Arquitectura de Hardware para Convolución Bidimensional con Memoria Limitada Aplicada al Procesamiento de Imágenes

Martin Casabella, Sergio Sulca, Ivan Vignolles
Escuela de Ingeniería en Computación
Facultad de Ciencias Exactas Físicas y Naturales
Universidad Nacional de Córdoba
Email: martin.casabella@gmail.com, ser.0090@gmail.com, ivignolles@alumnos.unc.edu.ar

***Abstract*—En este artículo se presenta una arquitectura de hardware para realizar una convolución 2D en una FPGA cuando no se puede instanciar suficiente memoria RAM para poder alojar la imagen completa. Se priorizó la velocidad de procesamiento, el uso eficiente de los recursos y un diseño escalable donde se pudieran agregar tantas operaciones de convolución en paralelo como se desee sin deber hacer grandes modificaciones en el diseño.**

1. Introducción

La convolución bidimensional discreta es ampliamente usada en mltiples campos de la ingenieria, siendo uno de ellos la visión artificial que ha tomado fuerza en la ultima década con los resultados obtenidos con las redes neuronales convolucionales. El creciente tamaño en los modelos, como tambien en la informacion disponible para el entrenamiento hacen de la velocidad de procesamiento un factor de gran importancia.

Por la naturaleza de los datos y de la operación de la convolución, un enfoque paralelo resulta mucho más eficiente que uno secuencial en lo que a velocidad de procesamiento respecta. Se utilizaron FPGAs para la implementacion, pues resuelven la necesidad de paralelismo como también la de poder probar distintos prototipos de arquitecturas de hardware.

La convolución discreta en 2D esta definida por la siguiente ecuación:

$$S(x,y) = \sum_i \sum_j I(x-i, y-j)K(i,j) \tag{1}$$

Al no ser necesaria la propiedad de conmutatividad en este caso, es posible eliminar el proceso de espejar una de las matrices [1] , obteniendo la función de correlacion cruzada

$$S(x,y) = \sum_i \sum_j I(x+i, y+j)K(i,j) \tag{2}$$

la cual será en realidad implementada, por lo que en lo que resta del artículo, cada vez que se nombre a la operación de convolución, se estará haciendo referencia a la ecuación 2.

Uno de los mayores desafíos fue trabajar con memoria RAM lo suficientemente escaza que no fuera capaz de alojar todos los pixels de la imagen que se desea procesar. Por lo que la imágen debe ser fraccionada y procesada en lotes, con todo el procesamiento adicional que esto implica.

Por una cuestión de simplicidad, todo el trabajo se desarrolló utilizando imagenes en escala de grises, al tener estas un único canal.

2. Análisis en punto flotante y punto fijo

No se arranca con un simulador del sistema completo desarrolado en python ya que las cuestiones del procesamiento de la imagen están más enfocadas a la etapa de implementación, sin embargo en la etapa de verificación del sistema hubo que hacer un análisis de como influyen los distintos filtros (kernels).

Como primer paso se tiene una imagen destinada a procesamiento, para el cual se debe tener en consideracin el uso de una resolucin finita, por ende se necesita efectuar un estudio riguroso acerca de cuantos bits de resolución mínimos admite el sistema con una pérdida de información mínima.

2.1. Análisis de comportamiento

Trabajado con un análisis a nivel del comportamiento del kernel en python, se observa que el rango de la imagen deben ser modificadas, para trabajar en un rango de 8 bits, entonces surge.

2.1.1. Maximum norm. Toma como norma el mayor valor absoluto de una tupla con n elementos

$$\|\mathbf{x}\|_\infty := \max(|x_1|, ..., |x_n|)$$

2.1.2. División con respecto a la norma. Tomando el valor máximo de la tupla en valor absoluta y dividiendo cada uno de los elementos del kernel

$$\hat{k}_{ij} = \frac{k_{ij}}{\|\mathbf{k}\|}$$

la razón de todo esto es llevar los valores iniciales del kernel al rango [-1 ; 1). Para poder tener una representación de 8 bits, para los mismos.

2.1.3. Expansión lineal dinámica de rango. Utilizados para el procesamiento de imágenes donde usualmente se traslada la imagen a un rango conveniente de trabajo.

$$I_N = (I - Min)\frac{newMax - newMin}{Max - Min} + newMin$$

Así como en el kernel, el objetivo aquí es llevar los valores de cada píxel de la imagen a [0 ; 1). Con el fin de tener una representación en 8 bits.

Para reducir el número bits a manejar en la implementación en hardware , ya sea en la etapa de cálculo de convolución como en la de transferencia de los datos por UART se escoge como ancho de palabra 8 bits (para la etapa de envío de datos).

Al obtener un cambio en el rango de la imagen que nos permite decir, que el producto de normalizar la señal, en principio, el rango de los datos cabe en 8 bits y 7 bits, al trabajar en python y al tener en cuenta que el kernel tiene componentes negativos se tiene.

- Representación utilizada en Imagen S(8,7)
- Representación utilizada en Kernel S(8,7)

En la operación inicial de la convolución se obtiene como resultado en S(16,14) por cada producto , y al tener un kernel de 3x3 se genera una suma de 9 elementos. Esto resulta en una representación S(20,14). Es decir, la salida del convolucionador consta de 20 bits.

Por las restricciones impuestas por los módulos UART, la máxima unidad de información son 8 bits, lo que implica que para enviar un solo dato , se requieren tres envíos.

Esto nos lleva a realizar una etapa de post procesamiento. Esta consistente en llevar el resultado de la convolución a rango positivo y mapear los bits menos significativos a uno más significativo, realizando un truncado de los menos significativos.

2.1.4. Métrica. Para poder establecer una Métrica y realizar una comparación , a fines de decidir la cantidad de bits de salida del procesamiento, con el fin de reducir ya sea la parte fraccionaria, o la entera, se forma una relación entre la operación a máxima resolución y otra en la cual se le reducen los bits.

Se genera una relación señal ruido de la estimación a 20 bits y el error pruducido al reducir la cantidad de bits. Esto para determinar la cantidad de bits mínima de trabajo.

- Error:

$$e_r = f(x)_{20b} - f(x)_{pos}$$

- Energia:

$$E = \frac{1}{n}\sum_{i=0}^{n-1} x_i^2$$

- SNR:

$$SNR = \frac{E_{20b}}{E_{error}}$$

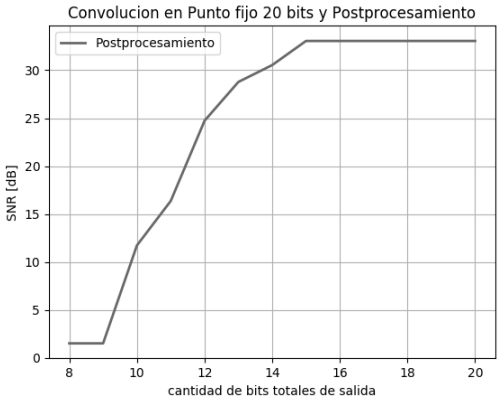

Fig. 1. Relación señal ruido de la estimación a 20 bits y el error pruducido al reducir la cantidad de bits.

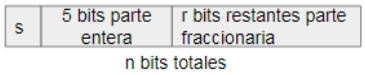

Fig. 2. Estructura de los bits resultante.

2.1.5. Representación en punto fijo. Al efectuar la convolución con un filtro unitario.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Al realizar el análisis luego de reducir los bits de la parte fraccionaria. Se parte de 8 bits en totales, en dónde se observa que la imágen mejora, y que a partir de 13 bits totales, 1 bit de signo, 5 bits parte entera y 7 en parte fraccionaria(figura 2) se tiene una SNR aproximada a 30 dB, la cúal se considera suficiente como se observa en la figura 1.

Se considera suficiente debido a observación de las imágenes, en dónde se aprecia que si nos quedamos con 8 bits de salida (lo que intuye al principio por las resoluciones el filtro y de la imágen) se tiene una calidad muy baja. pero a medida que se aumenta la cantidad de bits, la imágen mejora.

Al nivel de 13 bits se tiene una mejor apreciación de la imágen. Aunque si se realiza un análisis más fino se nota una degradación respecto de la original. Pero como el objetivo no es no es representar una imagen en su totalidad sino, hacer deteccin de borde o detectar elementos, etc. para la cual la precisin de la imágen queda en un segundo plano, esto se ve en la figura 3.

Para avanzar un paso más en la reducción de bits se realiza el mismo análisis al eliminar el bit más significativo de la parte entera. lo cual permite quedarse con una resolución de 12 bits como se oberva en la figura 4 y su estructura en la figura 5.

Se puede llevar el rango final a 8 bits, si se hace un cambio de rango igual al que se realizó para la imágen antes

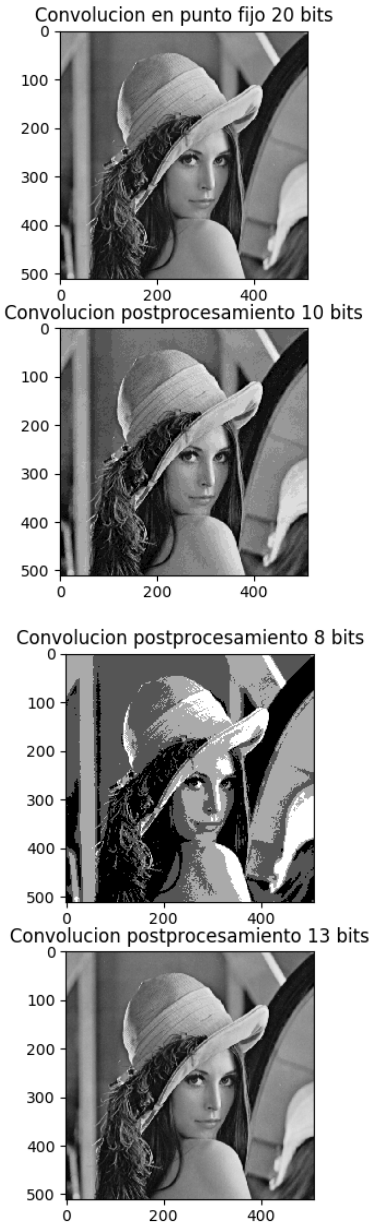

Fig. 3. Resultados de tomar distintos bits totales.

de procesarla. Para ello se requiere conocer el máximo y el mínimo valor de píxel de la imágen luego de filtrar, esto requiere que toda la imágen se encuentre en la memoria. Debido a las limitación esto no se podía llevar a cabo.

3. Arquitectura del sistema

3.1. Flujo de trabajo

El flujo de trabajo consta de tres instancias como se observa en la figura 6. En la primer instancia, se desarrolló un cliente en python donde se ingresa la imagen a procesar

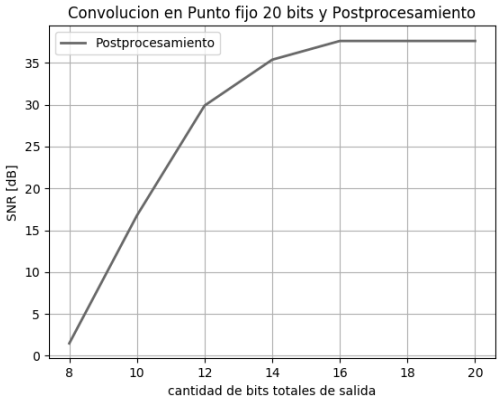

Fig. 4. Nueva SNR al reducir un 1 bit de la parte entera usando el mismo filtro.

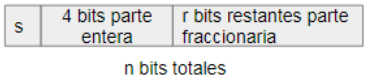

Fig. 5. Nueva estructura de los bits resultante.

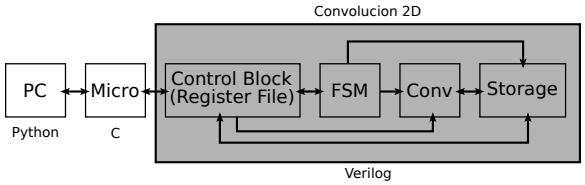

Fig. 6. Flujo de trabajo del sistema y lenguajes en los que se trabajó en cada instancia.

junto con los coeficientes del filtro que se desea aplicar, el filtro o kernel es una matriz que pertence al conjunto $\mathbb{R}^{3\times3}$. Toda la arquitectura del sistema ha sido diseñada para trabajar con filtros de esa dimensión, aunque no debería ser dificil generalizarla a filtros de otras dimensiones. La información es entonces separada en lotes, el cliente envía un lote, espera a recibir el mismo procesado y luego envía el lote siguiente.

Un microprocesador instanciado en la FPGA es el encargado de recibir los lotes, hace un desempaquetado de los datos y además le añade a los mismos una cabecera necesaria para la comunicación con el módulo de convolución. Asimismo una vez que el módulo finalizó el procesamiento, el microprocesador toma esa información procesada, la empaqueta y envía nuevamente a la computadora.

El módulo de convolución procesa el lote, internamente es el Bloque de Control (Control Block) quien hace de interfaz entre el resto del módulo y el exterior. La porción de la imagen es almacenada, los bloques de convolución (Conv) realizan la suma de los productos de los coeficientes del filtro con una sección de la imagen igual al tamaño del filtro, una vez que toda la porción de la imagen fue procesada, se envía el resultado y se espera a recibir una nueva porción. Una maquina de estados finitos (FSM) lleva cuenta del

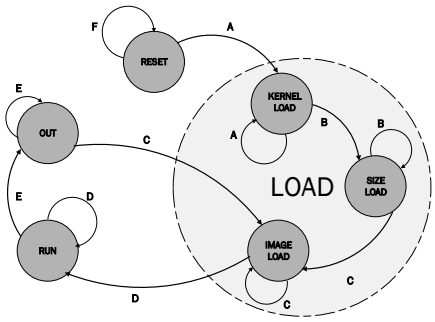

Fig. 7. Diagrama de estados del módulo de convolución. A - Carga de kernel, B - Carga de tamaño, C - Carga de imagen, D - Procesamiento, E - Out, F - Reset

ciclo de entrada→procesamiento→salida que ocurre en el módulo.

3.2. Módulo de convolución

En esta sección se describe en detalle la arquitectura del módulo de convolución.

3.2.1. Estados. Durante todo el ciclo de trabajo, el módulo atraviesa distintos estados como se aprecia en la figura 7. En primera instancia, está en un estado inicial de **reset** esperando a ser configurado. Permanecerá en ese estado hasta recibir la intruccion para pasar al estado de **kernel load** (carga de kernel) donde se carga al módulo el filtro que se desea aplicar. Una vez que se termina de cargar los coeficientes del filtro, se espera la instrucción para pasar a **size load** (carga de tamaño), en este estado se debe informar al módulo la altura de la imagen. Habiendo ocurrido la transición al estado siguiente, no se retornará a los estados previos hasta finalizar el procesamiento completo de la imagen. Al salir del estado de **size load** se pasa al de **image load** donde se carga una sección de la imagen. Pasados estos tres estados se finaliza la etapa de carga. En la instacia siguiente el módulo se encuentra en el estado de **run**. En este estado se realiza el procesamiento del lote de forma ininterrumpida, es decir, toda instrucción será ignorada hasta finalizar el filtrado donde una señal de aviso es emitida. Se espera entonces la instrucción para pasar al estado **out**. Con el lote ya procesado y la intrucción recibida se da inicio al envío de los datos filtrados. La sucesión de los ultimos tres estados mensionados **image load** → **run** → **out** se repetirá por cada nuevo lote hasta obtener la imagen procesada en su totalidad.

3.2.2. Almacenamiento de la información. El kernel o filtro que se quiere aplicar a la imagen se configura durante la etapa de carga y es almacenado en registros de los bloques de convolución. Con respecto a la imagen, se optó por utilizar columnas de memoria, donde cada columna almacena una columna de pixels. Queda entonces como una limitación la altura máxima de la imagen dada por el número máximo de pixels que es posible almacenar en una columna de memoria.

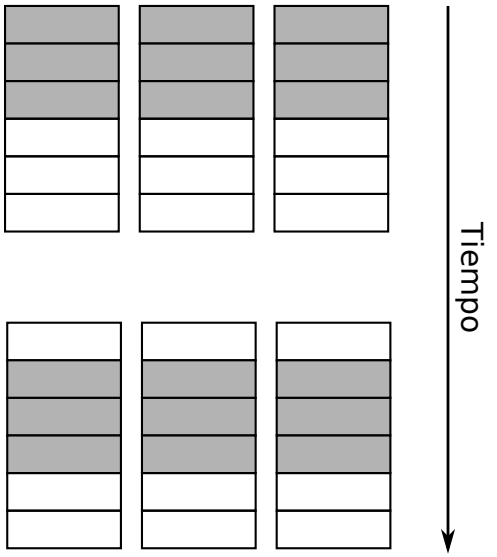

Fig. 8. En gris se observan los datos que han sido cargados a los registros del bloque de convolución, en el ciclo siguiente es posible ver como los valores de cada columna que ingresaron primero han sido decartados, y nuevos valores han sido cargados, produciendo un efecto de desplazamiento hacia abajo.

Cada bloque de convolución toma los valores de tres (o k, para un filtro que pertence a $\mathbb{R}^{k\times k}$) columnas adyacentes. Comienza cargando los tres primeros valores de las tres columnas en sus registros, realiza la suma del producto con los coeficientes del filtro, y hace un desplazamiento de los registros con los valores de la imagen, de forma que se descarten los valores mas antiguos de cada columna, y se cargue un nuevo valor por columna, como en una estructura FIFO (first in, first out). Como muestra la figura 8, esto es equivalente a ubicar el filtro sobre tres columnas de la imagen y desplazarlo verticalemente.

Luego de analizar distintas alternativas, se decidió que cada bloque de convolución que se tiene trabajando en paralelo produzca columnas procesadas adyacentes, es decir, que el bloque de convolución 1 produzca la primer columna de la imagen, el bloque 2 la segunda y asi sucesivamente. Una de las razones

4. Conclusion

The conclusion goes here.

Acknowledgments

The authors would like to thank...

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.