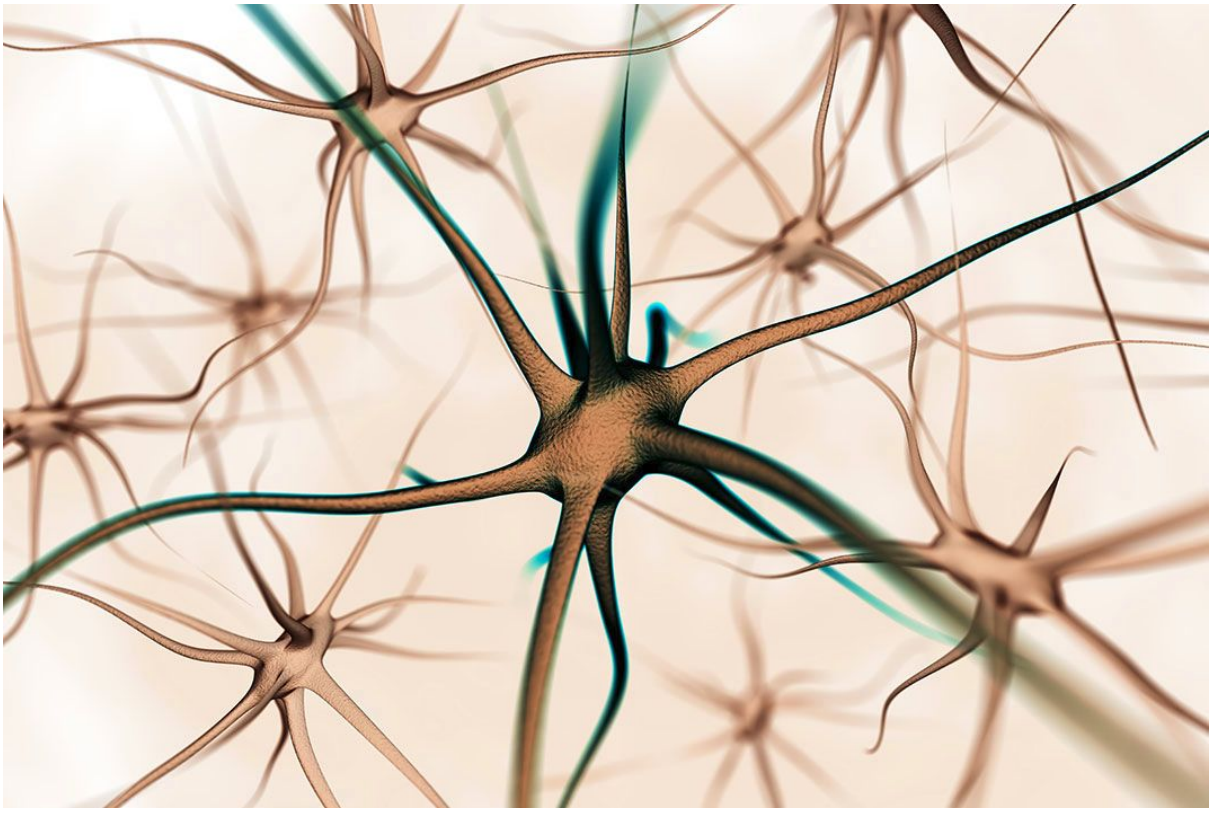
Practica 1 de Inteligencia Computacional:

Redes Neuronales



Matilde Cabrera González

Índice:

1. Introducción:

1.1 Explicación del problema.

Problema planteado en la asignatura Inteligencia Computacional del Máster de Ingeniería Informática de la Universidad de Granada. Vamos a entrenar una red neuronal y explicar los resultados.

En un primer paso entrenaremos una red neuronal de una capa (Perceptrón) y después entrenaremos una red neuronal multicapa (Backpropagation). Para ello tenemos una base de datos de entrenamiento y test “MNIST” que explicaremos a lo largo de este documento.

Cada red neuronal necesitará un preprocesamiento, un entrenamiento y una fase de test donde comprobaremos lo bien que reconoce los elementos del conjunto de la base de datos. Por último analizaremos los resultados.

1.2 Conceptos básicos.

Algoritmo de aprendizaje: asociado al modelo concreto de un red, permite ir modificando los pesos de las conexiones sinápticas de forma que la red aprenda a partir de los ejemplos que se le presentan.

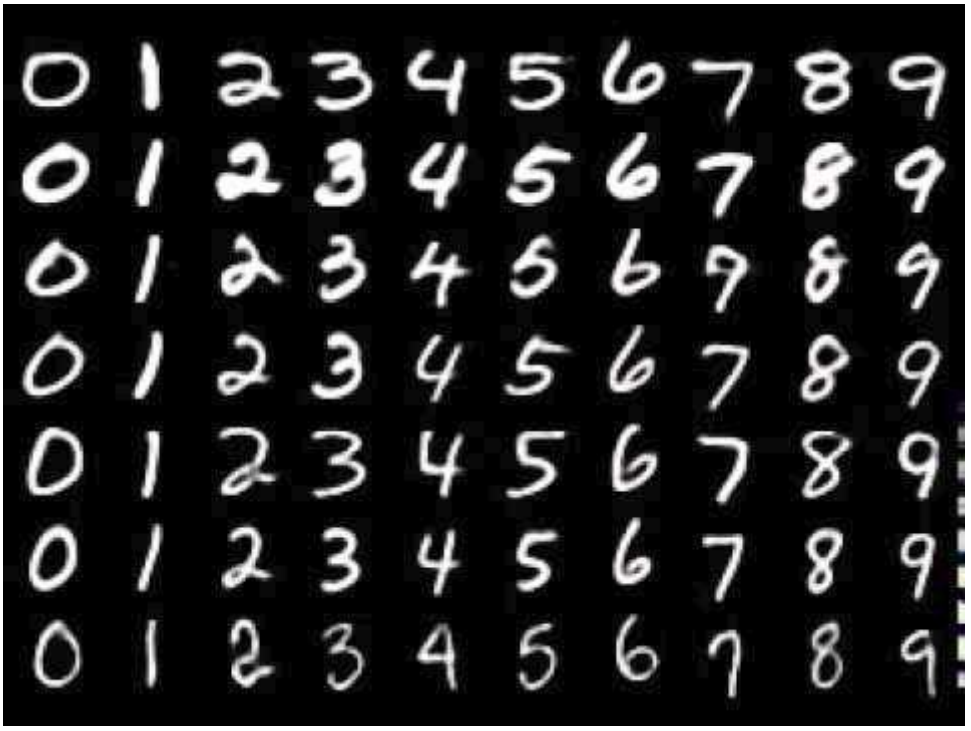
Test: proceso de validación para evaluar la “calidad” del aprendizaje conseguido.

Función de activación: dada una imagen de entrada, obtenemos la probabilidad de que pertenezca a cada uno de los dígitos. Sirve para quitar la linealidad entre capas de la red. A cada neurona se le asigna una función de activación que definirá la salida de la misma. En keras tenemos las siguientes funciones de activación: linear, softmax, relu, sigmoid, tanh, elu, selu, softplus, softsign, hard_sigmoid y exponential.

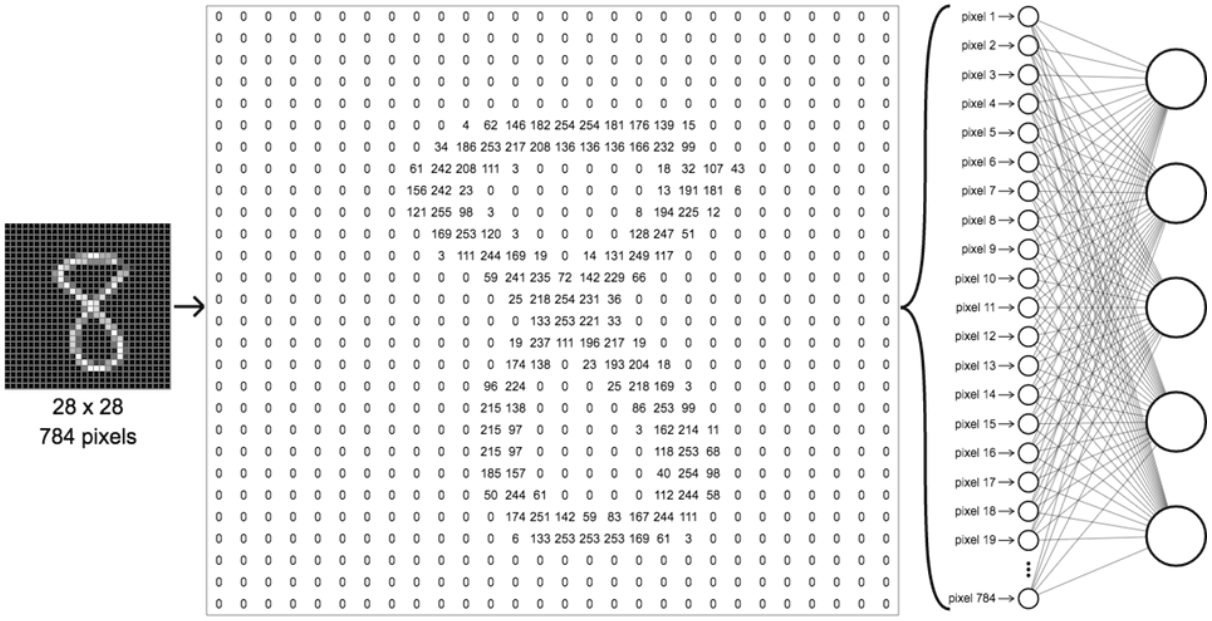
Features: variable de entrada para el entrenamiento de la red neuronal.
Loss: medida de error a la hora de catalogar un número.
Dense: capas conectadas, especificamos el número de neuronas en la capa y la función de activación.
Epochs: número fijo de iteraciones que se ejecutará el proceso de entrenamiento.
Batch_size: nº de instancias que se realizan antes de una actualización de pesos.

1.3 MNIST.

MNIST es un dataset o conjunto de datos que contiene imágenes de dígitos escritos a mano. Vamos a usar MNIST de la librería Keras, está formado por 60.000 ejemplos para entrenar y 10.000 para testear el aprendizaje.



Como podemos ver en la ilustración anterior, los datos almacenado son números del 0 al 9, cada número es una matriz de enteros 28*28 (pixeles), cada elemento de dicha matriz es un número entre el [0,255] de tipo *uint8* que representa la escala de grises, siendo 0 el blanco y 255 negro.



1.4 Entorno de trabajo.

- Sistema operativo Ubuntu 18,04 LTS
- Lenguaje de programación Python3.
- Pip como sistema de gestión de paquetes.
- Jupyter notebook, entorno de trabajo interactivo para el desarrollo de código.
- Librería Keras, es una API de redes neuronales de alto nivel, capaz de ejecutarse en TensorFlow, CNTK y Theano.
- Librería Scikit-learn, útil para Machine Learning en Python, es de código abierto, proporciona las diferentes librerías o paquetes:
- Plataforma TensorFlow para construir y entrenar redes neuronales.

- Instalación del entorno:

```
sudo apt install python3-pip
pip3 install tensorflow
pip3 install keras
pip3 install jupyter
pip3 install -U scikit-learn
```

- Lanzar jupyter desde la terminal: jupyter notebook

2. Red Neuronal.

2.1 Preprocesamiento.

Se han normalizado los datos, escalamos los valores de entrada de la red neuronal a valores de tipo *float32* en un rango [0,1].

Convertimos cada imagen (matriz de 28*28 pixeles), a un vector de 784 componentes.

Las etiquetas o labels para cada dato de entrada ([0-9]), se va a representar con un vector de 10 posiciones (diez números a representar), donde la posición correspondiente a la coincidencia de un número es un 1 y el resto un 0. Volcamos las etiquetas en un archivo llamado out.txt.

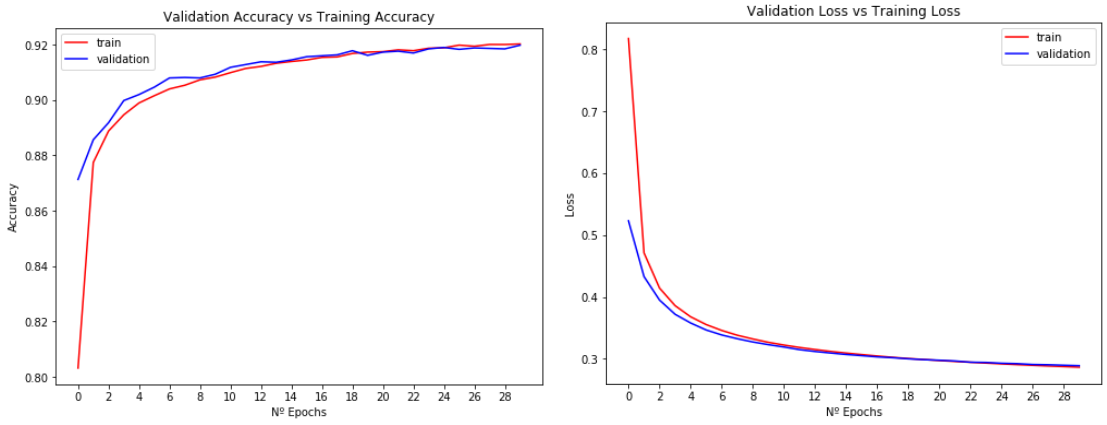
Dividimos el conjunto de entrenamiento en dos subconjuntos, es decir, reservamos un tanto por ciento del conjunto de entrenamiento para la validación.

2.2 Entrenamiento.

1ª Iteración:

Se ha empezado con una red lo más sencilla posible, con una sola capa, como entrada como es normal y para todas las redes que realicemos será el vector de componentes, como salida tendremos 10 neuronas y la función de activación será “softmax”. El algoritmo de entrenamiento usado es “sgd” (gradiente descendente), nº instancias 32, 5 iteraciones y función de pérdida “categorical_crossentropy”.

Vemos en los gráficos el siguiente resultado:



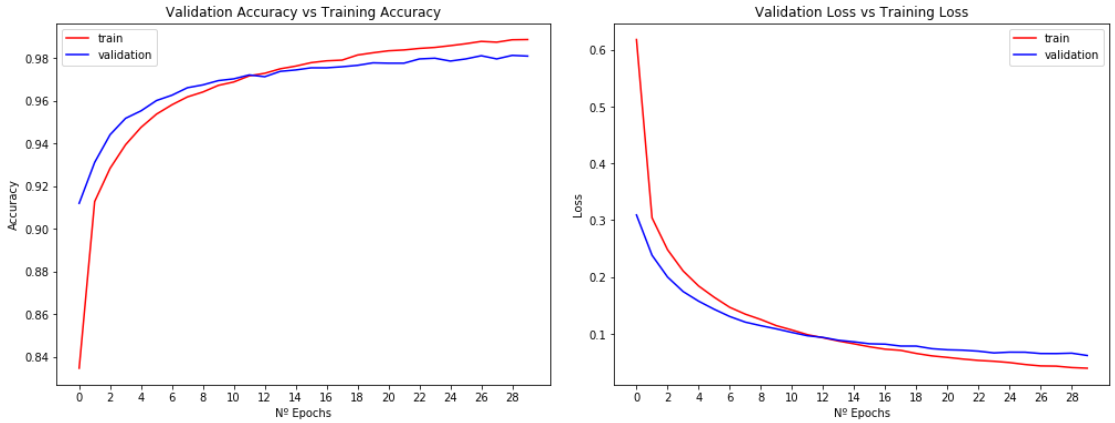
Con estos primeros datos ya obtenemos las siguientes tasas:

- Tasa de error sobre el conjunto de prueba: 7.929998636245728%
- Tasa de error sobre el conjunto de entrenamiento: 7.899999618530273%

2ª Iteración a destacar:

Hemos añadido dos capas a la red, con la primera capa de salida con 512 neuronas y la segunda con 256 neuronas, ambas de tipo“relu”, el tiempo pasa de 66 segundos a 365 segundos. Por otra parte la tasa de error mejora bastante:

- Tasa de error sobre el conjunto de prueba: 1.8800020217895508 %
- Tasa de error sobre el conjunto de entrenamiento: 0.7216691970825195 %



3ª Iteración:

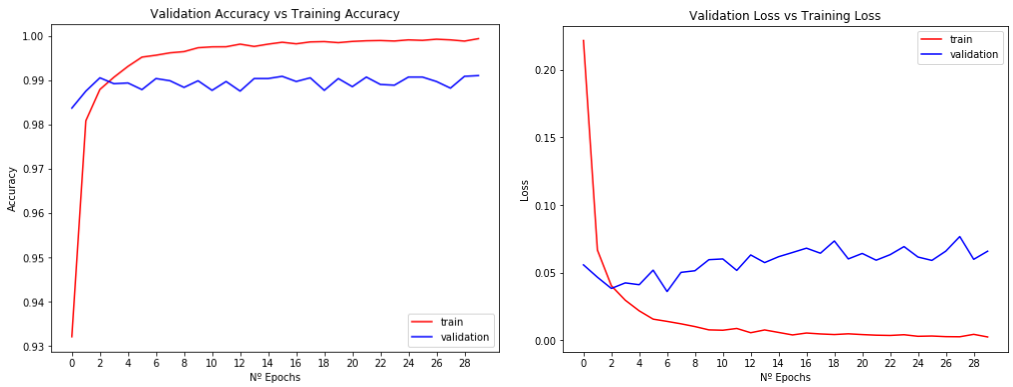
Incluimos una capa de convolución, ya no tenemos que convertir la imagen a un vector de 784 componentes, en su lugar añadimos una dimensión más. También añadimos “Flatten” para que las siguientes capas obtengan una secuencia de vectores.

Cambio al algoritmo de entrenamiento “adam”, aunque la mejora con los tiempos no compensa, vemos en las gráficas que las líneas de validación y training no se cruzan, por lo que nos permite un mejor resultado en gran cantidad de iteraciones, con 30 iteraciones conseguimos:

- Tasa de error sobre el conjunto de prueba: 1.50%
- Tasa de error sobre el conjunto de entrenamiento: 0.14%
- Tiempo de entrenamiento: 1131 segundos

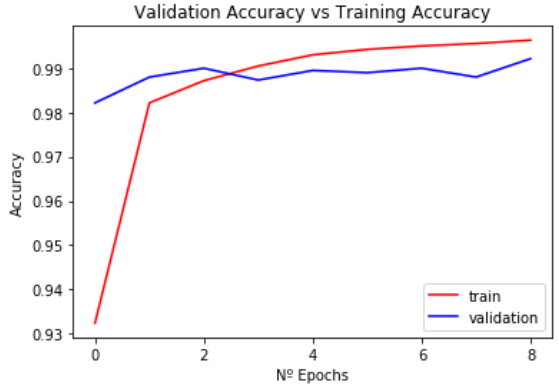
Siguientes iteraciones:

Le añadimos una capa de convolución y la tasa de error sobre el conjunto de prueba baja hasta 1.15%, el tiempo se dispara a 3488 segundos.



Incluimos “MaxPooling2D”, una capa de agrupación máxima para datos espaciales, bajamos a 9 iteraciones y probamos la parada temprana junto con ModelCheckpoint, no queremos pararlo antes de tiempo, por eso a la parada temprana le ponemos 200 iteraciones de paciencia, solo queremos ver cuando mejora y cual es el mejor tiempo guardado, ModelCheckpoint nos permite controlar el modelo guardando el mejor conjunto en un archivo. Obtenemos la mejor tasa de error hasta el momento:

- Algoritmo de entrenamiento usado: adam
- Valores de sus parámetros
- Nº instancias: 128
- Iteraciones del proceso de entrenamiento: 9
- Funcion de perdida para guiar el optimizador: categorical_crossentropy
- Tasa de error sobre el conjunto de prueba: 0.88%
- Tasa de error sobre el conjunto de entrenamiento: 0.17%
- Tiempo de entrenamiento: 861



Añadimos otras dos capas de convolución seguido de un MaxPooling2D, identicas a las anteriores, reducimos tanto el tiempo como la tasa de error de prueba, aunque la tasa de error de entrenamiento aumenta :

- Algoritmo de entrenamiento usado: adam
- Valores de sus parámetros
- Nº instancias: 128
- Iteraciones del proceso de entrenamiento: 10
- Funcion de perdida para guiar el optimizador: categorical_crossentropy
- Tasa de error sobre el conjunto de prueba: 0.75%
- Tasa de error sobre el conjunto de entrenamiento: 0.26%
- Tiempo de entrenamiento: 470 segundos

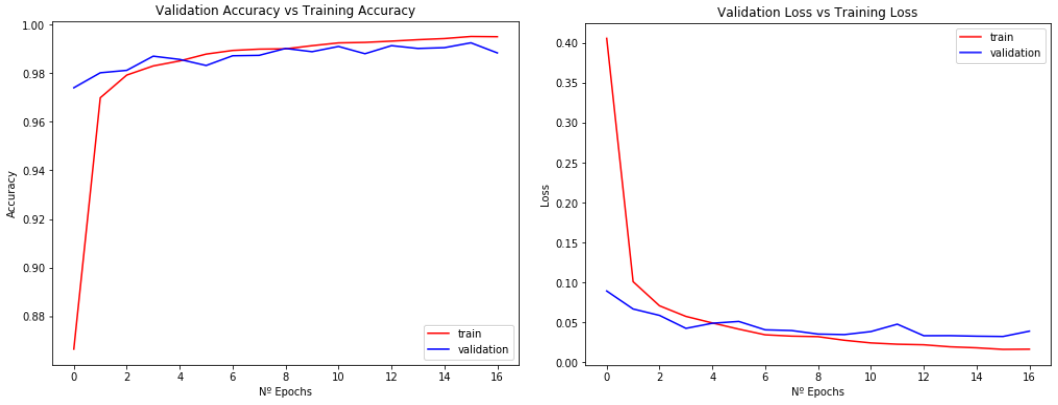
Añadimos una capa de convolución y otras dos posteriores seguidas de un MaxPooling, la topología de la red queda de la siguiente forma:

```
Conv2D(16, kernel_size=(3, 3),activation='relu')
Conv2D(32, (3, 3), activation='relu')
MaxPooling2D(pool_size=(2, 2))
Conv2D(16, (3, 3), activation='relu')
```

```
Conv2D(32, (3, 3), activation='relu')
Conv2D(64, (3, 3),padding='same', activation='relu')
MaxPooling2D(pool_size=(2, 2))
Conv2D(16, (3, 3), activation='relu')
Conv2D(32, (3, 3),padding='same', activation='relu')
MaxPooling2D(pool_size=(2, 2))
Flatten()
Dropout(0.1)
Dense(128, activation='relu')
Dense(10, activation='softmax')
```

El entrenamiento de la red da peores resultados, hacemos varias pruebas con todos los algoritmos de entrenamiento y la que mejor resultado da es el cambio del algoritmo de entrenamiento con RMSprop:

- Algoritmo de entrenamiento usado: RMSprop
- Valores de sus parámetros
- Nº instancias: 128
- Iteraciones del proceso de entrenamiento: 14
- Funcion de perdida para guiar el optimizador: categorical_crossentropy
- Tasa de error sobre el conjunto de prueba: 0.69%
- Tasa de error sobre el conjunto de entrenamiento: 0.24%
- Tiempo de entrenamiento: 836



Viendo que da mejores resultados el algoritmo de entrenamiento RMSprop y que la anterior red daba mejores resultados con adam, decidimos probar la anterior red con el algoritmo de entrenamiento RMSprop, los resultados mejoran notablemente:

- Tasa de error sobre el conjunto de prueba: 0.65%
- Tasa de error sobre el conjunto de entrenamiento: 0.28%
- Tiempo de entrenamiento: 531 segundos.

3. Conclusión.

Hemos ido entrenando nuestro conjunto de datos, desde lo más sencillo que hemos podido con una sola capa, hasta que hemos conseguido el mejor ajuste y tiempo posibles, hemos ido evitando en cada paso el sobreajuste y el tiempo excesivo, al final estos han sido los mejores resultados:

- Algoritmo de entrenamiento usado: RMSprop
- Valores de sus parámetros
- Nº instancias: 128
- Iteraciones del proceso de entrenamiento: 50
- Funcion de perdida para guiar el optimizador: categorical_crossentropy
- Tasa de error sobre el conjunto de prueba: 0.54%
- Tasa de error sobre el conjunto de entrenamiento: 0.16%
- Tiempo de entrenamiento: 779

Comentar que realmente sólo se han producido 12 iteraciones por la parada temprana, la implementación y sus resultados han sido los siguientes:

Model: "sequential_15"

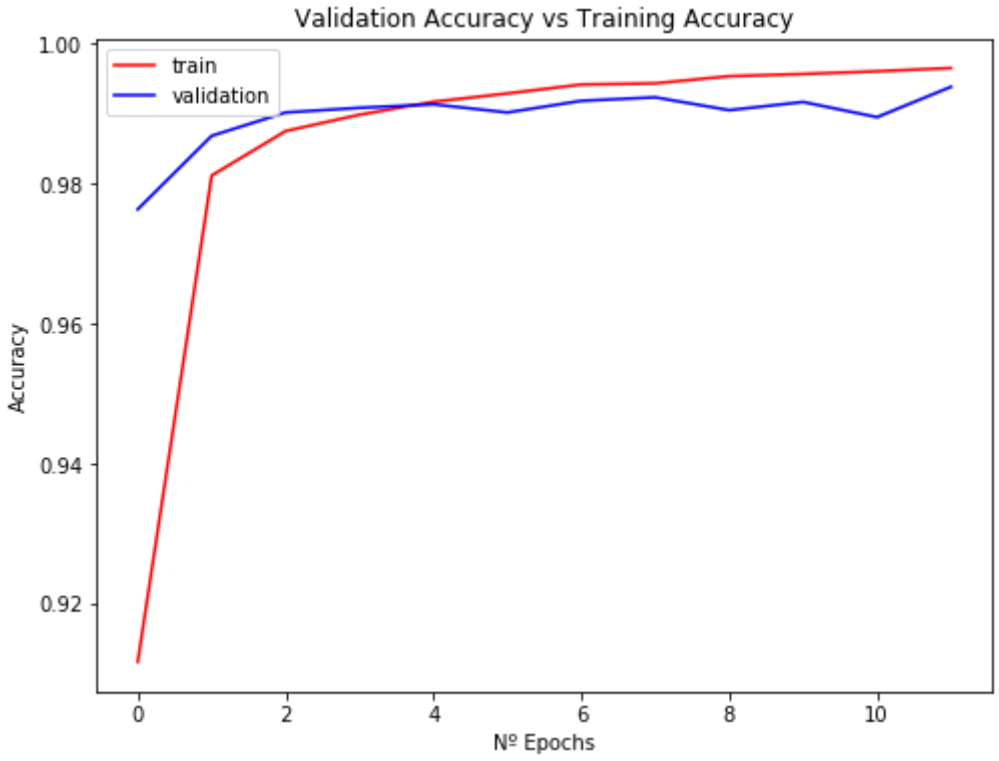
Layer (type)	Output Shape	Param #
conv2d_85 (Conv2D)	(None, 26, 26, 16)	160
conv2d_86 (Conv2D)	(None, 24, 24, 32)	4640
max_pooling2d_42 (MaxPooling	(None, 12, 12, 32)	0
dropout_14 (Dropout)	(None, 12, 12, 32)	0
conv2d_87 (Conv2D)	(None, 10, 10, 32)	9248
conv2d_88 (Conv2D)	(None, 10, 10, 64)	18496
max_pooling2d_43 (MaxPooling	(None, 5, 5, 64)	0
dropout_15 (Dropout)	(None, 5, 5, 64)	0
conv2d_89 (Conv2D)	(None, 3, 3, 64)	36928
conv2d_90 (Conv2D)	(None, 3, 3, 128)	73856
max_pooling2d_44 (MaxPooling	(None, 1, 1, 128)	0
flatten_14 (Flatten)	(None, 128)	0

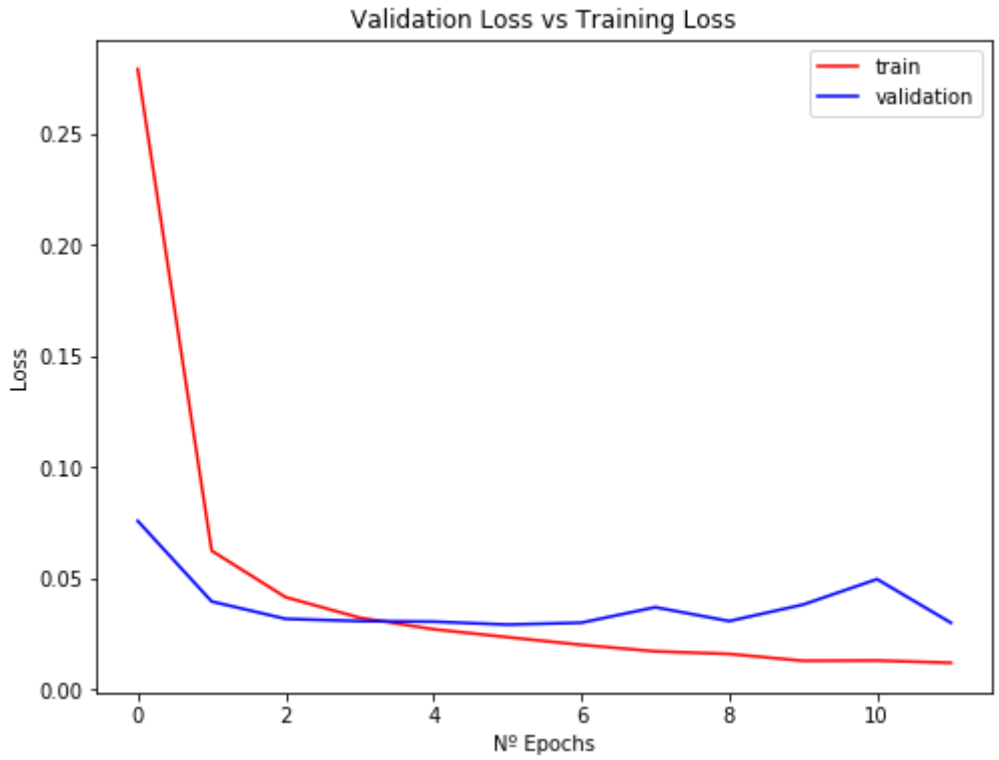
```
_________________________________________________________________
dropout_16 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_27 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_28 (Dense)             (None, 10)                1290
=================================================================
Total params: 161,130
Trainable params: 161,130
Non-trainable params: 0
_________________________________________________________________
 Entrenando datos
Train on 54000 samples, validate on 6000 samples
Epoch 1/50
54000/54000 [==============================] - 61s 1ms/step - loss: 0.2791 -
accuracy: 0.9117 - val_loss: 0.0759 - val_accuracy: 0.9763
Epoch 2/50
54000/54000 [==============================] - 65s 1ms/step - loss: 0.0625 -
accuracy: 0.9812 - val_loss: 0.0396 - val_accuracy: 0.9868
Epoch 3/50
54000/54000 [==============================] - 65s 1ms/step - loss: 0.0416 -
accuracy: 0.9875 - val_loss: 0.0318 - val_accuracy: 0.9902
Epoch 4/50
54000/54000 [==============================] - 65s 1ms/step - loss: 0.0324 -
accuracy: 0.9899 - val_loss: 0.0308 - val_accuracy: 0.9908
Epoch 5/50
54000/54000 [==============================] - 66s 1ms/step - loss: 0.0272 -
accuracy: 0.9917 - val_loss: 0.0306 - val_accuracy: 0.9913
Epoch 6/50
54000/54000 [==============================] - 65s 1ms/step - loss: 0.0236 -
accuracy: 0.9929 - val_loss: 0.0293 - val_accuracy: 0.9902
Epoch 7/50
54000/54000 [==============================] - 65s 1ms/step - loss: 0.0202 -
accuracy: 0.9942 - val_loss: 0.0301 - val_accuracy: 0.9918
Epoch 8/50
54000/54000 [==============================] - 65s 1ms/step - loss: 0.0172 -
accuracy: 0.9943 - val_loss: 0.0371 - val_accuracy: 0.9923
Epoch 9/50
54000/54000 [==============================] - 65s 1ms/step - loss: 0.0161 -
accuracy: 0.9954 - val_loss: 0.0308 - val_accuracy: 0.9905
Epoch 10/50
54000/54000 [==============================] - 65s 1ms/step - loss: 0.0130 -
accuracy: 0.9957 - val_loss: 0.0382 - val_accuracy: 0.9917
Epoch 11/50
54000/54000 [==============================] - 66s 1ms/step - loss: 0.0131 -
accuracy: 0.9961 - val_loss: 0.0497 - val_accuracy: 0.9895
Epoch 12/50
```

54000/54000 [==============================] - 66s 1ms/step - loss: 0.0121 - accuracy: 0.9965 - val_loss: 0.0301 - val_accuracy: 0.9938
Epoch 00012: early stopping

Tiempo de entrenamiento:
779.3884460926056
10000/10000 [==============================] - 4s 355us/step
Test accuracy: 0.9945999979972839
Test loss: 0.022537097497815
60000/60000 [==============================] - 21s 352us/step
Train accuracy: 0.998449981212616
Train loss: 0.006073477429414671
Algoritmo de entrenamiento usado: RMSprop
Valores de sus parámetros
Nº instancias: 128
Iteraciones del proceso de entrenamiento: 50
Funcion de perdida para guiar el optimizador: categorical_crossentropy
Tasa de error sobre el conjunto de prueba: 0.54%
Tasa de error sobre el conjunto de entrenamiento: 0.16%
Tiempo de entrenamiento: 779

En las imágenes que se muestran a continuación podemos ver una comparativa de los datos de validación y entrenamiento del resultado final.





4. Bibliografía.

https://keras.io
https://elvex.ugr.es/decsai/deep-learning/
https://elvex.ugr.es/decsai/computational-intelligence/#ec

5. Anexos

5.1. Primer código

```
#!/usr/bin/env python

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.utils import to_categorical
from keras.layers.core import Dense, Activation
import numpy as np
import matplotlib.pyplot as plt
import time
from sklearn.model_selection import train_test_split

# nº de instancias que se realizan antes de una actualización de pesos
batch_size = 32
```

```
# nº iteraciones que se ejecuta el proceso de entrenamiento
epochs = 30
# % reservado para datos de validación
test_size = 0.1
# mezcla los datos en cada época
shuffle=True

# medida de error
loss="categorical_crossentropy"
# Optimizacion
optimizer="sgd"

#precarga de los datos de mnist que contiene keras
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# normalización de los datos de entrada
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

x_train /= 255
x_test /= 255

# convertir imagen a un vector de 784 componentes.
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)

#categorial
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)

#dibido conjunto de entrenamiento y validación
X2_train, X_val, Y2_train, Y_val = train_test_split(x_train, y_train, test_size = test_size,
random_state=42)

#clase secuencial para definición de modelo
model = Sequential()
model.add(Dense(10, activation='softmax', input_shape=(784,)))

# comprobar architectura de nuestro modelo.
model.summary()
model.compile(loss=loss,
              optimizer=optimizer,
              metrics=['accuracy'])

# cuento el tiempo
start = time.time()
```

```
print(" Entrenando datos ")
snn = model.fit(x=X2_train, y=Y2_train,
                batch_size=batch_size,
                epochs=epochs,
                verbose=1,
                validation_data=X_val, Y_val),
                shuffle=shuffle)

# Tiempo de entrenamiento
end = time.time()
print(" Tiempo de entrenamiento: ")
print(end - start)

#evaluación del modelo
test_loss, test_acc = model.evaluate(x_test, y_test)

print('Test accuracy:', test_acc)
print('Test loss', test_loss)

train_loss, train_acc = model.evaluate(x_train, y_train)
print('Test accuracy:', train_acc)
print('Test loss', train_loss)
```

5.1. Código final

```
#!/usr/bin/env python

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.utils import to_categorical
from keras.layers.core import Dense, Activation, Dropout
from keras.layers import Conv2D, MaxPooling2D, Flatten
import numpy as np
import matplotlib.pyplot as plt
import time
from sklearn.model_selection import train_test_split

# nº de instancias que se realizan antes de una actualización de pesos
batch_size = 128
# nº iteraciones que se ejecuta el proceso de entrenamiento
epochs = 50
# % reservado para datos de validación
```

```
test_size = 0.1
# mezcla los datos en cada época
shuffle=True

# medida de error
loss="categorical_crossentropy"
# Optimizacion
optimizer="RMSprop"

#precarga de los datos de mnist que contiene keras
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# añadimos una dimension para convolucion.
x_train = x_train.reshape(x_train.shape+(1,))
x_test = x_test.reshape(x_test.shape+(1,))

# normalización de los datos de entrada
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

x_train /= 255
x_test /= 255

#categorial
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)

#dibido conjunto de entrenamiento y validación
X2_train, X_val, Y2_train, Y_val = train_test_split(x_train, y_train, test_size = test_size,
random_state=42)

#clase secuencial para definición de modelo
model = Sequential()

model.add(Conv2D(16, kernel_size=(3, 3),activation='relu',
                 input_shape=x_train.shape[1:]))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.1))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.1))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
model.add(Flatten())
model.add(Dropout(0.1))
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))


# comprobar architectura de nuestro modelo.
model.summary()
model.compile(loss=loss,
              optimizer=optimizer,
              metrics='accuracy'])

earlystop = EarlyStopping(monitor='val_loss',
                          min_delta=0.001,
                          verbose=1,
                          patience=6,
                          mode='auto')

# cuento el tiempo
start = time.time()

print(" Entrenando datos ")
snn = model.fit(x=X2_train, y=Y2_train,
                batch_size=batch_size,
                epochs=epochs,
                verbose=1,
                validation_data=X_val, Y_val),
                shuffle=shuffle,
                callbacks=[earlystop])

# Tiempo de entrenamiento
end = time.time()
print(" Tiempo de entrenamiento: ")
print(end - start)

#evaluación del modelo
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
print('Test loss', test_loss)

train_loss, train_acc = model.evaluate(x_train, y_train)
print('Test accuracy:', train_acc)
print('Test loss', train_loss)
```

5.3 Anexos código de obtención datos de entrega

```
#Algoritmo de entrenamiento
print('Algoritmo de entrenamiento usado: ', optimizer)
#valores de sus parámetros
print('Valores de sus parámetros')
print('Nº instancias: ', batch_size)
print('Iteraciones del proceso de entrenamiento: ', epochs)
print('Funcion de perdida para guiar el optimizador: ', loss)

#Tasa de error sobre el conjunto de prueba (%)
print('Tasa de error sobre el conjunto de prueba: ',1-test_acc)
#Tasa de error sobre el conjunto de entrenamiento (%)
print('Tasa de error sobre el conjunto de entrenamiento: ',1-train_acc)

print(" Tiempo de entrenamiento: ")
print(end - start)
```

```
# grafica comparativa de accuary validación y entrenamiento
plt.figure(0)
plt.plot(snn.history['accuracy'],'r')
plt.plot(snn.history['val_accuracy'],'b')
plt.xticks(np.arange(0, epochs, 2.0))
plt.rcParams['figure.figsize'] = (8, 6)
plt.xlabel("Nº Epochs")
plt.ylabel("Accuracy")
plt.title("Validation Accuracy vs Training Accuracy")
plt.legend(['train','validation'])

# grafica comparativa de loss validación y entrenamiento
plt.figure(1)
plt.plot(snn.history['loss'],'r')
plt.plot(snn.history['val_loss'],'b')
plt.xticks(np.arange(0, epochs, 2.0))
plt.rcParams['figure.figsize'] = (8, 6)
plt.xlabel("Nº Epochs")
plt.ylabel("Loss")
plt.title("Validation Loss vs Training Loss")
plt.legend(['train','validation'])

plt.show()
```

```
# Etiquetas asignadas a los casos del conjunto de prueba
result = model.predict(x_test)
class_result=np.argmax(result,axis=-1)

f = open("out.txt","w")
for i in range(len(class_result)):
    f.write(str(class_result[i]))
f.close()
```

```
# Comprobando la tasa de error
metrics = model.evaluate(x_test, y_test)

for i in range(len(metrics)):
    print( model.metrics_names[i]," = ",metrics[i])

result = model.predict(x_test)
class_result=np.argmax(result,axis=-1)
errores = 0

for i in range(len(class_result)):
    if class_result[i] != np.argmax(y_test[i]):
        print(i)
        errores+=1

print ("Errores ",errores)
```