Proyecto de la asignatura

(Actualizado el 04/02/2020)

1. Introducción

A continuación, se describe el proyecto que será la base de una parte importante de la evaluación del laboratorio. En particular, de las prácticas relacionadas con la programación paralela con las librerías OpenMP y MPI que estudiaremos en la asignatura.

Así, cada equipo de trabajo tendrá que trabajar sobre el algoritmo propuesto para este curso, del que realizará, en primer lugar, una implementación secuencial en lenguaje C o C++. El algoritmo seleccionado será analizado por los miembros del equipo, del que se extraerán sus características principales (dependencia de datos, complejidad computacional, necesidades de memoria, etc.) y sus posibilidades de paralelización, en función del modelo de programación en cada caso, y del paradigma de programación paralela elegido para su implementación. Esta última parte culminará en la sesión de laboratorio en la que se mostrarán los resultados experimentales obtenidos (aceleración obtenida para diferentes tamaños del problema y número de procesadores, escalabilidad, etc.) y su justificación al profesor de laboratorio.

Además, las especificaciones del programa a desarrollar permitirán que sea posible, tanto la comprobación de su correcto funcionamiento, como la medida de tiempos de su ejecución. Estos elementos se tendrán en cuenta para la evaluación del trabajo. En el siguiente apartado se describe con detalle el proyecto de este curso.

2. Descripción del proyecto:

Procesado de una Red Neuronal Convolucional

El proyecto propuesto para este curso se centra en el procesado de una Red Neuronal Convolucional (Convolutional Neural Network, CNN o ConvNet), que pertenece a un tipo de lo que se ha dado en llamar Redes Neuronales profundas (Deep Neural Networks, DNN), como se muestra en la Figura 2.1, donde se puede observar su arquitectura, que está formada por la etapa de entrada, la de salida y las etapas ocultas.

Este tipo de redes neuronales digitales implementan una de las técnicas de Inteligencia Artificial (Rusell-2015) que se conocen como Aprendizaje Profundo o *Deep Learning* (DL) (Skansi-2018). Se trata de un subtipo de aprendizaje automático (Machine Learning, ML) en el que un modelo aprende a realizar una clasificación directamente desde un conjunto de datos de entrada (imágenes, texto, o sonido). En el caso de ML, el especialista extrae manualmente las características relevantes de los datos de entrada, por ejemplo, de una imagen, mientras que con DL alimentamos directamente con las imágenes la DNN, que aprende las características automáticamente. El termino profundo se refiere al número de etapas en la red (cuantos más niveles, más profunda es la red). Las redes neuronales tradicionales contienen solo 2 o 3 etapas, mientras que las redes profundas pueden tener cientos.

Las DNN aprenden usando datos de entrenamiento, por ejemplo, un conjunto de imágenes, en el caso de reconocimiento de objetos, donde cada grupo de imágenes contiene una categoría

diferente de objeto. De esta forma, la red puede comenzar a asociar características de objetos específicos y clasificarlos con su correspondiente categoría. Así, cada etapa de la red neuronal toma los datos de la etapa previa y los pasa transformados a la siguiente, haciendo que la red aprenda directamente de los datos, sin influencia de qué características están siendo aprendidas.

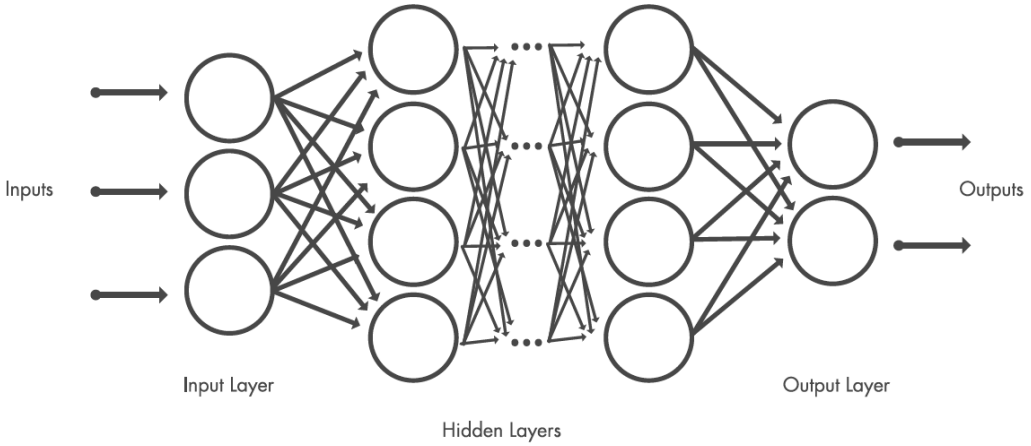


Fig. 2.1 Arquitectura de una DNN genérica (fuente:Mathworks).

El campo de aplicación de este tipo de DNN es muy amplio y va desde el reconocimiento de imágenes/sonidos, sistemas conversacionales (*chatbots*), etc. Esta tecnología requiere entrenar estas redes con un conjunto masivo de datos etiquetados, el incremento de la potencia computacional (computadores con un alto paralelismo, fundamentalmente basadas en aceleradores SIMD basados en GPU) y la disponibilidad de modelos pre-entrenados por expertos.

Una CNN es uno de los algoritmos más populares de aprendizaje profundo para imágenes y vídeo, y está compuesta por una etapa de entrada, una de salida y el conjunto de etapas ocultas entre ellas. En la Figura 2.2 se puede ver la arquitectura de una red CNN, donde se aprecia el bloque de extracción de características y el bloque de clasificación.

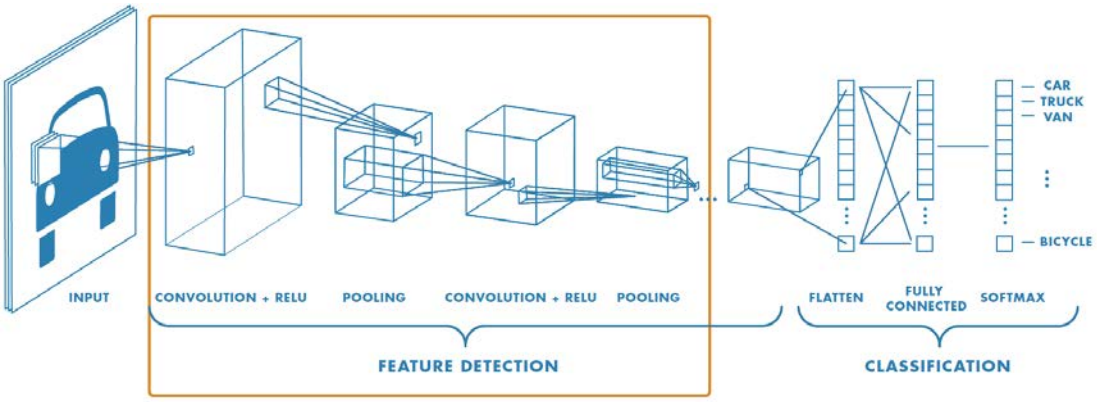
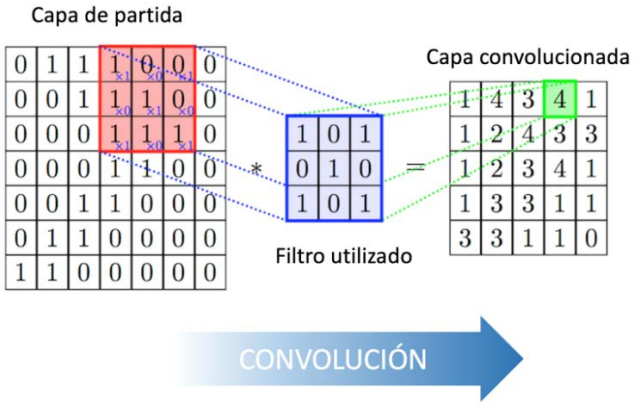


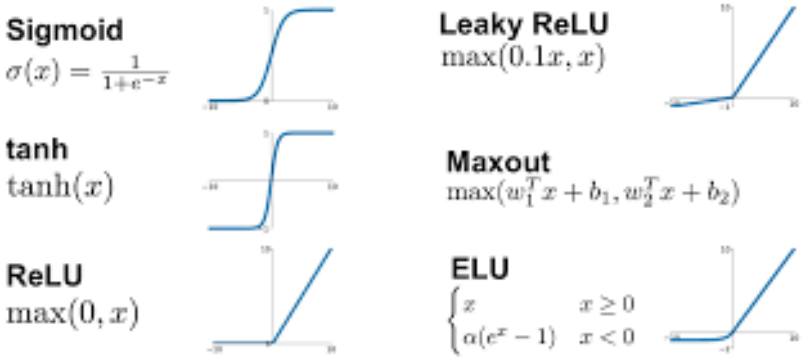
Fig. 2.2 Red Neuronal Convolucional (CNN) formada por bloque de detección de características(recuadro) y un bloque de clasificación (fuente: Mathworks).

El bloque de extracción está constituido por tres etapas, cada una de ellas con tres tipos de operaciones sobre los datos:

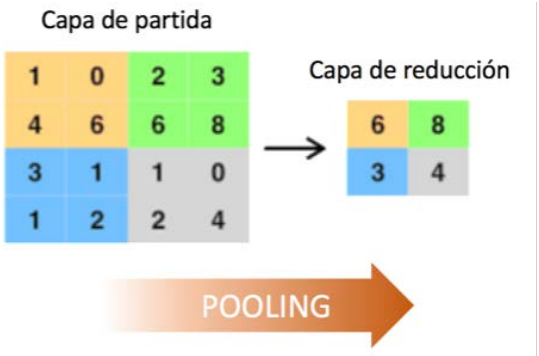
- **Convolución**. Toma la imagen de entrada y le aplica un conjunto de filtros convolucionales, cada uno de los cuales activa alguna característica de las imágenes.



- **Función de unidad lineal rectificada (ReLU).** Permite un aprendizaje más rápido y efectivo de la red, convirtiendo los valores negativos a cero y manteniendo los valores positivos. El ReLU es una opción, pero hay otras funciones, como se puede ver a continuación:



- **Pooling**. Simplifica la salida realizando un sub-muestreo no lineal, reduciendo el número de parámetros que la red necesita para aprender.



Cada una de estas operaciones se puede repetir sobre decenas o cientos de etapas, cada una de ellas aprendiendo a detectar características diferentes.

El bloque final de clasificación está formado por una red neuronal completamente conectada (Full Connected Network, FCN) que obtiene un vector de *k* dimensiones, donde *k* es el número de clases que la red es capaz de predecir. Este vector contiene las probabilidades para cada una de las clases o categorías de las imágenes que están siendo clasificadas. La etapa final de la arquitectura utiliza una función **softmax** para proporcionar la clasificación de salida.

En este proyecto nos vamos a centrar en los cálculos de una etapa del primer bloque de una CNN, el dedicado a la extracción de características, y no vamos a detenernos, en ningún caso, en el algoritmo de aprendizaje de la red neuronal.

Para ello, supongamos que partimos de una imagen de dimensión *n*n*. Así, cuando accedamos a los elementos de la imagen lo haremos utilizando dos coordenadas para indicar la posición en la imagen (*x*,*y*). Es decir, para una imagen A de dimensiones *n*n* (A[*n*][*n*]), nos referiremos a sus elementos como A(x,y), como a_{xy} (ver Figura 2.3), o como en el caso del lenguaje C como A[x][y] (donde los valores de *x* e *y* estarán en los rangos: *x*=0..*n*-1 e *y*= 0..*n*-1).

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix}$$

Fig. 2.3 Imagen representada en la matriz A de nxn elementos.

A continuación, se describen las características del programa que se tendrá que desarrollar y optimizar en las prácticas de este curso, y que tendrá que estar escrito en lenguaje C/C++:

Entrada de parámetros:

El programa tendrá que admitir en línea de comandos el parámetro que corresponde a las dimensiones de la imagen y el tipo de ejecución:

<executable> n t

donde *n*n* es el tamaño de la imagen y *t* indica el tipo de resultado. El primer parámetro se utilizará para la reserva dinámica de la imagen mediante una estructura continua en memoria, y en el caso de no pasarle ningún parámetro, se tomará por defecto el valor n=120. Asumiremos en cualquier caso que *n* será siempre múltiplo de 2.

El último parámetro, *t*, puede tener dos valores posibles ‘t’ y ‘d’. En el primer caso, el programa dará el resultado del tiempo de ejecución en segundos (con una precisión de microsegundos), mientras que en el segundo caso se mostrarán los valores de algunos elementos del resultado.

Per a facilitar la obtención de los resultados y el proceso de paralelización posterior, las dimensiones podrán ser múltiplos del número de núcleos (*cores*) o nodos a utilizar en cada caso.

Generación de los datos:

Los datos de la imagen sobre los que se trabajará serán valores acotados entre 0 y 1000, y se almacenarán en variables de tipo *float*. La función utilizada para la generación de los datos de la matriz de la imagen de entrada será la siguiente:

$$A(x,y) = \left(\frac{xy + x + y}{3n^2}\right) 10^3 .$$

Para realizar la convolución a la imagen de entrada, necesitaremos una batería de filtros convolucionales. Así, definiremos una estructura de *s* filtros convoluciones de tamaño *m*m* como el array C, que por tanto tendrá dimensiones *s*m*m* (C[s][m][m]). Nos referiremos a sus elementos como C(z,x,y) o, como en el caso del lenguaje C, como C[z][x][y]. Así, C[z][x][y] será el elemento *x*,*y* de filtro *z* de C.

Esta batería de filtros los generaremos aleatoriamente con la función *rand()* del siguiente modo:

```
srand(5);
for(k=0; k<s;k++){
    for(i=0; i<m;i++)
       for(j=0; j<m;j++){
          C[k][i][j]= (rand()*pow(-1,j+k))/RAND_MAX;
```

Con ello haremos que sea sencilla la comprobación del correcto funcionamiento de la solución implementada.

Algoritmo de procesamiento o cálculo:

El algoritmo de procesamiento de los datos de la imagen consistirá en la computación de una primera etapa de la obtención de características de una CNN. En particular, esta etapa estará formada por las siguiente funciones o pasos:

1) **Cálculo de la convolución de la imagen**.

   Dados los elementos de la imagen *A*(x,y), calcularemos la aplicación de los filtros de la siguientes manera:

   Asumiremos que la batería de filtros convolucionales está formada por *s*=20 filtros de dimensión *m*=5 (*C*[20][5][5])

   Para cada uno de los elementos de A dentro del rango: *x*= 2..*n*-3 e *y*= 2..*n*-3, y suponiendo que *B* es un array de dimensiones *s*n*n* (*B*[*s*][*n*][*n*]), se aplicará cada uno de los filtros según el siguiente procedimiento para obtener el array *B*:

$$B(z,x,y) = \sum_{i=-m/2}^{m/2} \sum_{j=-m/2}^{m/2} A(x+i, y+j) C\left(z, \frac{m}{2}+i, \frac{m}{2}+j\right).$$

2) **Aplicación de la función no lineal**.

   En el proyecto, con el fin de tener un alto coste computacional, utilizaremos una función ***sigmoid*** a los elementos de la matriz transformada *B*. Para ello aplicaremos la siguiente función para todos los elementos de *B*:

$$B(z,x,y) = \frac{1}{1+e^{-B(z,x,y)}}.$$

3) **Pooling**.

Como continuación de la etapa anterior, y con el fin de reducir la dimensionalidad del array *B*, se aplicará un submuestreo sobre la estructura de datos. Para ello, cada cuatro elementos del array *B* se convertirá en un elemento de una nueva matriz *R* aplicando la siguiente función:

$$R(z,x,y) = \max(B(z,2x,2y), B(z,2x,2y+1), B(z,2x+1,2y), B(z,2x+1,2y+1)),$$

donde *R* tiene dimensiones *s***n*/2**n*/2 (*R*[*s*][*n*/2][*n*/2]).

4) **Promediado**.

Para finalizar y poder comprobar el resultado del funcionamiento de las etapas, se obtendrá el valor promedio de los componentes de todos los planos convolucionales contenidos en *R* mediante el siguiente procedimiento:

$$M(x,y) = media\big(R(:,x,y)\big) = \frac{1}{s}\sum_{k=0}^{s-1} R(k,x,y)\,, \forall\, x,y: 0..\frac{n}{2}-1,$$

donde el array M tendrá dimensiones *n*/2**n* /2 (M[n/2][n/2]).

Salida de datos:

Como resultado de nuestro programa, se obtendrá el tiempo de ejecución en segundos y unos valores del resultado de procesamiento que servirán para comprobar el buen funcionamiento de la implementación. El tiempo de ejecución se calculará sin tener en cuenta la generación ni la impresión de los datos, utilizando la función gettimeofday().

Así, si t='t', la salida será el nombre del programa seguido del tiempo de ejecución en segundos:

<ejecutable>: xx.xxxxxx

Mientras que si t='d', la salida será el nombre del programa seguido de los valores de la imagen A(i,j) y los valores finales media M(i,j) en las posiciones i=j= 0,..,3:

<executable>: A(0,0): A(1,1): A(2,2): A(3,3): M(0,0): M(1,1): M(2,2): M(3,3)

El resultado al ejecutarlo en ***boe.uv.es*** con el valor por defecto (*n*=120) es el siguiente:

<executable>: 0.0000 : 0.0694: 0.1852 : 0.3472 :0.5000: 0.5180 : 0.5159: 0.3472 0.5108

3. Evaluación

La evaluación de los trabajos de cada grupo la llevará a cabo el profesor de laboratorio y tendrá dos partes: una previa a las sesiones de laboratorio y otra después de realizar las pruebas experimentales en el laboratorio.

La primera toma de contacto con el proyecto será el estudio del algoritmo propuesto y comenzará con un análisis de sus características. Una vez hecho esto, el grupo implementará una versión secuencial que cumpla con todos los requisitos indicados. Sobre esta implementación se

hará un análisis de coste computacional y un análisis de dependencia de datos de cara a su aceleración posterior. Este trabajo se presentará al profesor de laboratorio y se enviará a una actividad dedicada para ello en formato *doc* o *pdf* del aula virtual.

Posteriormente, para cada una de las técnicas de aceleración que se verán en la asignatura, el grupo de trabajo realizará las modificaciones necesarias sobre el código para adaptarlo a los diferentes modelos de programación. En cada caso, el código propuesto se mostrará al profesor y una copia del código se subirá a cada una de las actividades diseñadas al efecto.

Cada una de estas actividades de análisis precederá a las prácticas asociadas a cada modelo de programación y, en el laboratorio, el grupo de trabajo explicará al profesor las técnicas utilizadas para la implementación del código paralelo para, posteriormente, mostrar los resultados experimentales obtenidos.

Así, en los resultados mostrados al profesor se reflejarán el tiempo y aceleración obtenidas para el código respecto de la versión sin acelerar, variando el tamaño del problema y el número de elementos de proceso utilizados para su ejecución. También se mostrarán y justificarán los resultados de escalabilidad obtenidos en función del algoritmo, la implementación y la arquitectura del computador utilizado.

En la descripción de cada práctica se darán instrucciones más detalladas relativas al proceso de evaluación que serán aclaradas por el profesor de laboratorio.

Referencias

Skansi, S. (2018). *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. New York, NY, USA: Springer.

Russell, S., Norvig, P. (2015). *Artificial Intelligence – A Modern Approach* (3rd Edition). Prentice Hall.