

# Studio e realizzazione di modelli Deep Learning con Hardware Low Cost

Alessandro Napoletano<sup>1</sup> - Corso di Calcolatori Elettronici e Reti di Calcolatori -  
Università Politecnica delle Marche<sup>2</sup>

Gennaio 2022 - Febbraio 2022

**Abstract—**In questa relazione vengono spiegate le origini, il funzionamento e le capacità di calcolo richieste nel Deep Learning, sottobranca dell'intelligenza artificiale.  
Al termine verranno poi mostrati alcuni modelli realizzati con l'ausilio della libreria TFLite di TensorFlow, ottimizzata per le applicazioni DL con Hardware low cost

**Index Terms—**Accelerazione hardware, parallelizzazione, TPU, GPU, TensorFlow, TFLite

## I. INTRODUZIONE AL DEEP LEARNING

Il Deep Learning (apprendimento profondo, abbreviato in DL) indica quella branca dell'AI (Artificial Intelligence) che fa riferimento alla risoluzione di task tramite algoritmi ispirati dalla struttura e dalle funzioni del cervello biologico animale, mediante reti neurali artificiali. Il Deep Learning, fa parte di una più ampia famiglia di metodi del Machine Learning (apprendimento automatico, abbreviato in ML): Il ML utilizza algoritmi tradizionali per analizzare i dati, apprendere da essi ed in base a quelli prenderà delle decisioni, il tutto affiancato all'intervento dell'uomo. Il DL struttura invece gli algoritmi in modo da generare una rete neurale artificiale, la rete neurale apprende dai dati e prende decisioni in autonomia, senza che ci sia alcun intervento umano ad "insegnare come apprendere e come riconoscere" dai dati ricevuti.

Il Deep Learning si basa dunque sull'utilizzare un insieme di tecniche basate su reti neurali artificiali per realizzare modelli di apprendimento su più layer (livelli), dove ogni strato è necessario per il successivo affinché l'informazione venga elaborata in maniera sempre più completa.

Le architetture di Deep Learning vantano svariate applicazioni in svariati ambiti quali: computer vision, riconoscimento automatico della lingua parlata, elaborazione del linguaggio naturale, riconoscimento audio e nella Bioinformatica.

<sup>1</sup>Alessandro Napoletano è uno studente della facoltà di Ingegneria Informatica e dell'Automazione presso l'Università Politecnica delle MarcheGitHub : <https://github.com/AlexNapoletano>

<sup>2</sup>Università Politecnica delle Marche (UNIVPM), Università con polo d'Ingegneria dell'Informazione, situata nella città d'Ancona, Italia. Website: <https://www.univpm.it/Entra/>

## A. STORIA DEL DEEP LEARNING

Lo psicologo Frank Rosenblatt è considerato da molti come il padre del Deep Learning, le cui basi vengono descritte nel suo libro "Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms" del 1962. Nel 1958 presentò al mondo Perceptron, una rete neurale artificiale<sup>1</sup> alimentata da un computer con scopo molto chiaro: riconoscere le tessere contrassegnate a destra da quelle a sinistra. Perceptron suscitò molto scalpore grazie al suo successo, tuttavia la macchina si fermò per problemi di limitazioni del single layer.

Il sogno di Rosembatt riprese vigore nei primi anni Ottanta grazie ad Hinton e LeCun, i quali pubblicarono uno studio attraverso il quale veniva proposto un percorso per insegnare alle reti neurali a correggere gli errori; ma nonostante questi eventi passati ed ulteriori studi non citati, il termine "Deep Learning" fu ideato per la prima volta da Rina Dethcher nel 1986 e nelle reti neurali artificiali con Igor Aizenberg solo nel 2000, nel contesto dei boolean threshold neurons. Occorre arrivare ai giorni nostri per vedere sviluppi significativi e validi:

Nel 2012 furono presentati gli straordinari risultati ottenuti in un esperimento condotto dal professor Hinton a Toronto durante il contesting ImageNet, esperimento indirizzato al riconoscimento visivo: riconoscere uomini ed animali tramite il confronto con milioni di altre immagini, senza il bisogno di un intervento da parte dell'uomo.

Nel 2015 ImageNet tornò presentando i risultati di una ricerca che ha dell'incredibile. Riuscirono a creare un deep learning su 152 livelli di astrazione. Un livello notevole se confrontato con i 3 livelli sui quali ci si basava prima di questa ricerca.

Il concetto di livello di astrazione è di facile comprensione: se viene utilizzato un solo algoritmo ha un livello di astrazione, se utilizza due algoritmi ha due livelli di astrazione e così via. Microsoft riuscì ad effettuare 152 operazioni differenti sulla stessa immagine! Da questi concetti è deducibile una verità fondamentale,

<sup>1</sup>una rete neurale artificiale, è un modello computazionale composto da neuroni artificiali, ispirato vagamente dalla semplificazione di una rete neurale biologica. Esse ricevono segnali esterni su uno strato di nodi (unità di elaborazione) d'ingresso, ciascuno dei quali è collegato con numerosi nodi interni, organizzati in più livelli. Ogni nodo elabora i segnali ricevuti e trasmette il risultato a nodi successivi.

più sono i livelli di astrazione, più la macchina ha una grande capacità di apprendimento [tratto dall'articolo di intelligenzaartificiale.it indicato nelle reference].

## B. IL MACHINE LEARNING

Anche il Machine Learning quindi insegna ai calcolatori ciò che è naturale per l'essere umano, ovvero apprendere dall'esperienza. Gli algoritmi di Machine Learning trovano dei pattern naturali nella ricerca dei dati per effettuare decisioni migliori o per fare previsioni, questi algoritmi vengono oggi utilizzati in ausilio dell'uomo anche in ambiti dove sono richieste decisioni critiche, ad esempio:

- Finanza computazionale, per credit scoring ed algoritmi di trading.
- Biologia computazionale, ricerca e riconoscimento di tumori, scoperta di nuovi farmaci, e sequenziamento del DNA.
- Automotive, settore aeroespaziale e manufatturiero, per la manutenzione predittiva.
- Image processing e computer vision, per il riconoscimento facciale, rilevamento del movimento, rilevamento oggetti.

La più grande sfida del ML è quella di gestire i dati e trovare il giusto modello da applicare, questo perché i dati utilizzabili possono essere disordinati, confusionari e/o incompleti. La necessità dell'utilizzo del Machine Learning per la risoluzione di particolari task, nasce proprio per quei casi in cui abbiamo tantissimi dati, con le caratteristiche sopra indicate e tantissime variabili, ma non esiste alcuna particolare equazione per la risoluzione del compito stesso. La scelta del modello più adatto al dataset diventa quindi importantissimo, Il Machine Learning usa diversi tipi di algoritmi per allenare un modello a seconda dello scopo dello stesso (Figura 1):

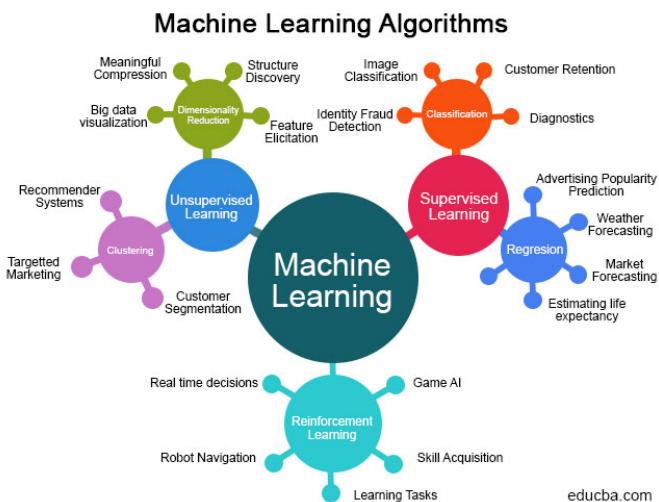


Fig. 1. Algoritmi di ML

"<https://cdn.educba.com/academy/wp-content/uploads/2019/08/Categories-of-Machine-Learning.jpg>"

I modelli Supervised (supervisionati) vengono utilizzati

quando c'è la necessità di allenare un modello a fare predizioni ed indentificazioni. Mentre gli Unsupervised (non supervisionati) vengono utilizzati principalmente per la suddivisione in categorie degli input, hanno inoltre un utilizzo diffuso nei motori di ricerca (Clustering).

Approfondiamo ora questi particolari modelli di sviluppo per algoritmi di Machine Learning.

**1) APPRENDIMENTO NON SUPERVISIONATO:** È una tecnica basata unicamente sugli input che verranno poi classificati ed organizzati sulla base di caratteristiche comuni per cercare di effettuare ragionamenti e previsioni sugli input successivi. L'apprendimento non supervisionato studierà quindi pattern nascosti o strutture intrinseche per trovare un inferenza tra i dati in input.

Un esempio comune di applicazione dell'apprendimento non supervisionato lo abbiamo nei motori di ricerca, dove data una o più parole chiave, questi programmi saranno in grado di creare una lista di link rimandanti alle pagine che l'algoritmo di ricerca ritiene attinenti alla ricerca effettuata. La validità di questi algoritmi è legata all'utilità delle informazioni che riescono ad estrarre dalla base di dati con l'attinenza dei link trovati e l'argomento contenuto in essi. I modelli unsupervised possono essere raggruppati in:

- Associazione- è il processo di comprensione delle relazioni, ad esempio come le persone che acquistano X tendano anche ad acquistare Y.
- Clustering- un problema di clustering, è quello dove si desidera comprendere la logica delle relazioni intrinseca nei dati, ad esempio raggruppare gli animali in base ad alcune caratteristiche in comune.

### 2) APPRENDIMENTO SUPERVISIONATO:

L'apprendimento supervisionato è una tecnica di apprendimento automatico che mira a costruire un modello in grado di elaborare autonomamente previsioni sui valori di uscita rispetto agli input, il tutto sulla base di esempi ideali costituiti da coppie di input/output fornite all'inizio. Questo tipo di algoritmo prende dunque dei set di input conosciuti con output conosciuti, ed allenano il modello a generare previsioni ragionevoli per rispondere ai nuovi dati. Per l'apprendimento supervisionato viengono utilizzate due tecniche:

- Tecnica di classificazione: questa tecnica prevede risposte discrete, i modelli di classificazione classificano i dati in input in categorie, ed in base degli input/output forniti all'inizio, allena il modello.
- Tecnica di regressione lineare: Si tratta di un metodo di stima del valore atteso di una variabile dipendente Y, partendo dai valori di altre variabili indipendenti X<sub>1</sub>, X<sub>2</sub>, ..., X<sub>k</sub>. Questo tipo di algoritmo può essere utilizzato ad esempio per la predizione della nazionalità di una persona.

**3) APPRENDIMENTO RINFORZATO:** Questa famiglia di modelli è composta da algoritmi che utilizzano gli errori stimati come ricompense o penalità. Se l'errore è grande, la penalità è alta e la ricompensa bassa. Se l'errore è piccolo, la penalità è bassa e la ricompensa alta.

La ricerca dell'errore di prova e la ricompensa ritardata sono le caratteristiche più rilevanti dell'apprendimento per rinforzo. Questa famiglia di modelli consente la determinazione automatica del comportamento ideale all'interno di un contesto specifico al fine di massimizzare le prestazioni desiderate. Il feedback sulla ricompensa è necessario affinché il modello impari quale azione è la migliore e questo è noto come "il segnale di rinforzo".

## II. COME FUNZIONA IL DEEP LEARNING

Come accennato nel precedente capitolo, il Deep Learning, basa il suo funzionamento sulla classificazione e selezione dei dati più rilevanti per giungere ad una conclusione emulando la metodologia di funzionamento del cervello umano che, per formulare una risposta ad un quesito, dedurre un'ipotesi logica ed arrivare alla risoluzione di un problema, mette in moto i propri neuroni e le relative sinapsi. I Neuroni biologici interconnessi formano le reti neurali cerebrali, le quali permettono a ciascun individuo di ragionare, fare calcoli, riconoscere suoni, immagini, volti, imparare ed agire.

Come già detto, il DeepLearning si comporta allo stesso modo sfruttando le Reti Neurali Artificiali (Artificial Neural Network, abbreviato in ANN), sul Modello degli studi di "McCulloch e Pitts del 1943"<sup>2</sup>, attraverso algoritmi di calcolo matematico-informatici basati su modelli costituiti da interconnessioni di informazioni.

Una rete neurale si presenta come un sistema "adattivo" in grado di modificare la sua struttura (i nodi e le interconnessioni) basandosi sia su dati esterni sia su informazioni interne che si connettono e passano attraverso la rete neurale durante la fase di apprendimento e ragionamento.

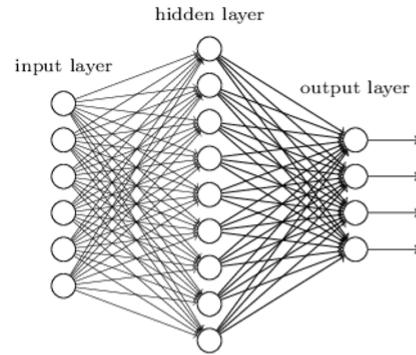
Il Deep Learning simula i processi di apprendimento del cervello attraverso sistemi artificiali per insegnare alle macchine non solo ad apprendere autonomamente, ma a farlo in modo più "profondo", dove per profondo s'intende "su più livelli".

Le Deep Neural Network (Reti neurali profonde, abbreviato in DNN) sfruttano un numero maggiore di strati intermedi chiamati "hidden layer", vale a dire i layer nascosti nella rete neurale per costruire più livelli di astrazione ed ottenere un risultato migliore tramite la convoluzione degli stessi. Quelle "tradizionali" contengono 2-3 layer, mentre le reti neurali profonde possono contenerne oltre 150. (Figura 2).

Proviamo a fare un esempio di funzionamento di una Deep Neural Network con task di riconoscimento visivo dei pattern: i neuroni del primo layer potrebbero imparare a riconoscere i bordi, i neuroni nel secondo layer potrebbero imparare a riconoscere forme più complesse come triangoli o rettangoli, create dai bordi. Il terzo layer potrebbe riconoscere forme ancora più complesse, il quarto potrebbe

<sup>2</sup>Esperimento sull'analogia Cervello/Macchina di Turing per dimostrare che il neurone era l'unità logica di base del cervello

"Non-deep" feedforward neural network



Deep neural network

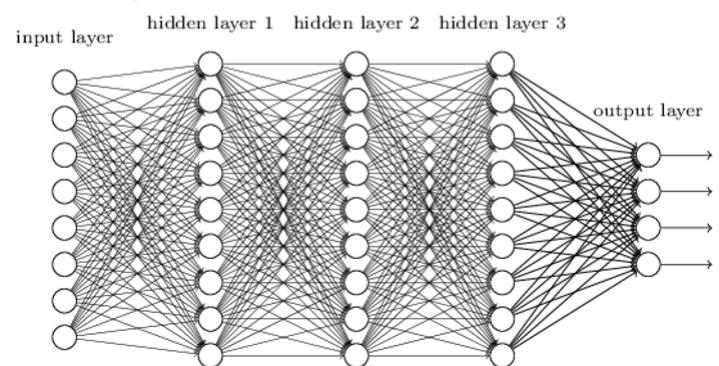


Fig. 2. NeuralNetwork e DeepNeuralNetwork  
["www.ai4business.it/intelligenza-artificiale/deep-learning/deep-learning-cose/"](http://www.ai4business.it/intelligenza-artificiale/deep-learning/deep-learning-cose/)

dedicarsi al riconoscere i dettagli e così via.... È chiaro che i molteplici livelli di astrazione possono dare alle DNN un enorme vantaggio nell'imparare a risolvere problemi sempre più complessi di riconoscimento, proprio perché ad ogni hidden layer vengono aggiunte informazioni ed analisi utili a fornire un output sempre più affidabile. Di contro la scalabilità della rete neurale è strettamente correlata ai data set, ai modelli matematici e alle risorse computazionali onerose.

### A. DIFFERENZA TRA DEEP LEARNING E MACHINE LEARNING

Seppur la richiesta di capacità computazionali del Deep Learning possa rappresentare un limite, la scalabilità della stessa, grazie all'aumento dei dati disponibili e degli algoritmi, è ciò che lo differenzia dal Machine Learning: i sistemi di DL migliorano le proprie prestazioni all'aumentare dei dati, mentre le applicazioni di Machine Learning una volta raggiunto un certo livello di performance non sono più scalabili neppure tramite l'aggiunta di ulteriori dati di training alla rete stessa (Figura 3) Questo perché nei sistemi di Machine Learning le caratteristiche di un determinato oggetto (nel caso di sistemi di riconoscimento visivo) vengono estratte, selezionate manualmente ed usate per creare un modello in

grado di categorizzare gli oggetti in base alla classificazione di quelle caratteristiche; nei sistemi di Deep Learning, invece, l'estrazione delle caratteristiche avviene in modo automatico: la rete neurale apprende in modo autonomo come analizzare dati grezzi e come svolgere un determinato task.

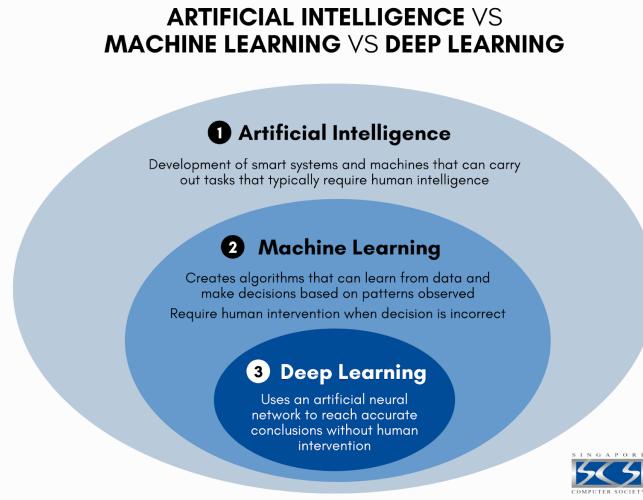


Fig. 3. Immagine tratta da:  
["www.scs.org.sg/articles/  
 machine-learning-vs-deep-learning"](http://www.scs.org.sg/articles/machine-learning-vs-deep-learning)

#### B. ADDESTRAMENTO DI UN SISTEMA DEEP LEARNING

Vista la differenza tra machine learning e deep learning, vediamo come viene effettuato, teoricamente, l'addestramento della rete neurale: Mentre gli algoritmi di Machine Learning sono lineari, gli algoritmi di Deep Learning sono caratterizzati da una gerarchia crescente di complessità ed astrazione data dalla quantità dei suoi layer. Ma come funziona di base l'apprendimento?

Immaginiamo che un bambino impari la parola "albero", indicando ciò che è un albero e ciò che non è un albero, affiancato dai genitori che risponderanno "Sì" o "No", il bambino imparerà a riconoscere effettivamente quella figura. Mano a mano che continua a puntare cose, il bambino prenderà sempre più coscienza delle caratteristiche che contraddistinguono un albero da tutto ciò che non lo è. Ciò che il bambino fa a sua insaputa, è chiarire un'astrazione complessa come il concetto stesso di albero costruendo una gerarchia in cui ogni livello di astrazione viene creato con la conoscenza che è stata acquisita dallo strato precedente. A differenza del bambino, che impiegherà un tempo indeterminato tra settimane/mesi ad apprendere, un'applicazione che utilizza algoritmi di Deep Learning può rilevare e ordinare milioni di immagini, identificando in pochi minuti e con precisione quali di queste contengano i set di dati, pur non avendo ricevuto alcun tipo di istruzione sull'identificazione dei dati nel training stesso.

Solitamente, nei sistemi di Deep Learning, l'unica accortezza degli scienziati è "etichettare" i dati inserendo ad esempio un

metatag "albero" all'interno delle immagini che contengono un albero, ma senza spiegare al sistema come riconoscerlo. Il sistema stesso, attraverso livelli gerarchici multipli, intuirà cosa caratterizza un albero (il tronco, le foglie, ecc.) e quindi come riconoscerlo.

Questi sistemi si basano su un processo di apprendimento "trial-and-error", ma purché l'output finale sia affidabile sono necessarie enormi quantità di dati! Pensare subito ai Big Data e alla facilità con cui oggi si producono e distribuiscono dati di qualsiasi forma e da qualsiasi fonte come facile risoluzione sarebbe però un errore: l'accuratezza dell'output richiede, almeno nella prima fase di addestramento, l'utilizzo di dati "etichettati" ciò significa che l'utilizzo di dati non strutturati potrebbero rappresentare un problema. Dati non etichettati possono essere analizzati da un modello di apprendimento profondo solo una volta che esso abbia raggiunto un certo livello di accuratezza. Non solo, i sistemi basati su Deep Learning sono difficili da addestrare a causa del numero stesso di strati nella rete neurale. Il numero di strati e collegamenti tra i neuroni nella rete è tale che può diventare difficile calcolare le "regolazioni" che devono essere apportate in ogni fase del processo di training, questo perché durante la fase di allenamento si usano i cosiddetti algoritmi di retropropagazione dell'errore (backpropagation) attraverso il quale si rivedono i pesi della rete neurale in caso di errori. In questo caso la rete propaga all'indietro l'errore in modo che i pesi delle connessioni vengano aggiornati in modo più appropriato.

#### III. ARCHITETTURE HARDWARE PER DEEP LEARNING

Se dal punto di vista delle potenzialità il Deep Learning può sembrare più interessante e complesso del Machine Learning, di contro però, il calcolo computazionale richiesto per il suo funzionamento è considerevole tanto quanto la spesa economica richiesta: le CPU più avanzate e le GPU top di gamma necessarie per sostenere gli elevati carichi di lavoro di un sistema di Deep Learning sono ancora eccessivamente costosi. Una possibile soluzione è quella di ricorrere a capacità computazionali via Cloud tramite "affitto", tuttavia anche questa soluzione attenua solo in parte il problema perché per creare una DNN è richiesta l'elaborazione di grandi quantità di dati utilizzando cluster di GPU di fascia alta per moltissime ore, non è quindi detto che acquistare "as a service" la capacità di calcolo necessaria mediante servizi online come ad esempio AWS<sup>3</sup> risulti economico.

Le principali soluzioni per le DNN sono basate su accelerazioni hardware ad elevato parallelismo, ovvero allo sviluppo di dispositivi hardware e di librerie software che permettano un utilizzo ottimale della parallelizzazione delle operazioni. Le architetture più adatte allo scopo sono: GPU e Tensor Core, acceleratori hardware FPGA ed infine Acceleratori ASIC.

<sup>3</sup>AWS= Amazon Web Services, mediante l'istanza EC2 permette l'affitto di GPU Server come soluzione di clouding computing

## A. HARDWARE ACCELERATOR PER L'INTELLIGENZA ARTIFICIALE

Possiamo dire che nei sistemi convenzionali, i processori lavorano in modo approssimativamente "sequenziale" ovvero, durante l'esecuzione di un algoritmo, le istruzioni vengono eseguite una per volta; i sistemi di accelerazione hardware invece ne migliorano l'esecuzione mediante il parallelismo delle istruzioni. La parallelizzazione garantirà principalmente un vantaggio in velocità riducendo notevolmente il tempo necessario al training della rete neurale, permettendo quindi di eseguire il task richiesto in modo più efficiente sia in termini di velocità che di consumo energetico.

Quando si parla di hardware per l'Intelligenza Artificiale, si fa riferimento ad alcuni tipi di acceleratori come ad esempio gli NPU (Neural Processing Unit) ovvero a classi di microprocessori, o microchip, progettati per consentire elaborazioni rapide di applicazioni nelle reti neurali in ambiti di visione artificiale, algoritmi di ML per la robotica, IoT e molte altre applicazioni ancora.

Poiché la necessità di risorse computazionali, sia per l'addestramento che per l'inferenza di reti neurali sempre più complesse, cresce in modo esponenziale ad oggi siamo in attesa di una nuova generazione di chip che abbiano peculiari capacità:

- più potenza di calcolo ed efficienza: le soluzioni hardware AI di nuova generazione dovranno essere più potenti e più efficienti in termini di costi e consumo energetico;
- cloud e edge computing: le nuove architetture in silicio dovranno supportare il deep learning, le reti neurali e gli algoritmi di visione artificiale, con modelli specifici per applicazioni in cloud e in edge;
- insight rapidi: essere in grado di fornire soluzioni di AI, sia software che hardware, alle aziende per analizzare molto più rapidamente il comportamento e le preferenze dei clienti, al fine di migliorare il servizio;
- nuovi materiali: vengono effettuate nuove ricerche per passare dal tradizionale silicio ai chip di elaborazione ottica per sviluppare sistemi molto più veloci rispetto alle tradizionali CPU o GPU;
- nuove architetture: nuovi tipi di architetture come i chip neuromorfi.

## B. ARCHITETTURE AD ELEVATO PARALLELISMO: GPU E TENSOR CORE

Le architetture vettoriali basate sul modello di computazione "Single Instruction Multiple Data (SIMD)" sono l'ideale per raggiungere alte prestazioni per carichi di lavoro caratterizzati da un massivo parallelismo dei dati soggetti alla computazione stessa.

Tra le prime architetture SIMD apparse sul mercato troviamo la famiglia di processori Intel SSE (per la gestione dello streaming) ed AVX (per il calcolo vettoriale) che include istruzioni vettoriali specializzate per reti neurali (VNNI).

In particolare, l'architettura AVX-512 rappresenta l'estensione SIMD a 512 bit dell'architettura Intel x86

proposta nel 2013. Il principale vantaggio introdotto da quest'architettura consiste nella compatibilità del codice sviluppato per processori x86.

Nonostante Intel abbia deciso nel 2018 di interromperne la produzione, la famiglia di processori Intel Xeon Phi<sup>4</sup> è stata per molti anni il concorrente diretto delle GPU di AMD e Nvidia.

La famiglia di schede grafiche Radeon di AMD è stata creata per applicazioni nel campo videoludico (Sviluppo di giochi tripla A e Gaming) e del Deep Learning.

Sono state annunciate da pochissimo delle nuove GPU AMD Radeon Super Resolution con tecnologia FSR (FidelityFX Super Resolution) che si concentreranno a migliorare la qualità d'immagine dei giochi senza alcun tipo di ottimizzazione. NVIDIA di contro utilizzerà sulle GPU GeForce la tecnologia DLSS (Deep-Learning Super Sampling). Anche sulle GPU viene adottato il modello di calcolo parallelo SIMD, ciò le rende ottime in plurimi ambiti applicativi, grazie anche al modello di programmazione Compute Unified Device Architecture (CUDA) di Nvidia. Tale modello consente ai programmati di sfruttare l'elevata capacità di parallelizzazione che contraddistingue le architetture delle GPU (raggiungono throughput elevatissimi per l'elaborazione parallela dei thread di dati) sia per applicazioni general purpose (GP-GPU) che per i carichi di lavoro di elaborazione grafica per la quale sarebbero state progettate.

*1) LA PARALLELIZZAZIONE - SIMD:* Il modello di calcolo SIMD (Single Instruction Multiple Data) consiste in un elevato numero di processori identici che eseguono la stessa sequenza di istruzioni su insiemi di diversi di dati.

I processori SIMD sono spesso usati dai supercomputer e, con alcune varianti, anche nei moderni microprocessori. Questo modello è composto da un'unica unità di controllo che esegue un'istruzione alla volta controllando più ALU che invece operano in maniera sincrona in modo che tutti gli elementi eseguano la stessa istruzione scalare, ma ciascuno su un dato differente. Una CPU basata su questo modello è anche detta Array Processor.

Nell'elaborazione di dati multimediali, spesso si incontrano algoritmi che possono avvantaggiarsi di un'architettura SIMD, ad esempio: per cambiare la luminosità di un'immagine si dovrebbe caricare ogni pixel nei registri del microprocessore, effettuarne la modifica della luminosità ed infine salvarne i risultati in memoria. Un processore SIMD eseguirebbe prima un'istruzione che caricherebbe contemporaneamente un certo numero di pixel (quanti dipende dall'architettura), poi li modificherebbe tutti assieme con un'altra istruzione ed in seguito li salverebbe in memoria. È chiaro che eseguire le operazioni a blocchi anziché sequenzialmente, fa sì che le operazioni vengano

<sup>4</sup>La suddetta tecnologia fu implementata nella famiglia di processori Intel Xeon Phi (Knights Landing) usati nei supercalcolatori e nella famiglia Xeon Skylake-X

svolte più efficientemente.

Per quello che riguarda la parallelizzazione tramite GPU, il procedimento è del tutto analogo, quello che cambia è semplicemente l'architettura interna hardware che approfondiremo leggermente sul paragrafo dedicato alle architetture ASIC. Va tenuto a mente che la parte critica dell'esecuzione SIMD con GPU è l'accesso alla Memoria Globale ed alla Shared Memory, decisamente più lente dei registri interni, necessitano quindi di algoritmi d'accesso ben ponderati onde evitare conflitti.

### C. ACCELERATORI BASATI SU FPGA ed ASIC

Le implementazioni hardware seguono generalmente due diverse stili di progettazione: da un lato i circuiti configurabili di tipo FPGA (Field Programmable Gate Array)<sup>5</sup> e dall'altro i circuiti hardware dedicati di tipo ASIC.

Ambo gli stili di progettazione sono focalizzati sulla riduzione della latenza nella fase di inferenza delle DNN nel rispetto dei vincoli sul consumo di potenza per applicazioni edge computing.

Inizialmente ideati per essere usati in fase di prototipazione e nello sviluppo di applicazioni Embedded, gli acceleratori FPGA hanno raggiunto fette di mercato significative grazie all'incremento della densità di integrazione dei processi tecnologici ed all'impiego di strumenti automatici di sintesi ad alto livello per la progettazione on-field. Seguendo questa tendenza, l'impiego di acceleratori FPGA nelle applicazioni Deep Learning si è concentrato soprattutto nello svolgimento di due task:

- Per accelerare la fase di training dei modelli nei data center e nei centri di supercalcolo
- Nelle fasi di inferenza nelle soluzioni on-the-edge.

Negli ultimi anni sono state proposte diverse soluzioni basate su acceleratori FPGA sia in ambito di ricerca che in quello commerciale.

#### 1) ACCELERATORI DEDICATI DI TIPO ASIC:

Nell'ambito degli acceleratori dedicati di tipo ASIC (Application Specific Integrated Circuit)<sup>6</sup> sono state diverse le soluzioni proposte sia in ambito di ricerca, come l'architettura Eyeriss proposta dal MIT e basata sul modello data flow, che in ambito commerciale.

L'architettura della Tensor Processor Unit (TPU) è stata sviluppata da Google per accelerare sia l'hardware per il calcolo tensoriale che gli accessi alla memoria. Questa è particolarmente adatta alla fase di apprendimento delle

<sup>5</sup>Un FPGA, in elettronica digitale, è un dispositivo logico programmabile ovvero un dispositivo hardware formato da un circuito integrato le cui funzionalità logiche di elaborazione sono appositamente programmabili e modificabili tramite opportuni linguaggi.

<sup>6</sup>Un ASIC è un circuito integrato creato per risolvere un'applicazione di calcolo/elaborazione ben precisa (special purpose): la specificità della progettazione, focalizzata sulla risoluzione di un unico task, consente di raggiungere delle prestazioni in termini di velocità di processamento e consumo elettrico difficilmente ottenibili con l'uso di soluzioni general purpose

DNN ed è supportata dal modello di programmazione Cloud TPU che sfrutta al massimo il parallelismo dei dati nell'elaborazione di matrici dense.

Dal lato GPU, troviamo invece l'acceleratore di Nvidia, il quale si rivolge al mercato dell'edge computing con un'architettura simile a quella della TPU, questa è caratterizzata da due componenti principali:

- La Global Memory - svolge una funzione analoga alla RAM per le CPU (accessibile sia da GPU che da CPU)
- Lo Streaming Multiprocessor (abbreviato in SM) - esegue le computazioni, ogni SM ha la sua Control unit, i suoi registri, la pipeline di esecuzione e le cache.

Lo Streaming Multiprocessor è composto da più CUDA Core, la Shared Memory è una cache condivisa tra i Core dello Streaming Multiprocessor che permette la gestione parallela dei thread, dove per thread si intende l'operazione che dovrà essere svolta.

Su GPU la logica è quindi "Lancio un thread per ogni elemento".

Andando un po' più nel dettaglio potremo dire che il Kernel è la funzione che viene chiamata dall'host e che verrà eseguita in CPU, il Kernel sarà composto da tantissimi thread che verranno gestiti in modo parallelizzato dai CUDA Core. Un CUDA Thread block sarà associato ad un CUDA Streaming Multiprocessor e più blocchi potranno poi accedere in modo concorrente ad uno Streaming Multiprocessor; la parte di esecuzione di un thread, quindi quella di un blocco e della convoluzione dei risultati è del tutto analoga al procedimento SIMD già approfondito.

L'architettura Orlando di STMicroelectronics è stata introdotta nel 2017 come system-on-chip a bassissima dissipazione di potenza per reti neurali adatte al mercato dei sistemi embedded ed ai dispositivi IoT.

In particolare, il system-on-chip di Orlando include un microcontrollore ARM Cortex, 8 Digital Signal Processor e un sistema riconfigurabile composto da 8 acceleratori convoluzionali dotati di connessione DMA.

## IV. TENSOR FLOW PER LO SVILUPPO DI APPLICAZIONI DEEP LEARNING

TensorFlow è una piattaforma end-to-end per l'apprendimento automatico, atta a semplificare l'utilizzo e la creazione di modelli Machine Learning sia per programmati esperti che per neofiti, questo grazie alla partecipazione unitaria tipica delle comunità Open Source ed anche a guide e tutorial fornite da TensorFlow stessa.

Tensorflow permette allo sviluppatore di utilizzare API di alto livello basate sullo standard "API Keras" per la definizione e per l'addestramento delle reti neurali, Keras è una API intuitiva che consente la prototipazione rapida, la ricerca e la produzione. Possiamo dire che questa è una "seconda" generazione di API che non richiede una conoscenza profonda del Deep Learning, ma ne basta una superficiale e permette svariati utilizzi di algoritmi, sia nell'ambito della ricerca scientifica, che nell'ambito della

produzione. Ad oggi queste API sono alla base di moltissimi prodotti commerciali di Google.

La piattaforma Tensorflow è compatibile con i principali sistemi operativi a 64bit come Windows, Linux e Mac OSX e le librerie disponibili possono funzionare su numerosi tipi di CPU e GPU. Sebbene anni addietro si parlasse di limitazione hardware, ad oggi esiste anche una variante dedicata al sistema operativo Android e Raspberry e quindi per hardware lowcost meno performanti.

#### A. TENSORFLOW LITE

TensorFlow Lite è un'architettura logica, una struttura (framework) che consente l'apprendimento automatico su dispositivi Mobile/IoT tramite l'utilizzo e la concessione di librerie, modelli preallenati, inferenza sul dispositivo ed API ad alto livello. La piattaforma è caratterizzata da ottimizzazione per l'apprendimento automatico sul dispositivo affrontando i vincoli di latenza, privacy, connettività, dimensione binaria e consumo energetico; offre pieno supporto a piattaforme multiple che vanno dagli smartphone Android ed iOS ai microcontrollori, oltre all'ottimizzazione dei modelli e delle accelerazioni hardware consentendo alte prestazioni di elaborazione. Il supporto linguistico include: Java, Swift, Objective-C, C++ e Python, ed è possibile realizzare e provare codice anche on Cloud tramite Google Colab. Nel nostro progetto le libreria di TFLite è stata richiamata per la creazione di un modello con metadati personalizzato. Le stringhe di codice necessarie post installazione sono le seguenti (Figura 4)

```

# Import pathlib
# Import pandas as pd
# Import seaborn as sns
import pydot
import cartopy
import libarchive
import matplotlib.pyplot as plt
# Serve per rappresentare classi di percorsi filesystem per diversi sistemi operativi
# Serve per fare analisi sui dati, è anche un manipulation tool
# Serve per fare grafici statistici

import numpy as np
# Serve per fare grafici
# Serve per fare calcoli matriciali

import tensorflow as tf
# Importa libreria di TensorFlow
assert tf.__version__.startswith('2')

tf.get_logger().setlevel('ERROR')
from absl import logging
logging.set_verbosity(logging.ERROR)

#####
# ESEMPIO D.L. RILEVAMENTO OGGETTI DI TFLITE #####
#####

```

Fig. 4. utilizzo della libreria TFLite

#### B. ANDROID STUDIO

Android Studio è un ambiente di sviluppo integrato (IDE) per lo sviluppo di applicazioni Smartphone appartenenti alla piattaforma Android. Il software uscì in anteprima nel 2013 e la prima build stabile risale a Dicembre 2014. Questo IDE è basato sul software JetBrains IntelliJ IDEA<sup>7</sup>, questo ambiente ci permette quindi di scrivere un App Android in linguaggio Java e provarlo sull'emulatore fornito dallo stesso.

La piattaforma è disponibile su Windows, Mac OSX e Linux e sostituisce il vecchio Android Development Tools

<sup>7</sup>JetBrain è un'azienda di sviluppo di software, IntelliJ IDEA è invece un IDE per la programmazione in JAVA

di Eclipse diventando così l'IDE principale di Google per lo sviluppo nativo di App Android.

Nel progetto, questo IDE avrà una certa importanza in quanto utilizzeremo un App di Object Detection già scritta e funzionante condivisa da TensorFlow tramite GitHub, applicheremo prima un modello preallenato di TFLite ed infine con un modello con metadati realizzato da me tramite i servizi di Google Cloud.

### V. STUDIO DEI MODELLI DI DEEP LEARNING CON TFLITE

Il progetto di sviluppo si dividerà in due parti:

- Nella prima parte, proveremo un modello preallenato fornito da TensorFlow Lite per l'Object Detection e lo metteremo alla prova con nuovi input.
- Nella seconda parte invece proveremo un modello preallenato personalizzato, mettendo alla prova anche quest'ultimo.

Un modello di Object Detection viene addestrato per rilevare la presenza e la posizione di più classi di oggetti, ad esempio un modello potrebbe essere addestrato a riconoscere l'insieme dei cartelli stradali in una foto nella quale sono presenti anche automobili, persone e quant'altro. Quando un'immagine viene fornita al modello, essa produrrà un riquadro di delimitazione attorno a ciascun oggetto riconosciuto associandolo ad un punteggio che indicherà la certezza del risultato. Per ottimizzare la fase di training, abbiamo detto che è consigliato inserire modelli con Metadati.

Come immediatamente intuitibile, si parla di algoritmi di Computer Vision, ma attenzione, normalmente ci si confonde quando parliamo di Object Detection, Image Classification (Figura 5) e Segmentation. L'Object detection ci indica la locazione degli oggetti riconosciuti nell'immagine/video e li contorna con dei riquadri di delimitazione accompagnati da un etichetta che specifica cos'è l'oggetto, all'occorrenza può essere personalizzata con un indice di "affidabilità di riconoscimento". L'image classification fornisce invece la categoria dell'oggetto riconosciuto senza indicarcelo nell'immagine.

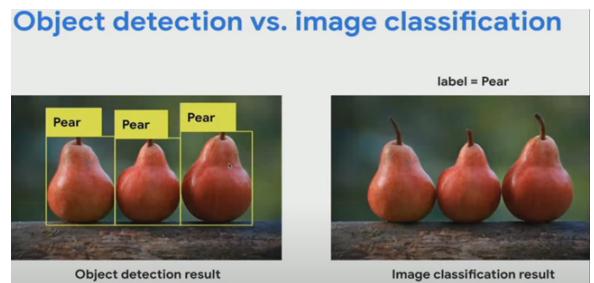


Fig. 5. Immagine tratta da: TensorFlow

La segmentazione<sup>8</sup>, per quanto molto diversa ad impatto visivo dal rilevamento oggetti, è in realtà concettualmente molto simile in quanto entrambe ci ritornano gli oggetti identificati. La differenza sta nel fatto che il risultato dell’analisi immagine soggetta a segmentazione ci da in uscita una maschera che mostra quali pixel appartengono all’oggetto riconosciuto colorandolo e separandolo dal resto, Figura 6).

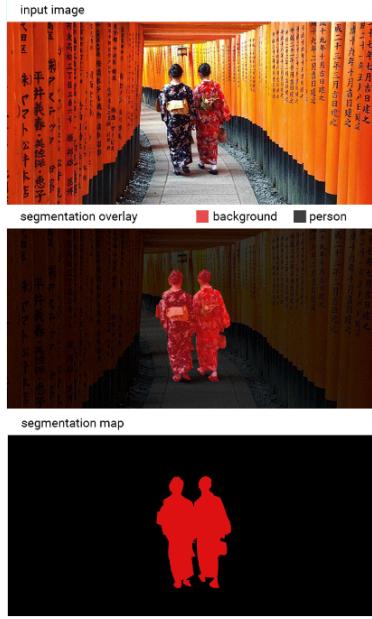


Fig. 6. Immagine tratta da:  
[“https://www.tensorflow.org/lite/examples/segmentation/overview”](https://www.tensorflow.org/lite/examples/segmentation/overview)

#### A. MODELLI TENSORFLOW LITE

Un modello TensorFlow Lite viene rappresentato in un formato portatile “FlatBuffers” identificato dall’estensione “.tflite”. Ciò offre diversi vantaggi come dimensioni ridotte ed inferenza più rapida, grazie all’accesso diretto ai dati, evitando passaggi di analisi e decompressione.

Un modello TensorFlow Lite può facoltativamente includere metadati (Figura V-A).

I metadati forniscono uno standard per le descrizioni dei modelli, essi sono infatti un’importante fonte di conoscenza su ciò che fa il modello e sulle sue informazioni di input/output. Essi sono costituiti da due parti:

- parti leggibili dall’uomo;
- parti leggibili dalla macchina, che possono essere sfruttate dai generatori di codici pipeline pre/post-elaborazione durante la fase di inferenza.

**1) FASE DI INFERENZA:** L’infereza si riferisce al processo di esecuzione di un modello sul dispositivo per effettuare previsioni in base ai dati di input. È possibile eseguire l’inferenza in due modi in base al tipo di modello utilizzato:

<sup>8</sup>La segmentazione dell’immagine è il processo di partizionamento di un’immagine digitale in più segmenti (insiemi di pixel, noti anche come oggetti immagine). L’obiettivo della segmentazione è semplificare e/o modificare la rappresentazione di un’immagine in qualcosa che sia più significativo e più facile da analizzare

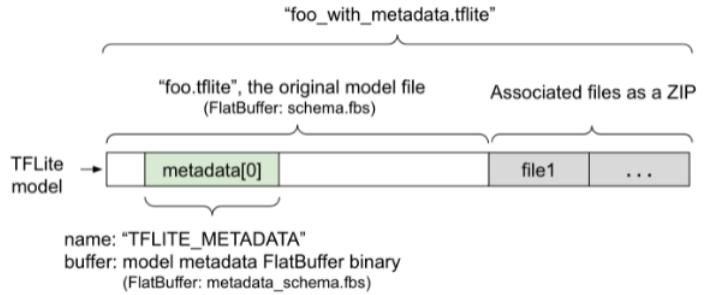


Figura 1. Modello TFLite con metadati e file associati.

- Modelli senza metadati: Utilizza l’API Interpreter di TensorFlow Lite.
- Modelli con metadati: permette di sfruttare le API predefinite utilizzando le librerie Opensource di TFLite o creare le proprie pipeline di inferenza personalizzate con le librerie di supporto.

È possibile migliorare le prestazioni sui dispositivi Android ed iOS sfruttando l’accelerazione hardware con GPU Delegate per ambo gli ecosistemi, NNAPI Delegate ed Hexagon Delegate solo per Android ed infine Core ML Delegate per iOS.

L’inferenza di modelli con metadati può essere semplice come poche righe di codice. I metadati di TensorFlow Lite contengono una ricca descrizione di ciò che fa il modello e di come utilizzarlo. Può consentire ai generatori di codice di generare automaticamente il codice di inferenza, ad esempio utilizzando la funzione di associazione ML di Android Studio o il generatore di codice Android TensorFlow Lite.

**2) API DI TFLITE E PIPELINE DI INFERENZA:** Le Application Program Interface (abbreviato come API) sono un insieme di procedure, spesso realizzate da compagnie esterne, che permettono il compimento di un dato compito del software. Il modo in cui le API vengono implementate è irrilevante per lo sviluppatore che, le utilizzerà semplicemente per comunicare con altri prodotti e servizi. Ciò porta quindi il codice ad una maggiore astrazione ed ad un più alto livello, spesso il termine API designa le librerie di un software di un linguaggio di programmazione.

La libreria delle attività di TensorFlow Lite ci fornisce interfacce di modello ottimizzate e pronte all’uso per attività di machine learning.

All’occorrenza è possibile creare pipeline di inferenza personalizzate tramite la libreria di supporto “TensorFlow Lite Support Library” personalizzando l’interfaccia del modello e/o creando la pipeline di inferenza stessa, questo perché la libreria di supporto contiene varietà di metodi utili e strutture dati per eseguire pre/post elaborazione e conversione dati. Questo processo non è stato però necessario per lo scopo del nostro studio, ed abbiamo utilizzato semplicemente API di default, sia per l’emulatore dello smartphone, che per tflite.

## B. MODELLO FORNITO E PREALLENATO DA TENSORFLOW LITE

Vediamo ora uno dei modelli preallenati per applicazioni mobile fornito da TensorFlow:

Nella pagina <https://tfhub.dev/tensorflow/collections/lite/task-library/object-detector/1> possiamo trovare una lista di modelli preallenati con dataset "COCO2017" ottimizzati per TFLite, i modelli preallenati sono utili quando vogliamo riconoscere oggetti generici senza pretendere troppo, ci risparmiano il tempo che si impiegherebbe ad allenare il modello ed a realizzare un dataset con tantissime immagini di riferimento da etichettare; tuttavia, un modello personalizzato ed allenato da noi può risultare più efficiente nel riconoscere oggetti tra immagini più complesse.

Per quanto riguarda il codice per l'applicazione Android scritta in Java o si possono scaricare App apposite da playstore oppure, possiamo scaricare Android Studio e decomprimere il file .zip scaricabile da questo link: <https://developers.google.com/codelabs/tflite-object-detection-android#2>.

L'approccio scelto è il secondo, una volta decompresso il file, troveremo una cartella di root chiamata "odml-pathways-main" con tutte le risorse necessarie a far funzionare l'app. Basterà importare su Android Studio il seguente path:

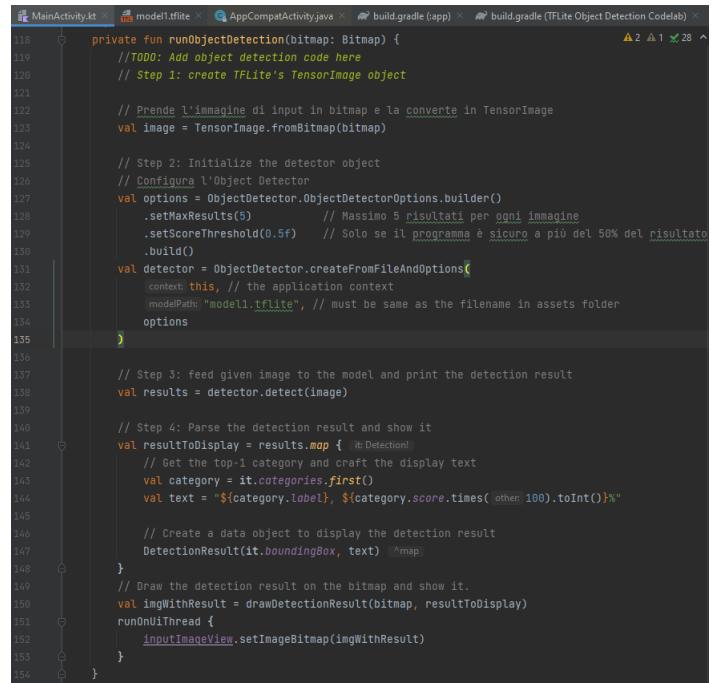
```
..\odml-pathways-main\object-detection\codelab2\android\starter.
```

Per eseguire il modello, l'App necessiterà l'esecuzione di tre passaggi: caricare l'immagine da elaborare, creare/configurare l'object detector e connettere i due passaggi precedenti fra loro, per fare tutto ciò basta implementare poche righe di codice al programma già scritto all'interno del "MainActivity.kt" (Figura 7) Una volta importato il programma, aggiunte le righe di codice inerente al training del modello sopra illustrate ed una volta caricato il modello "modell.tflite"<sup>9</sup>, basterà mandarlo in esecuzione o su emulatore o sul proprio smartphone<sup>10</sup>, il risultato ottenuto dall'esecuzione del software è stato il seguente. (Figura 8).

**1) APPROFONDIMENTI SUL CODICE:** L'app di avvio fornita da TensorFlow contiene del codice standard che fa già alcune cose di default, ci permette sia di scattare foto utilizzando la fotocamera del dispositivo che di usare alcune immagini di stock per provare il rilevamento di oggetti. Il metodo Java che utilizziamo per disegnare il rettangolo ed eseguire l'object detection è costituito, come detto nel paragrafo precedente, da tre parti:

<sup>9</sup>il modello va rinominato, ricercandolo sul sito citato corrisponde al nome: lite-model\_efficientdet\_lite0\_detection\_metadata\_1.tflite"

<sup>10</sup>dopo la ricerca del dispositivo sulla voce Device Manager in alto a destra, è necessario abilitare prima la modalità sviluppatore ed il debug tramite USB (all'occorrenza anche Debug WiFi). Android Studio permette un collegamento con i dispositivi Android anche tramite un QRCode



```

118 private fun runObjectDetection(bitmap: Bitmap) {
119     //TODO: Add object detection code here
120     // Step 1: create TFLite's TensorImage object
121
122     // Prende l'immagine di input in bitmap e la converte in TensorImage
123     val image = TensorImage.fromBitmap(bitmap)
124
125     // Step 2: Initialize the detector object
126     // Configura l'Object Detector
127     val options = ObjectDetector.ObjectDetectorOptions.builder()
128         .setMaxResults(5)           // Massimo 5 risultati per ogni immagine
129         .setScoreThreshold(0.5f)    // Solo se il programma è sicuro a più del 50% del risultato
130         .build()
131
132     val detector = ObjectDetector.createFromFileAndOptions(
133         context, // the application context
134         modelPath: "modell.tflite", // must be same as the filename in assets folder
135         options
136     )
137
138     // Step 3: feed given image to the model and print the detection result
139     val results = detector.detect(image)
140
141     // Step 4: Parse the detection result and show it
142     val resultToDisplay = results.map { itDetection! }
143         // Get the top-1 category and craft the display text
144         val category = it.categories.first()
145         val text = "${category.label}, ${category.score.times( other: 100).toInt()}%"
146
147         // Create a data object to display the detection result
148         DetectionResult(boundingBox, text) ^map_
149     }
150
151     // Draw the detection result on the bitmap and show it.
152     val imgWithResult = drawDetectionResult(bitmap, resultToDisplay)
153     runOnUiThread {
154         imgWithImageView.setImageBitmap(imgWithResult)
155     }
156 }

```

Fig. 7. Allenamento del modello

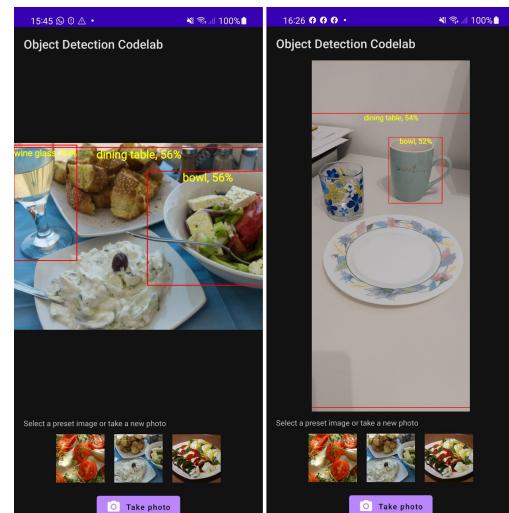


Fig. 8. Screenshot dei risultati ottenuti

- `fun runObjectDetection(bitmap: Bitmap)` - questo metodo<sup>11</sup> viene chiamato quando scegliamo un'immagine di preset o quando scattiamo una foto e scegliamo di effettuare l'object detection su quella. L'input per l'object detection è in formato bitmap.
- `data class DetectionResult(val boundingBoxes: Rect, val text: String)` - questa è una classe

<sup>11</sup>Tutti i linguaggi di programmazione forniscono la possibilità di definire, sotto un solo nome, interi gruppi di istruzioni o insiemi di linee di codice o blocchi di espressioni (statements), che dir si voglia. Così da poter riutilizzare porzioni di codice altrove.

In Java utilizziamo la logica definita dai blocchi istruzioni per rappresentare il comportamento di classi di oggetti e questi blocchi di codice prendono il nome di metodi.

che rappresenta il risultato dell'object detection. boundingBoxes è la forma rettangolare utilizzata dal programma per localizzare l'oggetto riconosciuto nella foto, e text è il risultato ottenuto in stringa da mostrare inerente all'affidabilità.

- `fundrawDetectionResult(bitmap: Bitmap, detectionResults: List<DetectionResult>):Bitmap` - questo metodo disegna i risultati del rilevamento ottenuto in `detectionResults` sul file `bitmap` in input e ne restituisce la copia modificata.

### C. MODELLO PERSONALIZZATO

Abbiamo visto e provato un modello preallenato già fornito da TensorFlow, ma come possiamo creare un modello personalizzato da noi?

Ad oggi la creazione di un modello personalizzato con metadati è un passaggio abbastanza semplice, Google fornisce la possibilità attraverso Google Cloud Platform (richiede iscrizione) di creare dei dataset, nel nostro caso per l'object detection, inserendo tantissime immagini. È possibile decidere se etichettare le immagini o meno, scegliendo quindi se usare un approccio supervised o unsupervised. Per migliorare l'efficienza del nostro modello, abbiamo scelto di etichettare tutte le 202 immagini selezionate. Il passaggio non è complicato, ma è estremamente dispendioso per ciò che riguarda il tempo necessario allo svolgimento (vanno etichettate tutte le classi da riconoscere per ogni foto, vedi Figura 9)

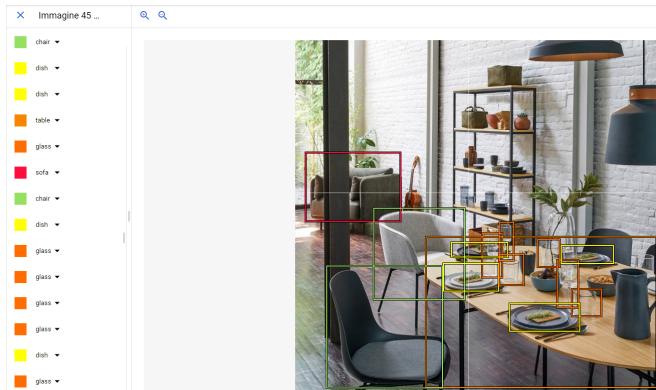


Fig. 9. fase di etichettatura dei dataset

La fase di training del modello può durare fino a più di 24h e viene eseguita autonomamente da Google. Una volta terminata questa, per implementare il nostro modello sull'App Android bisognerà eseguire i seguenti passaggi: Scaricare il modello preallenato, copiarlo ed incollarlo all'interno degli "asset" del programma su Android Studio ed aggiornarlo<sup>12</sup>. Dovremo inoltre modificare il codice Python su Google Colab. La cella "`model.evaluate(test_data)`" ci indicherà come l'algoritmo esegue la fase d'inferenza approssiandosi a dati del tutto nuovi. Il paramentro AP50 è

<sup>12</sup>si deve modificare il nome del modello all'interno della seguente stringa:

```
valdetector=ObjectDetector.createFromFileAndOptions(
```

uno dei più usati per l'object detection accuracy (un risultato di circa 0.39 è molto buono per applicazioni Mobile).

### VI. CONSIDERAZIONI FINALI

Dopo aver provato l'App con modello preallenato fornito da TFLite e con un modello custom preallenato da noi, possiamo osservare come si diversificano gli output sulle immagini stock fornite dall'App di Android Studio: Il modello da noi preaddestrato è stato in grado di riconoscere elementi, come i piatti, con precisioni più elevate, ma presentava lacune nel riconoscimento di bicchieri e tavolo. Al contrario il modello sviluppato da TensorFlow è stato in grado di riconoscere più elementi ma con inferiore precisione (vedi Figura 10).

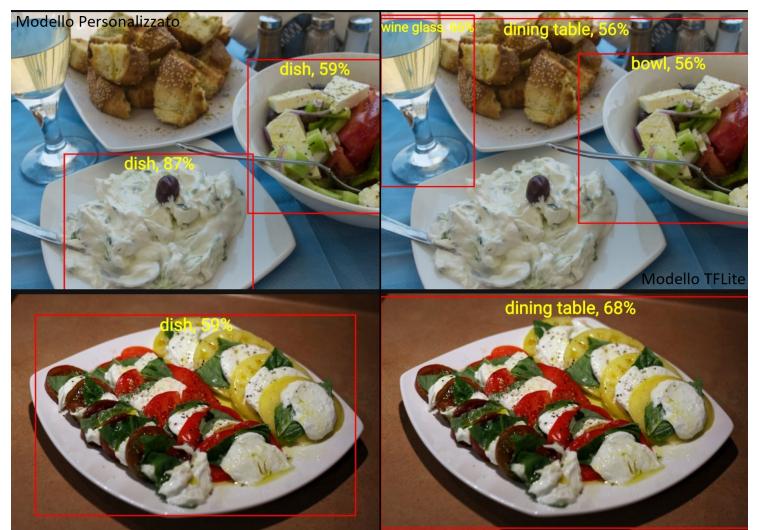


Fig. 10. Statistiche del modello personalizzato

Sempre tramite Google Cloud è possibile vedere le statistiche della fase di training, di tutte le immagini selezionate, una piccola parte è stata utilizzata senza etichette per verificare la precisione post training. Nel nostro caso le immagini di test sono state 37, ed i risultati riportati sono i seguenti: (vedi Figura 11)



Fig. 11. Statistiche del modello personalizzato

Un modello ad alta precisione fornisce pochi falsi positivi, mentre un modello a richiamo elevato produce meno falsi negativi. Dalle statistiche inerenti alle prestazioni, possiamo dedurre che per avere un modello più accurato sarebbero state necessarie più immagini nella fase di training, dato che il nostro modello riscontra un'alta confidenza (vedi Figura 12) nel riconoscere le classi con valori superiori al 0,5, come richiesto dall'App Android, tuttavia la vera efficacia si riscontra principalmente nei riconoscimenti relativi ad oggetti le cui etichette hanno valori superiori allo 0,62.



Fig. 12. Statistiche del modello personalizzato

## REFERENCES

- [1] W. Stallings, Computer Organization and Architecture - Designing for Performance. 10th Edition
- [2] Slide del corso di Calcolatori Elettronici e Reti di Calcolatori, Ingegneria Informatica e dell'Automazione presso Università Politecnica delle Marche
- [3] Slide CUDA Overview di Nvidia, Cliff Woolley Developer Technology Group
- [4] Supervised:  
[https://it.wikipedia.org/wiki/Apprendimento\\_supervisionato](https://it.wikipedia.org/wiki/Apprendimento_supervisionato)
- [5] Unsupervised:  
[https://it.wikipedia.org/wiki/Apprendimento\\_non\\_supervisionato](https://it.wikipedia.org/wiki/Apprendimento_non_supervisionato)
- [6] Reinforcement:  
[https://it.wikipedia.org/wiki/Apprendimento\\_per\\_rinforzo](https://it.wikipedia.org/wiki/Apprendimento_per_rinforzo)
- [7] Parallelizzazione SIMD:  
<https://it.wikipedia.org/wiki/SIMD>
- [8] Android Studio:  
[https://it.wikipedia.org/wiki/Android\\_Studio](https://it.wikipedia.org/wiki/Android_Studio)
- [9] Storia del DL:  
<https://www.intelligenzaartificiale.it/deep-learning/>
- [10] Differences Supervised and Unsupervised:  
<https://towardsdatascience.com/what-is-machine-learning-a-short-note-on-supervised-unsupervised-semi-supervised-and-aed1573ae9bb>
- [11] Deep Learning:  
<https://www.ai4business.it/intelligenza-artificiale/deep-learning/deep-learning-cose/>

- [12] MLvsDL:  
<https://www.scs.org.sg/articles/machine-learning-vs-deep-learning>
- [13] TensorFlow guide:  
<https://www.tensorflow.org/lite/guide>
- [14] Metadata TFLite:  
<https://www.tensorflow.org/lite/convert/metadata>
- [15] MachineLearning Algorithm image:  
<https://cdn.educba.com/academy/wp-content/uploads/2019/08/Categories-of-Machine-Learning.jpg>
- [16] Android Studio image:  
<https://developer.android.com/studio>
- [17] TFLite Object Detection examples:  
<https://developers.google.com/codelabs/tflite-object-detection-android#2>