**Relazione Progetto**

**Internet of Things based smart systems**

**Valutazione dell'Impatto della Riduzione della Precisione dei Pesi in una Rete Neurale nei Confronti dell'Accuratezza di Classificazione**

**Alessandro Messina, matricola O55000354**

**Orazio Scavo, matricola O55000414**

ANNO ACCADEMICO 2018/2019

Sommario

[***1. Introduzione*** - 3 -](#_Toc535319182)

[**1.1 Ambito applicativo** - 3 -](#_Toc535319183)

[**1.3 Flusso dell’analisi** - 3 -](#_Toc535319184)

[***2. Rete originale*** - 5 -](#_Toc535319185)

[**2.1 Progettazione della rete neurale** - 5 -](#_Toc535319186)

[**2.2 Allenamento della rete neurale** - 5 -](#_Toc535319187)

[**2.3 Valutazione dell’accuratezza della rete** - 5 -](#_Toc535319188)

[***3. Approximate Computing sulla rete neurale realizzata*** - 6 -](#_Toc535319189)

[**3.1 Scelta delle configurazioni approssimate** - 6 -](#_Toc535319190)

[***4. Valutazione dei risultati ottenuti*** - 7 -](#_Toc535319191)

[***5. Conclusioni*** - 8 -](#_Toc535319192)

# ***1. Introduzione***

Negli ultimi anni l’Approximate Computing è stato riscoperto come uno degli approcci più promettenti per la riduzione del consumo di energia in molte applicazioni caratterizzate da una certa tolleranza agli errori (si dice che queste applicazioni abbiano una *forgiving nature*). Questa tendenza è sicuramente legata alla crescente importanza che i consumi energetici assumono nella maggior parte dei dispositivi in uso al giorno d’oggi (dispositivi mobili, IoT, data center, etc.). La riduzione della quantità di energia e risorse richieste da parte di questi dispositivi comporta una serie di vantaggi non trascurabili, quale ad esempio nel caso di dispositivi IoT o mobili l’incremento della propria autonomia.

Uno degli ambiti che meglio si presta all’applicazione dell’Approximate Computing e quello delle *reti neurali,* in quanto spesso caratterizzate da una certa tolleranza agli errori, ma soprattutto capaci di auto-correggere gli errori introdotti dalle approssimazioni (*self error-healing*), grazie alla possibilità di eseguire ulteriori training, successivi all’approssimazione, dei parametri della rete. Con il retraining della rete, i pesi approssimati riescono a convergere verso dei valori tali da consentire una minimizzazione dell’effetto dell’errore introdotto dall’approssimazione e una conseguente massimizzazione dei vantaggi della stessa.

Uno dei principali problemi che nascono quando si lavora con delle reti neurali, avendo a disposizione delle risorse limitate, riguarda la quantità di memoria utilizzata per memorizzare i parametri della rete. Ridurre ad esempio il numero di bit per rappresentare tali pesi ha un impatto molto positivo sulla quantità di risorse richieste. La riduzione del numero di bit offre inoltre la possibilità di utilizzare circuiti aritmetici ridotti con una conseguente riduzione dell’area, potenza e possibilmente una riduzione del percorso critico e quindi un aumento della frequenza di clock. Una tale approssimazione ha però delle conseguenze anche sull'accuratezza della rete neurale. Lo scopo di questa trattazione è proprio quello di valutare tale impatto.

## **Ambito applicativo**

Per rendere più realistica l’analisi, è stata pensata una specifica applicazione in ambito IoT per la gestione di un parcheggio, in cui è richiesta la classificazione degli oggetti immortalati da una fotocamera posta all’entrata dello stesso, in modo da identificare diverse tipologie di veicolo o altri oggetti, la cui presenza in quel punto rappresenta una situazione non prevista. Per questo tipo di applicazione (classificazione) viene solitamente utilizzata una rete neurale di tipo convoluzionale (CNN), che data in input una certa immagine è capace di rilevare oggetti al suo interno e classificarli associandoli ad una tra un certo numero di categorie.

* 1. **Obiettivi**

L’obiettivo principale di questa trattazione è quello di valutare il *tradeoff* tra perdita di accuratezza e memoria risparmiata in seguito ad una riduzione del numero di bit utilizzati per rappresentare i pesi dei neuroni della rete. A tal fine, sono state scelte un certo numero di configurazioni, variabili sulla base di dove l’approssimazione viene applicata e in che misura, e per ognuna di esse è stata effettuata la valutazione di cui sopra.

Per rendere più agevole il processo di training e test delle varie configurazioni è stata inoltre implementata una semplice applicazione desktop in C++, NNAXIM [[1]](#footnote-1), che consente di analizzare il comportamento di una specifica configurazione o quello di tutte le configurazioni. Per la realizzazione del programma è stata utilizzata la libreria *TinyDNN[[2]](#footnote-2)*, che consente di implementare (sempre in C++) delle reti neurali a livello software.

## **1.3 Flusso dell’analisi**

I capitoli successivi approfondiscono i vari passi del flusso di lavoro seguito per l’analisi, il quale si articola nei seguenti punti:

* *Capitolo 2*. Progettazione, allenamento e valutazione dell’accuratezza della rete originale.
* *Capitolo 3*. Applicazione dell’Approximate Computing sulla rete realizzata. Per valutare l’impatto dell’approssimazione sulla rete originale sono state scelte 9 configurazioni diverse. Ogni configurazione è identificata dai layer coinvolti nell’approssimazione e dalla misura con la quale questa è applicata ad ogni peso dei neuroni di ognuno di questi layer. Per ognuna di queste configurazioni è stata valutata l’accuratezza prima e dopo dell’esecuzione di un riallentamento della rete (che consente in generale di mitigare l’impatto sulle prestazioni dovuto all’approssimazione dei pesi).
* *Capitolo 4*. Confronto dei risultati ottenuti per le diverse configurazioni. Una volta ottenuti tutti i valori di accuratezza per le varie configurazioni, questi sono stati graficati e confrontati opportunamente.
* *Capitolo 5*. Conclusioni sul lavoro svolto.

# ***2. Rete originale***

Data l’applicazione, è stato necessario trovare una rete neurale capace di effettuare la classificazione richiesta a partire dalle immagini di input prodotte dalla fotocamera del sistema. Una volta trovata la rete è stato necessario allenarla per ottenere i valori dei pesi per la configurazione base da usare successivamente come punto di partenza per l’applicazione dell’approssimazione nelle varie configurazioni. È stata inoltre valutata l’accuratezza della rete originale in modo da poter effettuare il confronto con le configurazioni approssimate.

## **Progettazione della rete neurale**

Per la classificazione è stata scelta una *Convolutional Neural Network* utilizzata all’interno dell’esempio *cifar10* di TinyDNN. Questa rete è composta da una serie di 3 blocchi di layer (ognuno dei quali costituito da un layer convoluzionale, un pooling layer e un relu activation layer) seguiti da un fully-connected layer, un relu activation layer e infine un ulteriore fully-connected layer. Per quanto riguarda i dataset di training e di test sono stati usati quelli di CIFAR10[[3]](#footnote-3). Questi contengono delle immagini 32x32px raffiguranti degli oggetti riconducibili ad una di queste 10 categorie: aeroplani, automobili, uccelli, gatti, cervi, cani, rane, cavalli, navi e camion. La rete neurale grazie al training impara quindi classificare gli oggetti presenti nelle immagini di input alla rete in una delle suddette categorie.

Tramite l’interfaccia offerta dalla libreria TinyDNN è stato possibile implementare le operazioni di training, test, caricamento dei dataset e quant’altro.

## **2.2 Scelte operative**

La rete progettata è caratterizzata da un’elevata complessità. Tale complessità si riflette sui tempi di allenamento della stessa e non avrebbe quindi consentito di lavorare agevolmente su di essa durante le fasi dello sviluppo del tool di simulazione. Al fine di disaccoppiare le fasi di sviluppo del tool e di training della rete neurale non approssimata, si è scelto inizialmente di lavorare su una rete estremamente più semplice, il cui obiettivo è quello di predire l’output di una funzione sinusoidale[[4]](#footnote-4). In questo modo è stato possibile sviluppare e testare velocemente le procedure di training e test sulla rete e la definizione delle varie configurazioni da applicare per l’approssimazione. Dopo aver verificato il corretto funzionamento delle procedure sviluppate è stato effettuato un *porting* dell’applicazione sul modello di rete descritto nel paragrafo precedente. L’applicazione è stata poi sfruttata per eseguire una sola volta il training e il test delle reti approssimate (e non) su un calcolatore più performante, riducendo così le latenze di esecuzione.

## **2.3 Valutazione dell’accuratezza della rete**

# ***3. Approximate Computing sulla rete neurale realizzata***

## **3.1 Scelta delle configurazioni approssimate**

* 1. Iterazione orizzontale (ovvero 2/3 iterazione max in cui applichiamo l’approssimazione a insiemi di neuroni diversi – e.g. tutti i neuroni, solo i neuroni hidden solo alcuni dei neuroni hidden)
     1. Iterazione verticale (ovvero 2/3 iterazioni max in cui applichiamo in maniera diversa l’approssimazione ai neuroni dell’insieme dell’iterazione di livello superiore, ovvero con un numero di bit di rappresentazione dei pesi diverso)
        1. Modifica dei pesi riducendo i bit di rappresentazione
        2. Valutazione dell'accuratezza della rete (Amod)
        3. Riallenamento la rete in cui i pesi sono rappresentati con un numero ridotto di bit (2 epoche max)
        4. Valutare l'accuratezza della rete (Amod2)

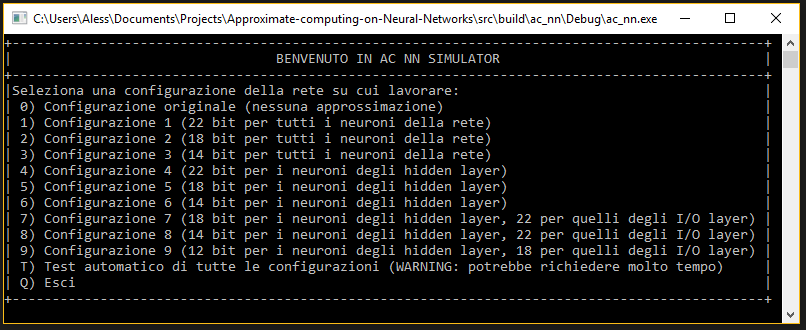
1. Ps. Fare riferimento ai paper quando scegli le configurazioni con solo gli hidden layer
2. Ps2. Fare notare le problematiche di progetto: a causa delle tempistiche del training abbiamo deciso di scegliere 3x3 iterazioni (orizzontali e verticali rispettivamente) e non un numero “quanto basta” come descritto nei paper.
   1. Es. In letteratura sono state descritte diverse soluzioni per applicare l’AC sulle NN. Noi abbiamo seguito un approccio ibrido cercando di approssimare per step successivi sia a livello orizzontale che a livello verticale. A causa delle limitate risorse hardware a disposizione non abbiamo però potuto eseguire un numero esaustivo di iterazioni, motivo per il quale abbiamo deciso di adottare un numero di iterazioni fisso nelle quali vengono utilizzate le configurazioni che sono state considerate più significative per la valutazione dei risultati finali.

## **3.2 AXIM Tool (AC simulator for NN)**

- Abbiamo reliazzato il programma per poter studiare la rete

- Descrizione directories del progetto (tipo dove trovi i file sorgenti, etc.)

- Come funziona il tool e cosa permette di fare



# ***4. Valutazione dei risultati ottenuti***

# ***5. Conclusioni***

1. NNAXIM (Neural Network Appromate Computing SIMulator) è disponibile su GitHub presso l’indirizzo <https://github.com/Taletex/NNAXIM>. Il nome del progetto è un tributo al simulatore NOXIM, disponibile sempre su GitHub all’indirizzo <https://github.com/davidepatti/noxim>. [↑](#footnote-ref-1)
2. TinyDNN è disponibile gratuitamente su GitHub presso l’indirizzo <https://github.com/tiny-dnn/tiny-dnn>. [↑](#footnote-ref-2)
3. <https://www.cs.toronto.edu/~kriz/cifar.html>. [↑](#footnote-ref-3)
4. Come base di partenza per l’implementazione di questa rete è stato utilizzato l’esempio *sinus\_fit* offerto da TinyDNN. [↑](#footnote-ref-4)