Clasificación de Imágenes. Redes Neuronales Convolucionales.

**Ejercicio 1.** Convoluciones 1D En el archivo “Práctica 4 - Convoluciones 1D.ipynb*”* encontrará un notebook donde deberás implementar algunas variantes de las convoluciones en 1D, es decir, sobre un vector.

**Ejercicio 2.** Convoluciones 2D En el archivo “Práctica 4 - Convoluciones 2D.ipynb*”* encontrará un notebook donde deberás implementar algunas variantes de las convoluciones en 2D, es decir, sobre una matriz o imagen.

**Ejercicio 3. MNIST** es una base de datos que contiene 70 mil imágenes en escala de grises de dígitos escritos a mano. Genere un modelo que permita clasificar las 10 clases.

**a) Carga de datos**

El código para descargar el dataset ya se encuentra en Keras y es posible cargarlo en memoria del siguiente modo:

from keras.datasets import mnist

(X\_train, Y\_train), (X\_test, Y\_test) = mnist.load\_data()

Los arreglos X\_train y X\_test poseen las imágenes del dataset y cada uno tiene un shape: (N, H,W), donde N es la cantidad de ejemplos, H el alto de la imagen y W el ancho. En el caso de MNIST tanto H como W valen 28.

**b) Visualización**

El primer paso para cerciorarse de que un conjunto de datos de imágenes está cargado correctamente es visualizar las mismas.

Para visualizar las imágenes, puede utilizar la función plt.imshow(). Por ejemplo, para ver la primera imagen del conjunto de entrenamiento, puede ejecutar:

plt.imshow(X\_train[0, :,:])

Para ver la tercera imagen del conjunto de prueba:

plt.imshow(X\_test[2, :,:])

Implemente un código que visualice 8 imágenes de cada clase en una grilla de 8x10. Utiliza la función [subplots](https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplots.html) de *matplotlib.pyplot* para generar la grilla.

**c) Aplanamiento**

Para clasificar con una red neuronal con capas del tipo x\*W+b (clase Dense en Keras), el formato de imagen (N,H,W) no sirve. Por ende los ejemplos deben adaptarse al formato (N,V), donde V es la cantidad de variables, que en este caso sería V=H\*W=28\*28= 784. La capa Flatten realiza esta operación por nosotros:

model.add(Flatten(input\_shape=(28,28)))

La operación Flatten se refiere a aplanar un arreglo n-dimensional, es decir, desarmar las estructuras de sus dimensiones y quedarse con todos los valores, pero ahora en un vector de una sola dimensión. En este caso, solo se desarman las dimensiones 1 y 2, la dimensión 0 queda porque es la que separa los ejemplos. Como no se pierde ningún valor, sólo se reordenan, la nueva dimensión V tiene tamaño 784=H\*V=28\*28.

*Nota: La API de Keras es inconsistente, y utiliza el parámetro input\_shape en la clase Flatten, mientras que en el resto de capas se utiliza el parámetro input\_dim. Recordamos que este parámetro sólo es necesario en la primera capa para especificar el tamaño de la entrada.*

**d) Entrenamiento del modelo**

Entrene un modelo para clasificar las imágenes, utilizando una *softmax* en la capa de salida y la entropía cruzada como función de error. Mida el error y el accuracy en el conjunto de test (y el de train). Compute la matriz de confusión, pero antes de mirarla ¿qué pares de clases le parece que van a confundirse más?.

**e) Normalización de las imágenes**

Los pixeles de las imágenes están codificados en el rango 0-255. Es más beneficioso para el entrenamiento de la red que estén normalizadas, con media 0 y varianza 1. La normalización debe realizarse a nivel de pixel, es decir, se debe calcular la media de todos los pixeles de todas las imágenes, luego la varianza de todos los pixeles de todas las imágenes y luego realizar la normalización. En este caso tanto la media como la varianza son escalares, ya que cada pixel está codificado por un solo valor. Implementa dicha normalización utilizando numpy, antes de las etapas de entrenamiento y clasificación.

**f) CNN en MNIST**

Diseñe un modelo de Redes Neuronales Convolucionales (CNN) para clasificar el conjunto de datos MNIST. Pruebe varios modelos distintos, variando los siguientes hiperparámetros (Recuerde evaluar en el conjunto de testing):

● Cantidad de capas Conv

● Cantidad de feature maps (kernels)

● Stride, padding y kernel\_size de las convoluciones

● Funciones de activación

● Uso de capas Max Pooling.

● Cantidad de neuronas ocultas

**Ejercicio 4.** Optimización de Hiperparámetros en MNIST. En este ejercicio vamos a probar variando los hiperparámetros de la red de forma sistemática para encontrar una arquitectura que se ajuste mejor al problema, es decir, la arquitectura más simple que obtenga un mejor accuracy en test. Comience utilizando una red de tres capas con la siguiente arquitectura:

Capa Convolucional (Conv2D) - Capa Max Pooling (MaxPooling2D) - Capa de (Regresión Logística (Dense)

Varíe sistemáticamente los parámetros de la tabla y tome nota del accuracy obtenido en train y test y la cantidad de épocas hasta que converge el modelo. Podemos considerar que el modelo converge si el accuracy de test alcanza su máximo y deja de cambiar por 5 épocas. Para eso, puede utilizar la clase EarlyStopping de Keras:

es = EarlyStopping(monitor='val\_acc', mode='min', patience=5, verbose=1)

E incluir el objeto como callback en el método fit:

history = model.fit(trainX, trainy, validation\_split=0.1, epochs=4000, verbose=0, callbacks=[es])

El parámetro *patience* le indica a EarlyStopping cuantas iteraciones sin cambio esperar.

Para realizar los experimentos, NO modifique manualmente los hiperparámetros cada vez que quiera probar una nueva combinación. En lugar de eso, escriba un script principal que:

1. Genere las combinaciones de hiperparámetros a probar
2. Por cada combinación, llame a otro script o función para entrenar y probar el modelo
3. Anote los resultados de cada modelo (junto con los hiperparámetros utilizados) en una tabla. Recomendamos guardar los resultados en un archivo .csv para luego inspeccionarlos manualmente.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Feature Maps en capa Conv2D** | **Tam. del kernel (kernel\_size)** | **Stride en capa Conv2D** | **Max Pooling**  **(MP)** | **Stride en MP** | **Función de activación** | **Épocas** | **Acc. en Train** | **Acc. en Test** |
| 4 | 3x3 | 1 | Si | 2 | ReLU |  |  |  |
| 16 | 3x3 | 1 | Si | 2 | ReLU |  |  |  |
| 64 | 3x3 | 1 | Si | 2 | ReLU |  |  |  |
| 128 | 3x3 | 1 | Si | 2 | ReLU |  |  |  |
| 4 | 7x7 | 1 | Si | 2 | ReLU |  |  |  |
| 16 | 7x7 | 1 | Si | 2 | ReLU |  |  |  |
| 64 | 7x7 | 1 | Si | 2 | ReLU |  |  |  |
| 128 | 7x7 | 1 | Si | 2 | ReLU |  |  |  |
| 64 | 3x3 | 2 | No | - | ReLU |  |  |  |
| 64 | 3x3 | 3 | No | - | ReLU |  |  |  |
| 64 | 3x3 | 1 | Si | 2 | TanH |  |  |  |
| 64 | 3x3 | 1 | Si | 2 | Sigmoide |  |  |  |

**Ejercicio 5. CIFAR10.** Este conjunto de datos posee imágenes de 10 clases de la vida cotidiana como diferentes animales y vehículos.

**a) Carga de datos**

from keras.datasets import cifar10

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

**b) Visualización**

Para visualizar las imágenes, también puede utilizar la función plt.imshow(), pero ahora debe tener en cuenta la dimensión extra:

plt.imshow(x\_train[0,:,:,:]) #se incluye la dimensión de canales

**c) Aplanamiento**

Las imágenes de CIFAR10 son a color, por ende las dimensiones de x\_train y x\_test son del tipo (N,H,W,C), donde C es la cantidad de canales (3 para imágenes a color). En el caso de CIFAR10, las imágenes son de tamaño 32x32, por ende el shape de estos vectores es (N,32,32,3). En este caso también tenemos que utilizar la capa Flatten al principio del modelo, pero ahora con el nuevo tamaño de entrada:

model.add(Flatten(input\_shape=(32,32,3)))

**d) Entrenamiento del modelo**

Entrene un modelo para clasificar las imágenes, utilizando una softmax en la capa de salida y la entropía cruzada como función de error. Mida el error y el accuracy en el conjunto de test (y el de train). Compute la matriz de confusión, pero antes de mirarla ¿qué pares de clases te parece que van a confundirse más? Recuerde normalizar los datos con anterioridad. En este caso, la normalización debe realizarse por cada canal.

**e) CNN en CIFAR10**

Repita el mismo procedimiento que para MNIST para el conjunto de datos CIFAR10. El entrenamiento para CIFAR10 generalmente tarda más tiempo que para MNIST debido a que las imágenes tienen aproximadamente tres veces la dimensionalidad y además mayor complejidad. Por ende, pruebe sólo algunas arquitecturas distintas.