POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Sviluppo e test di algoritmi di deep learning su hardware ottimizzato per machine learning

Relatore:
Prof. Enrico Masala

Candidato:
Giorgio Loi

Anno Accademico 2019/2020

Ringraziamenti

Con questa tesi si conclude il mio percorso in Ingegneria Informatica al Politecnico di Torino.
Ringrazio il professore Enrico Masala per avermi guidato durante questo lavoro.
Un grande ringraziamento va alla mia famiglia, in particolare a mia madre e le mie sorelle, senza i loro sforzi e il loro supporto non avrei potuto concludere questo bel percorso.
Grazie a tutto il mio splendido gruppo di amici che mi ha accompagnato in tutti questi anni e non mi ha mai fatto sentire solo.
In conclusione, voglio ringraziare Manuela che, sia come amica, sia come fidanzata ha sempre creduto in me e confortato nei momenti più difficili.

Indice

Ringraziamenti 3
Sommario 7
Capitolo 1 Introduzione 9
1.1 Intelligenza artificiale, machine learning e deep learning 10
1.2 Computer vision con il deep learning 13
1.3 Decentralizzazione del Machine Learning: 14
Capitolo 2 Reti neurali convoluzionali 15
2.1 Cosa è una rete neurale artificiale? 15
2.1.1 Architettura 16
2.1.2 Rappresentazione matematica 17
2.2 Addestramento delle reti 17
2.3 Il paradigma di train-test 18
2.4 Rete neurale convoluzionale 19
2.4.1 Panoramica architettura 20
2.4.2 Livello convoluzionale 21
2.4.3 Livello di Pooling 23
2.5 Normalizzazione batch 24
Capitolo 3 Neural Compute Stick & OpenVINO 26
3.1 Intel® Neural Compute Stick 2 26
3.1.2 Sviluppare con la Neural Compute Stick 30
3.2 Il toolkit Intel® OpenVINO 31
3.2.1 Model Optimizer 33
3.2.2 L'Inference Engine 34
3.2.3 Flusso di lavoro delle API dell'Inference Engine 35
3.2.4 OpenCV 37
3.3 Demo ed esempi 37
3.3.1 *simple_classifier_cpp_camera* 38
3.3.2 *mnist_calc* 39
3.3.3 *benchmark_ncs* 39

3.4 Misurare le performance per le predizioni di Deep Learning 41
3.4.1 Comparazione prestazioni tra NCS 2 e la CPU Intel Core i5-8500 42
3.4.2 Analisi risultati benchmark.....44
Capitolo 4 Visual attention detection 45
4.1 Analisi Demo.....45
4.2 Strumenti utilizzati 49
4.3 Come funziona.....49
4.4 Descrizione delle Classi della applicazione 50
4.4.1 Gestione delle predizioni.....50
4.4.2 Workflow main.cpp.....52
4.5 Modifiche aggiunte per l'estrazione dei parametri.....53
4.5.1 Sviluppo Endpoint e Heatmap.....56
4.5.2 Calcolo della heatmap 57
4.6 Test prestazioni.....58
Capitolo 5 Conclusioni e sviluppi futuri 59
Lista abbreviazioni 60
Lista delle figure.....61
Bibliografia.....63

Sommario

L’intelligenza artificiale ha avuto dei grandi progressi nel corso degli ultimi anni grazie all’avanzamento tecnologico. Tutt’ora è uno dei settori più attivi della ricerca. Ci sono una grande varietà di applicazioni che consentono di risolvere delle problematiche che prima erano impossibili da gestire. Tuttavia, la maggior parte delle applicazioni di intelligenza artificiale sviluppate al momento richiedono molta potenza computazionale, perciò si utilizzano computer desktop o server che hanno bisogno di una moderata potenza elettrica. Perciò oggi giorno si ha bisogno di sistemi scalabili e che richiedano meno risorse.

Tutta la community tecnologica condivide questa visione, per questo i maggiori produttori di hardware hanno iniziato il supporto per soluzioni per lo sviluppo di questo tipo di idee. Tra varie soluzioni, una è la serie di hardware prodotta dalla Intel, la Movidius Neural Compute Stick, per l’ottimizzazione di algoritmi di Deep Learning. La NCS si sfrutta attraverso il toolkit OpenVINO e supporta tutti i framework open-source più famosi per il Machine Learning.

Lo scopo di questa tesi è quella di riuscire a creare delle applicazioni e dare un’idea su come utilizzare algoritmi di ML su dispositivi con poca potenza computazionale con l’ausilio della Neural Compute Stick 2 della Intel. Ai fini della composizione di questa tesi, oltre alle dovute configurazioni per lavorare con la NCS 2, sono stati effettuati diversi benchmark per identificare le prestazioni e le potenzialità che potrebbe avere questo strumento.

Struttura tesi: Il capitolo 1 offre una descrizione degli obbiettivi del progetto di tesi dell’importanza oggigiorno della computer vision affiancata dal machine learning.

Nel secondo capitolo vengono introdotte le conoscenze sulle strutture delle reti neurali, in particolare sulle reti neurali artificiali convoluzionali.

Nel capitolo 3 vengono spiegati nel dettaglio gli strumenti utilizzati per la realizzazione del lavoro di testi e il loro funzionamento.

Il capitolo 4 presenta la fase di test e sviluppo delle funzionalità dell’applicazione di deep learning e dell’endpoint per la visualizzazione dei risultati, con spiegazione di tutte le sue componenti software.

Nell’ultimo capitolo ci saranno gli obiettivi raggiunti, vantaggi, svantaggi e possibili migliorie.

Capitolo 1 Introduzione

I sistemi di Deep Learning di questi tempi usano una architettura basata sul cloud con server che parallelamente processano le richieste di predizioni dalle applicazioni utente. Questo rende la collezione dei dati centralizzata, sotto il controllo dei service provider. Questo tipo di approccio porta a numerosi problemi riguardanti la privacy e la sicurezza dei dati o cattivi utilizzi. D'altra parte la ricerca e lo sviluppo della computer vision ha sfruttato i vantaggi dell'integrazione del deep learning per acquisire risultati significativi. In accordo con questo, i sistemi di rilevamento di oggetti stanno riscuotendo un grande successo realizzando applicazioni con discrete performance. Nonostante ciò queste applicazioni hanno bisogno di potenza computazionale e un consumo energetico non indifferente. Fortunatamente diverse soluzioni stanno diventando accessibili in termini sia di hardware che di software per rafforzare le applicazioni basate su deep learning in dispositivi poco performanti e a basso consumo.

Lo scopo della mia tesi è testare l'hardware e sviluppare un'applicazione adatta ad uno scenario basso costo che non richieda tanta potenza computazionale sfruttando il toolkit Intel OpenVINO e la sua Neural Compute Stick 2.

I risultati di questo lavoro potrebbero essere vantaggiosi per l'utilizzo della computer vision e del deep learning per una grande varietà di applicazioni accessibili a tutti.

1.1 Intelligenza artificiale, machine learning e deep learning

Con l'intelligenza artificiale intendiamo quelle operazioni caratteristiche dell'intelletto umano eseguite dal computer, dei costrutti matematici creati dall'intelligenza umana per eseguire un particolare task con le performance di un computer. È un conglomerato di molti campi della scienza che includono la filosofia, psicologia, le neuroscienze, l'ingegneria informatica, statistica e altri. Un approccio moderno all'IA è il Machine Learning (ML), dove gli algoritmi sono designati per leggere i dati e di apprenderne i pattern, le relazioni e trasformazioni da soli, modificandosi man mano che ricevono più informazioni su quello che stanno elaborando senza nessuna conoscenza pregressa. Il computer apprende la nuova conoscenza in vari modi: osservando l'ambiente esterno, imparando dai feedback delle sue azioni e suoi errori e da un database detto knowledge base. Per migliorare l'IA c'è bisogno di sistemi complessi, algoritmi avanzati, ottimizzati in modo tale da essere capaci di catturare informazioni nascoste del dato che normalmente non si potrebbero elaborare. Il ML esiste da diversi anni ma solo ultimamente sta diventando di importanza rilevante, questo grazie alle evoluzioni delle nuove tecnologie e dei computer che son sempre più capaci di elaborare più velocemente e sempre più grandi quantità di dati. Perciò, il ML è sempre più studiato, sempre più stanno aumentando le pubblicazioni e studi che immediatamente vengono applicate e adottate in ogni aspetto della vita, in diversi campi, come la finanza, la sanità, la computer vision, riconoscimento vocale. Queste sono due definizioni popolari del ML date da due famose persone di questo campo. La prima è proposta da Arthur Samuel, un ricercatore pioniere dell'intelligenza artificiale che ha descritto il machine learning come "*il settore di studi che fornisce al computer l'abilità al computer di apprendere senza essere esplicitamente programmato.*". Tom Mitchell, un informatico, ha fornito una definizione più moderna che è "*a computer programme is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance in tasks T, as measured by P, improves with experience E."* [1].

In generale, ogni problema di ML può essere diviso in due classificazioni: **Apprendimento supervisionato** e **apprendimento non supervisionato**. Nel primo il progettista fornisce una serie di esempi alla macchina. Ogni esempio è composto una serie di valori di input ed è accompagnato da un'etichetta in cui il progettista indica il risultato o un giudizio di valore. La macchina elabora i dati e apprende dagli esempi per individuare una funzione predittiva o una regola generale. Nel secondo l'agente non ha

esempi di input-output da testare. Non c'è nessuno che decide se un'azione è corretta oppure no, né una misura di riferimento per i feedback dell'ambiente. Può comunque apprendere l'esistenza di un legame tra alcuni fattori di input e output. In poche parole: la macchina impara dai proprio errori e dell' esperienza. La maggior parte delle applicazioni tecnologiche beneficia dal primo approccio soprattutto i metodi di deep learning.

Esiste una terza classificazione, chiamata **apprendimento per rinforzo**, nel quale un opportuno algoritmo si prefigge lo scopo di individuare un certo modus operandi, a partire da un processo d'osservazione dell'ambiente esterno. L'agente non ha esempi di input e output né una guida che gli dica il comportamento giusto da eseguire, ma ha una funzione di rinforzo associata a un obiettivo da raggiungere che gli permette di valutare i feedback delle azioni.

Dobbiamo stare attenti a non intendere come stessa cosa deep learning e machine learning, hanno un significato diverso anche se fanno entrambi parte dello studio dell'intelligenza artificiale. Il deep learning è un caso particolare del machine learning che si basa sull'utilizzo delle reti neurali artificiali con due o più strati (hidden layers), per elaborare le informazioni in modo non lineare. In questo modo è possibile evidenziare delle caratteristiche discriminanti anche da dati grezzi, come i pixel, immagini mediche o sequenze biologiche. [2]

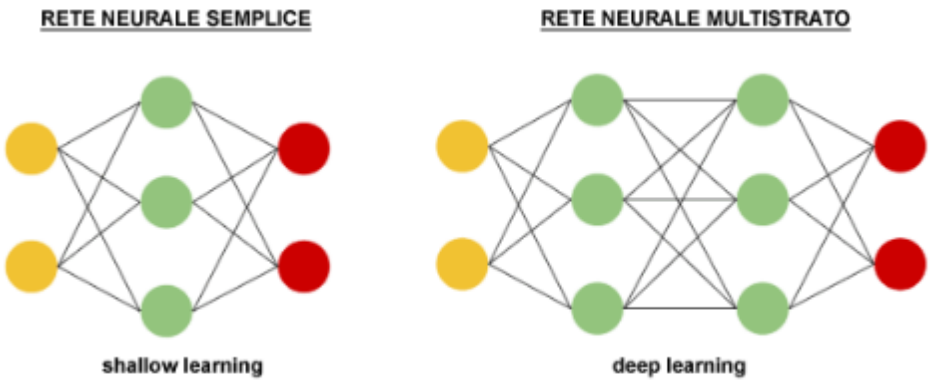

Figura 1: Differenza tra reti neurali [2]

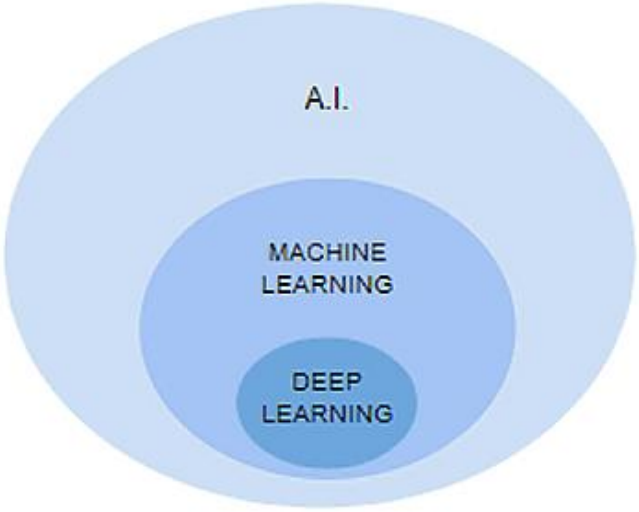

Figura 2: AI, ML e DL [2]

In generale gli algoritmi di deep learning sono ottimizzati in modo tale che le loro reti possano imparare (trovare il set migliore di regole) e predire da un insieme strutturato di dati. La rete migliora la sua precisione venendo continuamente allenata con una enorme quantità e diversità di dati. Per questo i sistemi di deep learning hanno bisogno solo di tanti dati ma anche di tantissimo tempo di training e si affidano pesantemente alle risorse computazionali. Per questa ragione, tendono a essere sviluppati in macchine molto potenti o in sistemi basati su cloud per il loro utilizzo. Tuttavia, questi tipi di setup stanno introducendo degli inconvenienti, come la richiesta di una potente connessione a bassa latenza per questo si sta desiderando sempre di più distribuire i sistemi di deep learning nei singoli dispositivi.

1.2 Computer vision con il deep learning

Il deep learning ha avuto grandi successi in varie aree della computer vision, come la classificazione delle immagini, rilevamento oggetti, la segmentazione semantica, generazione dell'immagine e stima della posa. In generale, i task di queste aree utilizzano le reti neurali convoluzionali (CNN) oppure le reti neurali ricorrenti (RNN) nel deep learning e vengono addestrate su un grande dataset di immagini a livello di pixel, perciò più è grande la dimensione dell'immagine e migliore sarà l'accuratezza ma sarà più costosa in termini di prestazioni computazionali. Sebbene raggiungano notevoli risultati, molti dei sistemi sono lenti e hanno bisogno della potenza di calcolo di molti hardware in parallelo come le GPU per processare i dati. Come risultato, molti sistemi di computer vision allo stato dell'arte fanno leva sull'hardware e della loro capacità di avere performance in real-time. Allo stesso tempo, molti aziende di hardware conoscono le tendenze del momento e hanno fatto grandi progressi in termini di supporto alla potenza di calcolo per gli hardware specializzati nel deep learning. Un esempio è la Intel, la compagnia ha introdotto la serie Neural Compute Stick (NCS), con la seconda versione uscita a fine 2018, un acceleratore hardware molto efficiente per il deep learning con a bordo la sua Vision Process Unit (VPU), che permette lo sviluppo di applicazioni su svariati hardware, affiancata dal suo potente framework capace di far lavorare insieme anche gli altri dispositivi Intel. Nvidia, oltre che rilasciare GPU sempre più potenti e che incrementano la capacità di elaborazione in parallelo, fornisce il Jetson system-on-module (SOM), computer che racchiudono tante GPU predisposte interamente per le applicazioni grafiche e di computer vision per i sistemi embedded con minori requisiti di alimentazione [3]. Anche Google contribuisce fornendo dei servizi a pagamento per usare la sua infrastruttura cloud, la Tensor Processing Unit (TPU).

Figura 3: Vari modelli di Nvidia Jetson [3]

1.3 Decentralizzazione del Machine Learning:

Convenzionalmente, i sistemi di ML sono posti in sistemi centralizzati basati su cloud dove l'utente finale fa richiesta per delle predizioni al server e aspettano poi di ricevere il risultato. Da una parte questa architettura aiuta a spostare tutte le risorse computazionali verso il server e permette scalabilità per il software a un grande range di dispositivi, dal PC ai sistemi embedded. Tuttavia questo setup è molto dipendente dalla connessione a internet e i vari protocolli di comunicazione penalizzano la latenza. Per esempio, in applicazioni che operano con uno stream video in input, può facilmente sovraccaricare la banda e il canale di comunicazione per il trasferimento dei dati con conseguente perdita di prestazioni. Per risolvere questo problema, si sta cercando di decentralizzare i processi di ML. Di questo cambiamento di paradigma ne beneficiano per esempio i sistemi di guida autonoma, sistemi molto sensibili ai ritardi di risposta, basti pensare che l'azione di frenata deve essere fatta quasi istantaneamente, le decisioni devono essere prese immediatamente. Proprio per questo la decentralizzazione dei sistemi di ML sta guadagnando molte attenzioni e ci sono diverse soluzioni fornite dai leader tecnologici del settore che venendo uniti ai sistemi in tempo reale di computer vision allo stato dell'arte e alla robotica, possono avere come risultato un potente strumento che può dare il via a tante innovazioni tecnologiche.

Capitolo 2 Reti neurali convoluzionali

2.1 Cosa è una rete neurale artificiale?

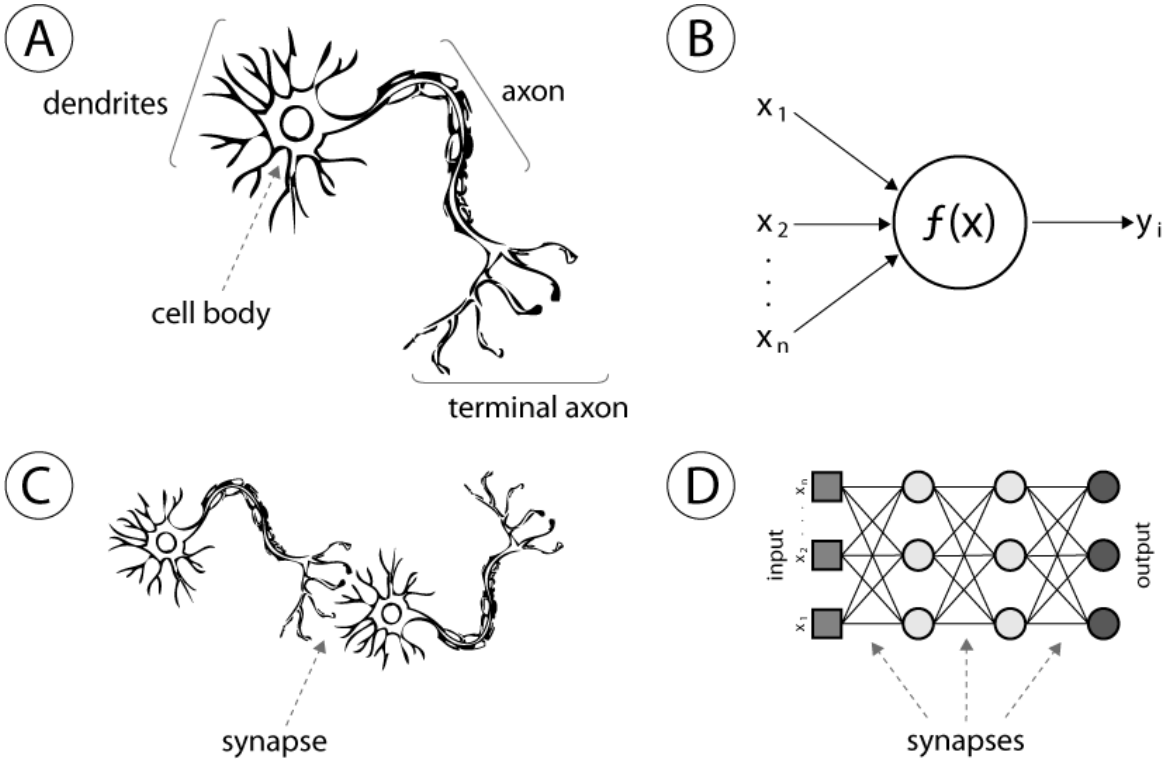

Figura 4: Comparazione tra neurone biologico e neurone artificiale [4]

Con le reti neurali artificiali (in inglese ANN, artificial neural network), normalmente chiamate anche solo come reti neurali, intendiamo dei paradigmi di processamento delle informazioni ispirati dal sistema del cervello umano con la motivazione di replicare la capacità della cognizione umana di risolvere dei task computazionali, questo approccio viene chiamato connessionismo [5]. Con l'andare avanti degli anni le reti neurali sono state sempre di più capaci di risolvere problemi impossibili per gli algoritmi standard e metodi statistici e perciò ha iniziato a riscuotere notevole successo, venendo adottati in svariati contesti e applicazioni. Biologicamente, un neurone è caratterizzato da tre regioni: il corpo cellulare, i dendriti e l'assone. Il primo è collegato con i dendriti per ricevere i segnali elettrici dagli altri neuroni. L'assone è il "collegamento" di output per gli altri neuroni e serve per spedire il segnale. L'assone ha molte terminazioni e il collegamento con altri neuroni si chiama sinapsi.

2.1.1 Architettura

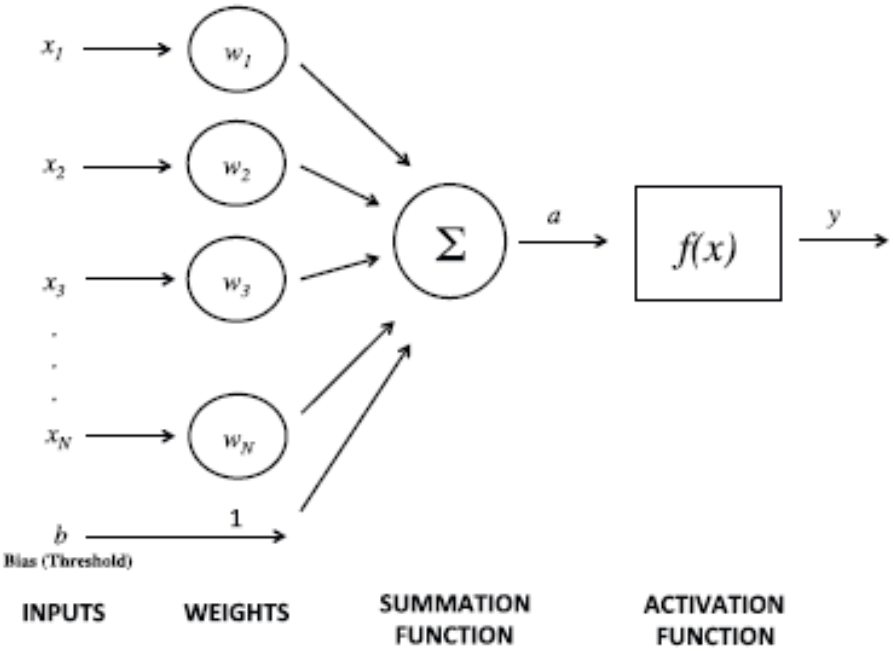

Figura 5: Neurone Artificiale [6]

Un neurone artificiale cerca di emulare il comportamento del neurone biologico. Nelle reti neurali ogni neurone corrisponde a un nodo computazionale che riceve delle informazioni da un certo numero di nodi di input (features) moltiplicati per dei pesi (weights). Le reti neurali solitamente sono composte da un set di nodi e da più livelli (layers). L'ultimo livello di solito viene considerato come lo livello di output e dipende dal tipo di problema che la rete neurale sta cercando di risolvere, se il livello ha un solo nodo di output allora è un problema di classificazione mentre se ne possiede più di uno è di regressione.

2.1.2 Rappresentazione matematica

$$a = \sum_{i=0}^{N} x_i w_i + b$$

Dove:

w - è la matrice dei pesi a due dimensioni del nodo
b – costante bias del nodo
x – matrice di input
a – matrice di output
[6]

La combinazione lineare è l'input per la cosiddetta funzione di attivazione che emette un segnale y: quest'ultima funziona in modo tale che quando il valore di a è diverso da uno stato costante, un segnale $y = f(a)$ può essere emesso ed è modellato in accordo con la funzione di attivazione f. Le funzioni di attivazione più comuni sono: Sigmoid [7], Tanh [8], ReLU [9]. È importante tenere a mente che la non-linearità delle funzioni di attivazione potenzia le reti neurali con la capacità di approssimare ogni funzione con un numero moderato di parametri.
L'esistenza della costante bias permette ai nodi di imparare alcune preferenze rispetto a un certo tipo di informazioni, a meno che non sia implementato un algoritmo di normalizzazione batch per normalizzare [10] l'output a, in quel caso il bias non sarebbe necessario. In poche parole, la rete neurale esegue una "forward operation" ovvero un processo che dato un input alla rete, vengono valutate le attivazioni di ciascun neurone, strato per strato, per poi raggiungere l'output finale [11].

2.2 Addestramento delle reti

L'addestramento di una rete neurale viene normalmente eseguita cercando di minimizzare una certa funzione di perdita (o di costo) $L(w)$, dove w rappresenta tutti i

pesi della rete neurale. Sono i parametri usati nella fase di addestramento. Dato un set di input e output, la funzione di perdita è essenzialmente una funzione dei parametri della rete e uno può minimizzare la funzione di perdita aggiustando il valore di w. Solitamente questa minimizzazione viene effettuata attraverso un'ottimizzazione con un gradiente discendente.

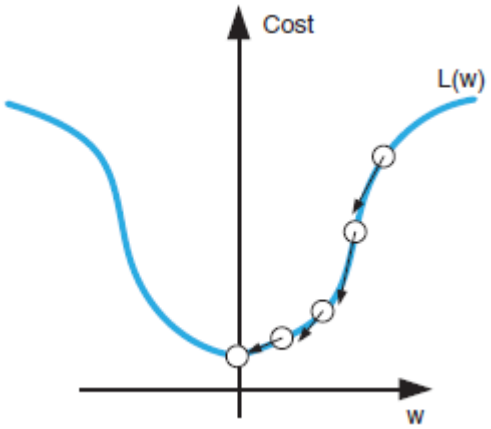

Figura 6: Illustrazione del processo del gradiente discendente. Il parametro w viene aggiustato e la funzione di perdita si muove nella direzione dove il suo valore diminuisce [11]

Il compito principale della fase di addestramento è quello di trovare i giusti pesi in modo tale da minimizzare la funzione di perdita.

2.3 Il paradigma di train-test

Quando si cerca di trovare i giusti pesi su un modello di ML bisogna fare attenzione ad evitare l'overfitting. Questo fenomeno succede quando un modello impara dai dati di addestramento "troppo" bene seguendo anche i possibili errori o rumori che potrebbe avere. Cercare di capire dai dati di test è ciò che si desidera, ma è altrettanto importante che il modello sia in grado di generalizzare per i campioni che ancora non ha visto. In questo caso riuscirebbe ad avere un comportamento migliore. Per cercare di controllare questo fenomeno sono state effettuate numerose ricerche. Un tipico approccio è quello di dividere i dati che secondo noi sono attendibili in due set: uno per la fase di addestramento e uno per la fase di test. Partendo dal primo, lo dividiamo ulteriormente

in altri due set, uno sarà l'attuale set di addestramento e l'altro sarà di validazione che sarà molto più piccolo del primo. il set di addestramento è usato per sintonizzare i pesi del modello, utilizzando il metodo di propagazione all'indietro. Quando il set è finito, le predizioni vengono eseguite sul test di validazione e vengono stimate le perdite per questo gruppo di dati. A questo punto, avviene una nuova divisione per i set di addestramento-validazione e si procede di nuovo come abbiamo visto prima. Ognuno di questi step vengono chiamate epoche. Le perdite avvenute dal set di training sono monitorate confrontandole con quelle del test di validazione per ogni epoca. cercando di prevenire situazioni dove nel primo set avviene una diminuzione della perdita mentre nel secondo aumenta. Il modo in cui vengono divisi i dati nei diversi set dipende da quando è grande il dataset e la natura del problema, ma in ogni caso la divisione dovrebbe essere bilanciata.

2.4 Rete neurale convoluzionale

Una rete neurale convoluzionale, in inglese convolutional neural network (CNN), è una classe di reti neurali artificiali che usano degli strati convoluzionali per filtrare gli input per ricavarne delle informazioni. L'operazione di convoluzione combina i dati in input, chiamati anche mappa delle feature, con un kernel di convoluzione, ovvero il filtro per formare una mappa delle feature trasformata. I filtri nei livelli convoluzionali sono modificati in base ai parametri imparati per estrarre l'informazione più utile per quel determinato task. La rete convoluzionale aggiusta automaticamente i valori per trovare la migliore feature per il task. Una CNN è capace di filtrare informazioni sia di una forma che di una texture di un oggetto quando viene confrontata con un generico task di rilevamento di oggetti quando differenti classi di oggetti tendono ad avere differenti forme ma molte volte oggetti diversi hanno la stessa forma e vengono caratterizzati da come appaiono e una CNN è molto potente da essere in grado di distinguere questi casi. Le CNN hanno dimostrato la loro efficacia in varie applicazioni di computer vision e riconoscimento vocale soprattutto per i sistemi di classificazione di immagini che includono riconoscimento facciale, rilevamento oggetti, segmentazioni oggetti. Per il processamento vocale includono rilevamento vocale, processamento del linguaggio parlato, classificazione testo, analisi testo e traduzioni linguistiche.

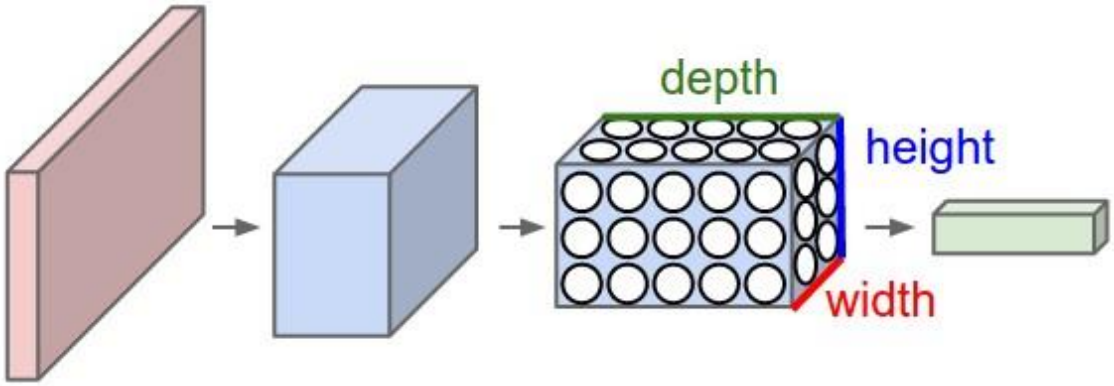

Figura 7: esempio di CNN [12]

2.4.1 Panoramica architettura

In generale, le CNN si basano sulle reti neurali artificiali e sull'idea degli strati di neuroni artificiali che trasportano le informazioni sui pesi e sui bias. Ma a differenza delle ANN i livelli di una CNN hanno i neuroni disposti in tre dimensioni: larghezza, altezza e profondità (per profondità ci riferiamo alla dimensione sul volume di attivazione, non alla profondità della rete neurale intera, infatti per profondità ci si può riferire al numero totale dei livelli di una rete).

Per esempio, le immagini di input nel dataset CIFAR-10, sono un volume di attivazioni, il volume ha dimensione 32x32x3 (larghezza, altezza e profondità rispettivamente). Come possiamo vedere nella figura in alto, i neuroni in un livello saranno connessi a solo una piccola regione dello strato, invece che in maniera completa come potevamo vedere nelle ANN. Ogni livello della CNN trasforma il volume dell'input 3D in un volume output in 3D. Nell'esempio nell'immagine lo strato rosso è l'input che contiene l'immagine, i valori di altezza e larghezza sono le dimensioni dell'immagine e la profondità sarà di solito 3 (canali RGB).

L'architettura delle CNN è progettata in modo tale che la codifica di certe particolari proprietà renda più efficiente il processamento dei dati a più dimensioni. I livelli nascosti, (hidden layers) sono una combinazione di livelli di convoluzione, livelli di pooling e i tradizionali livelli delle reti neurali (fully connected layer). [12]

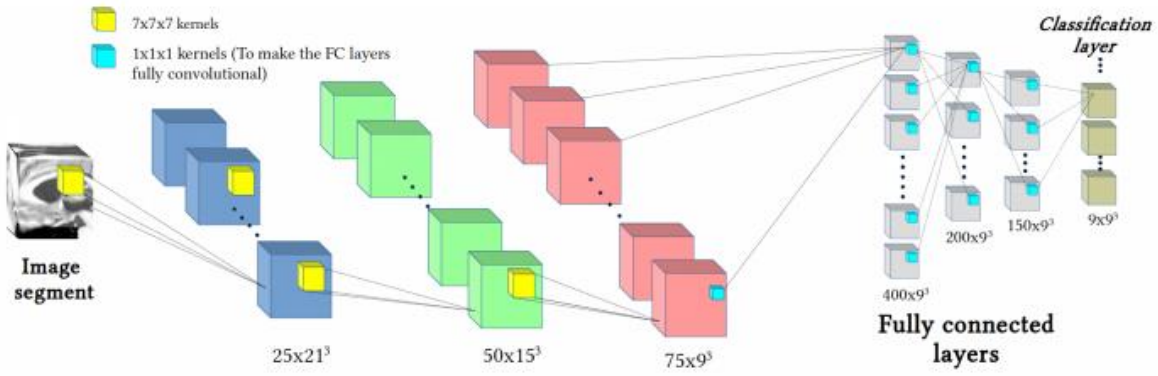

Figura 8: Esempio di architettura di una rete neurale convoluzionale [13]

2.4.2 Livello convoluzionale

In specifico, i livelli convoluzionali sono composti da nodi a 3D chiamati kernel o filtri, ogni kernel ha la sua dimensione in termini di larghezza, altezza e profondità. Ogni kernel è più piccolo di dimensione e si estende attraverso tutto il volume dell'input. Per esempio, un tipico kernel nel primo strato di una CNN potrebbe avere dimensione 5x5x3 (5 pixel di altezza e larghezza, 3 di profondità per i canali RGB). Durante la prima fase di propagazione, viene effettuata la convoluzione dei kernel lungo la larghezza e l'altezza del volume di input e calcola i punti tra i valori del kernel e quelli dell'input per ogni posizione. Come scorriamo il kernel lungo la larghezza e l'altezza del volume di input produciamo una mappa di attivazione a due dimensioni che fornisce il responso per quel dato filtro per ogni posizione spaziale. Come le reti neurali artificiali tutti i valori dell'output verranno normalizzati e poi trasformati dalla funzione non lineare, che tipicamente è la funzione ReLU. Durante la propagazione all'indietro, ogni nodo 3D di output calcolerà il gradiente per i pesi del suo filtro.

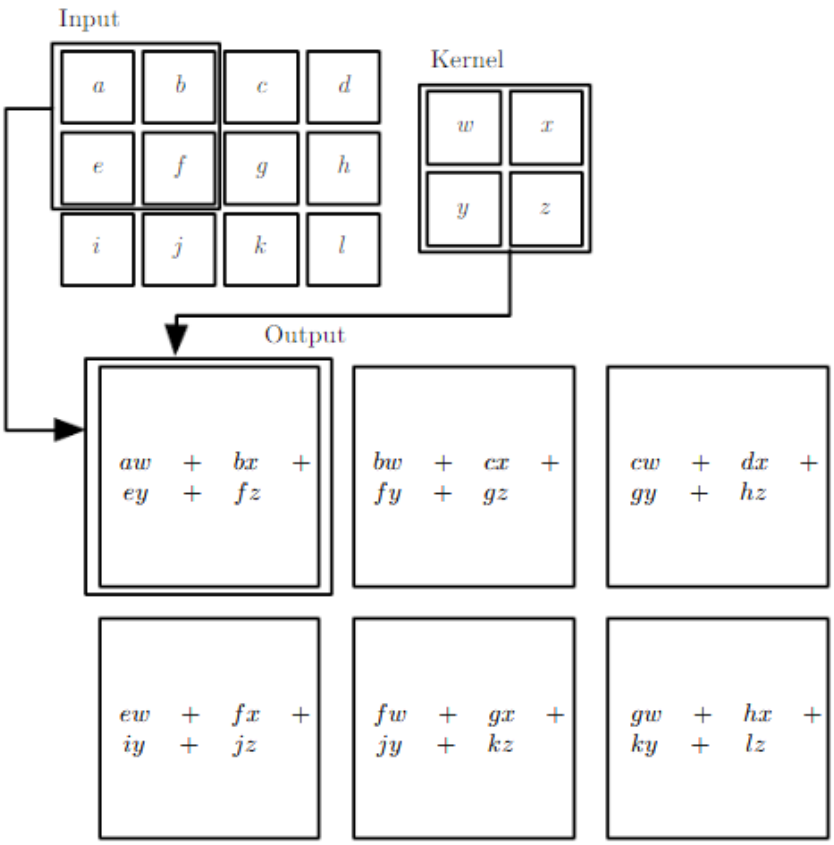

Figura 9: dimostrazione della operazione di convoluzione in 2D [14]

Il gradiente viene aggiunto attraverso ogni fetta della feature e viene usata per aggiornare i pesi nel filtro che ha generato quella determinata fetta. Le CNN addestrate hanno il loro livello di convoluzione come un set di filtri addestrati che possono estrarre determinate informazioni, come contorni in vari orientamenti oppure blob di colori in livelli superficiali, la forma e le texture dell'oggetto di addestramento nei livelli più profondi. È importante sottolineare che il livello convoluzionale è specificato secondo i seguenti parametri e devono essere scelti con attenzione: **dimensione del filtro, il numero di filtri, il passo e il padding a zero**. Il passo controlla come il filtro "scorre" attraverso il volume di input, solitamente è impostato in modo tale che il volume di output sia un intero e non una frazione. Il padding a zero è una tecnica di padding dell'input per l'aggiunta di zeri ai bordi. Questo è efficace perché permette al filtro di scorrere più frequentemente nei contorni del volume di input e catturare meglio le informazioni in queste aree, inoltre è conveniente perché ti dà maggior controllo sulla grandezza dell'output.

2.4.3 Livello di Pooling

Dopo i livelli di attivazione (ReLU layers), i programmatori possono scegliere di applicare un livello di pooling. Questo livello prende un filtro, solitamente di grandezza 2x2 e scorre per la stessa grandezza. Poi viene applicato al volume di input e produce come output i valori massimi in ogni subregione con cui il filtro ha effettuato la convoluzione. Questo permette la riduzione dei numeri dei parametri della rete, con conseguente alleggerimento della memoria e del costo computazionale.

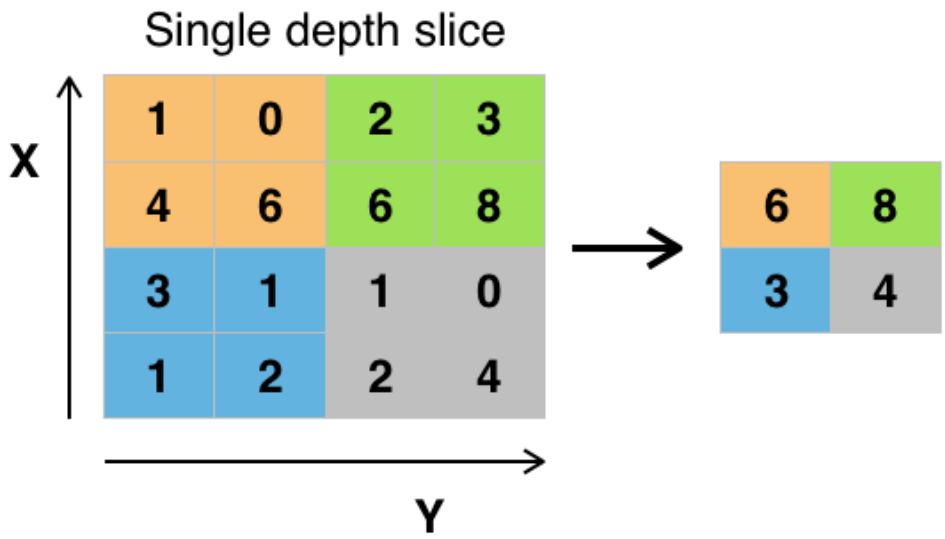

Figura 10: esempio di un pool con un filtro 2x2 e passo di 2 [15]

Altri tipi di pooling sono gli average pooling e L2-norm pooling. Il motivo per l'uso di questo livello sta nel fatto che una volta che sappiamo che una specifica feature sta nel nostro volume originale di input (con conseguente valore alto di attivazione), la sua esatta locazione non è importante come la sua relativa locazione rispetto alle altre feature. Questo livello, come abbiamo già detto, riduce la dimensione spaziale (quindi non la profondità) del volume di input, ma ha un altro scopo: controlla l'overfitting. Questo termine si riferisce quando un modello è molto sintonizzato con gli esempi di addestramento e perciò non è in grado di generalizzare bene per i set di validazione e test. Un sintomo dell'overfitting è avere un modello che prende il 100% o 99% nel set di addestramento (training), ma solo il 50% per i dati di test.

In sintesi:

- Un'architettura di una rete neurale convoluzionale è nel semplice caso una lista di livelli che trasformano un volume immagine in un volume di output.
- Ci sono vari tipi di livelli (quelli visti in questa tesi sono i più popolari e usati)
- Ogni livello accetta un volume di input in 3D e lo trasforma in un volume di output 3D attraverso una funzione differenziale.
- Alcuni livelli possono anche non avere parametri.

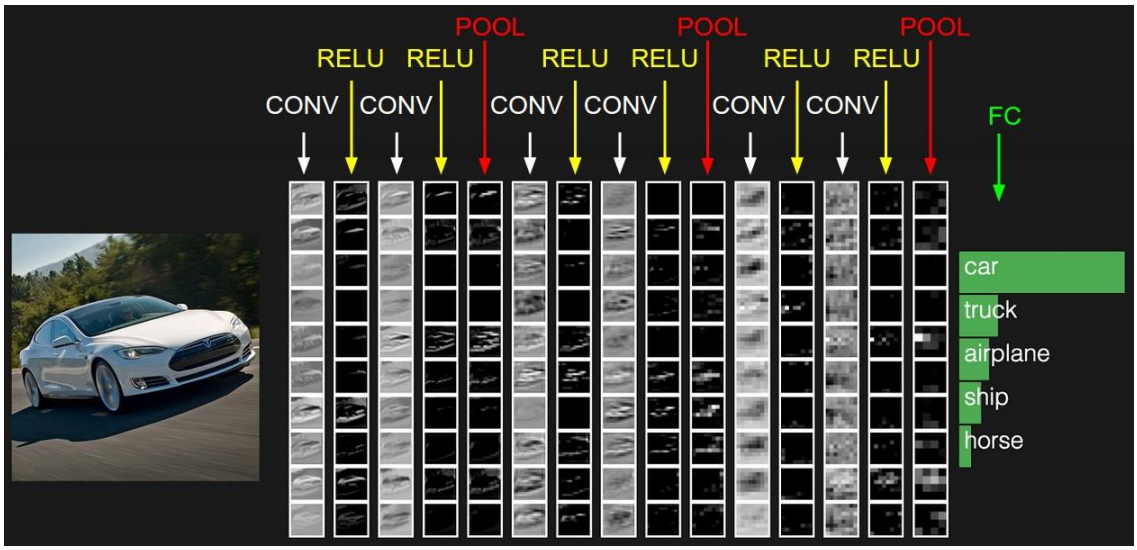

Figura 11: Esempio architettura CNN [12]

2.5 Normalizzazione batch

La normalizzazione batch è una delle ragioni per cui il deep learning è diventato molto famoso ed è diventato una feature standard per molte librerie di machine learning. La BN (batch normalization) è stata introdotta in seguito al problema del overfitting e al cambiamento di covarianza. Si riferisce al cambiamento della distribuzione dei dati di input usati per l'algoritmo di addestramento che si traduce in un cambiamento del comportamento dell'algoritmo di machine learning. È evidente quando i dataset di addestramento (train) e test provengono da differenti sorgenti e per questo avranno una diversa distribuzione. Quello che cerca di fare la BN è quella di provare a limitare questo

cambiamento normalizzando le attivazioni di ogni livello (l'output del livello precedente che è l'input del livello successivo) in modo tale che anche un piccolo cambiamento possa amplificarsi nel livelli successivi. [10]

Capitolo 3 Neural Compute Stick & OpenVINO

3.1 Intel® Neural Compute Stick 2

L'Intel® Neural Compute Stick 2 (Intel® NCS 2) è il nuovo kit della Intel che ci permette lo sviluppo di algoritmi di riconoscimento attraverso le reti neurali. A forma di chiavetta USB, la NCS 2 è dotata dell'ultima versione di VPU - la Intel Movidius Myriad X, che include un acceleratore di reti neurali chiamato Neural Compute Engine. Ha 16 processori SHAVE e un hardware dedicato che funge da acceleratore per le reti neurali. Rispetto alla versione precedente, NCS 2 offre prestazioni 8 volte superiori [16].

Figura 12: Intel Neural Computer Stick 2 [17]

La serie NCS è stata progettata per consentire una rapida prototipazione, validazione e sviluppo di applicazioni per il deep learning. Tutto questo è reso grazie al basso consumo della architettura VPU che permette alle applicazioni di IA di non aver bisogno di una connessione al cloud per effettuare le operazioni. La NCS2 è molto efficiente e ti

permette di effettuare 1 trilione di operazioni al secondo (1 TOPs) in pochi watt (da 1 a 3 W) ad un prezzo veramente competitivo.

Ecco una panoramica interna del dispositivo:

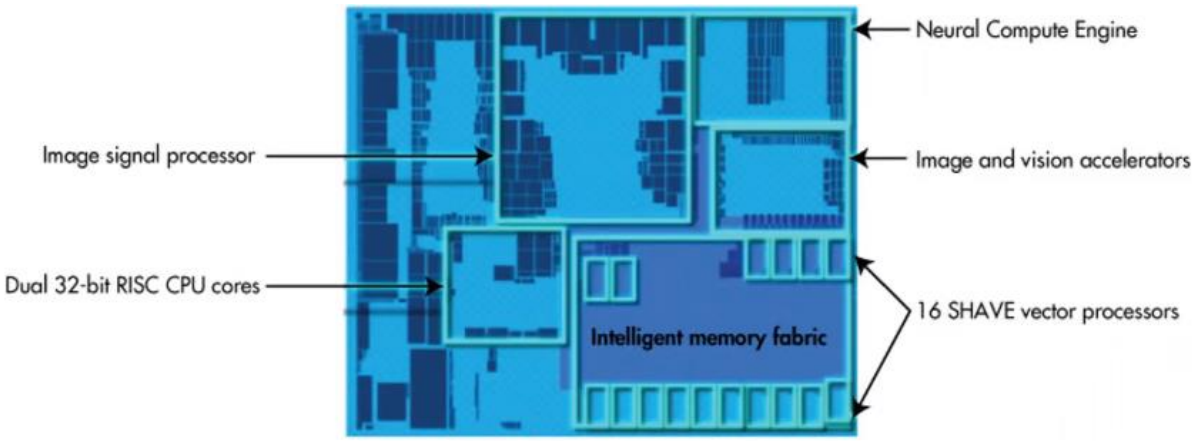

Figura 13: Intel Movidius Myriad X [18]

Possiamo notare gli acceleratori per l'imaging e per la computer vision, una CPU e i 16 processori a 128-bit SHAVE (streaming hybrid architecture vector engine). Il processo produttivo è di 16 nm FinFET.
Il formato nativo usato dal Myriad-X è il FP16, può eseguire più modelli contemporaneamente in modo concorrente, tenendo conto della sua potenza computazionale. Myriad X è la prima VPU a montare la Intel Neural Compute Engine, un acceleratore hardware dedicato con il supporto nativo ai formati FP16 e fixed-point 8-bit.
Modelli supportati dal plugin Myriad sono:

Caffe*:

- AlexNet
- CaffeNet
- GoogleNet (Inception) v1, v2, v4
- VGG family (VGG16, VGG19)
- SqueezeNet v1.0, v1.1
- ResNet v1 family (18, 50, 101, 152)
- MobileNet (mobilenet-v1-1.0-224, mobilenet-v2)

- Inception ResNet v2
- DenseNet family (121,161,169,201)
- SSD-300, SSD-512, SSD-MobileNet, SSD-GoogleNet, SSD-SqueezeNet

TensorFlow*:

- AlexNet
- Inception v1, v2, v3, v4
- Inception ResNet v2
- MobileNet v1, v2
- ResNet v1 family (50, 101, 152)
- ResNet v2 family (50, 101, 152)
- SqueezeNet v1.0, v1.1
- VGG family (VGG16, VGG19)
- Yolo family (yolo-v2, yolo-v3, tiny-yolo-v1, tiny-yolo-v2, tiny-yolo-v3)
- faster_rcnn_inception_v2, faster_rcnn_resnet101
- ssd_mobilenet_v1
- DeepLab-v3+

MXNet*:

- AlexNet and CaffeNet
- DenseNet family (121,161,169,201)
- SqueezeNet v1.1
- MobileNet v1, v2
- NiN
- ResNet v1 (101, 152)
- ResNet v2 (101)
- SqueezeNet v1.1
- VGG family (VGG16, VGG19)
- SSD-Inception-v3, SSD-MobileNet, SSD-ResNet-50, SSD-300 [19]

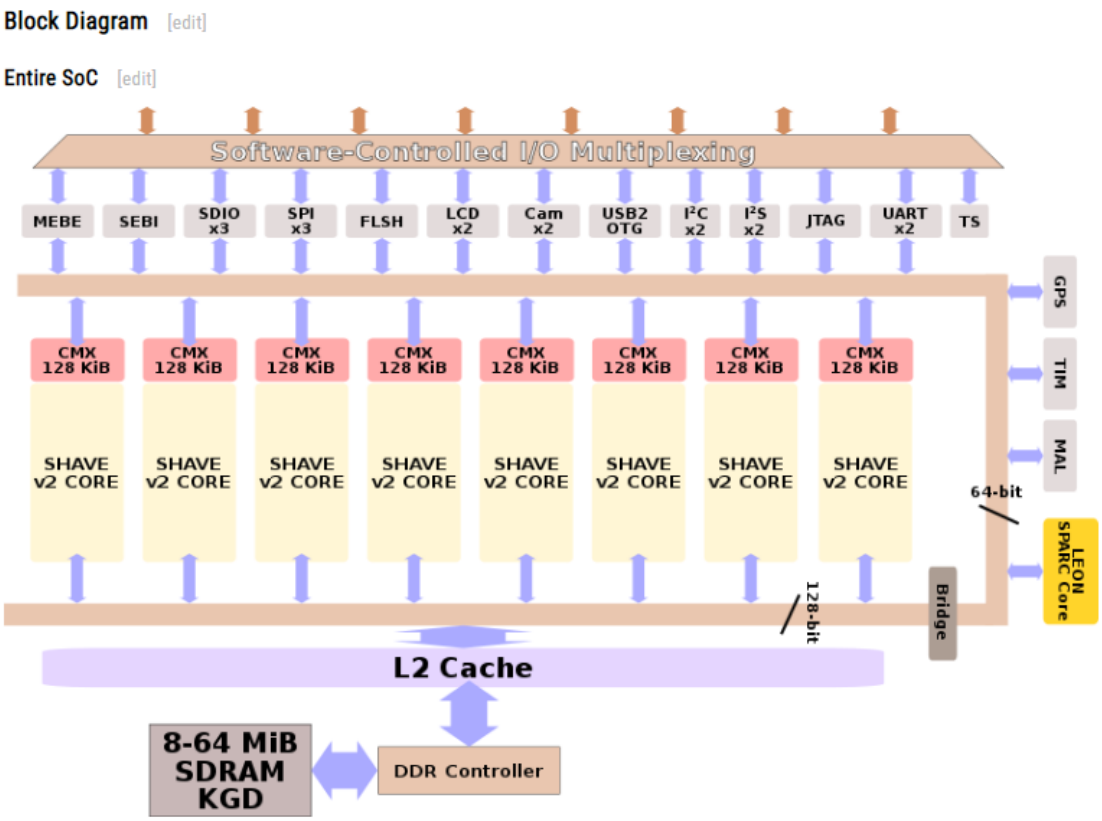

Figura 14: architettura SHAVE [20]

Nella figura 14 è rappresentato il diagramma a blocchi della microarchitettura SHAVE: gli acceleratori progettati da Movidius per i loro processori di computer vision. Inizialmente son stati progettati per l'accelerazione della fisica dei giochi, ma in seguito ad una bassa richiesta di questi tipi di acceleratori per la fisica per gli smartphone hanno costretto l'azienda a concentrarsi sull'image e vision processing. Rispetto la precedente versione (figura in alto) la Myriad X ha aumentato il numero di SHAVE core fino a 16 espandendo e potenziando gli acceleratori per la computer vision aggiungendo il supporto per una nuova pipeline per il processamento di immagini in 4K. [20]

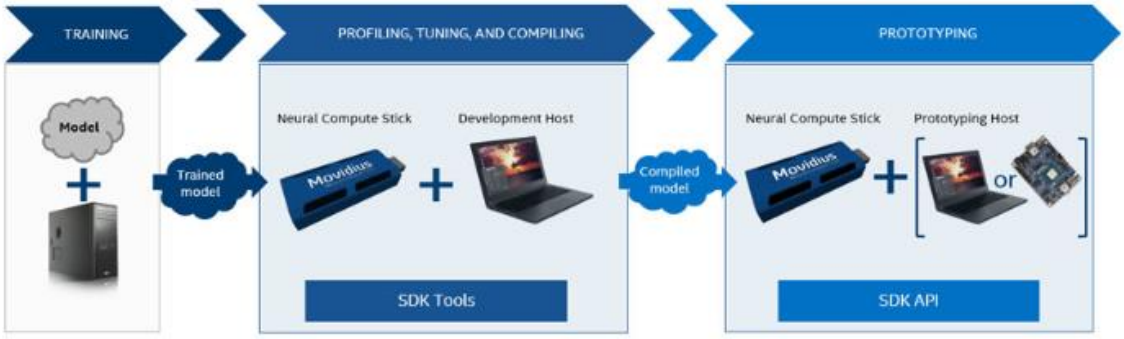

Figura 15: Il seguente diagramma mostra il tipico workflow per lo sviluppo con la NCS [21]

3.1.2 Sviluppare con la Neural Compute Stick

Il processo di sviluppo con la NCS è presentato nella figura 15. Un modello è progettato e allenato in un computer predisposto a questo tipo di task. Poi attraverso le API supportate viene profilato, sintonizzato, compilato e convertito nel formato supportato dalla VPU. Con il nuovo modello formattato gli utenti possono validare il modello e sviluppare le proprie applicazioni nella stessa macchina utilizzata o in altri dispositivi. L'hardware è fisicamente connesso alla macchina e viene fatto accesso ad essa attraverso i metodi forniti dalle API.

Grazie al formato IR (intermediate representation), si può sviluppare e testare una rete neurale su un tipo di processore come una CPU e utilizzare lo stesso modello in un range di processori come i processori Intel (CPU, GPU/Intel® Processor Graphics, VPU, FPGA). Il formato IR consente di far eseguire i modelli su vari framework come TensorFlow, Caffe e MXNet. Grazie alla architettura modulare del suo toolkit OpenVINO è possibile questa flessibilità per supporto a vari framework.

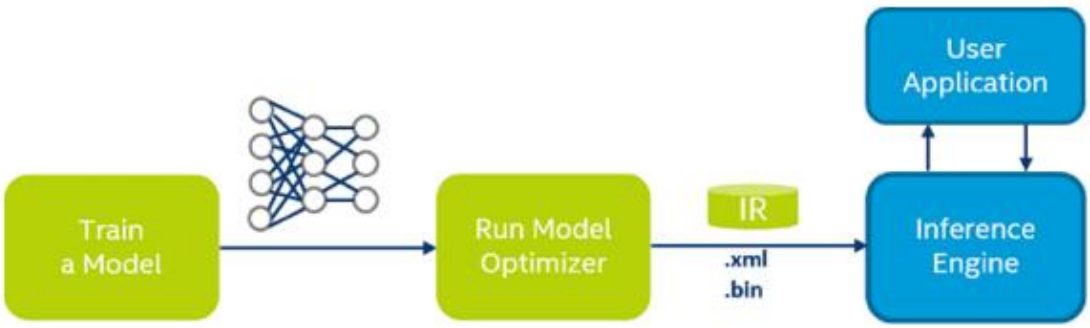

Figura 16: Diagramma del workflow del toolkit OpenVINO [22]

3.2 Il toolkit Intel® OpenVINO

Figura 17: Logo OpenVINO [23]

OpenVINO è il software principale per lo sviluppo con la NCS e gli altri dispositivi Intel. Permette la creazione di soluzioni per la computer e machine vision con ottime prestazioni in termini di velocità e accuratezza. OpenVINO integra l'elaborazione della fotocamera, ottimizza i calcoli degli algoritmi di DL e possiede dei tools per l'accelerazione per la CV. Questo significa che le soluzioni basate sulle CNN usando questo toolkit hanno le massime performance grazie alla possibilità di estendere il loro carico di lavoro sui diversi dispositivi della Intel (come le CPU, GPU, FPGA, VPU e IPU) semplicemente sfruttando le sue API. Rispetto al precedente kit di sviluppo, l'NCSDK [18], OpenVINO permette le predizioni basate sulle CNN con migliori kernel pre-ottimizzati e le chiamate alle API di OpenCV. In figura 16 si può vedere il diagramma del workflow di sviluppo del toolkit.

Il toolkit OpenVINO include i seguenti componenti:

- Intel® Deep Learning Deployment Toolkit
  - Model Optimizer
  - Inference Engine
- OpenCV
- Driver e applicazioni per OpenCL
- Intel® Media SDK
- OpenVX
- Demo ed esempi

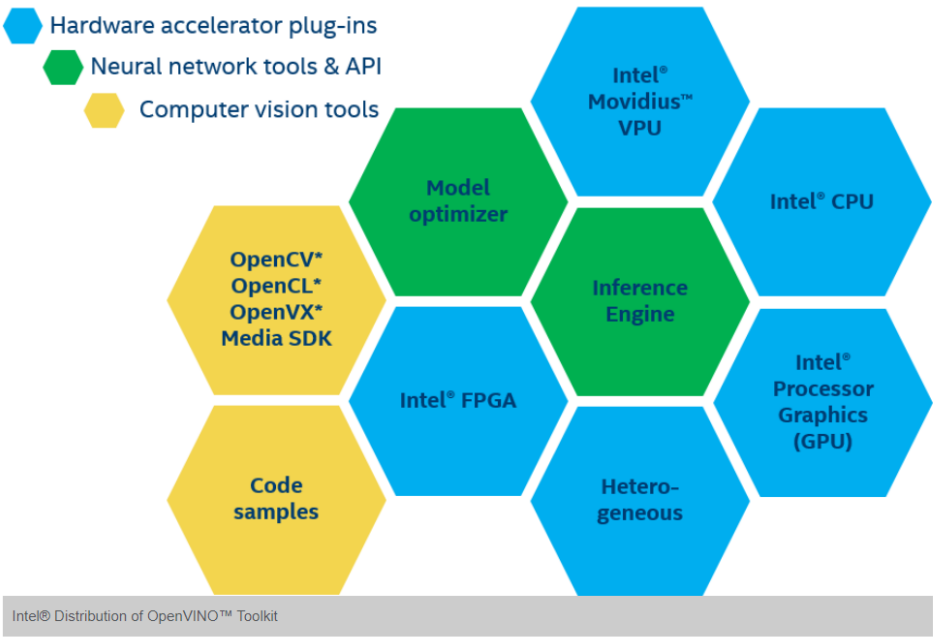

Figura 18: Rappresentazione grafica dei componenti software del toolkit [24]

Ecco riportata una tabella che mostra i formati dei modelli supportati dai vari dispositivi:

Supported Model Formats

PLUGIN	FP32	FP16	I8
CPU plugin	Supported and preferred	Supported	Supported
GPU plugin	Supported	Supported and preferred	Supported*
FPGA plugin	Supported	Supported	Not supported
VPU plugins	Not supported	Supported	Not supported
GNA plugin	Supported	Supported	Not supported

Figura 19: tabella dei formati supportati [25]

Inoltre non tutti i modelli pre-addestrati sono compatibili con il plugin Myriad-X.

3.2.1 Model Optimizer

Il Model Optimizer esegue un task molto complicato: fa la mappatura tra i diversi dispositivi (CPU, GPU, VPU, FPGA) e i vari modelli (Caffe, Mxnet, TensorFlow, Onnx). La rappresentazione dei vari modelli di deep learning cambia molto tra i vari framework, lo stesso per i vari dispositivi, hanno delle architetture differenti, diversi set di istruzioni e modalità di programmazione. Intel offre perciò delle API comuni a tutti questi dispositivi che ti consentono di implementare algoritmi per la Computer Vision senza sapere tutti i dettagli di programmazione di basso livello per tutti i dispositivi.

Come lavora il Model Optimizer?

Prima fase:
Converte tutti i modelli citati prima in un unico formato unico chiamato IR. Sono due file, uno in formato `.xml` che descrive la topologia, gli strati, la connettività e i parametri, l'altro è un file `.bin,` che contiene i dati addestrati e che tiene conto di tutti i pesi e dei valori del bias del modello.

Seconda fase:
Ottimizzazione: in questa fase vengono effettuate tutte quelle tecniche come la fusione dei layer, la normalizzazione batch etc. Sono ottimizzazioni hardware per i formati dei pesi: il Model Optimizer sceglie il formato più adatto per velocizzare le operazioni.

Terza fase:
Usa questi file IR per le predizioni sui vari dispositivi. [22]

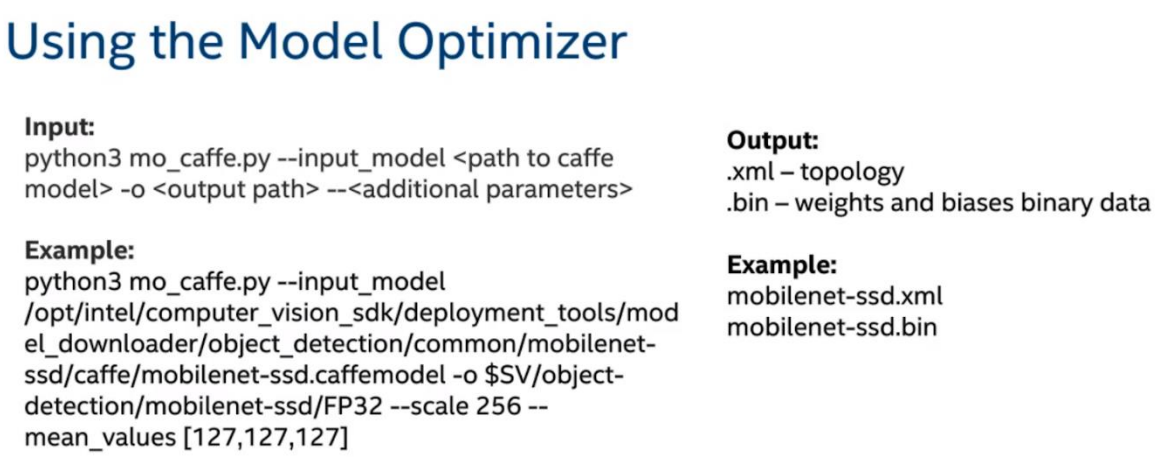

Figura 20: esempio utilizzo Model Optimizer [26]

3.2.2 L'Inference Engine

Dopo aver usato il Model Optimizer per creare un IR file, entra in gioco l'Inference Engine per dedurre il risultato dato un certo input.

L'Inference Engine è una libreria scritta in C++ con un set di classi che consentono la predizione delle immagini in input e fornire un risultato. La libreria in C++ fornisce una API per leggere il file in formato IR, imposta i formati degli input e output ed esegue il modello nel dispositivo. Il Model Optimizer processa il modello originale e mappa i layers per le operazioni supportate con risultato i file IR.

Alcune operazioni sarebbero diverse da dispositivo a dispositivo, per esempio l'operazione di moltiplicazione ha una implementazione diversa per la CPU rispetto la GPU, la prima è implementata in C, C++ o Assembler mentre la seconda viene fatta in OpenCL e altri modi. Proprio per questo l'Inference Engine ha diversi "plug-in" per ogni dispositivo supportato [27]. Lo sviluppatore deve solo specificare il dispositivo scelto e l'Inference Engine userà il plug-in giusto farà la giusta implementazione della rete per il dispositivo desiderato. Riassumendo: Il Model Optimizer farà le operazioni di ottimizzazione e diagnostica in base all'hardware, e l'Inference Engine sceglierà il plug-in giusto per il dispositivo desiderato per effettuare le specifiche ottimizzazioni.

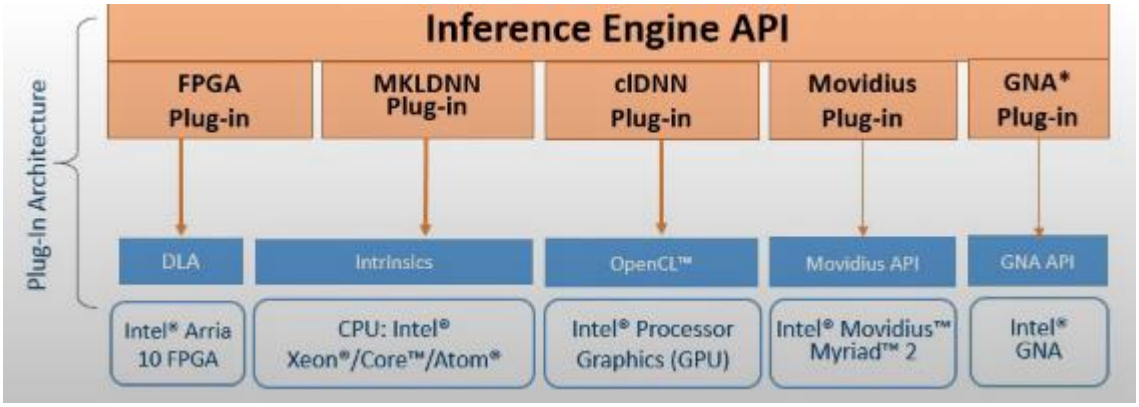

Figura 21: Plug-in dell'Inference Engine [27]

3.2.3 Flusso di lavoro delle API dell'Inference Engine

Il flusso di lavoro è suddiviso in vari step:

1. **Creare l'oggetto Inference Engine Core** - Crea l'oggetto `InferenceEngine::Core` per lavorare con i diversi dispositivi, tutti i plug-in sono gestiti internamente dall'oggetto `Core`.

2. **Leggere l'IR file** – Usando la classe `InferenceEngine::Core`, legge il file IR dentro all'oggetto della classe `InferenceEngine::CNNNetwork`. Questa classe rappresenta la rete nella memoria dell'host.

3. **Preparare i formati dell'input e output** – Dopo aver caricato la rete, specifica la precisione dell'input e dell'output e il layout della rete. Per queste specifiche usa, `InferenceEngine::CNNNetwork::getInputsInfo()` and `InferenceEngine::CNNNetwork::getOutputsInfo()`.

4. Passaggio delle configurazioni specifiche del dispositivo richiesto (`InferenceEngine::Core::SetConfig`), e le estensioni dei registri (`InferenceEngine::Core::AddExtension`).

1. **Compilazione e caricamento della rete nel dispositivo** – Usa il metodo `InferenceEngine::Core::LoadNetwork()` per il dispositivo specifico (es. CPU, GPU, etc.) per compilare e caricare la rete nel dispositivo.

2. **Impostare i dati in input** – Con la rete caricata, abbiamo un oggetto `InferenceEngine::ExecutableNetwork`. Usa questo oggetto per creare una `InferenceEngine::InferRequest` dove segnala i buffer di input da usare per l'input e per l'output. Specifica una memoria allocata del dispositivo e ci copia la rete direttamente.

3. **Esecuzione:**
    - Sincrona – metodo `InferenceEngine::InferRequest::Infer()`. Si blocca finché la predizione non è completata.
    - Asincrona – metodo `InferenceEngine::InferRequest::StartAsync()`. Può controllare lo stato con il metodo `InferenceEngine::InferRequest::Wait()`, aspettare o specificare una callback una volta finito.

4. **Ottenere l'output** – una volta che la predizione è completata prendere l'output dalla memoria. Lo fa con il metodo `InferenceEngine::InferRequest::GetBlob()`. [27]

3.2.4 OpenCV

Figura 22: Logo OpenCV [28]

OpenCV (Open Source Compute Vision Library) è una libreria software open source per la computer vision e il machine learning. È stata sviluppata per fornire una infrastruttura comune per le applicazioni di computer vision e per accelerare l'uso del machine learning nei prodotti commerciali. La libreria comprende più di 2500 algoritmi ottimizzati, che includono algoritmi allo stato dell'arte sia della computer vision che del machine learning. Questi algoritmi possono essere usati per il rilevamento e detenzione delle facce, identificazione di oggetti, classificazione delle azioni umane nei video, tracciamento movimento della telecamera, estrazione dei modelli 3D degli oggetti e tantissimi altri task. OpenCV vanta una community di più di 47 mila persone e più di 18 milioni di download. Questa libreria è usata in modo estensivo in aziende, gruppi di ricerca ed enti governativi. Possiede interfacce per C++, Python, Java e MATLAB e supporta i sistemi operativi Windows, Linux, Android e Mac OS. [28]

3.3 Demo ed esempi

Gli sviluppatori della Movidius forniscono un repository contenente una serie di demo ed esempi per testare la Intel Neural Compute Stick 2 e la Deep Learning Toolkit dell'Intel OpenVINO Toolkit. Il repository si chiama ncappzoo [29] ed è costantemente mantenuto aggiornato da vari sviluppatori. Tutto il contenuto di ncappzoo è open source ed è possibile scaricare il contenuto per sperimentare, modificare e imparare dalle loro demo. Le demo provate son state testate su un sistema operativo Linux Ubuntu 18.04, con un Intel Core i5-6500, 16 GB RAM e con la NCS 2.

3.3.1 *simple_classifier_cpp_camera*

Questa applicazione esegue una predizione di una immagine usando GoogLeNet [30], ma è possibile anche usare differenti modelli.
Prima di eseguire la predizione l'applicazione procede con i seguenti step:

1. Esegue il download del file Caffe prototxt ed effettua alcuni cambiamenti necessari per lavorare con la NCS 2 e l'OpenVINO Toolkit.
2. Scarica e genera i dati ilsvrc (un file dove sono presenti le etichette per le predizioni).
3. Scarica il file dei pesi del modello da uno specifico repository.
4. Compila il modello per generare i file IR usando il Model Optimizer.
5. Esegue `run.py` che permette la predizione dell'immagine in ingresso.

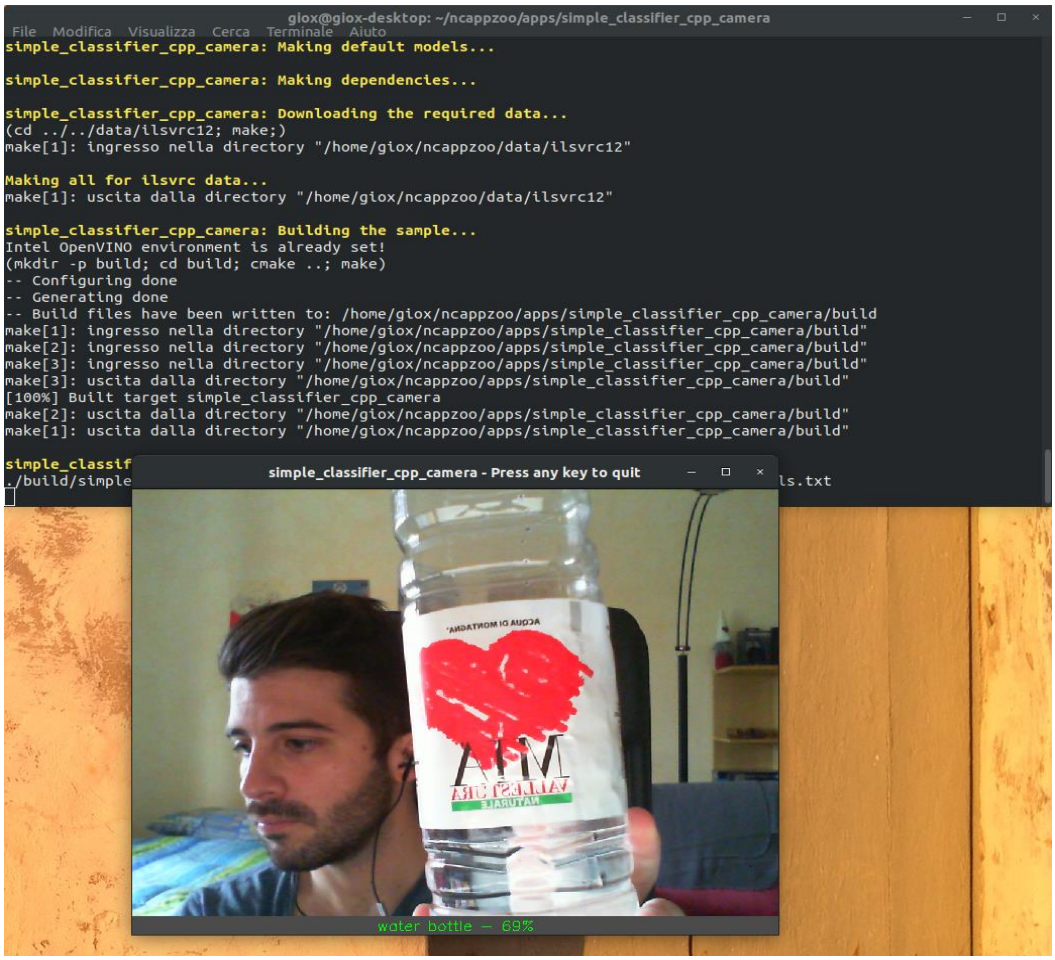

Figura 23:Esempio applicazione simple_classifier_cpp_camera

3.3.2 *mnist_calc*

Questo esempio utilizza la rete mnist_deep addestrata sul dataset MNIST [31] attraverso TensorFlow per la classificazione delle cifre scritte a mano come input di una calcolatrice. Le cifre vengono disegnate con il mouse o con il touchscreen.

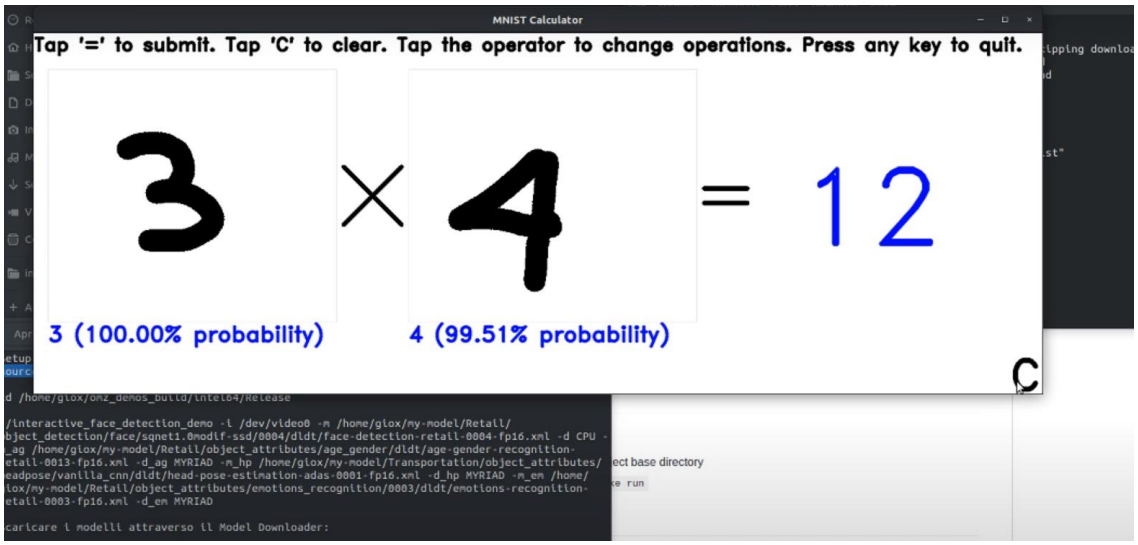

Figura 24: esempio applicazione mnist_calc

3.3.3 *benchmark_ncs*

Un'altra interessante applicazione presente nel repository è la benchmark_ncs, che ci consente di effettuare dei benchmark sulla nostra corrente configurazione e misurare vari parametri. In questo esempio vengono mostrati gli fps (frame al secondo o predizioni al secondo) che riusciamo a ottenere con un determinato dispositivo, modello e dataset.
L'esempio usa tre thread per ogni NCS e crea sei richieste asincrone per thread ed esegue mille predizioni usando la rete GoogLeNet.

Risultati del test effettuato con una NCS 2:

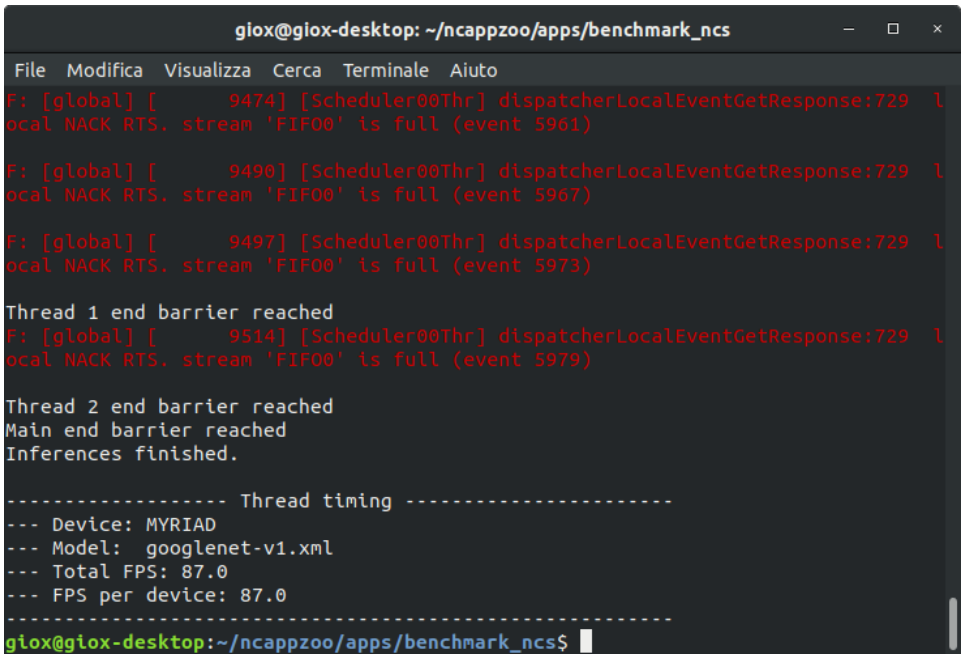

Figura 25 – benchmark test sulla NCS 2

Risultati del test effettuato con la Intel Core i5-6500:

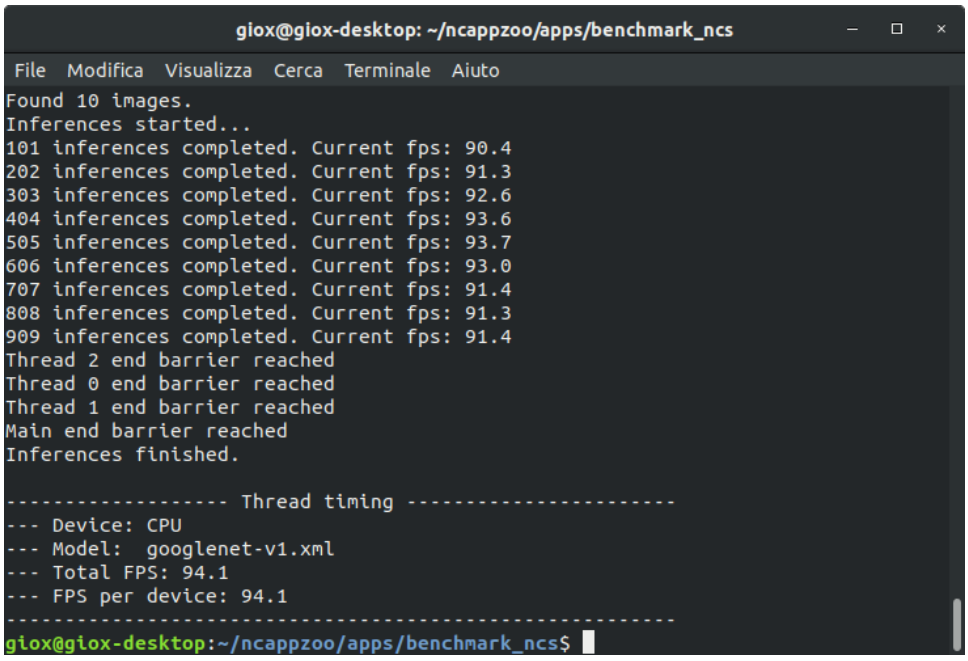

Figura 26: benchmark sulla CPU Intel Core i5-6500

Come possiamo vedere dai risultati le prestazioni della NCS 2 non si allontanano da quelle dell' i5-6500. Ai fini di un benchmark più completo si analizzano altri valori da tenere in considerazione che verranno mostrati nel prossimo paragrafo.

3.4 Misurare le performance per le predizioni di Deep Learning

Il toolkit Intel OpenVINO aiuta l'accelerazione delle predizioni di deep learning per una vasta gamma di processori e acceleratori della Intel. Al posto di utilizzare un unico dispositivo la Intel offre un vario portfolio di soluzioni hardware e software che, affiancati al toolkit OpenVINO possono raggiungere vari obiettivi in termini di performance, potenza e costo a seconda dei casi d'uso. In questo lavoro di tesi mi sono concentrato nel fornire una misura delle prestazioni tra il processore del mio PC desktop con a bordo una Intel i5-6500 e la NCS2 per avere un'idea delle potenzialità di quest'ultima per l'utilizzo in situazioni dove non è possibile avere particolari workstation o server per effettuare delle predizioni.

La misura delle performance per le predizioni tiene conto di varie variabili ed è estremamente dipendente dai casi d'uso. Di solito vengono usati 4 parametri per le misure che sono elementi chiavi per considerare una applicazione di deep learning di successo:

1. **Throughput** – Misura il numero di predizioni distribuite entro una certa soglia di latenza. Quando si analizza un sistema di deep learning si sceglie quel sistema che ha throughput che consente di avere un giusto trade-off tra latenza e potenza per un determinato costo che soddisfano i requisiti.
2. **Value** – Se il troughput è importante, è molto più critico nello sviluppo di applicazioni AI è l'efficienza di prestazioni o prestazioni per costo. Si misura in throughput/dollaro.
3. **Efficiency** – La potenza dei sistemi è un fattore molto importante da considerare. Quando si scelgono delle soluzioni di deep learning, l'efficienza energetica (throughput/watt) è un fattore critico da considerare.
4. **Total Benefit** (Applicabile per le piattaforme VPU) – Combina i fattori di value ed efficiency per comparare la resa degli hardware per le migliori prestazioni per watt e per dollaro per un determinato caso d'uso.

3.4.1 Comparazione prestazioni tra NCS 2 e la CPU Intel Core i5-8500

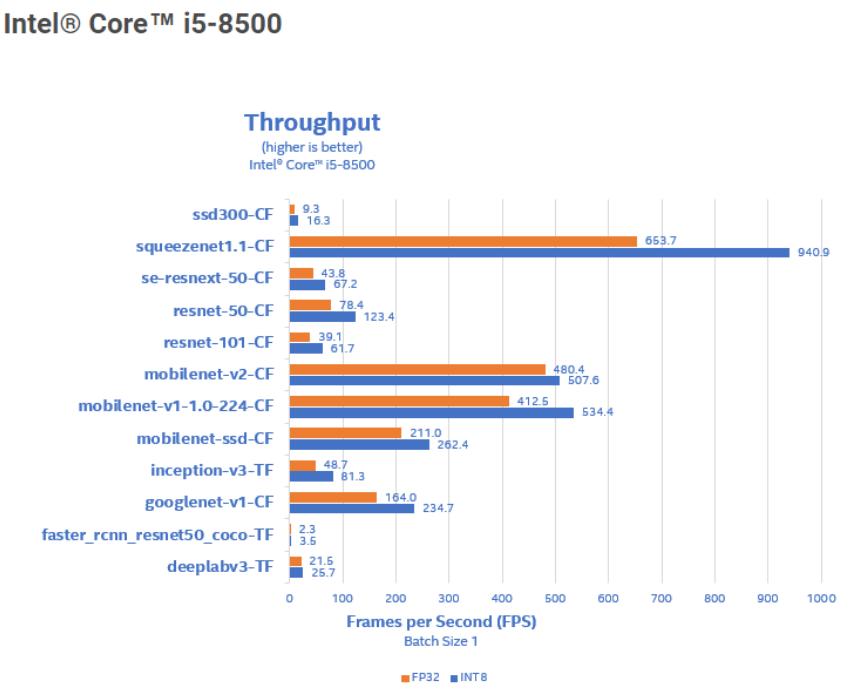

Figura 27

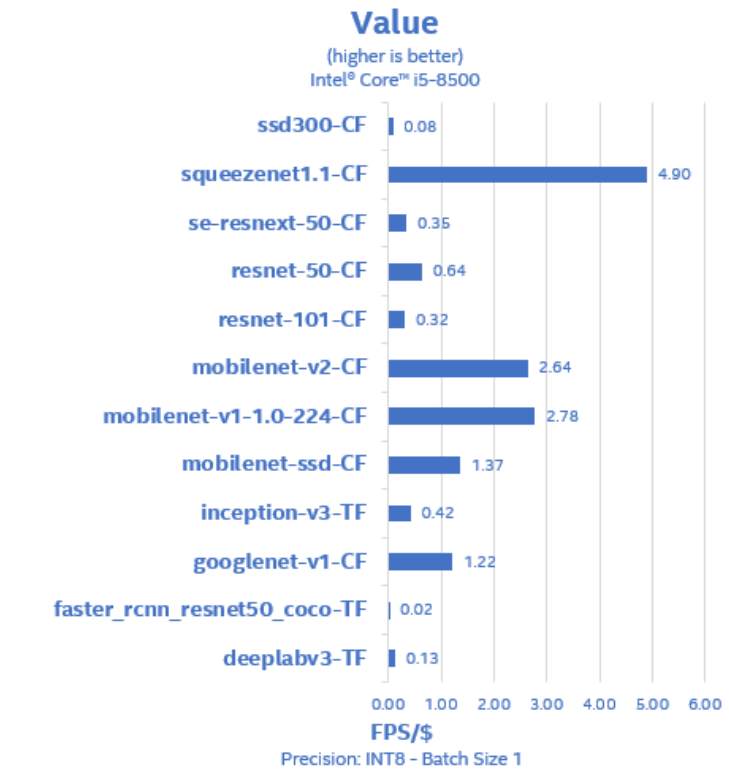

Figura 28

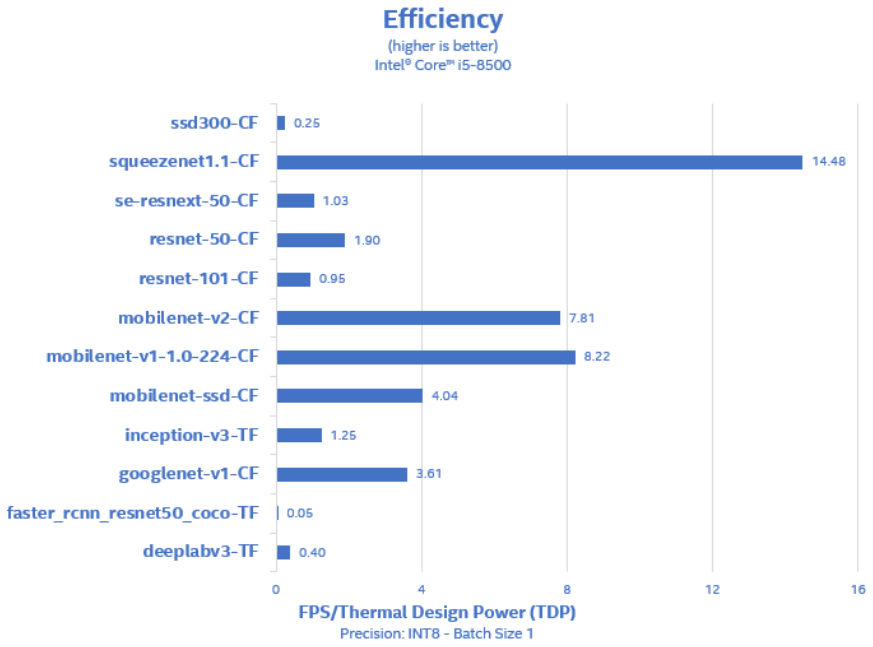

Figura 27,28,29: i valori del benchmark della CPU Intel Core i5-8500 con vari modelli.

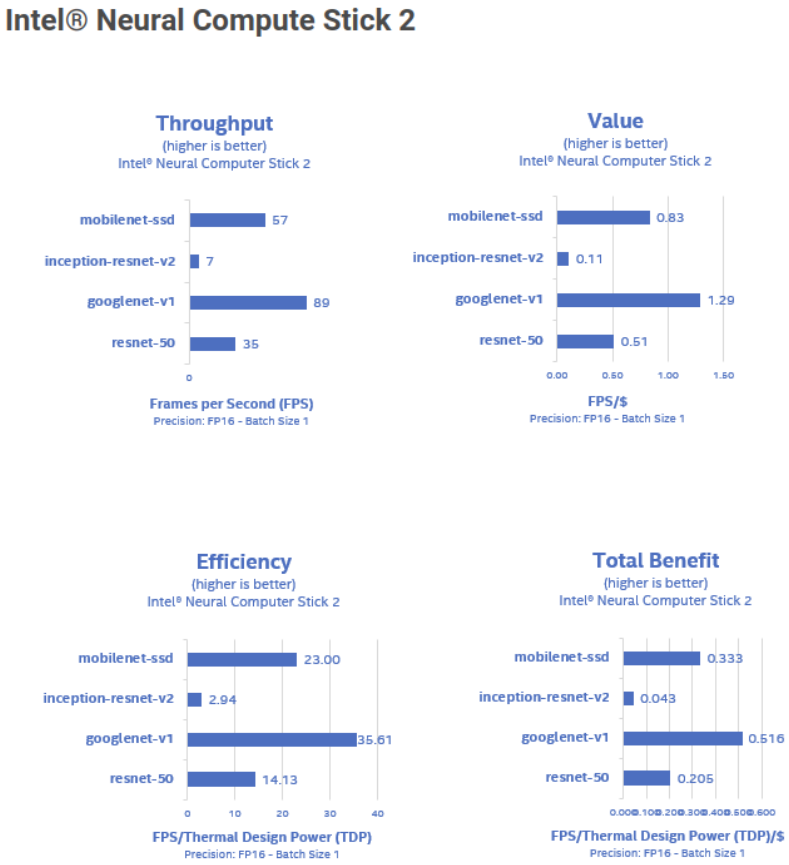

Figura 30: benchmark NCS 2 con vari modelli [32]

3.4.2 Analisi risultati benchmark

Se andiamo ad effettuare un confronto tra i modelli in comune (mobilenet-ssd-cf, inception-resnet-v2, googlenet-v1 e resnet-50) tra i due dispositivi possiamo notare un valore molto maggiore per quanto riguarda l'efficiency della NCS 2 rispetto la CPU a discapito di un minor valore di throughput. Come abbiamo detto in precedenza, le prestazioni variano a seconda del caso d'uso e dalla configurazione del sistema ma possiamo constatare l'efficacia della NCS 2 in quei contesti dove si ha bisogno di una applicazione di deep learning che giri su sistemi a basse prestazioni o con una durata della batteria ridotta come per esempio droni, robot mobili, computer portatili datati dove non si ha un accesso diretto e stabile alla rete.

Figura 31: NCS 2 collegata a un drone [33]

Capitolo 4 Visual attention detection

In questo capitolo verrà esposto il lavoro svolto su una applicazione di deep learning, utilizzando il toolkit OpenVINO e la Neural Compute Stick 2, illustrando nel dettaglio le tecnologie usate e le scelte di programmazione.
Dopo aver analizzato e modificato alcune demo presenti nel repository fornito dalla Intel si è scelta l'applicazione "interactive-face-detection-demo". L'idea su cui si basa il progetto è la modifica di questa applicazione e l'aggiunta di alcune funzionalità per adattarla a un possibile caso d'uso. Quindi l'obiettivo del mio lavoro è stato di creare un'applicazione che salvi i risultati di diverse predizioni, come l'età, sesso, posa della testa ed emozione prevalente, per poi poterle elaborare e visualizzare in un endpoint http seguita da una generazione di una heatmap sulla base dell'orientamento della testa per evidenziare le zone dove si è guardato più frequentemente.

4.1 Analisi Demo

La demo esegue un task di riconoscimento facciale usando una sequenza di reti neurali. Inoltre, esegue quattro richieste di predizione parallele per il riconoscimento dell'età, della posa della testa, delle emozioni e del rilevamento di punti di riferimento della testa. Son stati usati i seguenti modelli pre-addestrati:

- `Face-detection-adas-0001,` il modello principale per rilevare le facce. Questa rete si basa sulle caratteristiche della rete convoluzionale MobileNet, che include delle particolari operazioni di convoluzione che riducono la quantità di risorse necessarie eseguire i blocchi di convoluzione 3x3.

  Input

  La rete prende in input un'immagine [1x3x384x672] con il formato [BxCxHxW] dove:

  - B – dimensione del batch

- C – numero di canali
- H – altezza immagine
- W – larghezza immagine

La rete si aspetta un'immagine in BGR.

Output

L'output della rete è un blob con la forma: [1, 1, N, 7], dove N è il numero dei bounding box rilevati, la descrizione ha il seguente formato: `[image_id, label, conf, x_min, y_min, x_max, y_max]`

- `image_id` – id dell'immagine nel batch
- `label` – l'id della classe predetta
- `conf` – livello di confidenza della classe predetta
- `(x_min, y_min)` – coordinate dell'angolo in alto a sinistra del bounding box
- `(x_max, y_max)` – coordinate dell'angolo in basso a destra del bounding box

Figura 32: Esempio face detection [34]

- `age-gender-recognition-retail-0013,` riporta le stime dell'età e del sesso per ogni faccia rilevata. Questa rete è convoluzionale e riesce a riconoscere l'età e il sesso delle persone in un range di [18, 75] anni, non è applicabile per i bambini, dato che le loro facce non sono presenti del training set. Il dataset di validazione contiene più di ventimila soggetti che rappresentano diverse etnie, età e sesso.

  Input

  Come input la rete prende un'immagine con la seguente forma: [1x3x62x62] nel formato [1xCxHxW]. L'ordine dei colori aspettato è BGR.

  Output

  In output la mostrerà due etichette con l'età e il sesso della faccia rilevata.

- `head-pose-estimation-adas-0001`, riporta la posa della testa con gli angoli di Tait-Bryan (roll, pitch e yaw). È una rete convoluzionale basata sul dataset Biwi Kinect Head Pose.

  Input

  Come input la rete prende un'immagine con la seguente forma: [1x3x60x60] nel formato [1xCxHxW]. L'ordine dei colori aspettato è BGR.

  Output

  Il livello di output sarà nel seguente formato:

  1 – "angle_y_fc" [1,1] – Stima dell'angolo yaw (in gradi)
  2 – "angle_p_fc" [1,1] – Stima dell'angolo pitch (in gradi)
  3 - "angle_r_fc" [1,1] – Stima dell'angolo roll (in gradi)

  Ogni output contiene un valore float che rappresenta un valore nella rappresentazione in angoli Tait-Bryan (yaw, pitch e roll).

- `emotion-recognition-retail-0003`, questa rete riporta l'emozione (neutrale, triste, felice, sorpreso, arrabbiato) per ogni faccia rilevata. Basata sul dataset AffectNet [35].

  Input

  Come input la rete prende un'immagine con la seguente forma: [1x3x64x64] nel formato [1xCxHxW]. L'ordine dei colori aspettato è BGR.

  Output

  La probabilità per ogni emozione corrispondente.

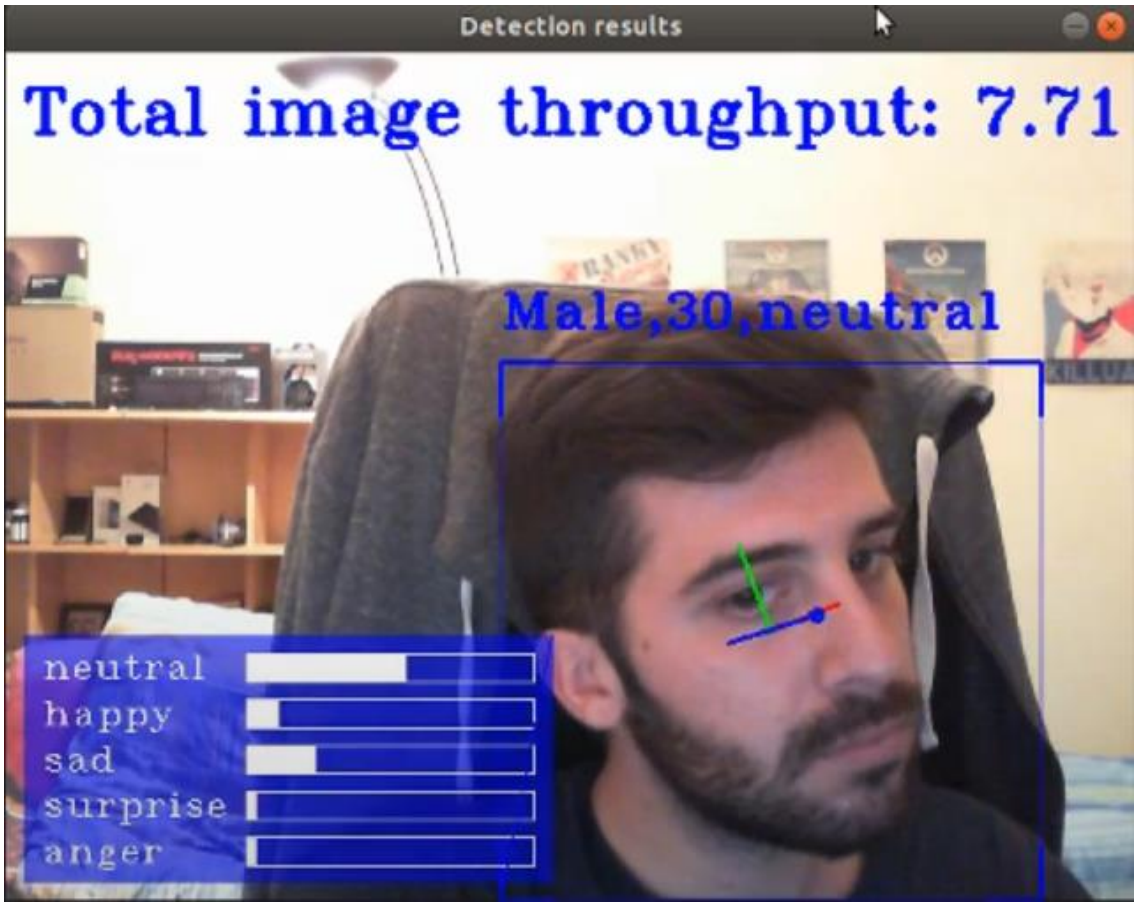

Figura 33 – Esempio output demo interactive_face_detection_demo

4.2 Strumenti utilizzati

Per effettuare i test e le modifiche è stato utilizzato il toolkit OpenVINO su un sistema operativo Ubuntu 18.04 a 64bit su un PC desktop con un Intel Core i5-6500, 16 GB RAM DDR4 insieme alla Neural Compute Stick 2.
Dopo aver installato tutte le dipendenze necessarie il toolkit OpenVINO si è eseguito il Verification Script per verificare il corretto funzionamento dei componenti del toolkit in seguito si sono installati i driver della NCS 2. L'applicazione è scritta interamente in C++.

4.3 Come funziona

L'applicazione legge i parametri passati dal terminale di comando e carica le reti a seconda di quali son state scelte. Per ogni frame catturato attraverso la libreria OpenCV esegue la predizione sulla rete Face Detection e successivamente esegue in modo parallelo le altre predizioni usando le altre reti passate per poi mostrare i risultati su schermo. Inoltre è possibile far eseguire una determinata predizione a un dispositivo diverso come si può vedere nella Figura 35.

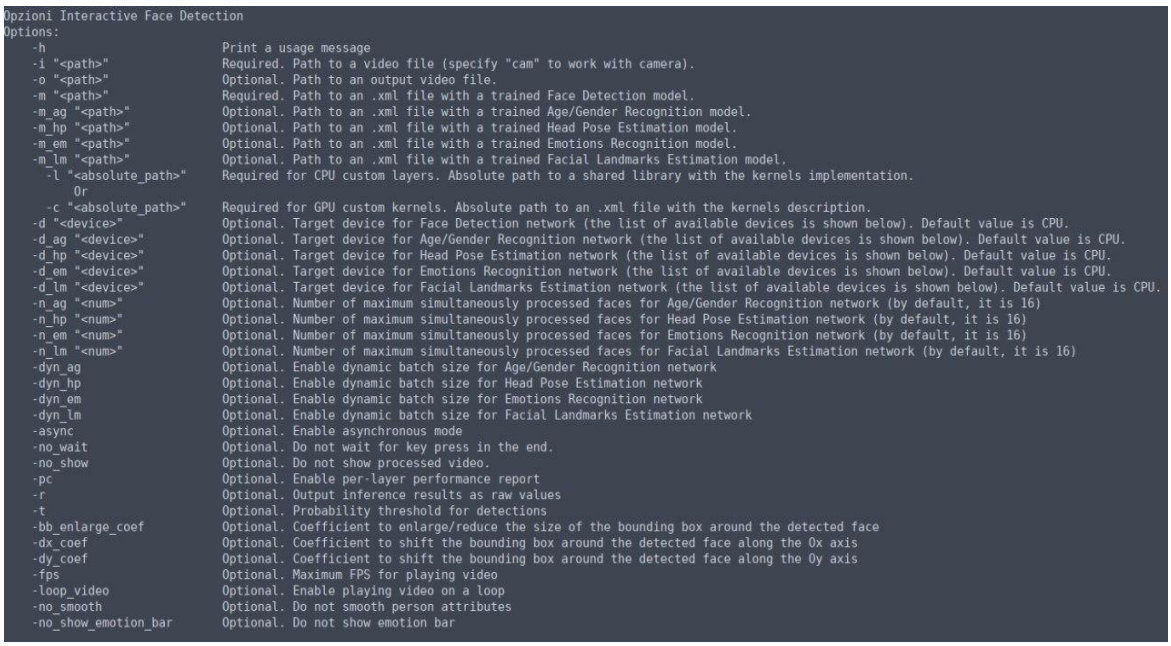

Figura 34: Lista di tutti i comandi per l'applicazione

```
Interactive Face Detection Demo
./interactive_face_detection_demo \
-i cam \
-m /home/giox/models/FP16/face-detection-retail-0004-fp16.xml \
-d CPU \
-m_ag /home/giox/models/FP16/age-gender-recognition-retail-0013-fp16.xml \
-d_ag MYRIAD \
-m_hp /home/giox/models/FP16/head-pose-estimation-adas-0001-fp16.xml \
-d_hp MYRIAD \
-m_em /home/giox/models/FP16/emotions-recognition-retail-0003-fp16.xml \
-d_em MYRIAD \
-r
```

Figura 35: Esempio comando, il primo modello è eseguito dalla CPU, mentre gli altri tre dalla NCS 2 (MYRIAD)

4.4 Descrizione delle Classi della applicazione

L'applicazione creata è strutturato da varie classi fondamentali per il suo funzionamento, alcune che gestiscono i risultati delle predizioni e altre predisposte per la generazione delle etichette, tabelle e bounding box.

- detectors.hpp
- detectors.cpp
- face.hpp
- face.hpp
- interactive_face_detection.hpp
- visualizer.hpp
- visualizer.cpp
- main.cpp

4.4.1 Gestione delle predizioni

Nei file detectors.hpp e detectors.cpp vengono definite le strutture che consentono di gestire le predizioni attraverso le reti. Vengono dichiarate 4 strutture, corrispettivamente `FaceDetection`, `AgeGenderDetection`, `HeadPoseDetection` e `EmotionDetection` che derivano dalla struttura `BaseDetection` e in ognuna di

queste strutture viene fatto l'override del metodo `InferenceEngine::CNNNetwork read()` che si occupa di leggere il modello della rete, controllare i parametri di input e fornire gli output per ogni task corrispondente in parallelo.

```
struct FaceDetection : BaseDetection
{
    struct Result
    {
        int label;
        float confidence;
        cv::Rect location;
    };
    std::ofstream outfile;
    std::string input;
    std::string output;
    double detectionThreshold;
    int maxProposalCount;
    int objectSize;
    int enquedFrames;
    float width;
    float height;
    float bb_enlarge_coefficient;
    float bb_dx_coefficient;
    float bb_dy_coefficient;
    bool resultsFetched;
    std::vector<std::string> labels;
    std::vector<Result> results;

    FaceDetection(const std::string &pathToModel,
                  const std::string &deviceForInference,
                  int maxBatch, bool isBatchDynamic, bool isAsync,
                  double detectionThreshold, bool doRawOutputMessages,
                  float bb_enlarge_coefficient, float bb_dx_coefficient,
                  float bb_dy_coefficient);

    InferenceEngine::CNNNetwork read() override;
    void submitRequest() override;

    void enqueue(const cv::Mat &frame);
    void fetchResults();
};
```

Figura 36: struct FaceDetection

Nel file face.cpp è presente la struttura Face che è incaricata per la gestione della descrizione delle facce riconosciute per ogni frame, mentre nel file visualizer.cpp sono presenti delle classi che eseguono delle generiche routine per la visualizzazione dei risultati ottenuti, le più importanti sono:

- la classe `PhotoframeVisualizer` – Usata per disegnare il rettangolo attorno la faccia.
- la classe `HeadposeVisualizer` – Usata per disegnare gli assi per la descrizione dell'orientamento della faccia.
- il metodo della classe `Visualizer`, `drawFace` – per disegnare la tabella delle emozioni.

4.4.2 Workflow main.cpp

Ora verranno descritte le fasi fondamentali implementate nell'entrypoint dell'applicazione.

Fase pre-processing:

- Analisi e validazione degli argomenti di input.
- Caricamento Inference Engine.
- Controllo dei frame.
- Istanziazione delle strutture `FaceDetection`, `AgeGenderDetection`, `EmotionDetection`, `HeadposeDetection.`
- Lettura dei modelli in formato IR e caricamento dei plugin.

Fase processing:

- Esecuzione simultanea delle reti per il riconoscimento della faccia, sesso, età, posa della testa e delle emozioni.

Fase post-processing:

- Elaborazione delle informazioni di output per ogni frame.
- Processamento dei risultati in formato leggibile.

- Visualizzazione dei risultati, delle informazioni sul numero di frame processati e del valore di throughput.

4.5 Modifiche aggiunte per l'estrazione dei parametri

Uno degli obbiettivi del lavoro è stato quello di aggiungere delle funzionalità a questa demo. L'idea era di riuscire ad estrapolare le informazioni delle predizioni di ogni frame e salvarli in un formato processabile, in modo tale da poterci effettuare delle operazioni in modo più semplice. Per implementare questa funzionalità si è aggiunta la libreria "JSON for Modern C++ [36]" che permette di salvare le informazioni delle predizioni in formato JSON.

Come si può vedere in figura 34, è presente il comando -r che permette di stampare a video sul terminale gli output delle predizioni, ed è qua che è stata aggiunta la scrittura delle informazioni sul file JSON, esattamente nel file detectors.cpp. Quindi, aggiungendo il comando -r dopo i vari comandi obbligatori l'applicazione creerà per ogni frame un oggetto JSON con le informazioni sulle informazioni ricavate precedentemente e le scriverà sul file.

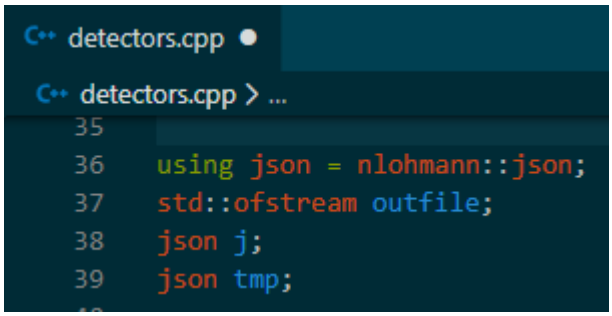

Figura 37: inizializzazione JSON

detectors.cpp ●

detectors.cpp > AgeGenderDetection::operator[](int) const

```cpp
AgeGenderDetection::Result AgeGenderDetection::operator[](int idx) const
{
    Blob::Ptr genderBlob = request->GetBlob(outputGender);
    Blob::Ptr ageBlob = request->GetBlob(outputAge);

    AgeGenderDetection::Result r = {ageBlob->buffer().as<float *>()[idx] * 100,
                                    genderBlob->buffer().as<float *>()[idx * 2 + 1]};
    if (doRawOutputMessages)
    {

        tmp["frameN"] = frame;
        tmp["soggetto"] = idx;
        tmp["male prob"] = r.maleProb;
        tmp["age"] = r.age;
        frame++;

    }

    return r;
}
```

Figura 38 – Estrapolazione dati riguardanti il sesso e l'età

detectors.cpp > ...

```cpp


HeadPoseDetection::Results HeadPoseDetection::operator[](int idx) const
{
    Blob::Ptr angleR = request->GetBlob(outputAngleR);
    Blob::Ptr angleP = request->GetBlob(outputAngleP);
    Blob::Ptr angleY = request->GetBlob(outputAngleY);

    HeadPoseDetection::Results r = {angleR->buffer().as<float *>()[idx],
                                    angleP->buffer().as<float *>()[idx],
                                    angleY->buffer().as<float *>()[idx]};

    if (doRawOutputMessages)
    {
        tmp["roll"] = r.angle_r;
        tmp["pitch"] = r.angle_p;
        tmp["yaw"] = r.angle_y;
        j.push_back(tmp);
        std::ofstream o("pretty.json");
        o << std::setw(4) << j << std::endl;
    }

    return r;
}
```

Figura 39: Estrapolazione dati dell'orientamento della testa

```cpp
std::map<std::string, float> EmotionsDetection::operator[](int idx) const
{
    // Vector of supported emotions
    static const std::vector<std::string> emotionsVec = {"neutral", "happy", "sad", "surprise", "anger"};
    auto emotionsVecSize = emotionsVec.size();

    Blob::Ptr emotionsBlob = request->GetBlob(outputEmotions);

    /* emotions vector must have the same size as number of channels
     * in model output. Default output format is NCHW, so index 1 is checked */
    size_t numOfChannels = emotionsBlob->getTensorDesc().getDims().at(1);
    if (numOfChannels != emotionsVecSize)
    {
        throw std::logic_error("Output size (" + std::to_string(numOfChannels) +
                               ") of the Emotions Recognition network is not equal "
                               "to used emotions vector size (" +
                               std::to_string(emotionsVec.size()) + ")");
    }

    auto emotionsValues = emotionsBlob->buffer().as<float *>();
    auto outputIdxPos = emotionsValues + idx * emotionsVecSize;
    std::map<std::string, float> emotions;

    if (doRawOutputMessages)
    {
        // add a number that is stored as double (note the implicit conversion of j to an object)
        outfile.open("inference-result.txt", std::ios_base::app);
        outfile << "[" << idx << "] element, predicted emotions (name = prob):" << std::endl;
        outfile.close();
    }

    for (size_t i = 0; i < emotionsVecSize; i++)
    {
        emotions[emotionsVec[i]] = outputIdxPos[i];

        if (doRawOutputMessages)
        {
            tmp[emotionsVec[i]] = outputIdxPos[i];
        }
    }
```

Figura 40: Estrapolazione risultati delle emozioni.

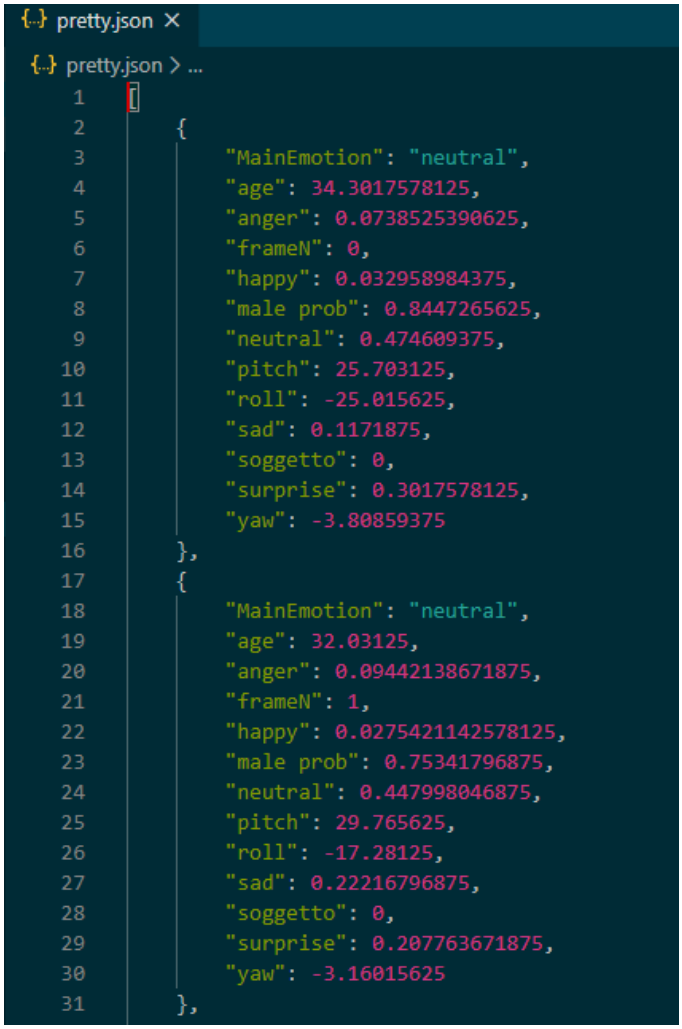

Figura 41: il file JSON con le informazioni sulle predizioni

4.5.1 Sviluppo Endpoint e Heatmap

Per usufruire delle informazioni ricavate dalle predizioni e presenti nel JSON è stata sviluppata una applicazione in HTML e Javascript. L'applicazione è stata pensata in modo tale che si riesca ad avere una panoramica generale delle informazioni presenti dal file JSON in modo semplice e intuitivo. L'endpoint prende in ingresso il file JSON, popolato con le informazioni delle predizioni per ogni frame e calcola il numero di frame processati, i soggetti, età media ed emozione prevalente. Inoltre è stata aggiunto il calcolo di una heatmap che è generata in base alla direzione in cui i soggetti stanno guardando più frequentemente, calcolata grazie alle informazioni sull'orientamento della faccia. Schiacciando il tasto "Visualizza Risultati" verrà generata la heatmap.

Figura 42: Endpoint http e heatmap

4.5.2 Calcolo della heatmap

Per il calcolo della heatmap è stata usata la libreria heatmap.js [37]. Per ogni frame si crea un punto nella heatmap e a seconda della frequenza di un punto in quell'intorno la zona diventa sempre più rossa.

```
        while (numFrame--) {
            val = 2;
            max = 10;
            min = 1;
            var point = {
                x: 400 + Math.floor(getTanFromDegrees(javaobj[numFrame].yaw) * distanza),
                y: 200 + Math.floor(getTanFromDegrees(javaobj[numFrame].pitch) * distanza),
                value: val
            };
            points.push(point);
            console.log(point);
        }
        var data = {
            min: min,
            max: max, // il valore per cui il pixel viene colorato in rosso
            data: points
        };
        heatmapInstance.setData(data);
    }
};
```

Figura 43: Creazione heatmap

Per il calcolo dei punti della heatmap è stata usata una formula trigonometrica, che dato l'angolo (yaw e pitch) e la distanza tra testa e il punto osservato troviamo la distanza rispetto al centro che corrisponderà alle coordinate x e y del punto osservato. Dopo aver istanziato il container della heatmap, nel file .css si sceglie la grandezza desiderata. Il conteggio dei pixel parte in alto a sinistra, avendo scelto una grandezza di 800x400 pixel, il centro della heatmap sarà quindi x:400 e y:200 che sarà il punto di osservazione della testa.

4.6 Test prestazioni

Hardware (4 modelli in parallelo)	**Throughput (fps)**
4 modelli con NCS 2	8.7
4 modelli con CPU i5-6500 (precisione fp32)	29.7
2 modelli con CPU(i5-6500) + 2 modelli NCS 2	14.3
4 modelli con CPU (Intel Celeron N3450)	4.9
2 modelli con CPU(Intel Celeron N3450) + 2 modelli NCS 2	5.9

Nella tabella in alto sono riportate i valori delle performance dell'applicazione con diverse configurazioni. Cercando di dividere il carico di lavoro tra CPU e NCS notiamo un migliorando rispetto al primo caso, dove la NCS esegue tutti i modelli in parallelo. Le prestazioni migliori le abbiamo usando esclusivamente la CPU, però in termini energetici non è ottimizzata come la NCS. Mentre utilizzando una CPU meno potente, la Intel Celeron N3450, facendo eseguire tutti i quattro modelli notiamo una netta diminuzione di prestazioni.

Capitolo 5 Conclusioni e sviluppi futuri

I risultati ottenuti non sono molto accurati come ci si potrebbe aspettare da sistemi più complessi equipaggiati con strumenti più potenti. Tuttavia, la soluzione implementata potrebbe funzionare in quelle realtà che non possono avere un supporto tecnologico costoso ma che vogliono modernizzarsi e utilizzare questi strumenti per aumentare la loro produttività o effettuare analisi specifiche.
Questa soluzione potrebbe aumentare le sue performance utilizzando in parallelo più NCS, parallelizzando il carico di lavoro.
Dei possibili casi d'uso dove si potrebbe applicare questo progetto potrebbero essere delle applicazioni di sentiment analysis, strumenti da usare nell'area della sanità, tracciamento degli oggetti nella sicurezza, sistemi di guida autonoma per i veicoli e tanti altri.

Questo progetto mostra come si potrebbe sfruttare il toolkit OpenVINO per lo sviluppo di applicazioni di deep learning in modo semplice e intuitivo anche senza l'ausilio di dispositivi potenti e costosi, grazie alla Neural Computer Stick 2.
Tutto questo è possibile sfruttando le numerose demo, modelli pre-addestrati in rete, o in casi particolari, usando una propria rete addestrata appositamente per ciò che si desidera, che verrà poi convertita nel formato compatibile con il toolkit.
Il framework è altamente integrabile con tante applicazioni industriali e strumenti di vario tipo consentendo un larga interoperabilità.
Questo lavoro è stato svolto anche con l'intento di voler contribuire a trovare un modo più semplice e immediato per affacciarsi allo sviluppo e test di applicazioni di machine learning, cercando di eliminare gli ostacoli che possono scoraggiare lo sfruttamento di questa innovativa tecnologia.

Lista abbreviazioni

ML	Machine Learning
DL	Deep Learning
ANN	Artificial Neural Network
IA	Intelligenza Artificiale
GPU	Graphic Processor Unit
VPU	Visual Processor Unit
SOM	System-on-module
NCS	Neural Compute Stick
API	Application Programming Interface
SHAVE	Streaming Hybrid Architecture Vector Engine
FPGA	Field-Programmable Gate Array
IPU	Image Processing Unit
CPU	Central Processing Unit

Lista delle figure

Figura 1: Differenza tra reti neurali 11
Figura 2: AI, ML e DL 12
Figura 3: Vari modelli di Nvidia Jetson 14
Figura 4: Comparazione neurone biologico e neurone artificiale 15
Figura 5: Neurone Artificiale 16
Figura 6: Gradiente discendente 18
Figura 7: Esempio CNN 20
Figura 8: Esempio architettura di una rete convoluzionale 21
Figura 9: Dimostrazione dell'operazione di convoluzione in 2D 22
Figura 10: Esempio di un pool con un filtro 2x2 e passo 2 23
Figura 11: Esempio architettura CNN 24
Figura 12: Intel Neural Compute Stick 2 26
Figura 13: Intel Movidius Myriad X 27
Figura 14: Architettura Shave 29
Figura 15: Diagramma workflow dello sviluppo con la NCS 30
Figura 16: Diagramma workflow del toolkit OpenVINO 30
Figura 17: Logo OpenVINO 31
Figura 18: Rappresentazione grafica dei componenti software del toolkit 32
Figura 19: Tabella dei formati supportati 32
Figura 20: Esempio Model Optimizer 34
Figura 21: Plug-in Inference Engine 35
Figura 22: Logo OpenCV 37
Figura 23: Esempio applicazione *simple_classifier_cpp_camera* 38
Figura 24: Esempio applicazione *mnist_calc* 39
Figura 25: Benchmark test sulla NCS 2 40
Figura 26: Benchmark test sulla CPU Intel Core i5-6500 40
Figura 27,28,29: benchmark della CPU Intel Core i5-8500 con vari modelli 43
Figura 30: Benchmark NCS 2 con vari modelli 44
Figura 31: NCS 2 collegata a un drone 46
Figura 32: Esempio bounding box 48

Figura 33: Esempio demo interactive_face_detection_demo ……... 49
Figura 34: Lista di tutti i comandi per l'applicazione ……... 50
Figura 35: Esempio comando, il primo modello è eseguito dalla CPU, mentre gli altri tre dalla NCS 2 (MYRIAD) ……..... 50
Figura 36: struct FaceDetection …… 51
Figura 37: inizializzazione JSON ……. 53
Figura 38:Estrapolazione dati riguardanti il sesso e l'età ……. 54
Figura 39: Estrapolazione dati dell'orientamento della testa ……..... 54
Figura 40: Estrapolazione dati delle emozioni. ……. 55
Figura 41: il file JSON con le informazioni sulle predizioni …… 56
Figura 42: Endpoint http e heatmap …….. 57
Figura 43: Creazione heatmap …….. 57

Bibliografia

[1] G. Balinggan, 2019. [Online]. Available: http://www.differencebetween.net/science/differencebetween-artificial-intelligence-and-human-intelligence/..

[2] A. Minini, 2019. [Online]. Available: http://www.andreaminini.com/ai/machine-learning/. .

[3] Nvidia, «Nvidia Jetson,» [Online]. Available: https://www.nvidia.com/it-it/autonomous-machines/embedded-systems.

[4] I. Liljeqvist, «The Essence of Artificial Neural Networks,» [Online]. Available: https://medium.com/@ivanliljeqvist/the-essence-of-artificial-neural-networks-5de300c995d6.

[5] Wikipedia, «Connessionismo,» [Online]. Available: https://it.wikipedia.org/wiki/Connessionismo.

[6] M. R. Gribskov, K. Nakai, S. Ranganathan e C. Schönbach, Encyclopedia of bioinformatics and computational biology.

[7] «Sigmoid Function,» [Online]. Available: https://en.wikipedia.org/wiki/Sigmoid_function.

[8] «Hyperbolic Function,» [Online]. Available: https://en.wikipedia.org/wiki/Hyperbolic_function..

[9] «Rectifier Linear Unit,» [Online]. Available: https://en.wikipedia.org/wiki/Rectifier_(neural_networks).

[10] «Batch Normalization,» [Online]. Available: https://en.wikipedia.org/wiki/Batch_normalization.

[11] N. Zheng e P. Mazumder, Learning in Energy - Efficient Neuromorphic Computing Algorithm and Architecture, 2019.

[12] «CS231n Convolutional Neural Networks for Visual Recognition,» [Online]. Available: https://cs231n.github.io/convolutional-networks/.

[13] C. D. I. B. A. J. Dolza, «3D fully convolutional networks for subcortical segmentation,» [Online]. Available: https://arxiv.org/pdf/1612.03925.pdf.

[14] «Deep Learning Notebook,» [Online]. Available: http://www.deeplearningbook.org/contents/convnets.html.

[15] A. Deshpande, «A Beginner's Guide To Understanding Convolutional Neural Networks,» 2019. [Online]. Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/.

[16] Intel, «Introducing the Intel Neural Compute Stick 2,» [Online]. Available: https://software.intel.com/en-us/articles/run-intel-openvino-models-on-intel-neural-compute-stick-2.

[17] «ncs2 img,» [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/intel-neural-compute-stick-2-and-open-source-openvino-toolkit.html.

[18] Movidius, «NCSDK,» [Online]. Available: https://github.com/movidius/ncsdk/releases.

[19] «Modelli supportati Myriad,» [Online]. Available: https://docs.openvinotoolkit.org/latest/_docs_IE_DG_supported_plugins_MYRIAD.html.

[20] Wikipedia, «Architettura Shave,» [Online]. Available: https://en.wikichip.org/wiki/movidius/microarchitectures/shave_v2.0.

[21] I. Movidius, «Workflow Diagram with NCS,» [Online]. Available: https://movidius.github.io/ncsdk/.

[22] Intel, «Model Optimizer,» [Online]. Available: https://docs.openvinotoolkit.org/latest/_docs_MO_DG_prepare_model_Prepare_Trained_Model.html.

[23] «Support Intuiface,» [Online]. Available: https://support.intuiface.com/hc/en-us/articles/360013098099-Interface-Asset-Face-Detection-with-OpenVINO-.

[24] Intel, «Introducing the Intel® Neural Compute Stick 2,» [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/run-intel-openvino-models-on-intel-neural-compute-stick-2.html.

[25] Intel, «Supported Devices,» [Online]. Available: https://docs.openvinotoolkit.org/latest/_docs_IE_DG_supported_plugins_Supported_Devices.html.

[26] Intel, «Using the Model Optimizer,» [Online]. Available: https://www.youtube.com/watch?v=kY9nZbX1DWM&list=PLDKCjIU5YH6jMzcTV5_cxX9aPHsborbXQ.

[27] Intel, «Inference Engine,» [Online]. Available: https://docs.openvinotoolkit.org/latest/_docs_IE_DG_inference_engine_intro.html.

[28] W. G. I. Intel Corporation, «OpenCV,» [Online]. Available: https://opencv.org/.

[29] «Neural Compute Application Zoo,» [Online]. Available: https://github.com/movidius/ncappzoo.

[30] «GoogLeNet,» [Online]. Available: https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet.

[31] «MNIST database,» [Online]. Available: https://it.wikipedia.org/wiki/MNIST_database.

[32] Intel, «performance benchmarks,» [Online]. Available: https://docs.openvinotoolkit.org/latest/_docs_performance_benchmarks.html#core-i5.

[33] Intel Movidius, «High performance, low power for AI inference,» [Online]. Available: https://www.intel.com/content/dam/support/us/en/documents/boardsandkits/neural-compute-sticks/NCS2_Product-Brief-English.pdf.

[34] Intel, «face-detection-adas-0001,» [Online]. Available: https://docs.openvinotoolkit.org/2019_R1/_face_detection_adas_0001_description_face_detection_adas_0001.html.

[35] M. H. Mahoor, «AffectNet,» [Online]. Available: http://mohammadmahoor.com/affectnet/.

[36] N. Lohmann, «Json for Modern C++,» [Online]. Available: https://github.com/nlohmann/json.

[37] P. Wied, «Heatmap.js,» [Online]. Available: https://www.patrick-wied.at/static/heatmapjs/.

[38] «Rete neurale artificiale,» [Online]. Available: https://it.wikipedia.org/wiki/Rete_neurale_artificiale.

[39] Y. B. a. A. C. I. Goodfellow, «ConvNets,» [Online]. Available: http://www.deeplearningbook.org/contents/convnets.html.