PC1 - Clasificación y detección de imágenes de números de la BD MNIST

Student:
Alex Avila Santos
Universidad Nacional de Ingeniería, Facultad de Ciencias,
e-mail: aavilas@uni.pe
Student:
Jared Miguel Hidalgo Esquivel
Universidad Nacional de Ingeniería, Facultad de Ciencias,
e-mail: jared.hidalgo.e@uni.pe
Student:
Konrad Benjamin Trejo Chavez
Universidad Nacional de Ingeniería, Facultad de Ciencias,
e-mail: konrad.trejo.c@uni.pe

Curso:
CC0D1- Deep Learning en visión artificial
PC1

Abstract

[Abstract]
Convolutional neural networks are a kind of deep neural network widely used in deep learning and in image detection and classification. We will use it to perform image detection. We will perform image detection using convolutional networks, for which we will use a database, MNIST, which contains a sample set of handwritten digits to train the convolutional network. Finally we will test for the various types of digits.
Keywords: CNN, MNIST, Learning

Contents

1 Introduction **2**

2 Theoretical Framework **2**
2.1 Redes Neuronales Convolucionales 2
2.2 MNIST 2

3 Metodología **3**

4 Results and Discussion **4**

5 Conclusiones **8**
Bibliografía9

1 Introduction

El avance de la tecnología a cambiado la forma de vida de millones de personas alrededor del mundo dado que nos ha permitido desarrollar labores inpensadas antes. Uno de las áreas que más ha avanzado es la inteligencia artificial la cual ha dado origen a nevas heraamientas y las aplicaciones. Una de las áreas donde los avances han sido más notables es el reconocimiento de imágenes, en parte gracias al desarrollo de nuevas técnicas de Deep Learning o aprendizaje profundo. Hoy en día tenemos ya al alcance de nuestra mano sistemas más precisos que los propios humanos, en las tareas de clasificación y detección en imágenes.

En el siguiente informe Realizaremos la detección de imágenes usando redes convolucionales para lo cual usaremos una basse de datos, MNIST, que contiene un conjunto de muestra de digitos manuscristos.

2 Theoretical Framework

2.1 Redes Neuronales Convolucionales

Las redes neuronales convolucionales son un clase de red neuronal profunda. Las redes neuronales convolucionales son redes de inspiración biológica que se utilizan en computadoras visión para clasificación de imágenes y detección de objetos. La motivación básica para lo convolucional red neuronal se obtuvo de la comprensión de Hubel y Wiesel del funcionamiento de la corteza visual del gato, en el que porciones específicas del campo visual parecían excitar neuronas particulares. Este principio más amplio se utilizó para diseñar una arquitectura dispersa para redes neuronales convolucionales. La primera arquitectura básica basada en esta inspiración biológica era el neocognitrón, que luego se generalizó a la arquitectura LeNet-5,red neuronal convolucional propuesta por Yann Lecun en 1989. En la arquitectura de red neuronal convolucional, cada capa de la red es tridimensional, que tiene una extensión espacial y una profundidad correspondiente al número de características

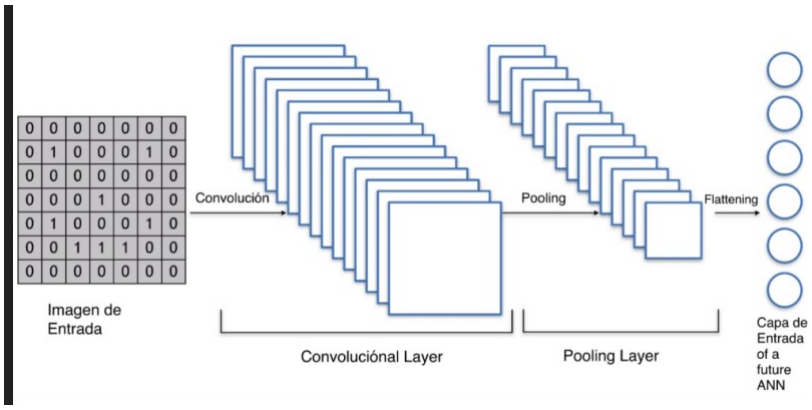

Figure 1: Imagen del proceso de convolución

A continuación definimos los pasos que realizará la red neuronal convolucional:

- **Convolución:** Estas consisten en tomar "grupos de pixeles cercanos" de la imagen de entrada e ir operando matemáticamente (producto escalar) contra una pequeña matriz que se llama kernel.
- **Max Pulling:** Este paso consiste en reducir la cantidad de neuronas antes de hacer la sigueinte convolución. Para esto aplicaremos max pulling que consiste en tomar una submuestra para reducir la entra de una matrix y poder aplicar la siguiente convolución.

2.2 MNIST

La base de datos MNIST de dígitos escritos a mano tiene un conjunto de entrenamiento de 60,000 ejemplos y un conjunto de prueba de 10,000 ejemplos. Es un subconjunto de un conjunto más grande disponible en NIST. Los dígitos se normalizaron en tamaño y se centraron en una imagen de tamaño fijo. Es una buena base de datos para las personas que desean probar técnicas de aprendizaje y métodos de reconocimiento de patrones en datos del mundo real mientras dedican un esfuerzo mínimo al preprocesamiento y formato.

Figure 2: Números del MNIST

3 Metodología

- Desarrollamos las convoluciones para reducir las imágenes.

Figure 3: Diagrama de la convolución

- Desarrollamos la red neuronal en java la cual tiene 169 neuronas de input y 3 capas intermedias.

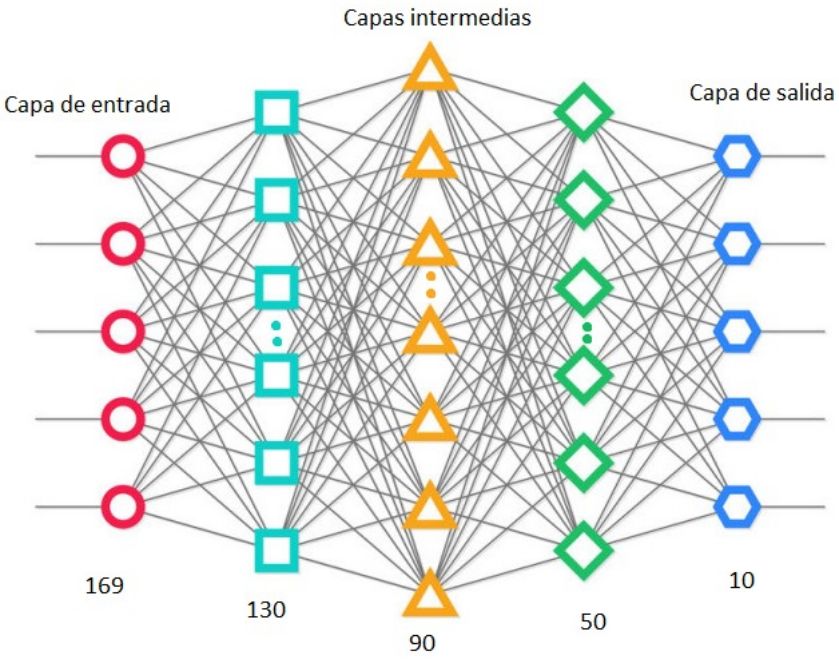

Figure 4: Diagrama de la red neuronal

- Guardamos los pesos entrenados
- testeamos las entradas

4 Results and Discussion

- método que nos permite realizar la convolución

```
public int[][] Convolucion01(BufferedImage image){
        Color colorAux;
        int color;
        int width = image.getWidth(null);
        int height = image.getHeight(null);
        BufferedImage image2 = new BufferedImage(width-2, height-2,BufferedImage.TYPE_INT_RGB);
        //The Laplacian of Gaussian
        int kernel[][] = {{0,1,0},{1,-4,1},{0,1,0}};
        int M[][] = new int[26][26];
        for(int y = 0; y < height-2; y++){
            for(int x = 0; x < width-2; x++){
                int total = 0;
                for(int i = y,a=0 ; i < y+3; a++,i++){
                    for(int j = x,b=0; j < x+3; b++,j++){
                        colorAux=new Color(image.getRGB(j, i));
                        total = total + colorAux.getBlue()*kernel[a][b];
                        if(total<0) total=0;
                        if(total>250) total = 250;
                    }
                }
                if(total > 50 ){
                    M[y][x] = 1;
                }
                else {
                    M[y][x] = 0;
                }
                color=(total << 16) | (total << 8) | total;
                image2.setRGB(x, y,color);
            }
        }
```

Figure 5: método convolución

- método que nos permite realizar el entrenamiento con la red convolucional y guardar las salidas que se usarán como input la red neuronal.

```
public void creartxtTraining(String n){

    try {
        String folder = "./trainingSample/trainingSample/"+n+"/";
        File folderFile = new File(folder);
        boolean resultado;
        int cont = 0 ;
        File[] files = folderFile.listFiles();
            for (File file : files) {
                boolean isFolder = file.isDirectory();
                if(isFolder) continue;
                System.out.println((isFolder ? "FOLDER: " : "  FILE: ") + file.getName());

                int M[][] = new int [26][26];
                String ruta_imagen = "./trainingSample/trainingSample/"+n+"/"+file.getName() ;
                System.out.println(ruta_imagen);
                BufferedImage image = ImageIO.read(this.getClass().getResource(ruta_imagen));
                M = Convolucion01(image);
                System.out.println();
                System.out.println();
                int P[][] = new int[13][13];
                P = Pooling(M,n);
            }
    } catch (IOException e) {
      System.err.println(e.getMessage());
    }
}
```

Figure 6: metodo de training

- método que nos permite realizar el pooling para reducir el tañano del las imágenes.

```
public int[][] Pooling(int matriz[][],String n) throws IOException{

        String ruta = "./input"+n+".txt";
        File file = new File(ruta);
        if (!file.exists()) {
            file.createNewFile();
        }

        FileWriter fw = new FileWriter(file.getAbsoluteFile(), true);
        BufferedWriter bw = new BufferedWriter(fw);
        int M[][] = new int[13][13];
        for(int y = 0; y < 13; y=y+1){
            for(int x = 0; x < 13; x=x+1){
                int max1 = Math.max(matriz[y][x],matriz[y][x+1]) ;
                int max2 = Math.max(matriz[y+1][x],max1);
                int max3 = Math.max(matriz[y+1][x+1],max2);
                M[y][x]= max3;
                 bw.write(max3 +" ");
            }
        }
        bw.write("\n");
        bw.close();
        fw.close();

    return M;
}
```

Figure 7: método pooling

```
public rna01(int ci_,int[] co_,int cs_,String pesos){
    ci=ci_;
    cm=co_.length;     // cantidad de medios inicializada
    //int co=co_;
    co=new int[cm];
    System.arraycopy(co_, 0, co, 0, cm);
    cs=cs_;
    weights=pesos;

    c = new int[cm+2];

    int n_neuron_m = 0;
    for(int i=0; i<cm; i++){
        n_neuron_m = n_neuron_m + co[i];
    }

    y = new double[n_neuron_m+cs];          //salidas de las neuronas
    s = new double[n_neuron_m+cs];          //entradas de las neuronas
    g = new double[n_neuron_m+cs];          //error retropropagado por la ne

    int prod=0;
    for(int i=1; i<co.length; i++){
        prod = prod + co[i-1]*co[i];
    }

    w = new double[ci*co[0]+prod+co[cm-1]*cs];    //pesos;

    c[0]=ci;
    //c[1]=co;
    System.arraycopy(co, 0, c, 1, cm);
```

Figure 8: Inicializador

```
///ci=0;//entrenamiento primero   /////HOPE
ci=cii;
ii = 0;//capa0*capa1
pls=0;
for(int i=0;i<c[1];i++){
    for(int j=0;j<c[0];j++){
        pls=pls+w[ii]*xin[ci][j];
        ii++;
    }
    s[i]=pls;  //i = i+ capa0
    y[i]=fun(s[i]); //i = i+ capa0
    pls=0;
}
int c_sum=0;

for(int n=1; n<cm; n++){
    //ii=ii+c[n-1]*c[n];
    pls=0;
    for(int i=0;i<c[n+1];i++){
        for(int j=0;j<c[n];j++){
            pls=pls+w[ii]*y[j+c_sum];
            ii++;
        }
        s[i+c_sum+c[n]]=pls;
        y[i+c_sum+c[n]]=fun(s[i+c_sum+c[n]]);
        pls=0;
    }
    c_sum=c_sum+c[n];
}
```

Figure 9: Entrenamiento ida

```
//++++capa_final g
for(int i=0;i<c[cm+1];i++){
    g[i+c_sum]=(xout[ci][i]-y[i+c_sum])*y[i+c_sum]*(1-y[i+c_sum]);
}

int prod=0;
for(int i=0;i<cm;i++){
    prod=prod+c[i]*c[i+1];
}
//++++capas_intermedias
for(int n=cm;n>0;n--){
    pls=0;
    for(int i=0;i<c[n];i++){
        for(int j=0;j<c[n+1];j++){
            pls=pls+w[prod+j*c[n]+i]*g[c_sum+j];      // c[1]=>c_sum
        }
        g[i+c_sum-c[n]]=y[i+c_sum-c[n]]*(1-y[i+c_sum-c[n]])*pls;
        pls=0;
    }
    prod=prod-c[n-1]*c[n];
    c_sum=c_sum-c[n];
}
```

Figure 10: Entrenamiento vuelta

```
for(int i=0;i<cm;i++){
    prod=prod+c[i]*c[i+1];
    c_sum=c_sum+c[i+1];
}
for(int n=cm;n>0;n--){
    ii=prod;
    for(int i=0;i<c[n+1];i++){
        for(int j=0;j<c[n];j++){
            w[ii]=w[ii]+g[i+c_sum]*y[c_sum-c[n]+j];
            ii++;
        }
    }
    c_sum=c_sum-c[n];
    prod=prod-c[n-1]*c[n];
}
//++++capal w
ii = 0;//capa0*capa1
for(int i=0;i<c[1];i++){
    for(int j=0;j<c[0];j++){
        w[ii]=w[ii]+g[i]*xin[ci][j];
        ii++;
    }
}
```

Figure 11: Entrenamiento actualización


```
for(int i=0;i<cm;i++){
    prod=prod+c[i]*c[i+1];
    c_sum=c_sum+c[i+1];
}
for(int n=cm;n>0;n--){
    ii=prod;
    for(int i=0;i<c[n+1];i++){
        for(int j=0;j<c[n];j++){
            w[ii]=w[ii]+g[i+c_sum]*y[c_sum-c[n]+j];
            ii++;
        }
    }
    c_sum=c_sum-c[n];
    prod=prod-c[n-1]*c[n];
}
//++++capal w
ii = 0;//capa0*capa1
for(int i=0;i<c[1];i++){
    for(int j=0;j<c[0];j++){
        w[ii]=w[ii]+g[i]*xin[ci][j];
        ii++;
    }
}
```


Figure 12: Entrenamiento actualización

```
int c_sum=0;
for(int n=0;n<cm;n++){
    pls=0;
    //ii=ii+c[n]*c[n+1];
    for(int i=0;i<c[n+2];i++){
        for(int j=0;j<c[n+1];j++){
            pls=pls+w[ii]*y[j+c_sum];
            ii++;
        }
        s[i+c_sum+c[n+1]]=pls;
        y[i+c_sum+c[n+1]]=fun(s[i+c_sum+c[n+1]]);
        pls=0;
    }
    c_sum=c_sum+c[n+1];
}
```

Figure 13: Uso de la red

```
-0.290722089653273 0.6425757226011302 0.10140810226271207 0.11209271905504786 -0.691065344069407 0.
4870531297319647 -1.0019946043831627 0.13098624836022146 -0.19987127242053968 0.4120910549472716 -0
.2859668570369239 -0.5189115175245531 -0.9759101653832888 1.027407601928352 0.22834592328052522 1.4
55 1.129560092628199 -0.7221484492971154 -0.7543341293843897 0.8530391178452015 0.03291054167994328
13812177 0.42346929051792215 -0.44434076088535956 0.2932642603991517 -0.5549842326431864 -0.8172173
9397 -0.17821929921039972 -1.9350587680729672 -1.7376876744754512 -0.027833275604610446 -0.52296516
92911219 -1.0569773536501612 0.7539157452269141 -0.009559404039074813 0.3470356541399692 0.84045903
5482 -0.2949930109111172 -0.5727974274762156 -1.5592303555129126 0.08522229017052334 0.300978278504
54 -0.16842463137613 56 0.2736986366918555 -0.7860521226095287 -1.8338879168061963 -0.91984950264222
912156 1.3679456556951919 1.1972355713459628 0.07984676446557713 -0.0840297550613051 3.830175494119
-0.3199984361517902 1.0274163819381115 -0.5399472611461571 -0.925276631880361 -0.003460824231805754
22072739795986 -1.5573674085905123 -1.4900872202538142 0.47723616295454174 -0.3305227293208888 1.39
76948807 0.3986850495852664 1.00204713240671 4.807132444938432 -0.2303965216992659 0.31174773617653
060435969282 -0.6241572414350222 -1.1554616745984136 3.5847816683652236 1.4162038368279466 0.438606
0.29000194743526847 0.28297648441058326 1.1624480642576416 -0.015475950895515187 -1.331103633847921
0.02879860067361322 -0.1700729898815292 -0.6434473197151213 0.8707666769089508 0.25298498009445797
4290732 1.8969115447501137 0.2654139016176771 0.6516156890351844 -0.20969179892499196 -0.2951930275
354793 -0.18661538662595406 0.7374432536798047 -0.05736632679854741 1.3144965697042958 1.1918857134
130178759339334 1.286313353629433 0.01217279498736297 -0.5144252638121068 0.05566801776448802 0.175
85188583856 0.1950918754853238 -0.548026408840607 0.4369302732458784 -0.6690959871760279 0.34450008
38347 1.0023714879921017 0.6491525742050701 1.8311264525105209 0.014090965553580755 0.1885771021254
0.8193968942212795 -0.6773907477110271 0.9702239812437953 -0.9374190087892866 0.7778924060170731 0.
1.0277364595426897 -0.3725776043337626 0.4736741455138319 -0.33338411458710804 -0.03491421457673509
7703633156954764 -0.2960973561682558 -1.23351610914865 0.4228279299965958 0.2551577893574511 -0.076
9 -0.2230240710754796 -0.6581547945461282 -1.0815974299128708 -0.3536260046151179 -0.02726698110973
8990347911 -1.138121109653632 -0.8059757010243532 -0.3914908948077503 -0.852508321533563 0.49556699
95639914564 0.29022281311080683 1.2629335129989105 0.08468693732619857 -0.02322372160009148 3.73345
8722521654837174 -1.5445366861650067 -1.6480878259119738 -1.5006078656748658 0.08448069986534205 0.
959764136 1.717735188744339 -1.0824735880887195 -0.5015591486630729 0.695680992870255 -0.8477837749
0.4755709203147269 -2.463562980902181 -2.8535375864902637 0.321886843511041 0.014589175934045066 0.
.3212160780959514 1.1910710378672895 1.3732680129818529 -0.8226326403579322 0.8245349015402385 -2.0
509580064831 -0.002377719104423554 0.6571590465640117 0.7153949489816979 1.137610732066716 1.971307
```

Figure 14: Pesos obtenidos

5 Conclusiones

- Se logró crear una red neuronal convolucional para clasificacion de imagenes.
- se logró desarrollar las funciones para el escalamiento de las imágenes.
- Se logró entrenar la red y generar los pesos

References

[1] http://yann.lecun.com/exdb/mnist/.

[2] https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/.

[3] Charu C. Aggarwal. Neural Networks and Deep Learning, 2018.