Escola Tècnica Superior d'Enginyeria ETSE-UV

Grado en Ingeniería Informática

Proyecto de Arquitectura de Computadores

Procesado de una Red Neuronal Convolucional

Jorge Calleja García

Ainhoa Pau Gallach

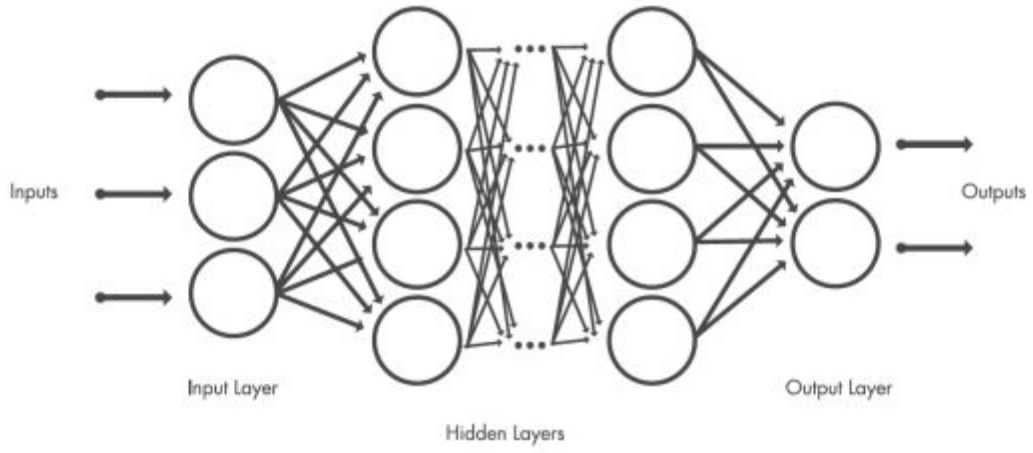
Cristhian Torrico Castelló

1) Objetivo y utilidad del algoritmo.

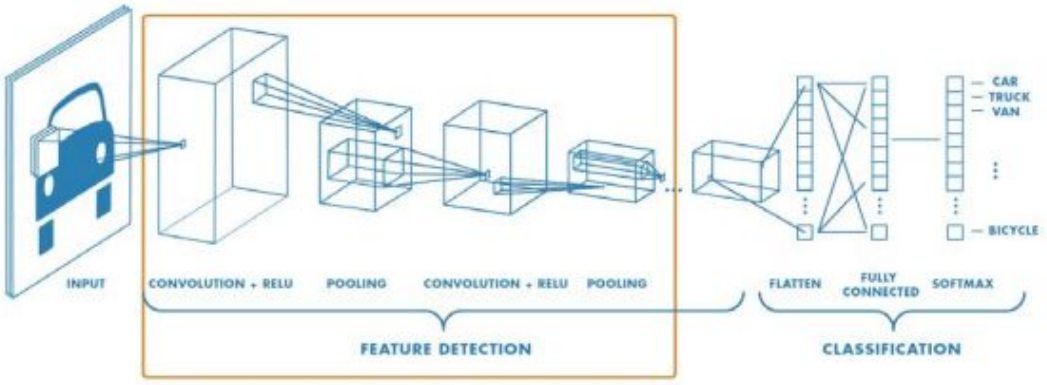
El proyecto propuesto se centra en el procesado de una Red Neuronal Convolucional (Convolutional Neural Network, CNN o ConvNet), que pertenece a un tipo de lo que se ha dado en llamar Redes Neuronales profundas (Deep Neural Networks, DNN)[1].

La red se compone de múltiples capas. En el principio se encuentra la fase de extracción de características compuesta de neuronas convolucionales y de reducción. A medida que se avanza en la red se disminuyen las dimensiones activando características cada vez más complejas. Al final se encuentran neuronas sencillas para realizar la clasificación.

El éxito de estos sistemas se debe en gran parte a la evolución de la capacidad computacional. La inteligencia artificial actualmente permite la solución de problemas en diversos campos que requieren un alto costo computacional. Estos problemas van desde el dise no de asistentes personales, traductores, vehículos autónomos, hasta sistemas de recomendaciones de productos multimedia y compras en línea entre otros [2].



Esquema de una CNN



2) Descripción del algoritmo.

El algoritmo está dividido en las siguientes partes:

Cuando llamamos al algoritmo hay que pasarle n y t:

$$< ejecutable > \ n \ t$$

Siendo n*n el tamaño de la imagen. El parámetro t tiene dos valores distintos, ‘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.

a) Generación de datos:

Hay que generar la siguiente matriz que será nuestra matriz de entrada:

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

b)Generación de filtros:

Para realizar la convolución de la imagen nos hará falta una batería de filtros de tamaño m*m. Será el array C. C[s][m][m]. El número de filtros convolucionales es s.

```
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;
                        }
                }
        }
```

c)Cálculo de la convolución de la imagen:

Teniendo A(x,y) y la batería de filtros 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.

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

d) Aplicación de la función no lineal:

Utilizaremos la función sigmoide a los elementos de B:

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

e) Pooling: Para reducir el array vamos a realizar un submuestreo sobre la estructura de datos. De cada cuatro elementos elegiremos el máximo para crear la nueva matriz R.

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

f) Promediado:

Finalmente para entender el resultado del funcionamiento de las capas se obtendrá el valor medio de todas las componentes de todos los planos convolucionales:

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

3) Análisis del coste y requerimientos de memoria.

- Necesidades de memoria:
  Para la matriz de imagen n*n A[n][n]
  int n => (4bytes + 4bytes) *n = 8n
  Para la matriz de Convulción con s filtros de tamaño m*m C[s][m][m]
  int s = 20;
  int m= 5; => (4bytes + 4bytes)*5 + 4bytes*20 = 120 bytes
  Para el cálculo de la convulción de la imagen n*n y los filtros s B[s][n][n]
  int s = 20;
  int n; => (4bytes + 4bytes) *n + 4 bits*20 = 8n + 80
  Para calcular la matriz del Pooling: R[s][n/2][n/2]
  int s = 20;
  int n; => 4 bits*20 + (4bytes + 4bytes)*(n/2) = 80 + 4n
  Para calcular la matriz del Promedio M[n/2][n/2]
  int n; => (4bytes + 4bytes)*(n/2) = 4n

- Análisis de costes:
  Generacón de matriz: 7n *n

  Generación de la batería de filtros: 3m*m*s

  Cálculo de la convulción de la imagen: 6m*m*(n - 4)*(n - 4)*s

  Aplicación de la función no lineal: 2(n - 4)*(n - 4)*s

  Pooling: *(n / 2)*(n / 2)*s

  Promedio: 1*(n / 2)*(n / 2)*s

4) Análisis de dependencia de datos.

```
for(int z=0; z<s ; z++)
{
        for(x=2; x<n-2; x++)
        {
                for(y=2; y<n-2; y++)
                {
                B[z][x][i]= 1/(1 + pow(2.718281828459045, -B[z][x][y]));
                }
        }
}
```

Vemos que si esta función la paralelizamos habrá dependencia de datos en el array B.
Al hacerlo paralelo cogerá valores incorrectos que aún no se han calculado.
Hay dependencia de lectura, al leerse a si mismo distintas posiciones.

```
int media = 0;
for (z=0; z<s; z++){
        for(x=0; x<n/2; x++){
                for(y=0; y<n/2; y++){
                        media += R[z][x][y];
                }
        }
}
media = media / s;
```

Vemos que si esta función la paralelizamos habrá dependencia de datos en la variable "media".

Dependencias de control:

```
int max;
for (z=0; z<s; z++){
        for(x=0; x<n/2; x++){
                for(y=0; y<n/2; y++){
                        if(B[z][2*x][2*y] > B[z][2*x][2*y + 1])
                                max = B[z][2*x][2*y];

                        if(max < B[z][2*x+1][2*y])
                                max = B[z][2*x+1][2*y];

                        if(max <  B[z][2*x+1][2*y+1])
                                max = B[z][2*x+1][2*y+1];

                        R[z][x][y] = max;
                }
        }
}
```

Vemos que si esta función la paralelizamos habrá dependencia de datos en el array B.
Al hacerlo paralelo, para hacer la comparación en los "if" cogerá valores incorrectos que aún no se han calculado.
Hay dependencia de lectura, al leerse a si mismo distintas posiciones.

5) Propuestas de paralelización.

Si convertimos cada matriz a vector mejoraremos mucho por el hecho que tendremos que utilizar menos bucles for.

Una matriz declarado de esta forma : float matriz[10][10]; Solo tiene un canal de entrada y no de salida, entonces no podemos paralelizar el cálculo de cualquier cosa con esta matriz deberíamos de cambiar de estructura.

Lo coherente sería transformar nuestras matrices a : `float **matriz;`
Se puede intentar bajar el número de bucles debido a que generan un gran problema computacional

6) Referencies:

Hemos trabajado sobre la información proporcionada por el profesor en el archivo 1. AC_Projecto-1920v1.pdf, en el que sus referencias son las siguientes.

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.

2. QUINTERO, Carlos, et al. Uso de Redes Neuronales Convolucionales para el Reconocimiento Automático de Imágenes de Macroinvertebrados para el Biomonitoreo Participativo. *KnE Engineering*, 2018, p. 585-596.

Concepto evaluado	Valor
Completitud: están todos los apartados (incluido el tiempo dedicado) (0-1)	
Claridad de la redacción (0-1)	
Objetivo y utilidad del algoritmo (0-1)	
Descripción del algoritmo (0-1)	
Estructura y tipo de datos, y memoria utilizada por el algoritmo (0-1)	
Análisis de costes del algoritmo (0-1)	
Análisis de dependencia de datos (0-1)	
Propuestas de paralelización: distribución de datos y tareas que se pueden realizar en paralelo (0-1)	
Referencias (al final del documento) (0-1)	
Formato del documento (pdf o doc) sin comprimir (0-1)	
Total	