Implementación de la convolución en Marie Assembler (Noviembre de 2024)

Juan D. Chicaiza, Emilio N. Soria, Francisco J. Alarcón

I. INTRODUCCION

La convolución es una operación matemática fundamental utilizada en una variedad de disciplinas, incluidas el procesamiento de señales, el análisis de imágenes, el aprendizaje profundo y la redes neuronales convolucionales. En términos generales, la convolución consiste en aplicar un filtro o kernel a una matriz de entrada generando una matriz de salida que resalta o transforma ciertas características de los datos originales. En el procesamiento de imágenes, por ejemplo, la convolución se utiliza para tareas como el suavizado, la detección de bordes y la aplicación de filtros personalizados.

La importancia de la convolución radica en su capacidad para extraer información relevante de datos en bruto, facilitando de análisis y la interpretación de patrones. En redes neuronales, la convolución permite detectar características como líneas, bordes y texturas en diferentes niveles de abstracción, lo que es crucial para el reconocimiento de imágenes y la visión por computadora. Por otro lado, en el procesamiento de señales, se aplica para filtrar ruido, detectar señales específicas o realizar operaciones de correlación.

Implementar la convolución en un entrono como MARIE, un lenguaje ensamblador con un conjunto de instrucciones reducido, demando un conocimiento de programación a bajo nivel y del funcionamiento de ciertos procesos de este. MARIE carece de operaciones directas para la multiplicación y división, lo que requiere un enfoque alternativo utilizando sumas y restas repetidas para simular estas operaciones. Este proyecto no solo demuestra la factibilidad de implementar convoluciones en un entorno limitado, sino que también ofrece una valiosa experiencia en la comprensión de los mecanismos internos de una computadora y la optimización del código.

II. METODOLOGÍA

La metodología utilizada para implementar la convolución en el ensamblador de MARIE se dividió en algunas partes. En primer lugar se creó un programa en C++ que sirva como guía para entender cómo funciona el proceso de convolución y como podría implementarse en el lenguaje ensamblador especificado. A continuación se muestra el pseudocódigo que sirvió como base:

```
Inicializar punteros y matrices (X, K, Y)

FOR i desde 0 hasta (tamaño de la imagen - 1):
  FOR j desde 0 hasta (tamaño de la imagen - 1):
    Inicializar Sum en 0

    FOR ki desde 0 hasta (tamaño del kernel - 1):
      FOR kj desde 0 hasta (tamaño del kernel - 1):
        Calcular ni = i + ki - padding
        Calcular nj = j + kj - padding

        SI (ni >= 0 Y ni < tamaño de la imagen) Y (nj >= 0 Y nj < tamaño de la imagen):
          temp_resultado = X[ni][nj] * K[ki][kj]
          Sum = Sum + temp_resultado

    Almacenar Sum en Y[i][j]

  Imprimir matriz X
  Imprimir matriz K
  Imprimir matriz Y
Fin del programa
```

Un detalle importante a señalar, es el padding. El padding es un proceso que se hace normalmente en la convolución que consiste en rodear a la matriz de ceros para cuando el kernel se sitúe en los contornos de la matriz, no exista conflicto al tener que encontrar algo por qué multiplicar los elementos del kernel sin su recíproco en la matriz. En este caso se realizó un proceso diferente en el que el padding se ignora omitiendo las posiciones que exceden los límites de la matriz de entrada, por ejemplo, digamos que el kernel se encuentra en las posiciones [0, 0] de la matriz de entrada. En ese caso las posiciones 0 en i y en j van a exceder los límites de la matriz de imagen. Por lo que se descartan.

Otra observación importante al momento de realizar el código fue el uso de un programa en Python que facilita la creación de matrices en lenguaje ensamblador. De esta manera se

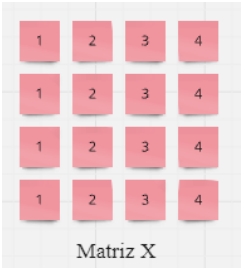
optimizó la manera de hacer diferentes pruebas con diferentes matrices. Esto se pudo realizar gracias a que las matrices y punteros a ellas estaban definidas al inicio del programa y solo bastaba con un programa que cuente el número de líneas que se van a generar teniendo en cuenta el inicio del programa. Así se pudieron obtener los índices de cada matriz, y el puntero inicial que salta a la posición donde inicia el programa, evitando de esta manera conflictos dentro del código.

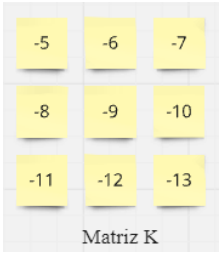
Finalmente, se destaca la importancia de las subrutinas, ya que gracias a ellas se pudieron optimizar y modular procesos como la impresión de las matrices, operaciones complejas como la multiplicación, bucles for, condicionales, etc.

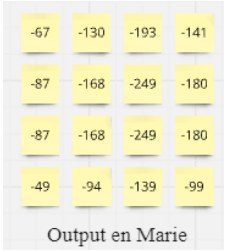
III. RESULTADOS

Se realizaron las siguientes pruebas para comprobar el buen funcionamiento del código. En ellas se puede observar que se realizó una correcta implementación. Siendo la matriz de resultado (“Output en Marie”), correcta con respecto al proceso de convolución entre la Matriz X y la Matriz K, que en este caso representan la matriz de imagen y la matriz del kernel respectivamente.

A. *Prueba 1*







B. *Prueba 2*

IV. CONCLUSIÓN

A partir de estos resultados, se puede concluir un funcionamiento correcto en el código. Se destaca principalmente la importancia de creación de subrutinas y de alternativas simples ante problemas como el manejo del padding. Y finalmente, se complementa con un buen uso y entendimiento de la programación a bajo nivel y del uso de recursos de programación limitados.

REFERENCIAS

[1] Cuartas, J. (2021, January 30). *El concepto de la convolución en gráficos, para comprender las Convolutional Neural Networks (CNN) o redes convolucionadas*. Medium. https://josecuartas.medium.com/el-concepto-de-la-convoluci%C3%B3n-en-gr%C3%A1ficos-para-comprender-las-convolutional-neural-networks-cnn-519d2eee009c

[2] Ramos, O. E. (2020). *Introducción a Visión Computacional (III)*. UTEC. Recuperado de (2024). Epizy.com. http://oramosp.epizy.com/teaching/201/rob-autonoma/clases/5_Intro_Vision_Computacional_III.pdf?i=1

[3] MARIE. (n.d.). *MARIE.js*. *https://github.com/MARIE-js/MARIE.js*