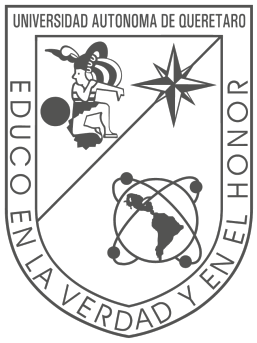
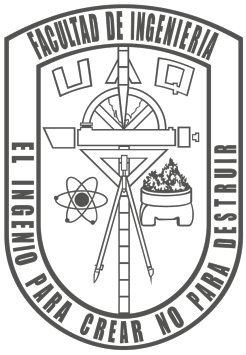
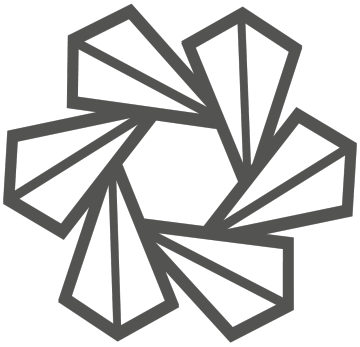
Universidad Autónoma de Querétaro

Facultad de Ingeniería
División de Investigación y Posgrado







Reporte 7

Redes Convolucionales

Maestría en Ciencias en Inteligencia Artificial
Optativa de especialidad II - Deep Learning

Aldo Cervantes Marquez
Expediente: 262775
Profesor: Dr. Sebastián Salazar Colores

Santiago de Querétaro, Querétaro, México
Semestre 2022-2
27 de Septiembre de 2022

Índice

1. Introducción

En la presente práctica se mostrarán la aplicación de una red convolucional a una base datos proporcionada por la librería de Tensor Flow de Python. Todo esto con el fin de observar como es que este tipo de redes tienen amplia aplicación en la clasificación de imágenes. Aplicando una configuración de red neuronal llamada LeNet-5.

2. Marco Teórico

2.1. Base de datos

La base de datos consta de un conjunto de valores que representan dibujos de números mediante imágenes de 1 solo canal de color. Consistente de 70000 imagenes de valores numéricos del 0 al 9 y en imagenes 28x28 pixeles, generando un arreglo de la siguiente manera (1,28,28) donde el primer valor es el número de imagen seleccionada y los otros dos la dimension de la misma. Por otro lado tenemos el vector de respuestas (x,0), el cual indica el valor en la posición del vector del numero de la imagen, el valor real de la imagen (es decir, el resultado) (véase Figura 1).

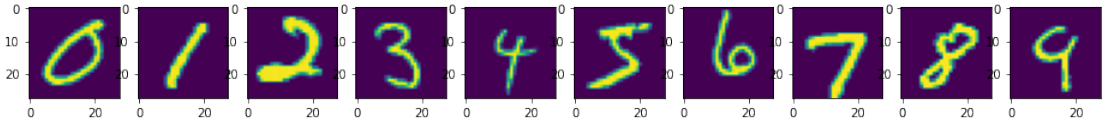


Figura 1: Imágenes de la base de datos.

2.2. Convoluciones

Las convoluciones constan de kernels que permiten modificar la imagen píxel a píxel, generalmente dichos kernels representan una matriz de valores, los cuales interactúan con la cantidad de datos en igual forma para realizar la convolución (véase Figura 2) [1, 2].

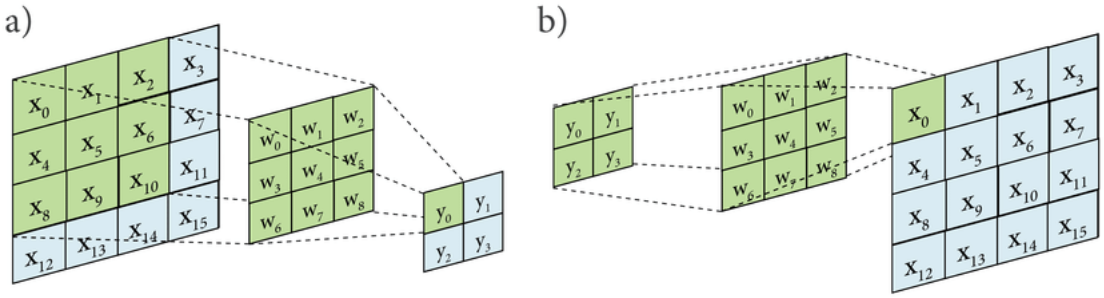


Figura 2: Paleta cúbica de colores RGB.

2.3. Pooling

Como sabemos, las convoluciones permiten resaltar partes de la imagen que nos podrían interesar, siempre conservando prácticamente en su totalidad la imagen. Por otro lado, la capa de pooling nos asegura que los patrones detectados en la capa convolucional se mantengan [3].
Además de que las capas de pooling no requieren de ningún parámetro de aprendizaje. Existen principalmente 3 tipos de Pooling: el maxpool, el minpool y el averagepool. El primero indica que el minímo de los valores de la sección de la matriz de pooling sera el seleccionado como resultado, mismo caso para el máximo y para el promedio del conjunto de valores (véase Figura 3).

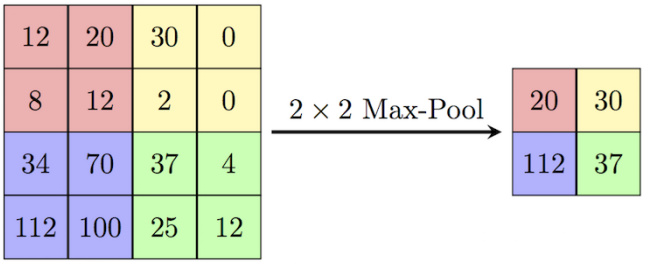


Figura 3: Ejemplo de Maxpool.

2.4. Red Convolucional

Consiste en en algoritmo que al igual que una red neuronal, asigna y actualiza pesos a los valores de la función y por lo tanto se optimizan los valores de los kernels (convoluciones) para poder reconocer patrones de siluetas, curvas, lineas, rostros, etc [4].

2.4.1. Red LeNet-5

Consiste en un arreglo de 7 capas propuesta por Yann LeCun en 1998, tiene como principal objetivo, resolver problemas con imágenes. Dichas capas se van alternando entre convoluciones y Pooling, en donde las capas de convolución son resultado de la multipliación de un kernel por la imagen, por lo que, dichos valores de la matriz del kernel, serán definidas como pesos de la red, el punto es ir disminuyendo el tamaño de la imagen y aplanarla para que pueda entrar a una red neuronal (véase Figura 4).

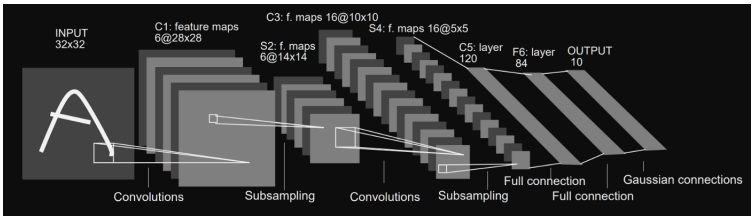


Figura 4: Estructura de red convolucional LeNet5.

3. Justificación

El uso de redes convolucionales permite poder identificar imágenes de una manera más precisa que con redes neuronales normales, puesto que involucra matrices de convolución (kernels) y por lo tanto, es posible obtener de una mejor manera relaciones de patrones de los valores mediante los mapas de características y finalmente aplanarlos en valores sintetizados pero muy útiles en la información que proporcionan [5].

4. Resultados

Primeramente se realizó una exploración de la base de datos, observando que había que agregar valores a las imágenes, esto fue posible con la función *pad* de la librería de *numpy*. Entonces la dimensión de los datos serán: $(1, 28, 28) \rightarrow (1, 32, 32)$

Después se realizó un cambio e la forma de los datos, agregando un 1 al final de lo descrito anteriormente (1,28,28), con el fin de especificar el número de canales que componían a las imágenes (estandarizando al RGB, pero en este caso era una imagen de un solo canal). Entonces la dimensión de los datos ahora será $(1, 32, 32) \rightarrow (1, 32, 32, 1)$.

Se normalizaron los datos en el intervalo $[0, 1]$ mediante la formula:

$$x_{norm}[a, b] = (b - a)\frac{x - x_{min}}{x_{max} - x_{min}} + a \tag{1}$$

Únicamente sustituyendo $a = 0, b = 0$ de este modo se tiene una mejor paridad de todos.

A continuación se realizó la configuración de ONE HOT en la salida como ya se había visto en practicas anteriores.

Finalmente se realizó el programa de la red LeNet5, la cual consta de redes convolucionales con un comportamiento lineal (como una neurona perceptrón):

$$f(\omega_i) = \omega_i x_i \tag{2}$$

Con una función de activación de tangente hiperbólica.

$$y(a) = tanh(a) \tag{3}$$

Sin embargo, para la ultima capa (capa de salida) se ocupa una estructura de RBF (Euclidean Radial Basis Function), el cual tiene una estructura conjunta con la función de activación [5, 6]:

$$y(i) = \sum_j (x_j - \omega_{ij})^2 \tag{4}$$

Cabe resaltar que para la medición del error se usó entropía cruzada, aunque el autor utilizó una funcion parecida a MSE pero con un castigo a las clases incorrectas:

$$E(W) = \frac{1}{P}\sum_{p=1}^{P}(yD^p(Z^p, W) + log(e^{-j} + \sum_i e^{-y_i(Z^p, W)})) \tag{5}$$

Finalmente se hicieron varias pruebas con distintos parámetros como se muestra a continuación:

Tabla 1: Resultados de red neuronal convolucional en Mnist.

Prueba	Función de activación	optimizador	Tasa de aprendizaje	Épocas	Exactitud	Batch	Tiempo de ejecución	Porcentaje de éxito de la prueba
1	Tangente hiperbólica	SGD	0.05	2	96.70%	32	79.42 s	97.4%
2	Tangente hiperbólica	SGD	0.02	2	95.85%	16	37.1490 s	96.83%
3	Tangente hiperbólica	SGD	0.25	3	98.38%	64	50.9887 s	98.54%
4	Tangente hiperbólica	Adam	0.25	3	10.06%	64	99.54 s	9.74%
5	Tangente hiperbólica	Adam	0.01	2	94.22%	32	60.01 s	95.29%
6	Tangente hiperbólica	Adam	0.025	2	84.04%	32	54.432 s	84.28%
7	Sigmoide (S)	SGD	0.001	2	11.24%	32	128.881 s	11.35%
8	Lineal (Relu x4,G)	Adam	0.001	2	14%	16	342 s	22.59%
9	Gaussiana (Gelu)	SGD	0.1	2	9.87%	32	163.7565 s	9.8%
10	Híbrida (R,S,S,G,Smax) **Cambio de capa de sal.**	Adam	0.01	2	98.02%	64	74.03676 s	98.29%

5. Conclusiones

1. El algoritmo funciona con mucha exactitud cuando se utiliza como función de activación la tangente hiperbólica y como ultima neurona una RBF (que es como el autor lo realiza) [5].

2. El uso de otras funciones de activación resultaron muy malas para aplicarse en este algoritmo, pues no rebasaban el 20 %.

3. Se observó que al utilizar diferentes funciones de activación y modificando el RBF, fue posible observar que se mejoraron los resultados en comparación a las demás funciones que iban solas a excepción de la tangente hiperbólica.

4. Fue algo muy notorio que únicamente cambiando el método de SGD a Adam (prueba 3 y 4), teniendo prácticamente los mismos hiperparametros, se obtuvo el valor máximo y mínimo de las pruebas respectivamente.

Se observó que fue posible realizar una red neuronal convolutiva que pudiera trabajar con diferentes funciones de activación y neurona en la capa de salida. Teniendo resultados concretos sobre los hiperparámetros que permten resolver el problema.

Referencias

[1] "2d convolution using python & numpy — by samrat sahoo — analytics vidhya — medium." https://medium.com/analytics-vidhya/2d-convolution-using-python-numpy-43442ff5f381. (Accessed on 09/17/2022).

[2] "5.2. imágenes rgb — introducción a la programación." https://cupi2-ip.github.io/IPBook/nivel4/seccion4-4.html. (Accessed on 09/17/2022).

[3] "Cómo crear red convolucional en keras - ander fernández." https://anderfernandez.com/blog/que-es-una-red-neuronal-convolucional-y-como-crearlaen-keras/. (Accessed on 09/26/2022).

[4] "Intro a las redes neuronales convolucionales — by bootcamp ai — medium." https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8. (Accessed on 09/26/2022).

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition." http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf, November 1998. (Accessed on 09/26/2022).

[6] "python - how to implement rbf activation function in keras? - stack overflow." https://stackoverflow.com/questions/53855941/how-to-implement-rbf-activation-function-in-keras. (Accessed on 09/26/2022).

redes convolucionales

September 27, 2022

Uso de librerias correspondientes para el desarrollo del proyecto

[29]:
```
import tensorflow as tf
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
import numpy as np
from sklearn.model_selection import train_test_split#60,20,20
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
(X,Y),(X1,Y1)=tf.keras.datasets.mnist.load_data()
```

[30]:
```
X.shape
```

[30]:
```
(60000, 28, 28)
```

Se observa una dimensión de 28x28 pero se ajustará a 32x32 como en el articulo

[31]:
```
x_conv_t=np.pad(X,pad_width=(((0,0),(2,2),(2,2))))
x_conv_p=np.pad(X1,pad_width=((0,0),(2,2),(2,2)))
print(x_conv_t.shape)
print(x_conv_p.shape)
# Se debe agregar un 1 al final para agregar el unico canal
x_conv_t=x_conv_t.reshape(x_conv_t.shape[0],32,32,1)
x_conv_p=x_conv_p.reshape(x_conv_p.shape[0],32,32,1)
```

```
(60000, 32, 32)
(10000, 32, 32)
```

Se realizará una normalización de los valores al intervalo de $[0, 1]$

$$x_{norm}[a, b] = (b - a)\frac{x - x_{min}}{x_{max} - x_{min}} + a$$

para posteriormente, realizar la configuración de ONE HOT.

[32]:
```
x_conv_t=(x_conv_t/x_conv_t.max())
x_conv_p=(x_conv_p/x_conv_p.max())
print(x_conv_t.max(),x_conv_p.max())
## ONE  Hot
from tensorflow.keras.utils import to_categorical
print(Y[3450])
```

```
yt_oh=to_categorical(Y,Y.max()+1)
yp_oh=to_categorical(Y1,Y1.max()+1)
print(yt_oh[3450])
print(Y1[200])
print(yp_oh[200])
#### Verificando dimensiones
print(x_conv_t.shape, yt_oh.shape)
print(x_conv_p.shape,yp_oh.shape)
```

```
1.0 1.0
4
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
3
[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
(60000, 32, 32, 1) (60000, 10)
(10000, 32, 32, 1) (10000, 10)
```

Una vez lista las las normalizaciones adecuadas, se plantea la estructura de la red convolucional.

0.1 Empezando por C1

[33]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model=Sequential()
model.add(tf.keras.layers.
 ↪Conv2D(6,(5,5),input_shape=(32,32,1),activation='tanh',padding='valid',strides=1))␣
 ↪#C1
```

1 S2

[34]:
```
model.add(tf.keras.layers.AveragePooling2D(pool_size=(2,2))) #S2
```

2 C3

[35]:
```
model.add(tf.keras.layers.
 ↪Conv2D(16,(5,5),activation='tanh',padding='valid',strides=1)) #c3
```

3 S4

[36]:
```
model.add(tf.keras.layers.AveragePooling2D(pool_size=(2,2))) #s4
model.add(tf.keras.layers.Flatten())
```

4 C5,C6

[37]:
```
model.add(Dense(120,activation='tanh')) #c5
model.add(Dense(84,activation='tanh')) #c6
```

Para el caso de **C7** que es la salida, se comenta que la estructura de la red es de forma RBF, el cual tiene la siguiente estructura:

$$y_i = \sum_j (x_j - \omega_{ij})^2$$

Se encontró un código que permite caracterizar este tipo de neurona.

Obtenido de: https://stackoverflow.com/questions/53855941/how-to-implement-rbf-activation-function-in-keras

[38]:
```
from keras.layers import Layer
from keras import backend as K

class RBFLayer(Layer):
    def __init__(self, units, gamma, **kwargs):
        super(RBFLayer, self).__init__(**kwargs)
        self.units = units
        self.gamma = K.cast_to_floatx(gamma)

    def build(self, input_shape):
        self.mu = self.add_weight(name='mu',
                                  shape=(int(input_shape[1]), self.units),
                                  initializer='uniform',
                                  trainable=True)
        super(RBFLayer, self).build(input_shape)

    def call(self, inputs):
        diff = K.expand_dims(inputs) - self.mu
        l2 = K.sum(K.pow(diff,2), axis=1)
        res = K.exp(-1 * self.gamma * l2)
        return res

    def compute_output_shape(self, input_shape):
        return (input_shape[0], self.units)
```

5 Agregando C7

[39]:
```
model.add(RBFLayer(10,0.5)) #c7
#model.add(Dense(10,activation='softmax')) #c7
```

Posteriormente se compila el modelo con las funciones de perdida y el optimizador, el cual nos indica el articulo que es:

Función de perdida (similitud con la entropía cruzada)

$$E(W) = \frac{1}{P}\sum_{p=1}^{P}(yD^p(Z^p, W) + log(e^{-j} + \sum_i e^{-y_i(Z^p,W)}))$$

Cuya intención es hacer competir a las clases y penalizarlas.

Optimizador (Gradiente descendiente y/o Adam)

[40]:
```
model.compile(loss='categorical_crossentropy',optimizer=tf.keras.optimizers.
↪SGD(learning_rate=0.25),metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_4 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_5 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_2 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_4 (Dense)              (None, 120)               48120
_________________________________________________________________
dense_5 (Dense)              (None, 84)                10164
_________________________________________________________________
rbf_layer_2 (RBFLayer)       (None, 10)                840
=================================================================
Total params: 61,696
Trainable params: 61,696
Non-trainable params: 0
_________________________________________________________________
```

6 Entrenamiento

Debido a la alta cantidad de datos, se realizará un batch para poder tener una muestra significativa de los valores

[41]:
```
import time
import random
m=random.randint(1,1000)
print(m)
print(x_conv_p[0:m,:,:].shape,yp_oh[0:m,:].shape)
t1=time.time()
```

```
model.fit(x_conv_t,yt_oh,epochs=3,verbose=1,batch_size=64)
t2=time.time()
print('Tiempo de entrenamiento', t2-t1)
```

```
21
(21, 32, 32, 1) (21, 10)
Epoch 1/3
938/938 [==============================] - 45s 46ms/step - loss: 0.2407 -
accuracy: 0.9277
Epoch 2/3
938/938 [==============================] - 41s 43ms/step - loss: 0.0754 -
accuracy: 0.9767
Epoch 3/3
938/938 [==============================] - 40s 42ms/step - loss: 0.0507 -
accuracy: 0.9842
Tiempo de entrenamiento 125.24375748634338
```

7 Prueba (Test)

[42]:
```
pred=model.predict(x_conv_p)
pred=np.argmax(pred,axis=1)
#label=np.argmax(yp_oh)
exactitud_test=0
for a in range(len(pred)):
    if pred[a]==Y1[a]:
        exactitud_test+=1
print('exactitud de la prueba= ',100*exactitud_test/len(pred),'%')
```

```
exactitud de la prueba=  98.34 %
```

[ ]:
```

```