Sviluppo, analisi e implementazione di un algoritmo ottimizzato per le reti di convoluzione.

Amir Al Sadi **amir.alsadi@studio.unibo.it**

Nicolò Saccone **nicolo.saccone@studio.unibo.it**

Relazione del progetto di Sistemi Digitali M

Anno Accademico 2019/2020

SOMMARIO

INTRODUZIONE

Per convoluzione del processo di elaborazione delle immagini si intende l'applicazione all'immagine sorgente di una matrice quadrata denominata filtro di convoluzione. Questa tecnica ha largo impiego nelle reti neurali.

Con rete neurale si indica un modello computazionale largamente utilizzato nell'ambito del machine learning e deep learning. Una rete neurale è formata da migliaia di unità computazionali simili a dei "neuroni" artificiali che, se non correttamente parallelizzati, possono essere poco efficienti. Il largo utilizzo di queste reti nei più svariati ambiti le ha rese oggetto di numerosi studi da parte della comunità scientifica, soprattutto nell'ambito del training. Per training si intende un processo nel quale la rete viene sollecitata con un numero consistente di ingressi, i quali servono a calibrare correttamente i "neuroni" in modo da stabilizzare la rete e renderla in grado di interpretare correttamente i successivi ingressi. Un particolare tipo di reti neurali è identificato dalle CNN (reti di convoluzione neurali o Convolutional Neural Networks), utilizzate soprattutto in ambito di applicazioni di visione artificiale come ad esempio il riconoscimento facciale o l'individuazione di oggetti in un'immagine.

Solitamente anche le CNN più semplici presentano un numero di processi convoluzionali in cascata dell'ordine delle migliaia. Risulta dunque necessario ottimizzare queste operazioni, che sono la parte più dispendiosa della rete. Ogni livello di convoluzione ha in ingresso un'immagine e un filtro (o Kernel) di dimensione ridotta rispetto al frame dell'immagine. L'operazione di convoluzione concentra il maggior costo computazionale nei prodotti. Per questo motivo, è di attuale interesse individuare dei metodi per ridurre la memoria e di conseguenza il numero di calcoli svolti. Un primo stratagemma utilizzato è quello di separare i kernel dei filtri di convoluzione, riducendo il numero di prodotti per pixel convoluto da NxN (dove N è la dimensione riga/colonna del kernel) a 2N senza perdita di informazione (per maggiori informazioni sugli algoritmi di convoluzione riferirsi a [1]). Oltre a questo, esistono dei metodi che sfruttano l'allocazione dei dati in variabili di dimensione inferiore rispetto a quella originale. La più comune consiste nel trasformare dati espressi in formato decimale in interi, sfruttando la notazione a virgola fissa.

Il principio di realizzazione del progetto proposto in questo documento utilizza un metodo di questa tipologia.

DESCRIZIONE DELL'ALGORITMO E SCELTE PROGETTUALI

L'algoritmo considerato lungo lo svolgimento di questa sperimentazione è descritto nell'articolo di IBM "*Deep learning with Limited Numerical Precision"* [2]. La scelta del caso di studio ha determinato alcune modifiche all'algoritmo originale.

Descrizione dell'algoritmo

Il problema più comune nei processi di convoluzione è la rappresentazione dei pesi del kernel, alcune volte troppo onerosi se rappresentati a piena precisione. Risulta dunque spesso dispendioso il processo di convoluzione standard delle reti, soprattutto se si considerano casi reali formati da migliaia di livelli di convoluzione in cascata.

Nel prodotto di convoluzione si hanno due fattori: il dato rappresentante il pixel e i pesi del kernel. Nella situazione trattata nell'articolo, entrambi i dati sono rappresentati con in formato float. Si è deciso di trattare pixel in ingresso con formato a 8 bit, invece che a 32.

Il processo di approssimazione è strutturato in questo modo: data la dimensione del dato (float a 32 bit) viene fissata una dimensione desiderata per la rappresentazione del dato, pari a WL. A questo punto si trasforma il dato iniziale in un decimale a virgola fissa, ossia un dato avente la parte intera pari a IL bit e la parte frazionaria pari a FL, in modo tale da avere WL=IL+FL.

Una volta fissati WL, IL e FL, si applica una **elaborazione** del dato. Prima di procedere alla conversione, bisogna specificare il significato dei dati utilizzati:

- X, ossia il dato in ingresso.
- $\varepsilon = 2^{-FL}$, ossia l'errore della misurazione.
- R = $<-2^{IL-1}, 2^{IL-1} - \varepsilon>$, ossia il range di valori acquisibili dal dato rappresentato.

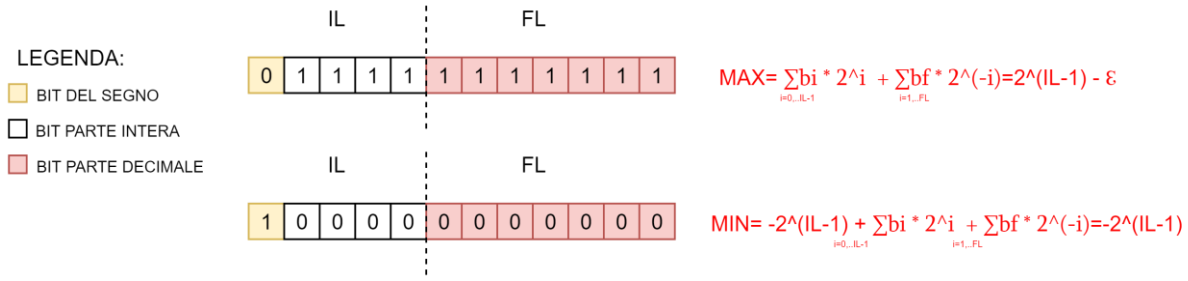

Figura 1: Notazione a virgola fissa

  La grandezza del range è diretta conseguenza della rappresentazione a virgola fissa: per una maggiore chiarezza la Figura 1 spiega nel pratico come calcolare questi estremi data una notazione a virgola fissa.
- [X] = N * ε, dove N è il più grande intero tale che [X] sia al più uguale a X.
- Z, il risultato successivo alla fase di elaborazione.

A questo punto, vengono proposti due metodi di approssimazione: quello *round-to-nearest* e quello *stocastico*.

Il metodo *round-to-nearest* approssima il valore [X] in base all'algoritmo:

$$Z = \begin{cases} [X] & \text{se } [X] \leq X < [X] + \varepsilon/2 \\ [X] + \varepsilon & \text{se } [X] + \varepsilon/2 \leq X \leq [X] + \varepsilon \end{cases}$$

Dunque, Z è il risultato dell'arrotondamento di [X] al multiplo di ε più vicino. Questo algoritmo, come trattato nelle conclusioni dell'articolo, risulta meno preciso di quello *stocastico*.

Per algoritmo stocastico si intende il secondo metodo proposto nell'articolo, che utilizza *l'arrotondamento stocastico*. Fissando $p = \frac{X-[X]}{\varepsilon}$:

$$Z = \begin{cases} [X] & \text{con probabilità p} \\ [X] + \varepsilon & \text{con probabilità } 1 - p \end{cases}$$

Questo algoritmo ha inoltre l'importante proprietà di avere errore quasi nullo. Dunque, all'aumentare della distanza di [X] da X aumenta la probabilità che il dato elaborato assuma valore X. Questo è coerente con l'idea del *round-to-nearest*, ma aggiungendo la probabilità dell'arrotondamento, si

approssima più efficacemente. Per questo motivo, in coerenza con i risultati ottenuti da IBM si è deciso di adottare questo secondo metodo lungo tutto il progetto.

La fase successiva alla **elaborazione** è quella di **conversione** del dato. Questa fase valuta se il dato elaborato richieda una saturazione o meno. Detto C il risultato della conversione:

- Se Z > 2^{IL-1}- Ɛ, C = $2^{IL-1} - Ɛ$.
- Se Z < -2^{IL-1}, C = -2^{IL-1}.
- Se Z ∈ R(range), C = Z.

I dati convertiti si collocano all'interno di un dato di dimensione IL+FL bits. La notazione fissa vera e propria si ottiene in questa fase, in quanto si compie uno shift di FL bits a sinistra di C (corrispondente alla moltiplicazione dell'intero per un fattore 2^{FL}). Il dato a questo punto non è più di tipo float ma di tipo intero con segno a WL (IL+FL) bits.

Si procede dunque al **calcolo della convoluzione dei dati**, tramite l'algoritmo classico e separabile. I risultati di questi prodotti vengono collocati in una variabile di dimensione pari a **IL + FL + (dimensione del pixel)** bits. A questo punto il dato viene shiftato a destra di FL bits (corrispondente alla moltiplicazione dell'intero per un fattore 2^{-FL}), e passa dal formato intero al formato float. Il dato così rappresentato viene nuovamente **elaborato** e **convertito** secondo i passi elencati in precedenza.

A questo punto, il pixel convoluto è rappresentato dalla parte intera del dato in uscita dalla catena.

Per ottenere un corretto passaggio dalla descrizione alla implementazione dell'algoritmo, sono state prese delle decisioni operative che hanno influito sull'analisi dei risultati ottenuti.

Caso di studio

Si è deciso di trattare in ingresso alla rete solo immagini a scale di grigio (grayscale) a 8 bit, in contrasto con i 32 bit ipotizzati nel paper. Infatti, si è constatato che la scelta di ridurre i bit per la rappresentazione del dato avesse diversi vantaggi:

- Rimanere coerenti con il corso di Sistemi Digitali M, che ha sempre trattato rappresentazioni dei dati a 8 bit (RGB o grayscale).
- Ridurre in partenza l'impiego di memoria per la rappresentazione dei dati, ottenendo già in partenza performance ottimizzate.
- Generalizzare a un caso più comune consentendo maggiore estendibilità e maggiore facilità nell'interpretazione dei risultati ottenuti. Infatti, con la scelta di adottare gli 8 bit di rappresentazione del dato, è venuta meno la necessità di approssimare anche i pixel dell'immagine, limitandosi unicamente ai pesi (rappresentati con float come specificato nel paper).

In queste condizioni è dunque relativamente semplice estendere la trattazione a sistemi aventi ingressi rappresentati con dei formati diversi da quello in questione.

Coerentemente con il corso di studi, si è deciso di adoperare la suite Vivado per riuscire a mappare efficacemente l'algoritmo sulla scheda Zedboard e soprattutto per poter quantificare le risorse utilizzate, in modo da confrontare l'efficienza dell'algoritmo nelle sue due versioni (tradizionale e con kernel separabile). Per ottimizzare i risultati della sintesi di Vivado HLS si è deciso di compiere esternamente i passi di elaborazione e conversione dei pesi della matrice. Questo ha prodotto il vantaggio desiderato di un risparmio immediato di clock e memoria allocata. Un'altra scelta di

ottimizzazione riguarda la generazione di numeri casuali utilizzati per determinare la probabilità dell'intervallo di arrotondamento. Le principali problematiche legate alla generazione dei numeri casuali riscontrati sono:

1. Vivado non contempla le librerie <time.h> e <rand.h>, dunque è stato necessario scrivere un modulo che generi in modo pseudo-casuale i numeri partendo da un seme iniziale.
2. Il modulo in questione genera numeri pseudo-casuali e quindi prevedibili prima dell'esecuzione. Questo svantaggio, però, non influisce in maniera sostanziale sui risultati ottenuti.
3. Il modulo in questione è molto dispendioso.

Per questo motivo, si è deciso di pre-generare i numeri casuali, riducendo notevolmente la memoria allocata (circa 8000 FF in meno).

Per stressare quanto più possibile la flessibilità dell'algoritmo, si è deciso di utilizzare due tipi distinti di kernel: uno 3x3 e uno 7x7. La scelta di utilizzare due matrici così lontane dimensionalmente sottolinea le eventuali perdite di informazioni date dalle approssimazioni. Questo può essere utile nell'ottica del confronto del numero di prodotti effettuati nella convoluzione, in quanto a fronte di 9 prodotti (nella convoluzione tradizionale) e 6 prodotti (nella convoluzione con kernel separabili) per un kernel 3x3, si hanno rispettivamente 49 prodotti (nella convoluzione tradizionale) e 14 prodotti (nella convoluzione con kernel separabili) per uno 7x7. Possiamo dunque aspettarci che all'aumentare del numero di prodotti, peggiorerà la qualità dell'immagine in output, a causa del maggior numero di approssimazioni.

Si è scelto in un secondo momento di adattare il codice prodotto su un progetto preesistente [3] in modo da verificare su Zedboard il lavoro svolto.

ANALISI DEGLI UTILIZZI DELLA MEMORIA

In prima istanza si è deciso di analizzare l'impiego di memoria dell'algoritmo proposto, confrontato con quello di un algoritmo di convoluzione tradizionale, utilizzando il tool Vivado HLS per calcolare l'impiego di memoria su Zedboard Zynq-7000 Development Board.

I test sono stati strutturati come segue: variando la dimensione del dato approssimato, sono stati registrati i valori di memoria allocata. Sono stati scelti come parametri significativi: latenza del clock, clock stimato, BRAM, flip flops e look up tables. Nel sorgente Vivado (disponibile in https://github.com/alsadiamir/SistemiDigitaliProgetto/) è stata utilizzata una matrice di appoggio per i calcoli, popolata con i valori già convoluti in modo da evitare la conversione della matrice all'interno della funzione di convoluzione. Questa scelta ha permesso di valutare l'algoritmo di convoluzione senza costi aggiuntivi legati alla conversione dei dati.

Test con Kernel 3x3

Lungo lo svolgimento di questo test si è utilizzato un kernel gaussiano 3x3, che ha l'effetto di sfocare l'immagine.

LEGENDA

CONVOLUZIONE TRADIZIONALE

CONVOLUZIONE CON KERNEL SEPARABILE

RIGA SENZA OTTIMIZZAZIONE (CONV. con FLOAT)

CASO KERNEL 3x3

RISORSE UTILIZZATE / N° BIT	LATENCY (Clock cycles)	LATENCY (Clock cycles)	CLOCK ESTIMATED (ns)	CLOCK ESTIMATED (ns)	BRAM (18k)	BRAM (18k)	FF(106400 DISPONIBILI)	FF(106400 DISPONIBILI)	LUT(53200 DISPONIBILI)	LUT(53200 DISPONIBILI)
10	308371	308372	8.585	8.585	2	2	3209	3255	6560	6446
12	308371	308372	8.609	8.585	2	2	3209	3262	6562	6458
14	308371	308372	8.638	8.585	2	2	3109	3280	6564	6494
16	308371	308372	8.638	8.585	2	2	3198	3298	6564	6511
24	308372	308372	8.585	8.585	2	2	3225	3507	6568	6559
32	308371	308371	8.585	8.585	2	2	3208	3059	6677	6407
32	308435	308412	8.666	8.666	2	2	11167	7564	18756	13453

Tabella 1: Risultati con Kernel 3x3

In Tabella 1 sono rappresentati i risultati della sintesi di Vivado HLS per la scheda Zedboard Zynq-7000 Development Board. Dal grafico si evince come i consumi di memoria di una convoluzione senza l'algoritmo proposto (riga rossa nella tabella) non reggano il confronto con quelli dell'algoritmo stocastico. Per quanto riguarda la convoluzione tradizionale, il numero di FF e LUT è circa tre volte superiore a quello del corrispondente algoritmo stocastico. Utilizzando l'algoritmo con kernel separabili invece, si nota che la simulazione con convoluzione senza approssimazione usa circa il doppio di FF e LUT rispetto all'equivalente stocastico.

Al diminuire del numero di bit allocati si può notare un costante trend lievemente in calo della memoria allocata, ad eccezione del caso a 32 bit. Calando da 32 a 10 bit non si nota una apprezzabile differenza di memoria allocata. Il risultato non rispecchia dunque completamente le aspettative. Questo dubbio è confermato dai risultati sugli indici di qualità delle immagini convolute, trattati successivamente.

I probabili motivi che giustificano i risultati possono essere:

- Ottimizzazioni interne a Vivado HLS.
- Cast interni al codice che potrebbero falsare la reale grandezza dei dati allocati.
- L'utilizzo di una matrice di appoggio inizializzata con tutti i pesi di valore massimo, in modo da simulare il caso peggiore, pari alla parte intera di $2^{IL-1} - \mathcal{E}$, è sicuramente una soluzione meno stabile rispetto all'utilizzo di una matrice con valori costanti. Infatti, considerando una matrice inizializzata a valori nulli (caso migliore in quanto si evitano i prodotti) si ottengono valori bassissimi (1/3 di FF e LUT). Il range di variabilità aggiunge dunque un parametro di incertezza nella sintesi.
- Il numero di bit assegnati alla parte intera (IL) e alla parte decimale (FL), come immediata conseguenza del punto precedente.

Test con Kernel 7x7

Nel caso del kernel 7x7 si è utilizzato un filtro di sfocatura gaussiano, coerentemente al caso 3x3.

LEGENDA

CONVOLUZIONE TRADIZIONALE

CONVOLUZIONE CON KERNEL SEPARABILE

RIGA SENZA OTTIMIZZAZIONE (CONV. con FLOAT)

CASO KERNEL 7x7

RISORSE UTILIZZATE / N° BIT	LATENCY (Clock cycles)	LATENCY (Clock cycles)	CLOCK ESTIMATED (ns)	CLOCK ESTIMATED (ns)	BRAM (18k)	BRAM (18k)	FF(106400 DISPONIBILI)	FF(106400 DISPONIBILI)	LUT(53200 DISPONIBILI)	LUT(53200 DISPONIBILI)
10	310620	310621	8.655	8.585	6	6	3688	3255	7254	6560
12	310620	310620	8.655	8.609	6	6	3690	3262	7256	6562
14	310620	310620	8.655	8.638	6	6	3692	3280	7258	6564
16	310620	310620	8.698	8.638	6	6	3684	3298	7243	6564
24	310620	310621	8.698	8.585	6	6	3684	3507	7243	6568
32	308371	310619	8.585	8.742	6	6	3746	3203	8073	6695
32	311252	310768	8.666	8.666	6	6	68349	19088	119461	33202

Tabella 2: Risultati con Kernel 7x7

In Tabella 2 sono riassunti i risultati delle convoluzioni. Il confronto fra l'algoritmo stocastico e quello tradizionale rimane coerente con il caso 3x3, registrando dalle otto alle dieci volte in meno le risorse allocate in favore dell'algoritmo stocastico.

Nota positiva di questa simulazione è una tendenza più marcatamente in calo della memoria rispetto al caso con kernel 3x3. Possiamo dunque concludere che l'aumento in numero di moltiplicazioni stabilisca in maniera più netta il guadagno di memoria.

ANALISI DELLA QUALITA' DELLE IMMAGINI

Ai test precedenti sono stati affiancati i risultati relativi alla qualità delle immagini convolute. Tramite l'ausilio di alcune API OpenCV, si è simulata una convoluzione con massima precisione e la si è confrontata con quelle elaborate dall'algoritmo stocastico.

Sono stati scelti tre indici che valutano la qualità delle immagini in uscita dalla rete:

- MSE: Mean Squared Error [4], ossia la somma delle distanze quadrate fra i valori dei pixel divisa per il numero di pixel.

$$MSE = \frac{\sum_{i=1}^{n}(xi - xci)^2}{n}$$

- RMSE: Root Mean Squared Error [4], ossia la radice di MSE.

$$RMSE = \sqrt{MSE}$$

- SSIM: Structural Similarity [5], che da una maggiore enfasi sulla somiglianza visiva delle immagini tramite un insieme complesso di fattori, a differenza di MSE e RMSE che si

concentrano solo sulle distanze assolute fra i pixel. Il valore varia da 0 a 1, con 1 come massima precisione.

Un ulteriore parametro di variabilità di questa rete sono i valori di IL e FL. Infatti, ci si aspetta che all'aumentare di FL la precisione del calcolo aumenti. Si è deciso, in questo senso, di utilizzare IL e FL diversi per l'arrotondamento stocastico classico e quello con kernel separabili.

I bordi delle immagini non sono stati trattati: questo potrebbe aver influito leggermente sui risultati di MSE e RMSE.

Test con Kernel 3x3

Nella Tabella 3 sono riportati i valori degli indici MSE, RMSE e SSIM relativi al caso con kernel 3x3.

LEGENDA

CONVOLUZIONE TRADIZIONALE

CONVOLUZIONE CON KERNEL SEPARABILE

CASO KERNEL 3x3

QUALITÀ IMMAGINE / N° BIT IL	FL	MSE	RMSE	SSIM	QUALITÀ IMMAGINE / N° BIT IL	FL	MSE	RMSE	SSIM
16	16	2.79	1.67	1.00	28	4	10.11	3.18	1.00
14	10	2.79	1.67	1.00	20	4	10.11	3.18	1.00
10	6	2.79	1.67	1.00	12	4	10.11	3.18	1.00
10	4	2.79	1.67	1.00	10	4	10.11	3.18	1.00
10	2	958.99	30.97	0.92	8	4	1356.57	36.83	0.89
8	2	1714.08	41.40	0.86	6	4	12793.68	113.11	0.27

Tabella 3: Indici di qualità nel caso di Kernel 3x3

Dai valori riscontrati, si nota come nel caso di un kernel 3x3 si registri una ottima similarità nell'immagine (SSIM = 1) utilizzando dati fino a 14 bit di ampiezza, con indici MSE e RMSE leggermente più alti nel caso separabile. Dai 12 ai 10 bit si nota una sostenuta differenza fra il caso tradizionale e separabile.

La perdita di qualità nel caso separabile è dovuta dal fatto che per ottenere il pixel convoluto vengono utilizzati i prodotti intermedi della convoluzione, già approssimati a loro volta. Per ottenere i risultati migliori dai due algoritmi, si è pensato dunque di utilizzare valori di IL e FL diversi.

Test con Kernel 7x7

I test con il kernel di ampiezza 7x7 confermano le tesi ipotizzate nel caso a kernel 3x3.

LEGENDA

CONVOLUZIONE TRADIZIONALE

CONVOLUZIONE CON KERNEL SEPARABILE

CASO KERNEL 7x7

QUALITÀ IMMAGINE – N° BIT IL	QUALITÀ IMMAGINE – N° BIT FL	MSE	RMSE	SSIM	QUALITÀ IMMAGINE – N° BIT IL	QUALITÀ IMMAGINE – N° BIT FL	MSE	RMSE	SSIM
16	16	5.29	2.3	1.00	22	10	5.11	2.26	1.00
10	14	5.29	2.3	1.00	14	10	5.11	2.26	1.00
10	6	2114.68	45.99	0.94	8	8	1328.66	36.45	0.90
10	4	3043.70	55.17	0.89	8	6	1343.64	36.66	0.88
10	2	3731.88	61.09	0.72	8	4	1418.83	37.67	0.88
8	2	1418.83	37.67	0.85	6	4	3227.36	56.81	0.41

Tabella 4: Indici di qualità nel caso di Kernel 7x7

Nella Tabella 4 sono riportati gli indici di qualità. A riconferma del trend del caso 3x3, aumentando il numero di prodotti diminuisce la qualità dell'immagine allocando meno bit per la rappresentazione del dato. Abbiamo dunque un effetto ancora più delineato nella convoluzione con algoritmo separabile.

IMPLEMENTAZIONE

Per ultimo si è deciso di implementare l'algoritmo su un progetto preesistente [3] per ottenerne un riscontro in un caso reale.

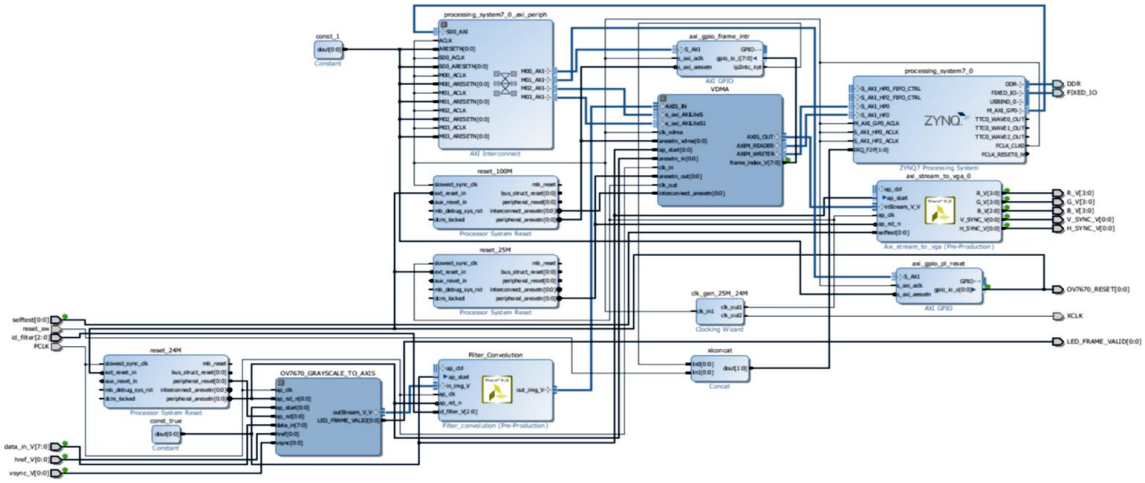

Figura 2: Design di SmartCamera

In Figura 2 è rappresentato il design hardware del progetto (denominato SmartCamera), che fornisce un numero di funzionalità particolarmente ampio, come ad esempio lo stream via UDP delle immagini in diversi formati o la configurazione via TCP dei filtri di convoluzione, oltre che una gestione embedded di lettura e scrittura in DDR e SD card.

Il modulo riadattato per l'esigenza è quello denominato "Filter Convolution", che aggrega in sé a sua volta due moduli che compiono convoluzione tradizionale e separabile in maniera mutuamente esclusiva. I filtri vengono applicati allo stream della telecamera OV7670, il cui protocollo di acquisizione di immagini è già gestito nel design.

I risultati riconfermano una grossa differenza di precisione anche utilizzando un filtro di tipo "motion blur" a favore dell'algoritmo stocastico senza separabilità. Dobbiamo precisare che il riadattamento del progetto iniziale su uno preesistente potrebbe non rispecchiare pienamente le performance temporali e di memoria, a causa dell'overhead introdotto dai componenti non necessari.

CONCLUSIONI

L'implementazione dell'algoritmo denota un grosso risparmio di risorse rispetto a una convoluzione a piena precisione. Inoltre, si può concludere la validità dell'algoritmo in ottica di training, in quanto a un grosso risparmio si associa una ottima qualità nelle immagini convolute, anche assegnando ai dati una dimensione relativamente bassa di bit. Va sottolineato che i tanti parametri di variabilità lasciano aperte diverse possibilità di miglioramento dell'algoritmo.

I possibili sviluppi futuri per questo tipo di progetto sono:

- Un ampliamento dei test e relativi aggiustamenti del codice HLS.
- Utilizzo di diversi filtri e confronti incrociati per determinare in che modo il tipo di filtro influisca sul rendimento della rete.
- Realizzazione di un design implementativo Vivado progettato sulle specifiche indicate nel progetto iniziale.

RIFERIMENTI

Fonti

[1] *http://vision.deis.unibo.it/~smatt/DIDATTICA/Sistemi_Digitali_M/PDF/08_Convolution_filters_HLS.pdf*

[2] *https://arxiv.org/abs/1502.02551*

[3] *https://github.com/smatt-github/SmartCamera*

[4] *https://en.wikipedia.org/wiki/Mean_squared_error*

[5] *https://en.wikipedia.org/wiki/Structural_similarity*

Indice delle figure

Indice delle tabelle