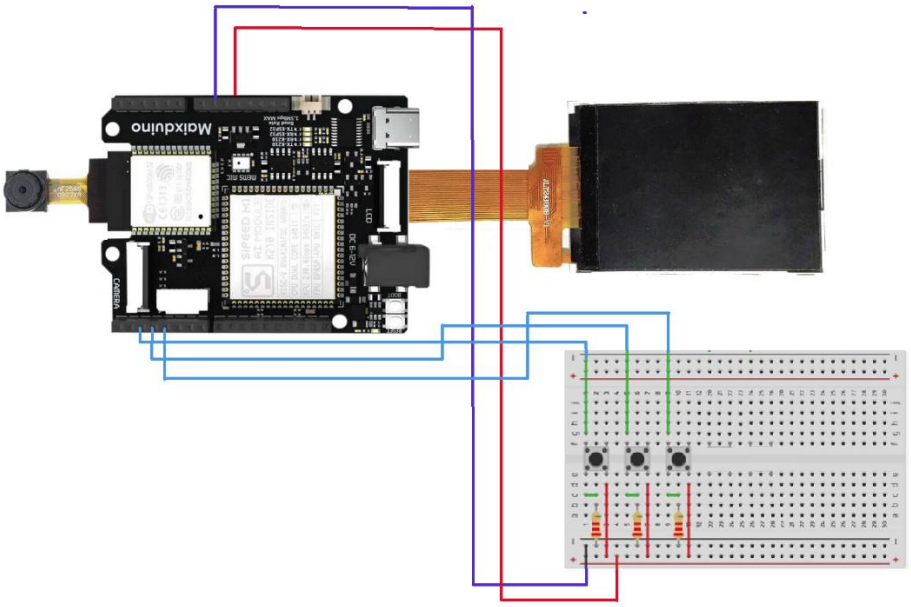

Figura 5.1: schema del circuito

Queste tre compnenti sono state fissate sopra a una tavoletta di legno sottile in modo da avere un'unica struttura portante e stabile.
Per la camera è stata utilizzata una placca di allumino fissata alla base della scheda e un piccolo pezzo di nastro adesivo per tenerli uniti.

5.2 Logica Operativa

Questo sistema ovviamente anche se munito di camera necessita di logica di operatività per esser utilizzata nel modo più semplice e corretto.
Ricordiamo che il dispositivo non ha nessun tipo di controllo sulla stabilità dell'immagine al momento della cattura quindi sarà necessario mantenere una posizione fisica più ferma possibile in modo da evitare di creare situazioni di sfocatura e quindi di immagine inutilizzabile.
Oltre a questo saranno montati sulla BradBoard dei tasti per poter implementare logiche minime necessarie per il raggiungimento del compito prefissato.
Nello specifico, saranno utilizzati 3 tasti e sono state individuate 3 funzionalità fondamentali.

- **Scatto**:

Questa modalità è rappresentara da un tasto che servirà appunto per scattare l'immagine e salvarla utilizzando un nome predefinito composto da due caratteri e dei numeri indicanti il numero di sequenza.

- **Cancella**

  Nel caso venisse scattata un immagine mossa o con target sbagliato è giusto prevedere la possibilità di eliminare l'ultima foto catturata, ovviamente questo lo possiamo determinare grazie all'utilizzo dello schermo LCD che ci permette di visualizare l'ultima immagine scattata.

- **Rinomina**

  Questa funzionalità è fondamentale nel caso si voglia collegare/scollegare l'alimentazione della scheda, magari per evitare di consumare eccessivamente energia in tempi di non attività o per cambio zona.

  Il nome con cui vengono salvate le foto è composto da una stringa e due variabili intere concatenate, ipotizzando che le variabili si chiamino X e Y abbiamo che Y viene incrementato ad ogni scatto e X invece è un valore che rimane costante.

  Premendo il bottone RINOMINA possiamo forzare l'incremento della variabile X in modo da rendere unici i nomi dei file creati e non sovrapporli nel caso si riavviase la scheda Maixduino per mancata alimentazione.

5.3 Implementazione

Prima di procedere con la visione del codice relativo alla cattura dell'immagine stessa partiamo col presupposto che il sistema creato, munito di tasto deve esser opportunatamente gestito a livello implementativo.

I pulsanti spesso generano transizioni di apertura/chiusura spurie quando vengono premuti, a causa di problemi meccanici e fisici: queste transizioni possono essere lette come pressioni multiple in un tempo molto breve, ingannando il programma.
Sarà mostrato come si sia proceduto ad effettuare il debounce di un ingresso, cioè controllare il segnale in un breve periodo di tempo per assicurarsi che il pulsante sia stato premuto correttamente. Senza il debouncing, premendo il pulsante una sola volta

si possono ottenere risultati imprevedibili con la conseguente esecuzione multipla delle istruzioni all'interno della funzione del bottone (Fig.5.3).

```
const int buttonPin = 2;

pinMode(buttonPin, INPUT);

buttonState = digitalRead(buttonPin);

loop{
buttonState = digitalRead(buttonPin);
if(buttonState==1){
   delay(50);
   buttonState = digitalRead(buttonPin);
   if(buttonState==0){

...
```

Figura 5.3: codice per il Debounce del bottone

Una volta predisposta l'implementazione ed eseguita sull'IDE è stato possibile verificare il corretto funzionamento dei tasti, privi di problemi di Bouncing.

Ora ci apprestiamo a sviluppare il codice che ci permette di salvare i byte dell'immagine sul modulo SD card.
In basso l'implementazione utilizzata in linguaggio C/C++ (Fig.5.4).

```
uint8_t* img88 = camera.getRGB888();
imgFile = SD.open(filename, FILE_WRITE);

if (imgFile) {
for (uint32_t i=0;i<230400;i+=2) {
imgFile.write(*(img88+i+1));
imgFile.write(*(img88+i));
}
tft.drawImage(0, 0, camera.width(), camera.height(), (uint16_t*)img);
}
else
{
Serial.println("File opening failed");
}
imgFile.close();
```

Figura 5.4: codice di scrittura di un immagine

Il primo passaggio è ottenere i pixel dell'immagine, per questo avremo un puntatore che ci indicherà dove sono immagazzinati i dati nel registro della scheda.
I dati ottenuti dai registri saranno poi scritti sull'SD card, per la scritura è stato utilizzato un ciclo iterandolo per il numero di pixel complessivo, ovvero altezza * larghezza *dimensione colori, che nel nostro caso sarà 320*240*3 = 230400.
La scrittura all'interno del loop è stata effettuata con una modalità di ordinamento dei byte di tipo BigEndian.
Dopo aver creato il file viene mostrato sullo schermo LCD il risultato visivo dell'immagine e viene chiuso lo stream da ulteriori modifiche.
Effettuando varie prove è stato appurato il corretto funzionamento del meccanismo generando file di grandezza 225kb per ciascuno.

6-Creazione del DataSet

Per il Dataset utilizzato in questo progetto si è scelto di campionare 5 differenti tipologie di suolo, come inizialmente è stato già accennato saranno prese immagini relative a tipi di suolo differenti :

- Ghiaiato
- Stradale
- Pedonale
- Sintetico
- Fuori Strada

Si è ricaduto in questa scelta per via della semplicità e la facile reperibilità di queste classi nella vita di tutti i giorni.
Questo progetto di tesi si è focalizzato sullo sviluppo nella sua interezza del sistema creato e sulle fasi necessarie per poter arrivare all'obbiettivo finale nella maniera più semplice, le immagini scelte per il Dataset sono immagini di esempio, qualora si volesse applicare lo stesso concetto per immagini differenti a fronte di un campionamento più specifico verso tematiche diverse è possibile riutilizzare gli stessi concetti espressi.

6.1 Fase di campionamento

In questa fase si è proceduto a catturare le immagini che comporranno poi nostro Dataset.
Sappiamo che per raggiungere il nostro scopo abbiamo bisogno di classificare 5 tipologie di terreno.

Sono state effettuate immagini provenineti da diverse zone di Bologna periferia, San Lazzaro di Savena, Imola e Borgo Tossignano (Fig. 6.1 e Fig. 6.2).
In basso, evidenziate in blu, possiamo vedere tramite OpenStreetMap le zone oggetto di campionamento:

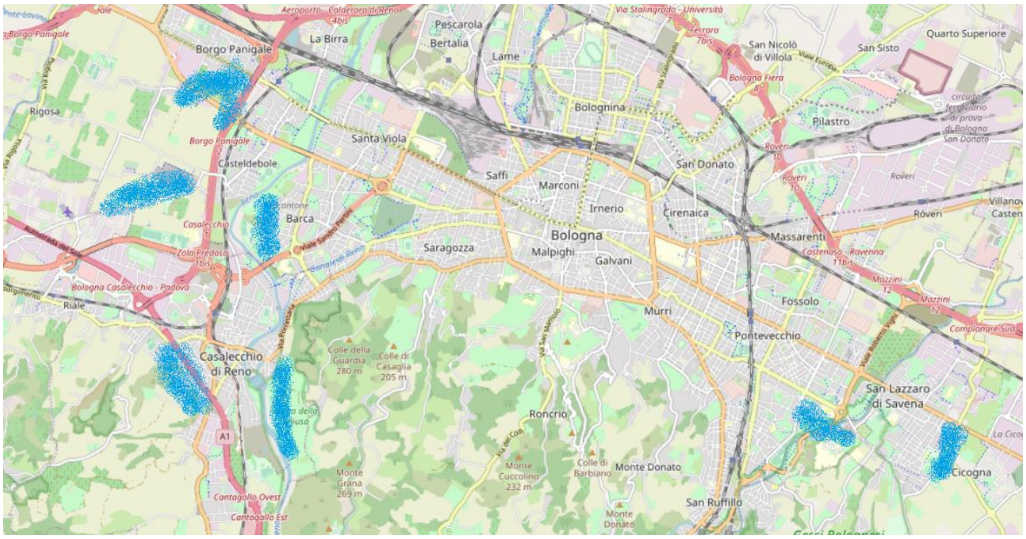

Figura 6.1: mappa della zona di Bologna periferia

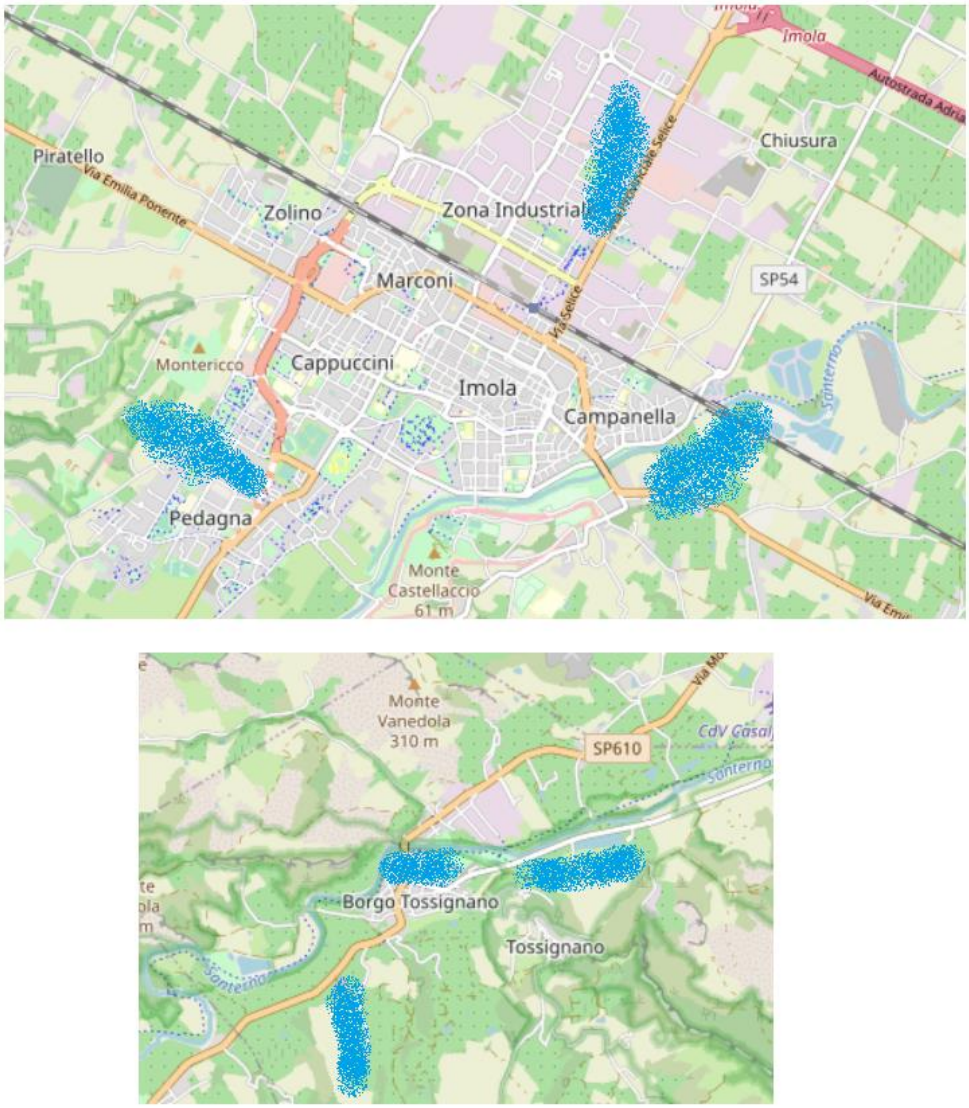

Figura 6.2: mappa sulle zone campionate di Imola e Borgo Tossignano

Tutte sono state scattate in condizioni di moderata luminosità per via del tempo meteorologico tendenzialmente nuvoloso, tipico del periodo invernale in cui si è preceduto a effettuare il lavoro.

Da notare che in questa fase sono stati presi un totale di circa 200 elementi per ogni classe, le rilevazioni effettuate sono state differenziate il più possibile per ottenere immagini molto diverse tra di loro.
Nel corso del campionamento delle immagini sono stati fatti controlli preventivi effettuati manualmente sui campioni ottenuti. Moltissimi dei dati catturati sono stati scartati per via di interferenze dovute all'esposizione alla luce ambientale.
Le immagini cosi ottenute sono state caricate su GitHub sulla repository personale dedicata alla Tesi in oggetto.
Principalmente le immagini catturate sono state prese in condizioni di scasa luminosità, molte risultano catturate durante orari pomeridiani e con condizioni atmosferiche nuvolose o parzialmente solari.

6.2 Elaborazione delle immagini

Per le immagini salvate dalla scheda Maixduino sulla nostra Micro SD card utilizzeremo Colab con il linguaggio Pyton per poterle decodificare ed effettuare modifiche ai pixel. Tramite le librerie PIL infatti è possibile ricorstruire l'immagine a parteire dai Byte presenti nei file che abbiamo generato.
Di seguito viene fornito il codice in Python (Fig.6.3):

```
H,W = 240,320
name="Risc-V-Thesis-Cnn/GO0_0"
file = open(name+'.TXT', "rb")

data = list()
byte = file.read(1)

while byte:
   data.append(int.from_bytes(byte, "big"))
   byte = file.read(1)
ndArrayInt = np.array(data)
file.close()

img_all = ndArrayInt.reshape(3,H,W).transpose(1,2,0)

print(img_all.shape)
testImage = Image.fromarray(img_all.astype(np.uint8),mode='RGB')
testImage.save('example.PNG')

plt.imshow(image67)
plt.title ('color image by open cv')
plt.show()
```

Figura 6.3: codice per la decodifica dell'immagine da forma binaria

Dal codice mostrato nell'mmagine sopra(Fig. 6.3) vediamo che è stato necessario convertire ogni singolo byte a numeri interi, è stata utilizzata una variabile per memorizzare tali conversioni.

Viene poi effettuata l'operazione transpose in combinazione con la funzione reshape, la prima ci permette di scambiare gli assi dell'ndArray in questo modo si ottiene la trasposizione della matrice, ovvero una matrice ottenuta sostituendo le colonne con le righe e successivamente effettuando l'operazione del reshape modifichiamo la forma dei tensori per adattarlo alla dimensione dell'immagine a colori che dovrà risultare.
Questo ci permette di avere un ndarray risultante di queste dimensioni (240, 320, 3).
I dati risultanti saranno poi convertiti a unsigned int a 8 bit e passati alla funzione Image.fromwarray messa a disposizaione dalla libreria PIL, insime al paramtro "mode" che indica le tre dimensioni dello spazio dei colori.
A questo punto non resta che salvare e visualizzare l'immagine a schermo tramite matplotlib.

In basso un immagine di esempio visualizzata a schermo (Fig. 6.4):

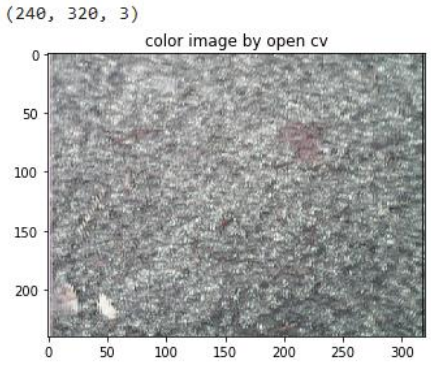

Figura 6.4: immagine catturata dal sensore OV2640

Data Augmentation

Nella fase di camponamento discussa nel sottoparagrafo precedente abbiamo detto che sono state classficate circa 200 imamgini per ogni classe del Dataset.

La rete neurale che andremo ad utilizzare ovviamente avrà bisogno di una quantità di dati superiore al fine di avere un buon risultato in fase di training.

Per questo motivo vi sono delle tecniche che vengono utilizzate per ampliare il Dataset partendo da immagini esistenti e generandone altre applicando trasformazioni sulle immagini, queste tecniche si chiamano di Data Augmentation.

Gli Augmented Data si utilizzano proprio per risolvere problematiche di overfitting, ampliando il dataset di addestramento attraverso un maggior numero di dati a disposizione.

Un esempio schematico del funzionamento di alcune di queste tecniche puo essere riassunto nell'immagine sottostante (Fig.6.5):

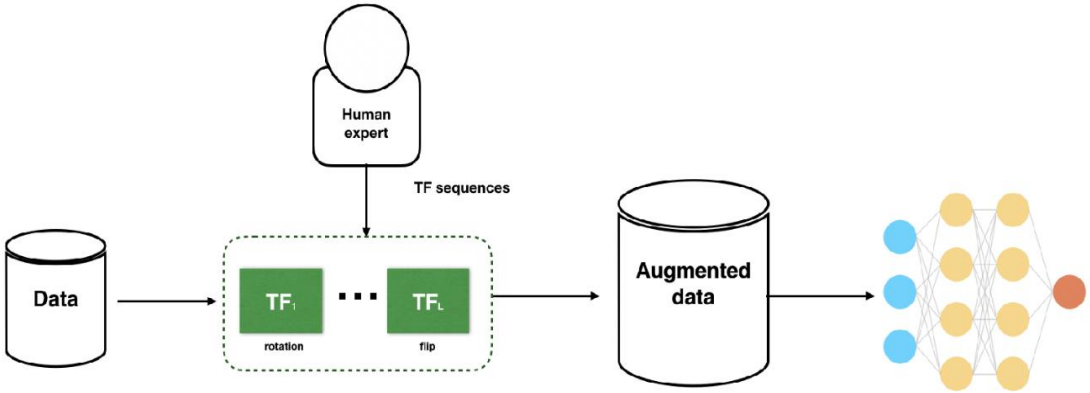

Figura 6.5: incremento dei dati e addestramento[SHA20]

Tra i benefici che puo portare l'utilizzo delle tecniche di Data Augmentation abbiamo:

- Il miglioramento dell'accuratezza di predizione del modello,
- Aumento dei dati presenti nel DataSet,
- Diminuzione dell'overfittting,
- Migliore capacità di generalizzazione del modello,
- Possibilità di sopperire a problemi di equilibrio del numero dei sample per classe.
- Riduzione dei costi di raccolta ed etichettatura dei dati,
- Previene il problema di riservatezza legata ad alcuni dati sensibili e privati.

Le tecniche di Data augmentation possono essere applicate in vari domini, per quanto riguarda le immagini possiamo dividere alcune tecniche in base a due tipologie:

Position augmentation

Sostanzialmente le posizioni dei pixel di una immagine vengono modificate.

- Scaling

  L'immagine viene ridimensionata alla dimensione specificata, ad es. la larghezza dell'immagine può essere raddoppiata oppure scalata.
- Cropping

  Nel Cropping, viene selezionata una porzione dell'immagine, ad esempio se avessimo un immaginie di 400x400 possiamo prendere la parte centrarle con dimensione 100x100, questo permette quindi di avere una porzione specifica.
- Flipping

  Nel flipping, l'immagine viene capovolta orizzontalmente.

- Rotation

  Semplice rotazione dell'immagine, questa può esser effettuata indicando l'angolazione della rotazione.
- Translation

  In questa modalità l'immagine viene spostata lungo l'asse x o lungo l'asse y.
- Affine transformation

  La trasformazione affine conserva punti, linee rette e piani. Può essere utilizzata per il ridimensionamento, la traslazione, il taglio, la rotazione, ecc.

Color augmentation

L'aumento del colore o il jittering del colore riguarda l'alterazione delle proprietà del colore di un'immagine modificandone i valori dei pixel.

- Brightness

  Un modo per aumentare i dati è modificarne la luminosità dell'immagine. L'immagine risultante diventa più scura o più chiara rispetto a quella originale.
- Contrast

  Il contrasto è definito come il grado di separazione tra le aree più scure e più luminose di un'immagine. È inoltre possibile modificare il contrasto dell'immagine.
- Saturation

  La saturazione è la separazione tra i colori di un'immagine.
- Hue

  Modifica la tonalità di un immagine, può essere descritta come l'ombra dei suoi colori

Per i nostri specifici scopi non utilizzeremo tutte le tecniche sopracitate ma ne useremo solamente 2, ovvero Flip (Fig.6.6) e Crop (Fig.6.7).

Image Flipping:

```
! pip install ipyplot
import cv2
import ipyplot

path = 'Risc-V-Thesis-Cnn/DATASET/STRADALE/GO1_23.png'
img = cv2.imread(path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

horizontal_flip = img[:, ::-1, :]
vertical_flip = img[::-1, :, :]
```

Figura 6.6: codice di esempio di flip di un immagine

Image Crop:

```
import random
import cv2
import ipyplot

def randomCrop(img, width, height):
    assert img.shape[0] >= height
    assert img.shape[1] >= width
    x = random.randint(0, img.shape[1] - width)
    y = random.randint(0, img.shape[0] - height)
    img = img[y:y+height, x:x+width, :]
    return img

path = 'Risc-V-Thesis-Cnn/DATASET/STRADALE/GO1_23.png'
img = cv2.imread(path)
cropped_image = randomCrop(img, 224, 224)
ipyplot.plot_images([img,cropped_image], max_images=20,
img_width=224,img_=224)
```

Figura 6.7: codice di esempio di Crop di un immagine

Nel nostro DataSet saranno applicate ai dati di training le tecniche di random crop e horizontal Flip mentre i dati di validation e di test avranno soltanto un crop centrale di dimensione 224x224 (Fig.6.8).

```
rootdir = '/content/test_dir/'
test_directory = '/content/test_dir/'

for subdir, dirs, files in os.walk(rootdir):
    for file in files:
        f = os.path.join(subdir, file)
        im_res = Image.open(str(f))
        width, height = im_res.size   # Get dimensions
        left = (width - 224)/2
        top = (height - 224)/2
        right = (width + 224)/2
        bottom = (height + 224)/2
        # Crop the center of the image
        im = im_res.crop((left, top, right, bottom))
        os.remove(subdir+'/'+file)
        im.save(subdir+'/'+'cropped_'+file)
```

```
rootdir = '/content/val_dir/'
test_directory = '/content/val_dir/'

for subdir, dirs, files in os.walk(rootdir):
    for file in files:
        f = os.path.join(subdir, file)
        im_res = Image.open(str(f))
        width, height = im_res.size   # Get dimensions
        left = (width - 224)/2
        top = (height - 224)/2
        right = (width + 224)/2
        bottom = (height + 224)/2
        # Crop the center of the image
        im = im_res.crop((left, top, right, bottom))
        os.remove(subdir+'/'+file)
        im.save(subdir+'/'+'cropped_'+file)
```

Figura 6.6: elaborazione central Crop

In questo codice si nota che il crop viene centrato in base alla grandezza specificata dell'immagine e una volta creata l'immagine nuova la vecchia sarà rimossa automaticamente.

Il seguente lavoro si svolge su entrambe le cartelle test_dir e val_dir.

7-Training e test del modello sulla scheda

Sappiamo che Maixduino ha il modulo per reti neurali K210 incorporato e che si può effettuare direttamente sulla scheda il trainig del modello, per questo progetto però si è scelto solamente di effettuare l'esecuzione di un modello già addestrato, nel nostro caso la fase di addestramento sarà effettuata su un sistema di calcolo differente.

7.1 Creazione del modello in Keras

Per la crezione e l'addestramento della rete, come gia discusso precedentemente, saranno utilizzate le API di Keras.
La fase dell'addestramento di una rete è solo una delle operazioni che sono tipicamente svolte per la progettazione e validazione di una rete neurale.

Le tipiche fase da attuare sono le seguenti:

- **Caricamento del set di dati**
  In questa fase viene inizializzato il set di dati, questi possono essere dati contenuti online in repository pubbliche o private oppure dati personalizzati.
  In questa fase vengono acquisiti i dati che saranno utilizzati poi nelle prossime fasi.
- **Analisi esplorativa dei dati**
  Normalmente un dataset, per via della grandezza, viene preso da fonti esterne, in questo caso i dati acquisiti sono all'utente pressoche sconosciuti, prima di fare ogni ragionamento è necessario comprendere bene che tipo di dati si sta per elaborare.
  In questa fase si analizzeranno i dati comprendendone le caratteristiche intrinseche.
- **Pre-elaborazione dei dati**
  Qui si possono eventualmente effettuare pre elaborazioni sui dati del Dataset, effettuare operazioni di Data Augmentation o correzioni dei dati.
- **Creazione del modello**

Fase in cui sarà creato tramite le api il modello con ogni strato adeguatamente parametrizzato.

- **Compilazione e Addestramento del modello**
  In questa fase si addestrarà la rete sui dati precedentemente gestiti, sarà importate analizzare i log sull'andamento delle Epoche per determinare la qualità dell'addestramento.
- **Utilizzo del modello per fare previsioni su dati reali**
  Nella fase finale verrà effettuato un test sul modello addestrato per verificare l'efficienza su dati mai visti prima.

7.2 Training della rete

Nel paragrafo 4 abbiamo discusso come creare il modello, ora vediamo come procedere per poter addestrare la rete.

```
train_datagen=ImageDataGenerator(preprocessing_function=preprocess_input)

train_generator=train_datagen.flow_from_directory('/content/train_dir',
                                                  target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                                  color_mode='rgb',
                                                  batch_size=32,
                                                  class_mode='categorical', shuffle=True)

val_datagen=ImageDataGenerator(preprocessing_function=preprocess_input)

val_generator=val_datagen.flow_from_directory('/content/val_dir',
                                                  target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                                  color_mode='rgb',
                                                  batch_size=32,
                                                  class_mode='categorical', shuffle=True)
```

Figura 7.1: caricamento validation e test da folder

```
Found 4566 images belonging to 5 classes.
Found 180 images belonging to 5 classes.
```

Figura 7.2: log del numero delle classi e esemplari totali

Il codice appena mostrato (Fig.7.1 e Fig. 7.2) descrive come è stato caricato il dataset all'interno della sessione in Colab, il generatore è utile per poter applicare tecniche di data augmantation sui dati semplicemente inserendo dei parametri specifici nelle funzioni principali anche se nel nostro caso non si è optato per questa soluzione poiché i dati saranno già modificati separatamente.

Semplicemente non impostando i parametri relativi ai DataAugmentation potremmo caricare il dataset indicando le caratteristiche principali.

La classe incaricata di effettuare il caricamento è ImageDataGenerator, e fornisce tre diverse funzioni che sono:flow(),flow_from_directory(), flow_from_dataframe.().

Nel nostro caso utilizzeremo la seconda, questa infatti sarà adattata a strutture di dati organizzati gerarchicamente a folder, nel nostro caso è la situazione ideale.

Quando invocheremo la funzione flow_from_directory() gli passeremo come parametri la locazione fisica della cartella e la grandezza e tipologia di immagini su cui si rispecchia.

Un parametro importante è il rimescolamento autoamtico (shuffle) in fase di caricamento, semplicemente settando il parametro shuffle=True abiliteremo questa funzionalità.

Con la modalità di classe andiamo a indicare una struttura basata su categorie di classificazione.

Una volta caricato vi sarà la fase di compilazione e training del modello (Fig.7.3).

```
finetune_model.summary()
finetune_model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
step_size_train=train_generator.n//train_generator.batch_size


callback = keras.callbacks.EarlyStopping(monitor='accuracy',
                                         patience=2,
                                         restore_best_weights=True)

history = finetune_model.fit_generator(generator=train_generator,validation_data=val_generator,
                                       steps_per_epoch=step_size_train,epochs=EPOCHS, shuffle=True, callbacks=callback)

finetune_model.save('/content/my_model.h5')
```

Figura 7.3: fase di compilazione e addestramento della rete

Per compilare un modello possiamo usare l'attributo "compile", questo ci permette di specificare dei parametri importanti come:

- Optimizer: questo parametro indica l'ottimizzatore che vogliamo usare. Esistono diversi ottimizzatori come già accennato nel sottoparagrafo 1.4.
- Loss: questo parametro specifica la funzione di perdita che vogliamo utilizzare per il modello.
- Metrics: in questo campo possiamo passare la metrica in base alla quale vogliamo che il modello venga valutato.

Per la classificazioni di immagini a colori l'ottimizzatore tra i piu indicati è Adam.

Dopo diverse esecuzioni effettuate, il risultato migliore ottento è rappresentato dall'immagine seguente (Fig.7.4):

```
Trainable params: 82,205
Non-trainable params: 1,832,976
_________________________________________________________________
Epoch 1/10
142/142 [=========] - 177s 1s/step - loss: 0.7349 - accuracy: 0.7201 - val_loss: 0.1744 - val_accuracy: 0.9389
Epoch 2/10
142/142 [=========] - 166s 1s/step - loss: 0.2608 - accuracy: 0.9171 - val_loss: 0.1077 - val_accuracy: 0.9667
Epoch 3/10
142/142 [=========] - 163s 1s/step - loss: 0.1532 - accuracy: 0.9552 - val_loss: 0.1221 - val_accuracy: 0.9556
Epoch 4/10
142/142 [=========] - 168s 1s/step - loss: 0.1131 - accuracy: 0.9649 - val_loss: 0.1026 - val_accuracy: 0.9667
Epoch 5/10
142/142 [=========] - 170s 1s/step - loss: 0.0929 - accuracy: 0.9689 - val_loss: 0.1386 - val_accuracy: 0.9667
Epoch 6/10
142/142 [=========] - 169s 1s/step - loss: 0.0796 - accuracy: 0.9722 - val_loss: 0.1070 - val_accuracy: 0.9778
Epoch 7/10
142/142 [=========] - 174s 1s/step - loss: 0.0698 - accuracy: 0.9744 - val_loss: 0.1351 - val_accuracy: 0.9667
Epoch 8/10
142/142 [=========] - 173s 1s/step - loss: 0.0588 - accuracy: 0.9832 - val_loss: 0.1237 - val_accuracy: 0.9722
Epoch 9/10
142/142 [=========] - 169s 1s/step - loss: 0.0524 - accuracy: 0.9813 - val_loss: 0.0880 - val_accuracy: 0.9778
Epoch 10/10
142/142 [=========] - 165s 1s/step - loss: 0.0460 - accuracy: 0.9837 - val_loss: 0.1080 - val_accuracy: 0.9778
```

Figura 7.4: log dell'output del training

Da notare che il numero di perametri addestrabili sono 82,205 mentre quelli bloccati dall'apprendimento sono 1,832.976.

Parlando invece dei dati ottenuti possiamo dire che analizzando i risultati notiamo che inizialmente la differenza tra accuracy e val_accuracy incrementa continuamente arrivando a massimo a 0.9778, nello stesso tempo l'accuracy non scosta molto dai dati di validation.

La Loss diminuisce costantemente e 0.0460 è un risultato positivo anche se migliorabile.

Il modello appena addestrato infine deve esser salvato per poter proseguire con la conversione e adattamento all'esecuzione sulla nostra board Maixduino.

Per il salvataggio di un modello si possono avere più strade, possono essere salvati solo i pesi calcolati o l'intera struttura del modello.

Un primo modo per salvare il nostro modello è quello di serializzare i dati in un formato json e di salvarli in file, per esempio "modello.json".
Mentre per caricare il modello sarà sufficiente portare il file "modello.json" salvato ed utilizzare una funzione di keras per istanziarlo.
Nel nostro caso però salveremo invece tutta la struttura nel formato H5, per farlo ci basterà invocare la funzione "Save".

L'ultima fase è quella di test del modello utilizzando dati nuovi mai visti prima, questi dati sono detti dati di Test.
Tramite la funzone "evaluate_generator()" in cui indichiamo la fonte dei dati gli step utiizzati (FIg.7.5) e il livello di log(Fig. 7.6).

```
test_datagen=ImageDataGenerator(preprocessing_function=preprocess_input)

test_generator=test_datagen.flow_from_directory('/content/test_dir',
                                                target_size=(IMAGE_SIZE,IMAGE_SIZE),
                                                color_mode='rgb',
                                                batch_size=32,
                                                class_mode='categorical', shuffle=True)

loss, acc = finetune_model.evaluate_generator(test_generator, steps=10, verbose=1)

print('\nTest accuracy:', acc)
print('\nTest loss:', loss)
```

Figura 7.5: valutazione finali sui dati di test

```
Test accuracy: 0.9900000095367432

Test loss: 0.028084181249141693
```

Figura 7.6: log dei valori accuracy e loss

Analizzado i risultati possiamo notare che abbiamo ottenuto un accuratezza del 99% e una Loss che si aggira pocomeno al 3 %.

Una volta concluso il training del modello e salvato il relativo file h5 su file, per poterlo utilizzare sulla scheda Maixduino dovremo convertire il modello H5 in modelli conformati differenti al fine di arrivare a quello che interessa a noi nel nostro caso .kmodel.
Il modello viene prima convertito in .tflite (Fig.7.7):

```
#convert keras to tflite format
!tflite_convert  --output_file=/content/model.tflite --keras_model_file=/content/my_model.h5
```

Figura 7.7: conversine modello in tflite con nncase

Infine viene richiamato il comando ncc per convertire il modello tflite in kmodel, per farlo bisognerà però indicare la cartella dei dati di test che verrà utilizzata da ncc per essere valutata (Fig. 7.8).

```
#convert tflite to kmodel format
%cd /content/Maix_Toolbox
!./ncc/ncc -i tflite -o k210model --dataset /content/test_dir /content/model.tflite /content/model.kmodel
```

Figura 7.8: conversine modello in kmodel con nncase

Il log del codice (Fig.7.9) ci mostra il consumo delle KPU e dellea memeoria principale

```
KPU memory usage: 2097152 B
Main memory usage: 188160 B
```

Figura 7.9: consumo del modello

7.3 Deployment e test del modello sulla scheda

Per il testing del modello sulla scheda è stato creato un programma in grado di catturare l'immagine e visualizzare a schermo la label relativa alla sua predizione.
Per poter ottenere i risultati è possibile richiamare il metodo "getResult" della KPU.
Il codice utlizzato in C/C++ per determinare il risultato è illustrato nella porzione di codice seguente (Fig.7.10):

```
if(KPU.getResult((uint8_t **)&output, &output_size)){
Serial.println(labels[argmax(output, output_size)]);
Serial.println(output[0]);
Serial.println(output[1]);
Serial.println(output[2]);
Serial.println(output[3]);
Serial.println(output[4]);
float  max=0;
int index=0;
for(int h=0;h<5;h++){
  if(output[h]>max){
    max=output[h];
    index=h;
}
}
tft.println(max);
tft.println(labels[index]);
}
else
{
Serial.println("failed to predict");
}
```

Figura 7.10: codice per la visualizzazione dei risultati in C/C++

Il risultato della predizione viene determinato da un puntatore di puntatore dell'indirizzo della variabile output di tipo puntatore a float.

Questo codice è dimostrativo e viene stampato l'intero risultato composto da tutte le percentuali di predizione assegnate per ogni classe.

Ciclando il risultato si determina la label predetta e successivamente tramite lo schermo LCD viene visualizzato il risultato.

Come possiamo vedere dall'immagine sottostante, il serial Monitor ci mostra i valori predetti arrotondati (Fig. 7.11).

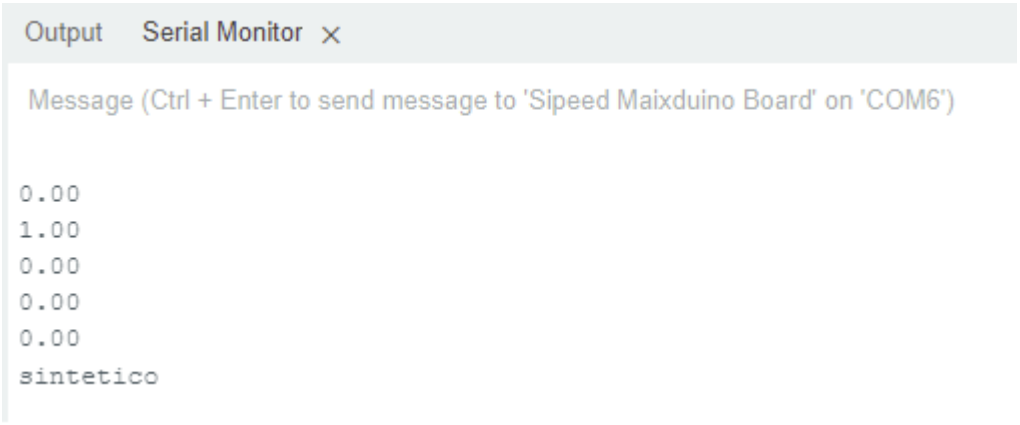

Figura 7.11: log dei risultati di una predizione

8-Conclusioni e sviluppi futuri

La rete creata è in grado di esser utilizzata su dispositivi embedded a basso consumo e i risultati riportati nel capitolo 7 hanno mostrato come il dataset creato si è dimostrato molto valido per l'addestramento della rete neurale.
Grazie all'utilizzo del transfer learning è stato possibile ottenere risultati molto incoraggianti e constituiscono una buona base di partenza per implementazini future più complesse.
Seppure i risultati ottenuti sono buoni, il margine di miglioramento è ampio, ed una futura ottimizzazione dei parametri di training unita a un arricchimento dei dati già presenti nel dataset potrebbe portare a risultati ancora migliori.

Durante la realizzazione di questo progetto però ci sono state anche delle criticità,
il sensore CMOS OV2640 seppur esser reputato versatile e molto utilizzato presenta delle limitazioni, è stato verificato che la lunga esposizione alla luce naturale crea interferenza nelle immagini creando situazioni di alterazione dei pixel, rendendo le immagini inutilizzabili.
Le foto scattate per la creazione del dataset sono state migliaia ma solo alcune centinaia sono state reputate valide.
Un'altra criticità riscontrata riguarda il porting di Maixduino su Arduino in c/c++, questa libreria è incompleta e dispone di limitate funzionalità, questo rende più compicata la configurazione delle proprietà del sensore, che a questo punto possono esser solamente modificate tramite istruzioni di lettura e scrittura sui registri.
Per quanto riguarda la scheda Maixduino, è stato riscontrato un malfunzionamento dello slot per l'SD card che rendeva impossibile la lettura e scrittura, la scheda è stata successivamente cambiata.
Concludendo, questo progetto è una buona base per poter poi creare qualcosa di più complesso, anche se con limitazioni si è riuscito nell'intento, ottenendo buone prestazioni a fronte di risultati più che incoraggianti.

9.1 Bibliografia

[CUL43]
McCulloch & Pitts Publish, A Logical calculus of the ideas immanent in nervous activity, 1943

[ROS58]
Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), 386–408. https://doi.org/10.1037/h0042519

[MAY19]
Mayank Banoula, "What is Perceptron: A Beginners Guide for Perceptron", 2019, https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron

[ENG17]
Engin Pekel,"A COMPREHENSIVE REVIEW FOR ARTIFICAL NEURAL NETWORK APPLICATION TO PUBLIC TRANSPORTATION",2017, https://www.researchgate.net/figure/Feed-forward-and-recurrent-ANN-architecture_fig1_315111480

[SAG17]
SAGAR SHARMA,"Activation Functions in Neural Networks",2017,https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[HUM18]
humanativaspa,"Reti neurali feed-forward – CNN – Convolutional Neural Network", 2018,https://humanativaspa.it/reti-neurali-feed-forward-cnn-convolutional-neural-network/

[HUM18]

humanativaspa,"Reti neurali feed-forward – CNN – Convolutional Neural Network", 2018,
https://humanativaspa.it/reti-neurali-feed-forward-cnn-convolutional-neural-network/

[GEO06]
Geoffrey E. Hinton, "A Fast Learning Algorithm for Deep Belief Nets", LETTER , 2006, pp.28

[JAC10]
Jackie Fenn,"Hype Cycle for Emerging Technologies", Gartner, 2010,

[GAR22]
Gartner ,Hype Cycle for Supply Chain Strategy,2022,https://www.gartner.com/en/newsroom/press-releases/2022-09-12-gartner-hype-cycle-for-supply-chain-strategy-shows-supply-chain-resilience-at-peak-of-inflated-expectations

[DIE22]
Diego Velez,"Depthwise-Separable convolutions in Pytorch",2022,https://faun.pub/depthwise-separable-convolutions-in-pytorch-fd41a97327d0

[ATU18]
Atul Pandey,"Depth-wise Convolution and Depth-wise Separable Convolution",2018, https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec

[JAV19]
JavaPoint, "Gradient Descent in Machine Learning", 2019, https://www.javatpoint.com/gradient-descent-in-machine-learning

[AND17]
Andrew G. Howard, Menglong Zhu, Bo Chen, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", 2017, https://arxiv.org/abs/1704.04861v1

[EDU22]
Eduardo Corpeño,"An Introduction to RISC-V—Understanding RISC's Open ISA", 2022, https://www.allaboutcircuits.com/technical-articles/introductions-to-risc-v-instruction-set-understanding-this-open-instruction-set-architecture/

[SIP19]
Sipeed, "Sipeed Maixduino Datasheet v1.0",Sipeed Technology, 2019, https://cxem.net/ckfinder/userfiles/a879191a668fd1d6c525814f0fccb5e3/files/MAIXDUINO/Sipeed_Maixduino_Datasheet_V1_0.pdf

[OMN06]
Omnivision,"OV2640 Datasheet V1.6",2006,https://www.uctronics.com/download/cam_module/OV2640DS.pdf

[BAR20]
Barth Development,"RGB565 Color Picker",2020,http://www.barth-dev.de/online/rgb565-color-picker/

[SIT13]
Sitronix, "ST7789V Datasheet V1",2013 ,https://pdf1.alldatasheet.com/datasheet-pdf/view/1132511/SITRONIX/ST7789V.html

[MIC20]
Michael Larabel,"Linux 5.7 Begins Landing Support For The Kendryte K210 Dual-Core RISC-V SoC",RISC-V,2020, https://www.phoronix.com/news/Linux-5.7-RISC-V

[HAR20]
Harley Davidson Regua,"Introducing Transfer Learning as Your Next Engine to Drive Future Innovations", 2020, https://medium.datadriveninvestor.com/introducing-transfer-learning-as-your-next-engine-to-drive-future-innovations-5e81a15bb567

[SHA20]
Sharon Y. Li,"Automating Data Augmentation: Practice, Theory and New Direction",2020,http://ai.stanford.edu/blog/data-augmentation/

[JAS21]
Jason Brownlee,"How to Choose an Activation Function for Deep Learning", 2021,https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/