UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI **INGEGNERIA**

Corso di Laurea triennale in **Ingegneria Elettronica**

IMPLEMENTAZIONE DI UNA RETE NEURALE QUANTIZZATA PER LA CLASSIFICAZIONE DI PEDONI FINALIZZATA ALL'INTEGRAZIONE SU SISTEMA EMBEDDED

IMPLEMENTATION OF A QUANTIZED NEURAL NETWORK FOR PEDESTRIAN CLASSIFICATION AIMED AT INTEGRATION ON AN EMBEDDED SYSTEM

Relatore:

Prof. **Laura Falaschetti**

Tesi di Laurea di:

Luca Iacussi

A.A. **2022** / **2023**

INDICE

INTRODUZIONE

L'Intelligenza Artificiale (AI) nasce con l'avvento dei computer e la sua data di nascita viene fissata nel 1956. L'Intelligenza Artificiale è un ramo dell'informatica che si occupa dello studio della programmazione e progettazione di sistemi, sia hardware che software, che permettono alle macchine di adottare un sistema di apprendimento molto simile a quello del cervello umano. In particolare, tali macchine sono in grado di acquisire caratteristiche prettamente umane come, ad esempio, le percezioni visive, spazio-temporali e decisionali. Inizialmente questi sistemi erano creati per essere in grado di effettuare ragionamenti logici, spesso legati al mondo della matematica, come ad esempio la dimostrazione di alcuni teoremi partendo da determinate informazioni. Negli anni seguenti sono stati progettati sistemi in grado di risolvere problemi logici sempre più complessi, grazie allo studio e allo sviluppo di nuove tecnologie.

Durante la seconda metà degli anni Sessanta, il "nuovo" obiettivo era di ricercare soluzioni a problematiche più vicine alla realtà dell'uomo, problemi di vita quotidiana, le cui soluzioni possano variare a seconda dell'evoluzione di determinati parametri durante il processo di elaborazione delle informazioni acquisite. Lo scopo della ricerca del tempo era di cercare di

riprodurre software e macchine che potessero ragionare e prendere delle decisioni in base all'analisi di diverse possibilità.

Nel 1969 si ebbe un importante passo avanti per la ricerca sull'Intelligenza artificiale, fondato sulla programmazione di sistemi che, grazie ad una serie di informazioni di base, erano in grado di produrre soluzioni specifiche per determinati scenari. A partire da questi anni fino ad oggi, la ricerca sull'Intelligenza Artificiale conobbe un importante sviluppo, e allargò anche i propri ambiti geografici.

Un esempio importante di applicazione dei "sistemi intelligenti" sono i sistemi utilizzati sui veicoli, che, in questo modo, sono in grado di guidare senza che vi sia un conducente umano al volante. Tali applicazioni hanno raggiunto, attraverso studi e ricerche, dei gradi di sicurezza sempre più elevati, grazie anche all'uso di apparecchi come sensori e telecamere che, come occhi e orecchie umane, sono in grado di percepire tutto ciò che avviene durante la guida. Partendo dalle informazioni ricevute tramite queste apparecchiature il sistema risulta essere in grado di prendere decisioni, in relazione alle necessità che si presentano, e quindi ad effettuare manovre di sicurezza.

I sistemi di Intelligenza Artificiale più utilizzati sono le reti neurali, delle strutture basate su livelli. Queste reti, una volta addestrate, sono in grado di

effettuare previsioni per quanto concerne l’obiettivo richiesto al modello stesso.

Il progetto è basato sullo studio di una rete neurale di partenza, progettata per rilevare e classificare oggetti presenti nelle immagini in ingresso alla rete stessa. La rete considerata è una rete neurale convoluzionale, nello specifico con architettura U-Net. Il codice è stato scritto in TensorFlow, piattaforma end-to-end per l'apprendimento automatico. È stata utilizzata l’API Keras, scritta in Python e in grado di funzionare su TensorFlow. La rete iniziale ha delle ottime prestazioni, ma un’occupazione di memoria molto elevata e un tempo di risposta piuttosto lungo. Lo scopo dello studio è applicare una quantizzazione alla rete, per migliorare questi parametri, per poi analizzare la differenza di prestazioni tra la prima e la seconda rete. L’obiettivo principale è ottenere un’occupazione di memoria limitata ed una velocità di risposta particolarmente breve.

Lo scritto è stato strutturato in più parti: una prima sezione è dedicata a dei richiami della teoria dell’AI e di deep learning, concentrandosi principalmente sulle reti neurali convoluzionali. Verrà poi accennata della teoria di base della quantizzazione e degli ambienti e piattaforme utilizzate per la scrittura del codice della rete. In seguito, verrà presentato l’esperimento effettuato: la struttura della rete, il dataset utilizzato per la fase di apprendimento del

modello, la tecnica di quantizzazione utilizzata e il modo in cui è stata applicata alla rete stessa. Verranno poi studiati i risultati ottenuti dalle due differenti reti, quantizzata e non, attraverso i quali è possibile trarre conclusioni per quanto concerne l'effetto della quantizzazione sulla rete di partenza.

Capitolo 1

BASI DI INTELLIGENZA ARTIFICIALE, DEEP LEARNING E RETI NEURALI CONVOLUZIONALI

1.1 Intelligenza artificiale

L'Intelligenza Artificiale (AI) è un campo della tecnologia che si occupa della creazione e progettazione di sistemi utilizzati per la risoluzione di problemi di vario genere. L'AI si divide in due categorie principali.

L'AI debole, detta anche AI ristretta, è un'AI addestrata e orientata ad eseguire attività specifiche. L'intelligenza artificiale debole è alla base della maggior parte delle applicazioni di AI che ci circonda oggi: abilita alcune applicazioni molto robuste e i veicoli autonomi. In questo caso viene riprodotto il comportamento del cervello umano, ma esclusivamente per svolgere ciò per cui sono stati progettati i modelli. Al di fuori di tale ambito di funzionamento, queste macchine non hanno un potere decisionale.

L'AI forte è composta da intelligenza artificiale generale (AGI) e superintelligenza artificiale (ASI). L'intelligenza artificiale generale (AGI), è una forma di AI secondo la quale una macchina avrebbe un'intelligenza pari a quella umana. Ciò significa che la macchina possiede una coscienza, ed acquisisce capacità di risoluzione di problemi. Si tratta di una riproduzione

del comportamento della mente umana non solo per quanto concerne la risoluzione di problemi o capacità di apprendimento, ma nel suo intero. Negli ultimi anni, grazie a ricerca e sviluppo, sono stati creati dei modelli di AGI, non ancora messi in commercio, che sono veri e propri robot che si comportano come esseri umani: sono in grado di compiere azioni fisiche pari a quelle dell'uomo, ma sono anche dotati di una coscienza, di emozioni. Dunque, possono risultare molto utili, ad esempio, nel caso di persone sole e/o con difficoltà, in quanto sarebbero in grado di dare sostenimento fisico e morale.

I sistemi e programmi di Intelligenza artificiale sono progettati, come affermato in precedenza, per simulare il comportamento del cervello umano. Il comportamento dell'uomo, quando ha necessità di trovare una soluzione ad uno specifico problema, si basa su tre parametri fondamentali, ovvero una conoscenza non sterile, una coscienza che permetta di prendere decisioni non solo secondo la logica e la capacità di risolvere problemi in maniera differente a seconda dei contesti e delle condizioni (di ogni tipo) nei quali ci si trova.

Per raggiungere lo scopo citato, vengono utilizzate reti neurali ed algoritmi, definiti ed applicati a differenti situazioni. Questo ha permesso ai sistemi intelligenti di migliorare sempre più le diverse capacità di apprendimento e, di conseguenza, anche decisionali. In tal senso la ricerca, negli anni, si è basata

principalmente sullo sviluppo di algoritmi che fossero in grado di imitare comportamenti differenti in relazione all'applicazione per la quale sono progettati e agli stimoli ambientali. Tali algoritmi, inseriti all'interno di sistemi intelligenti, permettono loro di prendere decisioni in base al contesto in cui operano. Nel caso degli algoritmi connessi ai sistemi intelligenti dei veicoli, ad esempio, un'automobile senza conducente può decidere, in situazioni di pericolo, se sterzare o frenare a seconda della situazione, ovvero in relazione alle informazioni processate dalla rete, inviate dai vari sensori.

Negli ultimi anni l'ambito di Intelligenza Artificiale ha conosciuto un importante sviluppo, grazie al continuo progresso tecnologico, che permette di progettare sistemi sempre più complessi, in grado di compiere compiti sempre più sofisticati.

1.2 Deep learning

Il machine learning è una branchia dell'intelligenza artificiale prevede l'utilizzo di dati e algoritmi per permettere ad un sistema di imitare il modo in cui gli esseri umani apprendono informzioni.

Il deep learning è un sottoinsieme del machine learning. Si tratta di una rete neurale con tre o più livelli, compresi quelli di input e di output. Queste reti neurali hanno il compito di simulare il funzionamento del cervello umano: data una grande quantità di dati al sistema, quest'ultimo sarà in grado di imparare dalle informazioni elaborate, così da poter svolgere il compito a lui assegnato. L'elemento alla base delle reti neurali è il "livello"; maggiore sarà il numero di livelli, maggiore sarà la precisione della rete.

Il deep learning si distingue dal classico machine learning per il tipo di dati con cui lavora e per le modalità con le quali apprende. Gli algoritmi di machine learning utilizzano dati strutturati ed etichettati per fare previsioni. Gli algoritmi di deep learning, invece, possono acquisire ed elaborare dati non strutturati, come testi e immagini, ed automatizzano l'estrazione di componenti. Ad esempio, si può immaginare di avere un insieme di foto di animali domestici e di volerle categorizzare per "gatto", "cane", "criceto" ed altre specie. Gli algoritmi di deep learning sono in grado di determinare quali

caratteristiche (le orecchie, muso, coda, eccetera) sono più importanti per contraddistinguere le diverse specie di animali, per poi classificarle.

Machine learning e deep learning sono anche in grado di effettuare diversi tipi d'apprendimento, che possono essere suddivisi in apprendimento controllato e apprendimento non controllato. Il primo utilizza dataset etichettati per categorizzare o fare previsioni; ciò richiede un intervento umano per etichettare correttamente i dati di input. L'apprendimento senza supervisione, al contrario, non richiede dataset etichettati, ma rileva pattern nei dati, raggruppandoli in funzione delle caratteristiche che li distinguono.

Per riprodurre i ragionamenti tipici dell'essere umano sono utilizzati reti neurali ed algoritmi. Questo ha permesso ai sistemi intelligenti di migliorare sempre più le loro capacità di apprendimento e, di conseguenza, anche decisionali. La ricerca si è basata principalmente sullo sviluppo di algoritmi in grado di imitare comportamenti differenti in relazione all'applicazione d'utilizzo ed agli stimoli ambientali. Tali algoritmi, inseriti all'interno di sistemi intelligenti, permettono loro di prendere decisioni in base al contesto in cui operano. Nel caso degli algoritmi connessi ai sistemi intelligenti dei veicoli, ad esempio, un'automobile senza conducente determina le manovre da effettuare, a seconda delle informazioni ricevute dalla rete, che rappresentano ciò che "vede" il veicolo.

Il sistema di apprendimento su cui si fondano le reti neurali permette loro di ottenere una conoscenza di base e una conoscenza appresa attraverso l'esperienza, non sterile. Così il sistema aggiorna e affina la propria conoscenza, migliorando le proprie capacità decisionali e, quindi, le proprie prestazioni. Si può facilmente notare come ci sia un chiaro parallelismo con ciò che succede per gli esseri umani: l'uomo, dovendo effettuare decisioni per il maggior tempo della sua giornata, migliora tale abilità attraverso ciò che ha già vissuto, ovvero le esperienze passate. Così, tenendo in considerazione ciò che ha affrontato precedentemente, è in grado di effettuare ragionamenti e prendere le scelte migliori. Per riassumere il concetto si può dire che il sistema viene allenato: la rete svolge il compito per la quale è progettata, ma con le sole conoscenze iniziali. Così, durante questa fase di "train", il modello ha il compito di immagazzinare nuove informazioni, attraverso le quali è in grado ampliare la propria capacità decisionale e migliorare le prestazioni.

Il deep learning è utilizzato in applicazioni e servizi di intelligenza artificiale che migliorano l'automazione, per svolgere compiti sia analitici che fisici. Questa tecnologia è alla base di prodotti o servizi di uso quotidiano (assistenti digitali, sistemi rilevamento frodi con carte di credito) e di applicazioni più sofisticate, come ad esempio sistemi di rilevamento di oggetti per la guida autonoma.

1.3 Reti neurali convoluzionali

Le reti neurali di deep learning sono progettate per imitare il comportamento del cervello umano attraverso una combinazione di input di dati, pesi e distorsione. Questi elementi, lavorando insieme, permettono alla rete di riconoscere e classificare gli oggetti all'interno delle immagini che vengono date in ingresso alla rete.

Le reti neurali profonde presentano una struttura composta da più livelli di nodi interconnessi. Ogni livello ha il compito di incrementare i risultati ottenuti dal precedente, in modo tale da migliorare la previsione e la categorizzazione degli oggetti. Questo processo di progressione di elaborazioni di dati attraverso la rete viene chiamato “propagazione in avanti”.

Tra i vari livelli che costituiscono una rete neurale profonda, abbiamo due livelli particolarmente importanti, ovvero il livello di input e il livello di output. Entrambi questi livelli sono detti “visibili”, in quanto sono quelli attraverso i quali si ha l’interazione tra rete e uomo. Il livello di input è il punto in cui il modello di deep learning acquisisce i dati da elaborare, ovvero le immagini; il livello di output è il punto in cui viene effettuata la previsione finale e la classificazione. Si può notare come, già considerando la sola struttura della rete, questa somigli a quella del cervello umano, pensando di

sostituire i livelli della rete con sezioni dell'organo umano, in quanto entrambi sono interconnessi tra loro, e l'elaborazione dei dati avviene attraverso gli "strati" che compongono il modello.

Un altro importante processo che avviene all'interno delle reti neurali viene definito "backpropagation". Utilizza algoritmi, come la discesa del gradiente, per calcolare gli errori di previsione e, in relazione ad essi, permette di regolare pesi e distorsioni della funzione attraversando a ritroso i livelli per addestrare il modello. Questa combinazione di processi permette dunque alla rete di diventare sempre più precisa, quindi di ottenere prestazioni sempre migliori.

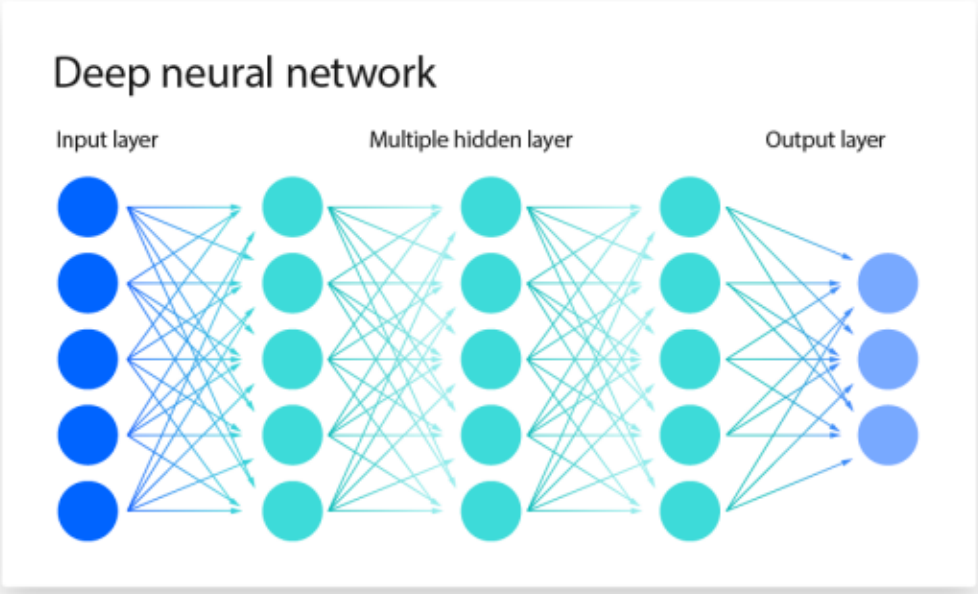

Figura 1 Semplice rappresentazione di una rete neurale a più livelli

Esistono diversi tipi di reti neurali, ma due tra questi sono quelli più importanti e maggiormente utilizzati.

Le reti neurali ricorrenti (RNN), invece, sono utilizzate maggiormente nelle applicazioni di riconoscimento del linguaggio naturale e di riconoscimento vocale.

Le reti neurali convoluzionali (CNN), invece, vengono utilizzate principalmente nelle applicazioni di visione artificiale e classificazione delle immagini, in quanto sono in grado di rilevare caratteristiche all'interno di un'immagine, permettendo così di rilevare e classificare gli oggetti.

Le reti neurali convoluzionali (CNN) si contraddistinguono dalle altre reti neurali grazie alle loro migliori prestazioni con immagini, input vocali e segnali audio. Le CNN si ispirano alla struttura biologica di una corteccia visiva, che contiene disposizioni di cellule semplici e complesse, che si attivano in relazione alle sottoregioni di un campo visivo, definite campi recettivi. Una volta ottenuti i risultati di questo studio, i neuroni in uno strato convoluzionale si connettono alle sottoregioni degli strati precedenti. I neuroni in ogni strato sono disposti 3D, trasformando un input 3D in un output 3D. Le unità nascoste (neuroni) in ogni strato apprendono combinazioni non lineari degli input originali, attraverso il processo denominato “estrazione delle caratteristiche”. Le attivazioni apprese da un

livello diventano gli input per il livello successivo. Una volta che questo processo è stato effettuato per ogni livello della rete, le caratteristiche apprese diventano gli input per il classificatore alla fine della CNN.

Le reti neurali convoluzionali contengono tre tipi di livello principali, ovvero:

- Livello convoluzionale
- Livello di pooling
- Livello completamente connesso (FC, Fully-connected)

Oltre a questi livelli più significativi è doveroso citare altri tipi di livelli: il livello ReLu ed il livello di pooling. A questi vanno aggiunti i livelli di input e di output.

Un livello di input di immagini ha il compito di passare le immagini alla rete ed applicare la normalizzazione dei dati. Le dimensioni di un'immagine corrispondono all'altezza, alla larghezza e al numero di canali di colore di dell'immagine presa in considerazione, corrispondenti ai valori RGB della stessa. Per un'immagine in scala di grigi, il numero di canali è uno, mentre per un'immagine a colori è tre.

Il livello convoluzionale è il primo livello di una rete CNN. I livelli convoluzionali possono essere seguiti da altri livelli convoluzionali. Ad ogni livello, aumenta la complessità della rete, così come aumenta la porzione

dell'immagine che viene identificata. I primi livelli si concentrano su funzioni semplici per il riconoscimento degli oggetti, ad esempio i colori e i contorni. Mentre i dati dell'immagine avanzano attraverso livelli più profondi della CNN, vengono riconosciuti elementi o forme più grandi e specifiche. Una volta che i dati dell'immagine sono passati attraverso tutti i livelli della rete, la stessa sarà in grado di riconoscere gli oggetti presenti nell'immagine.

Il livello convoluzionale è l'elemento costitutivo per eccellenza di una CNN, in quanto è il punto in cui si verifica la maggior parte dei calcoli. È caratterizzato da pochi componenti: dati di input, un filtro e una mappa delle funzioni. Il filtro, definito anche kernel, ha il compito di spostarsi attraverso i campi recettivi dell'immagine, verificando in questo modo la presenza della funzione. Questo processo viene definito "convoluzione".

Il rilevatore di funzioni è un array bidimensionale di pesi, che rappresenta parti dell'immagine. Quindi, il filtro viene applicato ad un'area dell'immagine e viene calcolato un prodotto di punti tra i pixel di input e filtro. Il risultato dell'operazione viene poi inserito all'interno di un array di output. Successivamente, il filtro si sposta di un passo, cambiando l'area d'immagine considerata, e ripete il processo fino a quando il kernel non avrà attraversato immagine nella sua interezza. L'output finale della serie di prodotti di punti

dall'input e dal filtro viene chiamato “mappa delle funzioni” o “mappa di attivazione”.

Dopo ogni operazione di convoluzione, la rete CNN applica una trasformazione ReLU (Rectified Linear Unit) alla mappa delle funzioni, introducendo così la non linearità nel modello. Un livello ReLU esegue un'operazione di soglia su ciascun elemento, per cui qualsiasi valore di input inferiore a zero viene impostato su zero, senza modificare la dimensione del suo input.

Come precedentemente affermato, il livello convoluzionale iniziale può essere seguito da un altro livello di convoluzione. Quando ciò accade, la struttura della CNN può diventare gerarchica: i livelli successivi possono vedere i pixel all'interno dei campi ricettivi dei livelli che li precedono. Ad esempio, ipotizziamo di dover determinare se un'immagine contiene una moto. Una moto può essere immaginata come un insieme di parti. È composta da un telaio, un manubrio, due ruote, due fanali, eccetera. Ogni singola parte della moto compone un modello di basso livello nella rete neurale, ma la loro combinazione rappresenta un modello di alto livello, creando una gerarchia di funzioni all'interno della CNN.

I livelli di pooling, definiti anche sottocampionamento, eseguono la riduzione della dimensionalità, andando a ridurre il numero di parametri nell'input.

Similmente al livello convoluzionale, l'operazione di pooling applica un filtro sull'intero input. Al filtro, però, in questo caso non è associato alcun peso.

Possono essere definite due principali tipologie di pooling:

- Pooling massimo: mentre viene applicato sull'input, il filtro seleziona il pixel con il valore massimo da inviare all'array di output. In generale è utilizzato più frequentemente rispetto al pooling medio.
- Pooling medio: mentre viene applicato sull'input, il filtro calcola il valore medio all'interno del campo ricettivo da inviare all'array di output.

Si può facilmente intuire come il livello di pooling determini la perdita di molte informazioni, ma, in realtà offre anche una serie di vantaggi alla CNN. Innanzitutto, permette di ridurre la complessità della rete: viene diminuito il numero di connessioni e questo comporta anche una diminuzione del numero di parametri e pesi condivisi. Tutto ciò permette di ottenere una rete più rapida e leggera in termini di memoria.

Il nome del livello completamente connesso (FC) è particolarmente azzeccato per la descrizione delle caratteristiche di tale strato della rete: ogni suo nodo nel si connette direttamente a un nodo nel livello precedente. Questo livello ha il compito di eseguire l'attività di classificazione in base alle funzioni

estratte tramite i livelli precedenti e i loro filtri. Mentre i livelli convoluzionali e di pooling utilizzano le funzioni ReLu, i livelli completamente connessi solitamente sono basati su una funzione di attivazione softmax per classificare gli input in modo appropriato. Nel caso dei problemi di classificazione, solitamente, lo strato completamente connesso è seguito da un livello softmax e, poi, da un livello di classificazione.

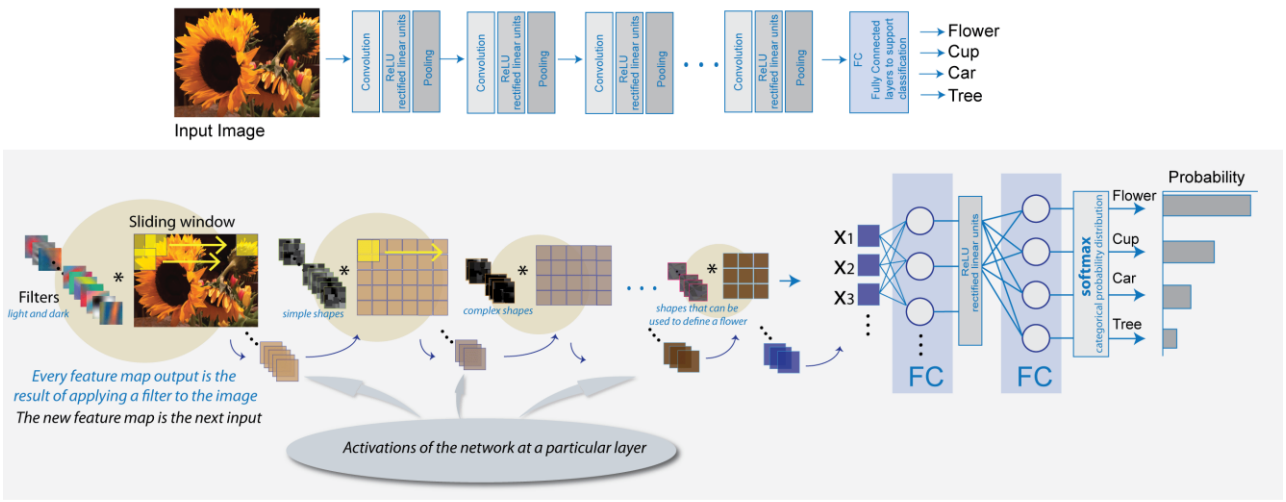

Figura 2 Schematica rappresentazione del funzionamento generale di una CNN

Per l'esperimento che verrà preso in considerazione viene utilizzata una rete CNN basata su architettura U-Net. La U-Net è fondata sulla struttura encoder-decoder, ed è composta da due differenti percorsi. Il primo viene definito "contraction path" ed ha la funzione di encoder, ovvero di captare il contesto

dell’immagine tramite una mappa; è composto da vari llivelli convoluzionali e maxpooling. Il secondo è chiamato “extension path” ed ha la funzione di decoder, uguale e contraria rispetto alla precedente. Questa operazione permette di ottenere una localizzazione precisa degli oggetti che compongono l’immagine. Durante la fase di encoder le dimensioni delle immagini vengono ridotte, per analizzare e classificare ogni pixel, ignorando la sua posizione all’interno dell’immagine. Nella fase di decoder, al contrario, le loro dimensioni vengono aumentate, fino ad ottenere la risoluzione del dato in ingresso, per raggruppare i pixel che formano un determinato oggetto che sarà localizzato, e poi classificato, all’interno dell’immagine di output.

La U-Net tradizionale è composta da 4 layer convoluzionali per ogni path, dunque 8 totali. Inoltre, ne è presente un altro, utilizzato per passare dalla fase di encoder a quella di decoder, che va sommato ai precedenti.

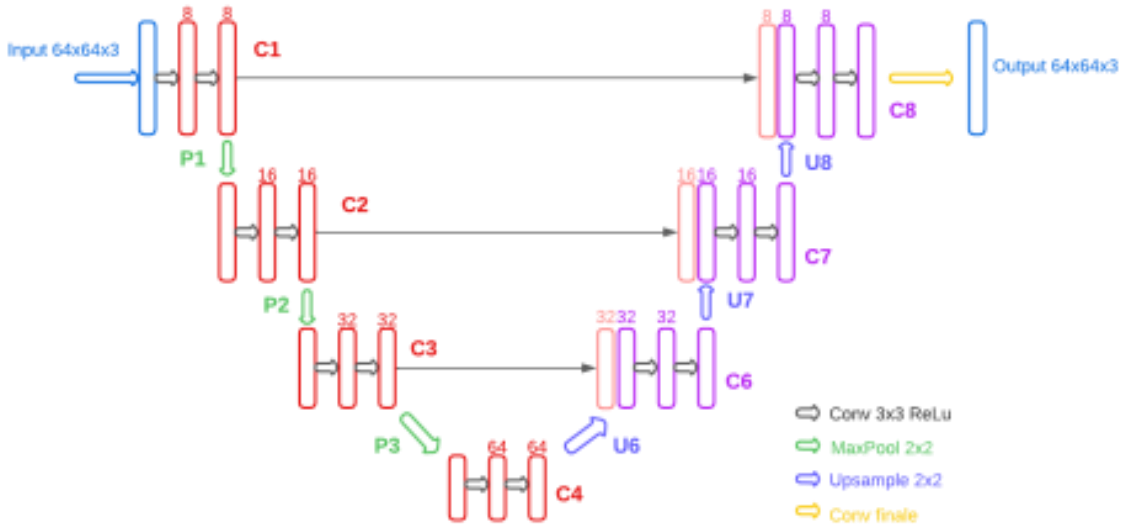

Figura 3 Struttura di una rete CNN basata su architettura U-Net

1.4 Quantizzazione

In funzione dello studio proposto è doveroso accennare delle basi di teoria della quantizzazione, in questo caso applicata a reti neurali convoluzionali. In termini puramente teorici la quantizzazione è un'operazione che permette di rappresentare un numero infinito di valori attraverso l'utilizzo di un set finito di valori. La rappresentazione ottenuta sarà caratterizzata da una perdita di informazione, in quanto gli infiniti valori iniziali verranno approssimati ad un numero limitato di "livelli di valore". Quest'ultimi vengono definiti attraverso l'utilizzo delle possibili combinazioni dei bit utilizzati per rappresentare i valori iniziali. Maggiore è il numero di bit usati, maggiore sarà il numero di valori quantizzati definibili, e, di conseguenza, migliore sarà la precisione della rappresentazione.

La quantizzazione solitamente viene applicata alle reti CNN per ottimizzarle, ottenendo dei sistemi in grado di elaborare le informazioni più velocemente. Inoltre, semplificando la rete, permette di ottenere una riduzione di memoria occupata dalla stessa. Allo stesso tempo, però, il modello potrebbe presentare delle prestazioni peggiori, in particolare per quanto concerne la precisione delle previsioni effettuate. Occorre dunque procedere per tentativi, cercando di ottenere dei buoni compromessi tra benefici e svantaggi. Le caratteristiche della rete vanno poi valutate tenendo in considerazione anche la futura

applicazione del modello. Nel caso di modelli progettati per applicazioni di guida autonoma, per ovvi motivi di sicurezza, il tempo di risposta della CNN deve essere particolarmente breve e la precisione molto elevata.

L'applicazione dell'operazione di quantizzazione ad una rete neurale adottata nello studio proposto, consiste nella sostituzione di alcuni dei suoi livelli con dei livelli che svolgono le stesse funzioni, ma sono quantizzati. Un livello quantizzato è un livello che effettua operazioni quantizzate ed attiva funzioni quantizzate, in termini semplici, semplificate rispetto a quelle inziali.

Per entrare nello specifico dell'operazione utilizzata per ottimizzare la rete di partenza, è necessario entrare nel dettaglio della scrittura del codice della rete. Il codice è stato scritto in linguaggio Python, utilizzando le librerie Tensorflow. La vera e propria stesura del codice viene effettuata sulla piattaforma online Google Colab. TensorFlow è una piattaforma end-to-end usata per l'apprendimento automatico. Opera su array multidimensionali o tensori, implementando operazioni matematiche standard sui tensori e molte altre operazioni specifiche per il deep learning. Nel caso in cui si abbia la necessità di effettuare l'esecuzione di calcoli di dimensioni particolarmente grandi, TensorFlow può utilizzare hardware dell'acceleratore come le GPU, che consente di effettuare le operazioni molto più rapidamente. TensorFlow

può essere usato come una qualsiasi libreria Python, ma fornisce anche importanti strumenti per ottimizzare le prestazioni del modello.

Per la creazione del modello proposto viene utilizzata l'API Keras. Quest'ultima è un'API di deep learning scritta in Python e in grado di funzionare su TensorFlow. Keras, innanzitutto, è facile da utilizzare, in quanto presenta un'interfaccia semplice e coerente. Inoltre, si presenta come facilmente modulabile, flessibile, e componibile, collegando insieme blocchi configurabili. Le strutture dati principali di Keras sono i livelli ed i modelli. Il tipo più semplice di modello è il "modello sequenziale", definito come una pila lineare di strati differenti. Nel caso si abbia la necessità di creare architetture maggiormente complesse, occorre utilizzare l'API funzionale Keras.

Per effettuare la quantizzazione dei livelli Keras della rete neurale presa in considerazione viene utilizzata un'estensione di Keras, ovvero il QKeras. QKeras è un'estensione di quantizzazione di Keras che fornisce una sostituzione immediata per alcuni livelli. In particolar modo verranno sostituiti quelli che creano il maggior numero di parametri e di livelli di attivazione, e che eseguono operazioni aritmetiche, in modo da poter creare in modo veloce una versione profondamente quantizzata della rete. QKeras permette anche di quantizzare il modello nella sua interezza, ma, in generale,

è maggiormente utilizzato il processo di quantizzazione livello per livello. L'obiettivo principale di tale estensione è di permettere allo sviluppatore di ottenere una rete quantizzata, partendo da una non quantizzata, in maniera molto intuitiva, semplice e rapida.

Capitolo 2

IMPLEMENTAZIONE DELLA RETE NEURALE QUANTIZZATA

2.1 Struttura rete neurale

L'esperimento che verrà preso in considerazione consiste nello studiare gli effetti della quantizzazione applicata ad una rete neurale progettata per riconoscere gli oggetti all'interno delle immagini che essa acquisisce in input. La rete neurale di partenza al quale viene fatto riferimento, è stata definita, ottimizzata, studiata ed approfondita nell'articolo di Falaschetti L. et al [1]. La base dello studio sarà dunque fondata su questa rete, di cui, facendo riferimento all'articolo citato, verrà presentata una descrizione.

La rete di partenza è una rete neurale CNN, basata su architettura U-Net. Come precedentemente affermato, l'architettura U-Net si fonda sull'utilizzo di quattro livelli convoluzionali per ognuno dei due percorsi nella quale può essere divisa. In più, va aggiunto un ulteriore livello che permette di passare dalla fase di decoder alla fase di encoder. Tale progettazione della rete permette di ottenere un modello con 1.9 milioni di parametri, e presenta delle prestazioni molto elevate. La rete è stata poi ottimizzata, così da ottenerne una forma semplificata, più veloce e meno pesante in termini di memoria occupata.

La modifica operata sulla rete consiste nell'eliminare due livelli convoluzionali, uno per percorso, simmetrici tra loro: il quarto ed il sesto della struttura. Difatti quest'ultimi contengono il maggior numero di filtri e contribuiscono alla formazione di numerosi parametri. Inoltre, si è scelto anche di diminuire il numero di filtri di ogni layer convoluzionale, dimezzandone il numero. Effettuando le modifiche citate, è possibile ottenere una rete con 120 mila parametri, una precisione leggermente inferiore e un'occupazione di memoria molto minore rispetto al modello iniziale. Inoltre, vengono effettuate delle modifiche anche per quanto riguarda il numero di classi utilizzate per la categorizzazione, passando da 30 ad 8. Lo studio che verrà presentato si concentrerà su tale rete ottimizzata e, partendo da essa, verranno valutati gli effetti della quantizzazione alla quale sarà sottoposta.

Come affermato nella fase introduttiva, esistono differenti algoritmi che sono implementati per permettere alla rete neurale di riconoscere gli oggetti presenti nelle immagini di input. L'algoritmo che viene utilizzato nel caso della rete CNN considerata è denominato "Semantic Segmentation", segmentazione semantica. La segmentazione delle immagini è un processo di analisi delle immagini end-to-end, che divide un'immagine digitale in un determinato numero di segmenti e classifica le informazioni contenute in ogni sezione dell'immagine. L'attività di segmentazione dell'immagine attribuisce

etichette ai singoli pixel, così da contraddistinguere i confini e le forme specifici dei diversi oggetti e regioni dell'immagine, classificandoli sfruttando informazioni come colore, contrasto, posizionamento all'interno dell'immagine ed altri attributi.

Per valutare le prestazioni della rete CNN che si ottiene, è necessario definire i parametri più significativi che caratterizzano il modello.

L'accuracy ha il compito di indicare la precisione della classificazione effettuata dal modello. Rappresenta, dunque, il parametro che permette di valutare la "pura" qualità della rete, senza tener conto di altri parametri. Un'altra importante caratteristica da tenere in considerazione è la size del modello, ovvero la sua occupazione di memoria. Questo parametro risulta essere molto importante, in quanto il modello viene progettato per essere poi caricato su di una board o altri dispositivi solitamente dotati di una disponibilità di memoria limitata. Il tempo di risposta della rete deve essere valutato anche in relazione all'applicazione del modello stesso: in base all'ambito nel quale sarà utilizzato, può essere necessario ottenere una risposta dalla rete in tempi molto brevi, come, ad esempio, nel caso di applicazioni per guida autonoma. Viene definita anche una "funzione di perdita" (loss function), che ha il compito di misurare la differenza tra la

previsione effettuata dal modello e la classificazione realistica degli oggetti nelle immagini di input.

Infine, viene definito un altro parametro molto importante, ovvero l'IoU (Intersection over Union), che misura la percentuale di sovrapposizione tra due riquadri di delimitazione degli oggetti sulla loro area combinata. L'intervallo di valori che può assumere va da 0 ad 1, ovvero, rispettivamente, sovrapposizione minima e sovrapposizione completa.

Per quanto riguarda la funzione di perdita si è scelto di utilizzare la "jaccard loss", in quanto permette di ottenere un alto valore del parametro IoU (migliore segmentazione), a discapito, però dell'accuracy.

Inoltre, per la creazione del modello si è scelto di usare l'ottimizzatore Adam optimizer, un metodo di discesa del gradiente stocastico basato sulla stima adattiva dei momenti del primo e del secondo ordine. Tale metodo permette di risolvere problemi di grandi dimensioni; risulta anche essere molto efficiente e richiede una piccola quantità di memoria.

I livelli della rete sono principalmente convoluzionali, con attivazione ReLu. Per la definizione della struttura viene utilizzata la funzione Keras che permette di creare un livello qualsiasi, ovvero tf.keras.layers.(). Il tipo di livello che si intende creare deve essere poi specificato, come messo in

evidenza dal codice, a seguito della chiamata della funzione. Tra le parentesi che seguono il tipo di layer indicato vengono inseriti i suoi attributi, le sue caratteristiche., che possono essere scelte, entro certi limiti, dallo sviluppatore.

Il lavoro di creazione di una rete neurale correttamente funzionante è diviso in alcune fasi fondamentali: la definizione della struttura del modello, la fase di importazione del dataset di immagini, la fase di train, di validation e di test.

Di seguito è riportata la sezione di codice nella quale viene definita la struttura della rete.

```
# Input layer

inputs = tf.keras.layers.Input((img_rows, img_cols, img_channels))

# Contraction path

c1 = tf.keras.layers.Conv2D(8, (3,3), activation='relu',
kernel_initializer='he_normal', padding='same')(inputs) # 16=feature
dimensions, 3x3=kernel, the initializers gives the starting weights
that must be updated
c1 = tf.keras.layers.Dropout(0.1)(c1)
c1 = tf.keras.layers.Conv2D(8, (3,3), activation='relu',
kernel_initializer='he_normal', padding='same')(c1)
```

```
p1 = tf.keras.layers.MaxPooling2D((2,2))(c1)

c2 = tf.keras.layers.Conv2D(16, (3,3), activation='relu',
kernel_initializer='he_normal', padding='same')(p1)
c2 = tf.keras.layers.Dropout(0.1)(c2)
c2 = tf.keras.layers.Conv2D(16, (3,3), activation='relu',
kernel_initializer='he_normal', padding='same')(c2)
p2 = tf.keras.layers.MaxPooling2D((2,2))(c2)

c3 = tf.keras.layers.Conv2D(32, (3,3), activation='relu',
kernel_initializer='he_normal', padding='same')(p2)
c3 = tf.keras.layers.Dropout(0.1)(c3)
c3 = tf.keras.layers.Conv2D(32, (3,3), activation='relu',
kernel_initializer='he_normal', padding='same')(c3)
p3 = tf.keras.layers.MaxPooling2D((2,2))(c3)

# Middle path

c4 = tf.keras.layers.Conv2D(64, (3,3), activation='relu',
kernel_initializer='he_normal', padding='same')(p3)
c4 = tf.keras.layers.Dropout(0.1)(c4)
c4 = tf.keras.layers.Conv2D(64, (3,3), activation='relu',
kernel_initializer='he_normal', padding='same')(c4)

# Expansive path

u6 = tf.keras.layers.Conv2DTranspose(32, (2,2), strides=(2,2),
padding='same')(c4)
```

```
u6 = tf.keras.layers.concatenate([u6,c3])
c6 = tf.keras.layers.Conv2D(32, (3,3), activation='relu',
kernel_initializer='he_normal', padding='same')(u6)
c6 = tf.keras.layers.Dropout(0.2)(c6)
c6 = tf.keras.layers.Conv2D(32, (3,3), activation='relu',
kernel_initializer='he_normal', padding='same')(c6)

u7 = tf.keras.layers.Conv2DTranspose(16, (2,2), strides=(2,2),
padding='same')(c6)
u7 = tf.keras.layers.concatenate([u7,c2])
c7 = tf.keras.layers.Conv2D(16, (3,3), activation='relu',
kernel_initializer='he_normal', padding='same')(u7)
c7 = tf.keras.layers.Dropout(0.2)(c7)
c7 = tf.keras.layers.Conv2D(16, (3,3), activation='relu',
kernel_initializer='he_normal', padding='same')(c7)

u8 = tf.keras.layers.Conv2DTranspose(8, (2,2), strides=(2,2),
padding='same')(c7)
u8 = tf.keras.layers.concatenate([u8,c1])
c8 = tf.keras.layers.Conv2D(8, (3,3), activation='relu',
kernel_initializer='he_normal', padding='same')(u8)
c8 = tf.keras.layers.Dropout(0.2)(c8)
c8 = tf.keras.layers.Conv2D(8, (3,3), activation='relu',
kernel_initializer='he_normal', padding='same')(c8)

# output layer
```

```
outputs = tf.keras.layers.Conv2D(n_classes, (1,1),
activation='softmax')(c8)
```

Il primo livello è il livello di input, che ha il compito di accettare in ingresso le immagini che dovranno essere processate dal modello. L'ultimo livello presente è invece quello di output, che svolge l'attività di classificazione degli oggetti presenti nelle immagini in ingresso al modello, ovvero è lo strato attraverso cui si ottiene la risposta della rete.

È possibile riconoscere anche la struttura della U-Net definita precedentemente: dopo il primo livello di input, abbiamo due percorsi differenti, contraddistinti dai commenti nel codice, ovvero quello di encoder e di decoder. Nel punto medio tra le due parti si trova un livello che permette di passare da una fase all'altra.

Mentre la fase di importazione del dataset verrà affrontata in seguito, è doveroso descrivere brevemente le fasi di train, validation e test, che sono alla base di ogni applicazione di machine learning. La fase di train è la fase di apprendimento della rete, attraverso il set di immagini dedicati a tale operazione presenti nel dataset. Grazie all'elaborazione delle informazioni acquisite durante tale fase, la rete sarà in grado di riconoscere i vari oggetti

presenti nell'immagine che viene processata, applicando la classificazione richiesta.

La fase validation si occupa di validare i risultati ottenuti durante il training. Ciò si effettua dando in ingresso al modello delle "nuove" immagini che non conosce, con assieme le rispettive maschere, su cui la rete effettuerà la previsione di riconoscimento e classificazione. In base alle prestazioni ottenute durante tale fase può essere valutato l'effetto della fase del train sulla CNN, ovvero se è pronta per poter funzionare correttamente. Tale valutazione viene effettuata considerando la differenza tra ciò che è realmente presente nell'immagine e la risposta della rete.

In seguito, viene effettuata la fase di test della rete. In questo caso il modello viene testato, ovvero ne vengono valutate le prestazioni attraverso lo studio degli indicatori più importanti: precisione (accuracy), valore di perdita (val. loss), indice IoU, size e tempo di risposta (time/step). Risulta molto difficile riuscire ad ottenere valori ottimi per ogni indice considerato: l'obiettivo è quello di ottenere un equilibrio tra gli indici. Nonostante questo, è importante anche considerare l'applicazione futura del modello: in questo caso il nostro obiettivo principale è avere un'occupazione di memoria il più piccola possibile, un'accuracy comunque accettabile (altrimenti non farebbe il suo dovere) e un tempo di risposta breve.

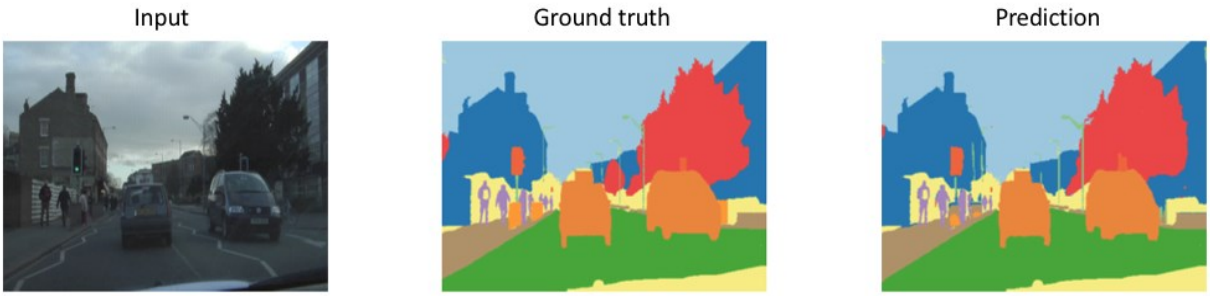

Figura 4 Nella figura sono mostrate, in ordine, immagine di input, relativa maschera, e previsione effettuata dalla rete.

2.2 Dataset ed elaborazione immagini

Il dataset utilizzato per la rete oggetto di studio è denominato Cityscapes , ed è composto da due cartelle: la prima contiene le immagini originali, la seconda le relative maschere. Le immagini hanno una risoluzione pari a 1024x2048 e sono circa 5000 suddivise in train, test e validation set. Le maschere corrispondono inizialmente a circa 30 classi, utilizzate per classificare ciò che è presente nelle immagini, che rappresentano situazioni tipiche che si possono incontrare durante la guida. Come precedentemente affermato, però, il numero di classi è passato da 30 ad 8 a seguito dell'ottimizzazione della rete precedentemente descritta. Le etichette ottenute in uscita dopo tali operazioni hanno dimensioni pari a 64x64x17, caratteristiche di una rappresentazione categorica.

Le cartelle di train, test e validation sono a loro volta divise con il nome delle varie città da cui sono state prese tali immagini. Le cartelle con all'interno le maschere, inoltre, contengono quattro diversi tipi di maschera; per la realizzazione del task richiesto vengono utilizzate quelle a colori, identificate dal tag "gtFine_color.png".

I dati sono stati importati tramite Google Drive, sul quale erano state precedentemente caricate le cartelle del dataset. A causa di limitazioni di memoria è risultato necessario considerare un numero minore di città per le

immagini di train. La funzione utilizzata per l'importazione è tf.keras.utils.image_dataset_from_directory().

Per poter ottenere performance elevate e una bassa occupazione di memoria è necessario lavorare con immagini con risoluzione di 64x64x3. Adottando questa strategia si ha una minore occupazione di memoria ma, allo stesso tempo, le prestazioni non vengono peggiorate.

Effettuando l'importazione delle immagini si hanno:

- 1592 immagini di train;
- 200 immagini di test;
- 500 immagini di validation.

A causa di un'imperfezione presente nella cartella delle maschere si è dovuto suddividere il dataset di train in un 90% destinato al training e un 10% destinato al testing.

Dopo aver estratto le maschere di train, test e validation, deve essere effettuata un'operazione di mapping tra colore dei pixel e classe corrispondente. In questo modo gli oggetti presenti nell'immagine, composti da insiemi di pixel, vengono identificati e classificati con le categorie possibili attraverso l'utilizzo di colori diversi.

Le prestazioni del modello ottenuto possono essere valutate considerando i valori degli indicatori di accuracy, IoU, quantità di memoria occupata, valore di perdita (val. loss) e del tempo di risposta della rete. I valori di tali parametri, relativi alla rete di partenza, sono riportati di seguito.

SIZE	ACCURACY	IoU	VAL. LOSS	TIME/STEP
1.554 MB	0.8597	0.5611	0.5545	412

I valori di precisione (accuracy), dell'indice IoU e del valore di perdita (val. loss) risultano essere molto buoni: ciò significa che la rete funziona correttamente. Sono presenti, però, dei parametri che presentano valori meno buoni, in particolare per quanto riguarda la memoria occupata e la velocità di risposta del modello.

Di seguito sono riportati i grafici dei parametri di precisione e val. loss della rete di partenza, ottenuta come descritto.

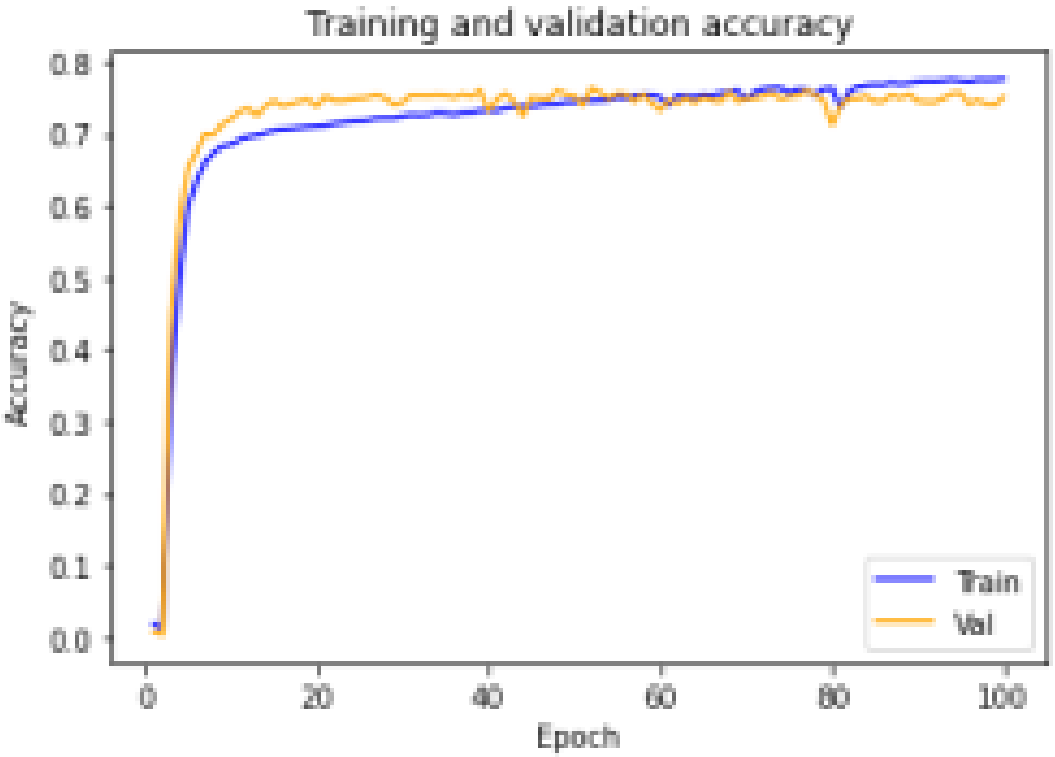

Figura 5 Grafico Accuracy

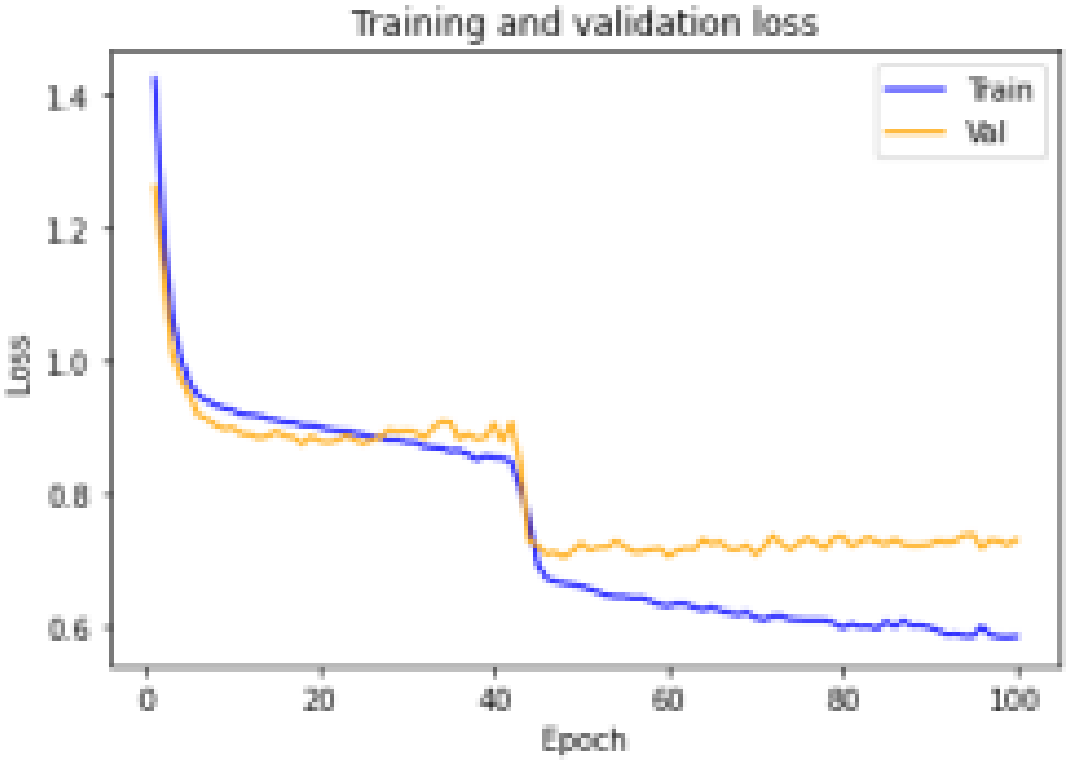

Figura 6 Grafico Val. loss

2.3 Quantizzazione della rete

Per ovviare alle imperfezioni che caratterizzano la rete precedentemente descritta, si sceglie di effettuare una quantizzazione della stessa, grazie all'utilizzo dell'estensione di Keras QKeras. Il QKeras consente di intervenire sulla rete livello per livello, permettendo dunque di mantenere invariati alcuni dei layers della rete di partenza, e sostituendone solo alcuni. Nel caso dello studio descritto si è deciso di intervenire sui livelli convoluzionali presenti nella struttura della rete, in quanto sono quelli che contengono il maggior numero di filtri e parametri. Il resto della rete, invece, rimane invariato. Applicando la quantizzazione con l'estensione QKeras, si effettua una vera e propria sostituzione del livello originale con un livello quantizzato, del quale vanno specificate le caratteristiche.

Di seguito è riportata la sezione di codice che contiene la descrizione della struttura della rete quantizzata.

```
# Input layer

inputs = tf.keras.layers.Input((img_rows, img_cols, img_channels))


# Contraction path
c1 = QConv2D(8, (3,3), activation='relu',
kernel_quantizer="quantized_bits(16,0, alpha=1)",
padding='same')(inputs) # 16=feature dimensions, 3x3=kernel, the
quantizers gives the starting weights that must be updated
c1 = tf.keras.layers.Dropout(0.1)(c1)
```

```
c1 = QConv2D(8, (3,3), activation='relu',
kernel_quantizer="quantized_bits(16,0,alpha=1)", padding='same')(c1)
p1 = tf.keras.layers.MaxPooling2D((2,2))(c1)

c2 = QConv2D(16, (3,3), activation='relu',
kernel_quantizer="quantized_bits(16,0,alpha=1)", padding='same')(p1)
c2 = tf.keras.layers.Dropout(0.1)(c2)
c2 = QConv2D(16, (3,3), activation='relu',
kernel_quantizer="quantized_bits(16,0,alpha=1)", padding='same')(c2)
p2 = tf.keras.layers.MaxPooling2D((2,2))(c2)

c3 = QConv2D(32, (3,3), activation='relu',
kernel_quantizer="quantized_bits(16,0,alpha=1)", padding='same')(p2)
c3 = tf.keras.layers.Dropout(0.1)(c3)
c3 = QConv2D(32, (3,3), activation='relu',
kernel_quantizer="quantized_bits(16,0,alpha=1)", padding='same')(c3)
p3 = tf.keras.layers.MaxPooling2D((2,2))(c3)

# Middle path

c4 = QConv2D(64, (3,3), activation='relu',
kernel_quantizer="quantized_bits(16,0,alpha=1)", padding='same')(p3)
c4 = tf.keras.layers.Dropout(0.1)(c4)
c4 = QConv2D(64, (3,3), activation='relu',
kernel_quantizer="quantized_bits(16,0,alpha=1)", padding='same')(c4)

# Expansive path

u6 = tf.keras.layers.Conv2DTranspose(32, (2,2), strides=(2,2),
padding='same')(c4)
u6 = tf.keras.layers.concatenate([u6,c3])
c6 = QConv2D(32, (3,3), activation='relu',
kernel_quantizer="quantized_bits(16,0,alpha=1)", padding='same')(u6)
c6 = tf.keras.layers.Dropout(0.2)(c6)
c6 = QConv2D(32, (3,3), activation='relu',
kernel_quantizer="quantized_bits(16,0,alpha=1)", padding='same')(c6)

u7 = tf.keras.layers.Conv2DTranspose(16, (2,2), strides=(2,2),
padding='same')(c6)
u7 = tf.keras.layers.concatenate([u7,c2])
c7 = QConv2D(16, (3,3), activation='relu',
kernel_quantizer="quantized_bits(16,0,alpha=1)", padding='same')(u7)
c7 = tf.keras.layers.Dropout(0.2)(c7)
c7 = QConv2D(16, (3,3), activation='relu',
kernel_quantizer="quantized_bits(16,0,alpha=1)", padding='same')(c7)
```

```
u8 = tf.keras.layers.Conv2DTranspose(8, (2,2), strides=(2,2),
padding='same')(c7)
u8 = tf.keras.layers.concatenate([u8,c1])
c8 = QConv2D(8, (3,3), activation='relu',
kernel_quantizer="quantized_bits(16,0,alpha=1)", padding='same')(u8)
c8 = tf.keras.layers.Dropout(0.2)(c8)
c8 = QConv2D(8, (3,3), activation='relu',
kernel_quantizer="quantized_bits(16,0,alpha=1)", padding='same')(c8)

# output layer

outputs = tf.keras.layers.Conv2D(n_classes, (1,1),
activation='softmax')(c8)
```

I livelli convoluzionali sono stati sostituiti con livelli che svolgono le stesse funzioni, ma quantizzati, attraverso l'utilizzo del termine "*QConv2D*". All'interno delle parentesi sono contenuti gli attributi che descrivono le caratteristiche del livello stesso. Viene utilizzata la funzione *kernel_quantizer=" quantized_bits(16, 0, alpha=1)"* per definire la quantizzazione applicata. In questo caso le operazioni aritmetiche utilizzate per elaborare le informazioni, così come quelle dei filtri, sono quantizzate a 16 bit. Il valore del parametro alpha viene impostato, per semplicità, ad 1.

2.4 Risultati sperimentali

Per analizzare le prestazioni del modello quantizzato, è riportata di seguito una tabella contenente i valori dei parametri più significativi della rete quantizzata, come per la rete di partenza.

SIZE	ACCURACY	IoU	VAL. LOSS	TIME/STEP
0.53 MB	0.8555	0.5453	0.5420	54

Per valutare in maniera migliore gli effetti dell'operazione effettuata, evidenziandone i benefici, inseriamo i valori dei parametri delle due reti in un'unica tabella.

	SIZE	ACCURACY	IoU	VAL.LOSS	TIME/STEP
N.Q.	1.55 MB	0.8597	0.5611	0.5545	412
Q.	0.53 MB	0.8555	0.5453	0.5420	54

Dai valori presenti in questa tabella è facile notare come l'utilizzo di livelli convoluzionali quantizzati comporti un'importante diminuzione della quantità di memoria occupata dalla rete e del tempo necessario alla rete per effettuare la previsione.

Di seguito vengono riportati i grafici dell'accuracy e del val. loss della rete neurale quantizzata.

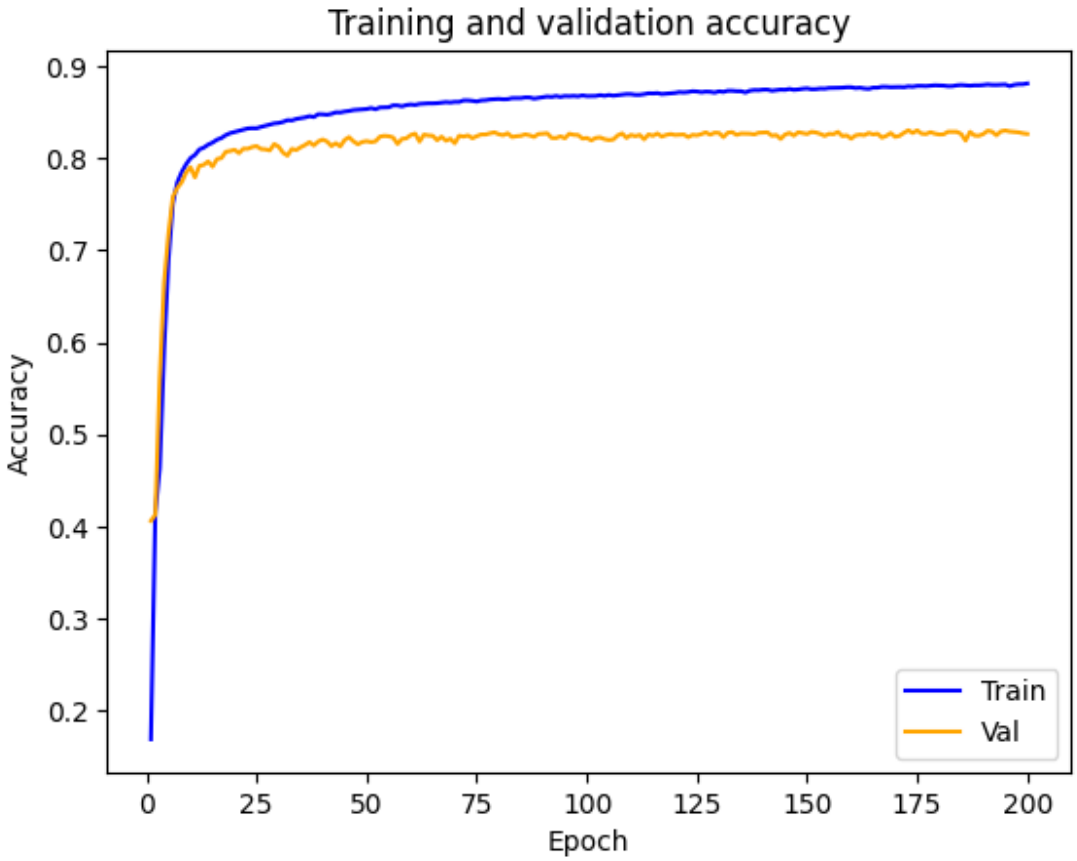

Figura 7 Grafico Accuracy rete quantizzata

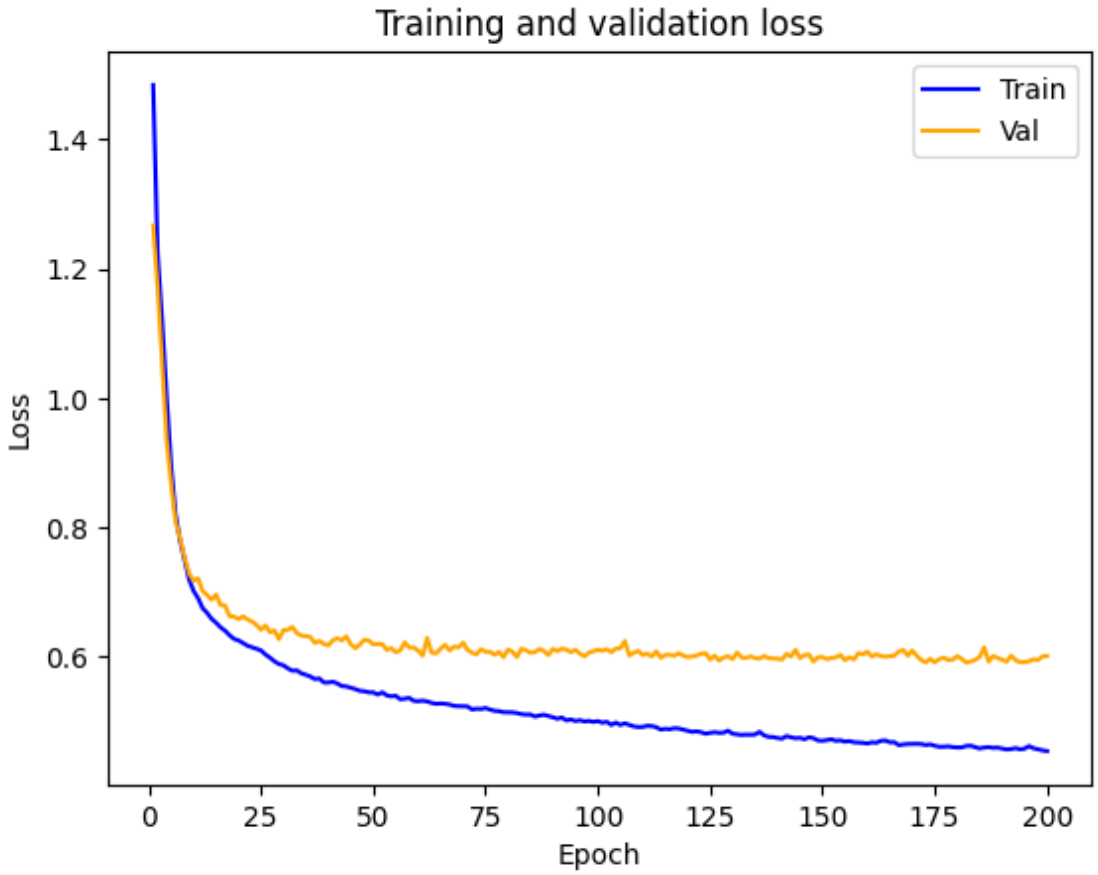

Figura 8 Grafico val. loss della rete quantizzata

CONCLUSIONI

Per effettuare un'analisi completa dei risultati ottenuti occorre studiare i valori dei parametri maggiormente significativi. L'obiettivo principale dell'esperimento effettuato era quello di ottenere una rete neurale quantizzata, partendo da una CNN iniziale, che presentasse un'occupazione di memoria bassa ed una velocità di risposta particolarmente elevata. La quantizzazione effettuata sulla rete oggetto di studio possiede queste caratteristiche, di conseguenza è possibile affermare che gli obiettivi prefissati sono stati raggiunti. In particolare, l'occupazione di memoria è diminuita di un rapporto di circa un terzo.

Facendo un'analisi più ampia, è importante notare come, attraverso l'operazione di quantizzazione, si riescano ad ottenere questi attributi, senza avere un'importante diminuzione del livello di prestazione della rete. Difatti, facendo riferimento alla tabella contenente i valori dei parametri delle due reti, i valori della precisione (accuracy), dell'indicatore IoU, e della perdita (val. loss) risultano essere molto simili tra loro. Ciò significa che tali valori sono equivalenti per quanto concerne il corretto funzionamento della rete; dunque, permette alla nuova rete di effettuare previsione tanto buone quanto la precedente.

Una bassa occupazione di memoria permette di ottenere un modello che, in futuro, può essere caricato su microcontrollori dotati anche di memoria limitata.

Un tempo di risposta breve, è strettamente necessario per le applicazioni di guida autonoma, per permettere al veicolo di effettuare manovre col giusto tempismo per evitare, ad esempio, incidenti o situazioni di pericolo. Inoltre, è doveroso specificare, che a questo tempo di risposta del modello, deve essere poi sommato il tempo impiegato per trasmettere alla parte meccanica del veicolo la previsione del modello.

Lo studio effettuato ha dato dunque i risultati sperati. Il modello proposto può quindi essere considerato valido per l'applicazione su veicoli a guida autonoma. Per ovvie ragioni di sicurezza, però, il modello dovrà essere ulteriormente testato, in quanto tale ambito di applicazione richiede che le reti effettuino delle previsioni molto precise e "sicure".

Lo studio effettuato, in termini più pratici ed applicativi, è finalizzato al caricamento della rete quantizzata su un microcontrollore, con lo scopo di realizzare un'applicazione su sistema embedded, ovvero sistemi elettronici di elaborazione a microcontrollore progettati per uno specifico utilizzo. Nel caso del modello ottenuto, verrebbe utilizzato per applicazioni di veicoli a guida autonoma. La rete ha il compito di passare informazioni riguardo ciò che è

presente davanti al veicolo, grazie all'utilizzo di sensori visivi (videocamere) implementati sul veicolo, identificando e classificando tutti gli oggetti o persone presenti. In relazione alle previsioni effettuate dal modello l'automobile sarà in grado di effettuare le manovre adatte alle situazioni che si presentano.

Lo studio proposto ha permesso al modello di essere più facilmente utilizzabile. Infatti, i microcontrollori sono dispositivi che possono avere una bassa disponibilità di memoria: una rete con un'occupazione troppo elevata potrebbe avere difficoltà ad essere poi effettivamente caricata su tali dispositivi e, quindi, ad essere utilizzata.

Riguardo allo studio proposto e all'applicazione futura della rete, è necessario specificare che il modello non è stato caricato su microcontrollore. Ma, analizzando i risultati ottenuti e le conclusioni effettuate sullo studio, il modello sembra avere tutte le caratteristiche per poter essere utilizzato in maniera efficiente su sistemi embedded, nel particolare, per applicazioni su auto a guida autonoma.

BIBLIOGRAFIA E SITOGRAFIA

[1] Falaschetti, L., Bruschi, S., Alessandrini, M., Biagetti, G., Crippa, P., & Turchetti, C. (2023). An U-Net Semantic Segmentation Vision System on a Low-Power Embedded Microcontroller Platform.Procedia Computer Science, 225, 4473-4482.

https://www.ibm.com/it-it/topics/artificial-intelligence

https://www.ibm.com/it-it/topics/deep-learning

https://www.ibm.com/it-it/topics/convolutional-neural-networks

https://it.mathworks.com/help/deeplearning/ug/introduction-to-convolutional-neural-networks.html

https://it.mathworks.com/help/deeplearning/gs/create-simple-deep-learning-classification-network.html

https://it.mathworks.com/solutions/image-video-processing/semantic-segmentation.html

https://github.com/google/qkeras/tree/master

https://www.tensorflow.org/overview?hl=it

https://keras.io/getting_started/

RINGRAZIAMENTI

Desidero ringraziare tutta la mia famiglia, che mi è stata vicina durante gli anni di studio presso l'Università, supportandomi e sopportandomi.

Tengo a ringraziare anche il mio gruppo di amici che, da sempre, mi sono stati accanto, permettendomi di superare periodi più o meno complicati del percorso di studi e della vita.

Desidero ringraziare la professoressa Falaschetti, che mi ha dato l'opportunità di effettuare un'attività di tirocinio molto interessante, dalla quale è stato possibile poi realizzare lo studio proposto nella tesi.