Universidad Nacional de Rosario

Facultad de ciencias exactas, ingeniería y agrimensura

Introducción a la Inteligencia Artificial

Trabajo práctico 5

Alumno:

Petruskevicius Ignacio - Lucio Trincheri

Junio 2022

IIA - Trabajo práctico 5

Petruskevicius Ignacio, Lucio Trincheri LCC UNR FCEIA 17/06/2022

1. Data set

El conjunto de datos **load-digits** de Scikit-Learn consta de imágenes con una resolución 8 x 8 píxeles donde cada uno puede adoptar 256 tonos de grises. En estas imágenes se encuentran impresos números del 0 al 9.

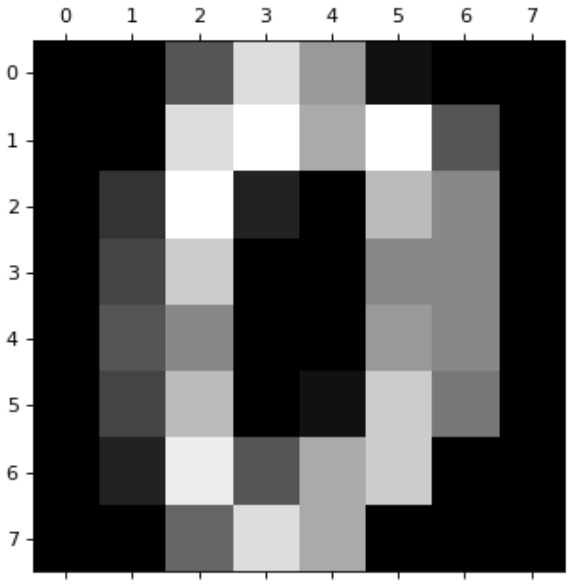

Figura 1: Ejemplo de imagen del conjunto de datos (representa un 0)

Esto se traduce a que el **data set** cuenta con 1797 imágenes, con al rededor de 180 instancias por cada clase que corresponden a cada dígito posible del 0 al 9 (10 clases en total). A su vez, cada imagen al poseer una resolución de 8 x 8 se trata de un vector de entrada de 64 parámetros, con valores del 0 al 255 representando el tono de gris.

Cada algoritmo divide las instancias en los conjuntos que requiera, ya sea conjunto de entrenamiento, de validación o de evaluación.

2. Árbol de decisión

2.1. Valores por defecto

En esta sección analizamos el árbol obtenido luego del entrenamiento con los valores por defecto, los mismos son:

- criterion: 'gini'
- max-depth: None

- min-samples-split: 2
- min-samples-leaf: 1
- max-leaf-nodes: None

Lo primero que notamos es que el árbol es extenso con mucha cantidad de hojas y nodos, con una profundidad de 13 niveles. Esto se debe a que el valor del parámetro que define cuantas instancias se necesitan para dividir el conjunto es muy bajo (min-samples-split: 2), generando muchas divisiones con pocas instancias cada una, lo que en conjunto con la falta de limitación en cuanto a la máxima profundidad (max-depth: "None"), permite dividir cada conjunto hasta hojas con una sola instancia (min-samples-leaf: 1). Esto causa un sobre-entrenamiento, el cual lo podemos observar dado que los valores de exactitud sobre el conjunto de entrenamiento llega a ser 1 y en el de datos solo a 0.83.

2.2. Modificaciones

Con el objetivo de ver como los cambios en los hiper-parámetros afectan los valores de exactitud, construimos una función que varia los mismos entre rangos coherentes y entrena un árbol. Luego se calculan los valores de exactitud tanto sobre el conjunto de entrenamiento como en el de evaluación y la diferencia entre los mismos (delta). De esta manera podemos tener una gran cantidad de configuraciones, donde podemos observar como los hiper-parámetros modifican la exactitud, y de ahí seleccionar bajo distintos criterios cuales son convenientes. La función nuestra en particular varía los siguientes parámetros:

Hiper-parámetro	Rango
Max depth	$[6, 14]$
Min samples split	$[2, 10]$
Min samples leaf	$[1, 5]$
Max leaf nodes	[15,Max depth·6]

Vale aclarar que **Max leaf nodes** está definido en base a la máxima profundidad para que el tiempo de ejecución no sea muy largo.

En particular, intentamos buscar parámetros que cumplan con una exactitud sobre el conjunto de testeo mayor a 0.8 y una diferencia con respecto al conjunto de entrenamiento menor al 5 %. No encontramos ninguna configuración que satisfaga ese requerimiento, pero si algunos casos que se le acercan. A continuación mostraremos un resumen de los resultados obtenidos y que valores de los parámetros se ajustan mejor.

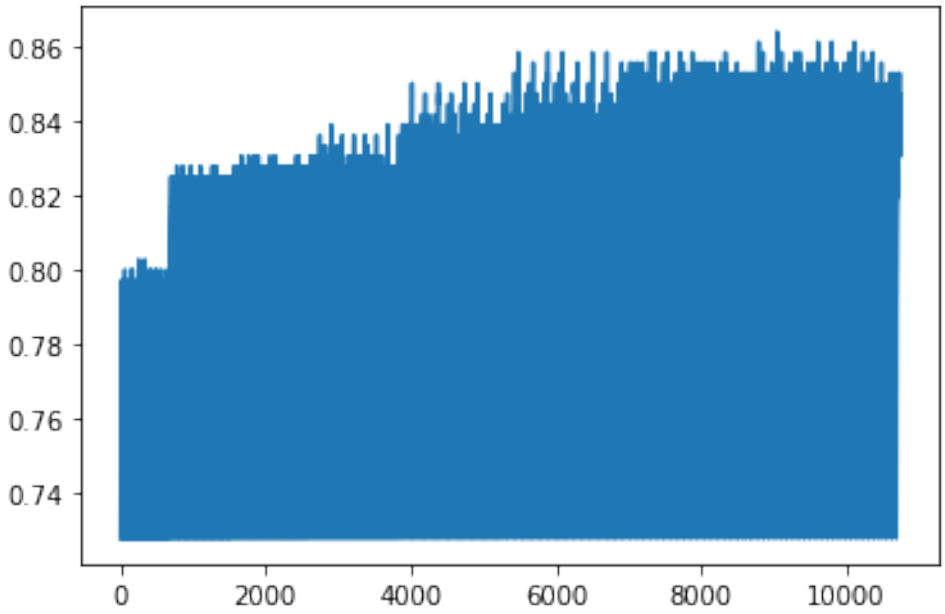

Figura 2: Exactitud sobre el conjunto de evaluación

Vemos que a partir de una cierta máxima profundidad, la exactitud tiende a estancarse excepto por aislados casos. Esto quiere decir que un buen resultado rondará esta profundidad, puesto que al crecer mucho se puede generar sobre-entrenamiento como veremos adelante.

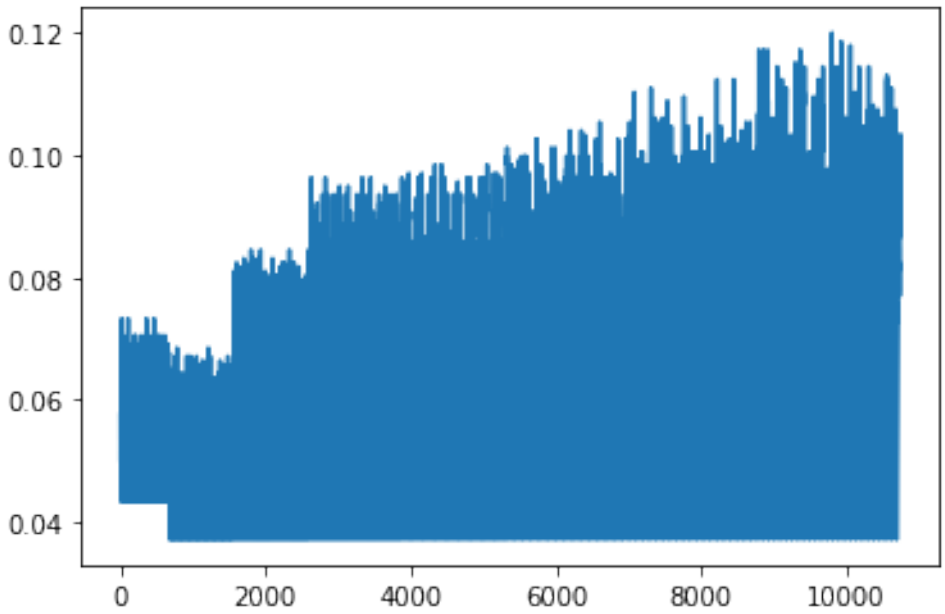

Figura 3: Delta exactitud entre los conjuntos de evaluación y entrenamiento

Efectivamente se puede observar como luego de que la máxima profundidad alcance el valor 8, la diferencia crece y por lo tanto se comienzan a ver casos de sobre-entrenamiento.

Teniendo esto claro, comenzamos a buscar los mejores resultados con el siguiente criterio, seleccionar los valores de hiper-parámetros que generan mayor exactitud pero a su vez menor delta. Así, creamos una función que evalúa que tan bueno es un árbol mediante la ponderación de la exactitud sobre el conjunto de evaluación y el delta obtenido en ese árbol. Con ella obtuvimos estas configuraciones:

Exactitud	Delta	Max depth	Min sample split	Min sample leaf	Max leaf nodes
0.8194444444444444	0.04972744142890284	13	9	3	32
0.8138888888888889	0.04832405474367896	7	9	4	30
0.8055555555555556	0.04832405474367896	13	9	4	26
0.8111111111111111	0.04971004407330082	13	9	4	29
0.8083333333333333	0.04900835073068888	13	9	4	28
0.8083333333333333	0.049704244954766885	7	9	4	29
0.8	0.04272790535838544	13	9	4	25
0.7916666666666666	0.03714335421016013	13	9	4	21
0.7944444444444444	0.04062862444908377	13	9	4	23
0.7944444444444444	0.04202041289723968	13	9	4	24

Y sus gráficas correspondientes:

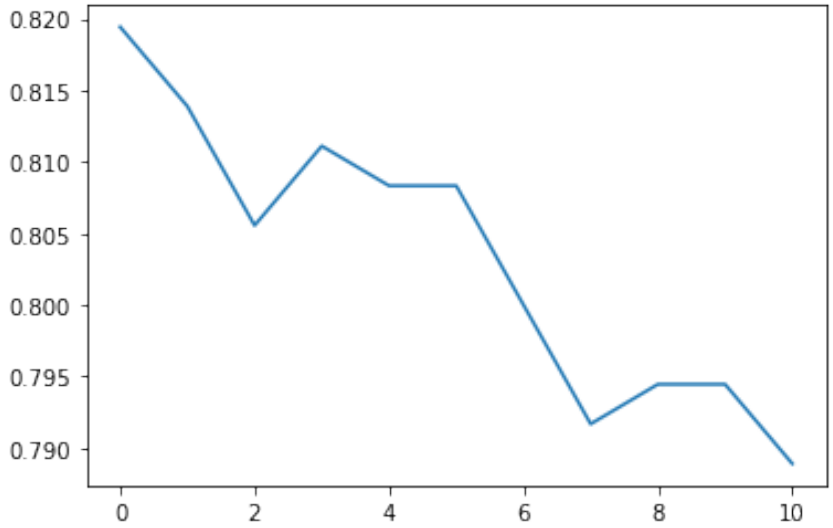

Figura 4: Exactitud de los mejores árboles sobre el conjunto de evaluación. x = nro árbol, y = exactitud

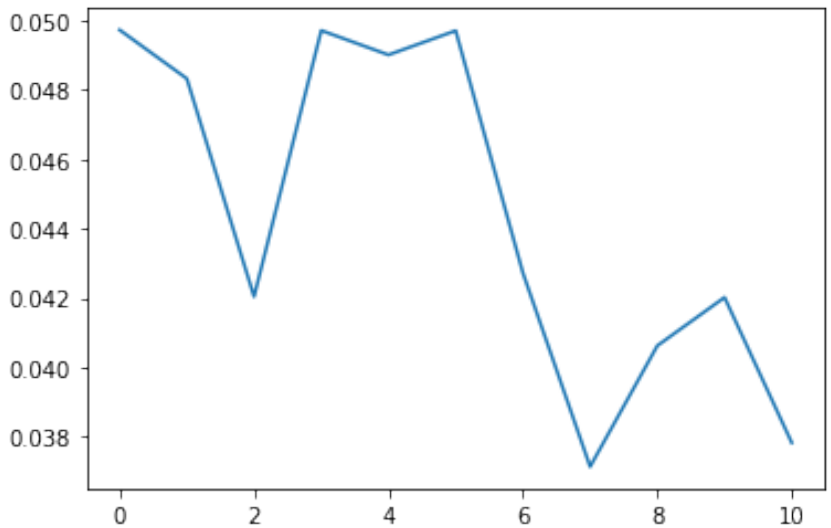

Figura 5: Delta exactitud de los mejores árboles entre los conjuntos de evaluación y entrenamiento

Cabe aclarar que aunque en este caso obtuvimos árboles que cumplan con una exactitud sobre el conjunto de evaluación mayor al 80 % y un delta entre conjuntos menor al 5 %, la función de scikit-learn que divide los valores en los conjuntos de entrenamiento y de evaluación es aleatoria. Esto conlleva que cada ejecución que divida el conjunto de instancias de entrada pueda llevar a situaciones donde sea posible una exactitud menor y un delta mayor a los pedidos (y esto sucedió entre las veces que fuimos trabajando sobre el dataset).

2.3. Resultados

Teniendo en cuenta que los arboles de decisión solo realizan divisiones lineales del dominio, y dada la complejidad del problema concluimos que los resultados a los cuales llegamos son muy buenos. Además, esto viene complementado por el bajo tiempo de entrenamiento y de evaluación de nuevas instancias (en el orden de los segundos). Por otro lado, este tipo de herramientas no permiten aprovechar completamente una gran cantidad de casos de entrenamiento, lo cual prevemos que será una ventaja interesante para las redes neuronales. Algunas comparaciones con las redes se realizaron al final del informe.

3. Redes neuronales

3.1. Hiper-parámetros por defecto

Analizaremos los resultados obtenidos con los hiper-parámetros por defecto tanto utilizando la librería **Scikit-Learn** y **TensorFlow**.

3.1.1. Scikit-Learn

Se utilizan 2 capas ocultas con función de activación **ReLU**, un "learning-rate" de 0.0001 y una cantidad máxima de iteraciones igual a 500.

Con estos valores se obtiene una **exactitud** sobre el conjunto de entrenamiento de 0.9234516353514266 y de 0.9 sobre el conjunto de evaluación.

3.1.2. TensorFlow

En este caso también se utilizan 2 hidden layers con **ReLU**, "learning-rate" de 0.0001 pero una cantidad máxima de épocas igual a 400.

Y obtuvimos sobre el conjunto de entrenamiento una exactitud de 0.9965205192565918 y sobre el conjunto de evaluación una exactitud de 0.9388889074325562.

3.1.3. Conclusiones

En ambos casos se obtienen muy buenos resultados para valores por defecto, ampliamente superiores a su contra parte del árbol de decisión (teniendo en cuenta que los valores de la red neuronal todavía no fueron modificados), por lo que entendemos que son satisfactorios. Además el ratio entre el tiempo de entrenamiento y la exactitud es bastante bueno teniendo en cuenta la exactitud obtenida.

3.2. Variando el “learning-rate”

Para variar el “learning-rate” se diseño una simple función que crea y entrena distintas redes con los valores pedidos.

A continuación se encuentran las diferentes gráficas que representan los valores de exactitud variando el valor de “learning-rate” al entrenar la red (esto fue realizado con la librería TensorFlow). En el epígrafe se encuentran los valores: “learning-rate”, exactitud máxima y la época donde se obtuvo esa exactitud. Luego se explicará como el valor de “learning-rate” cambia el comportamiento del modelo obtenido.

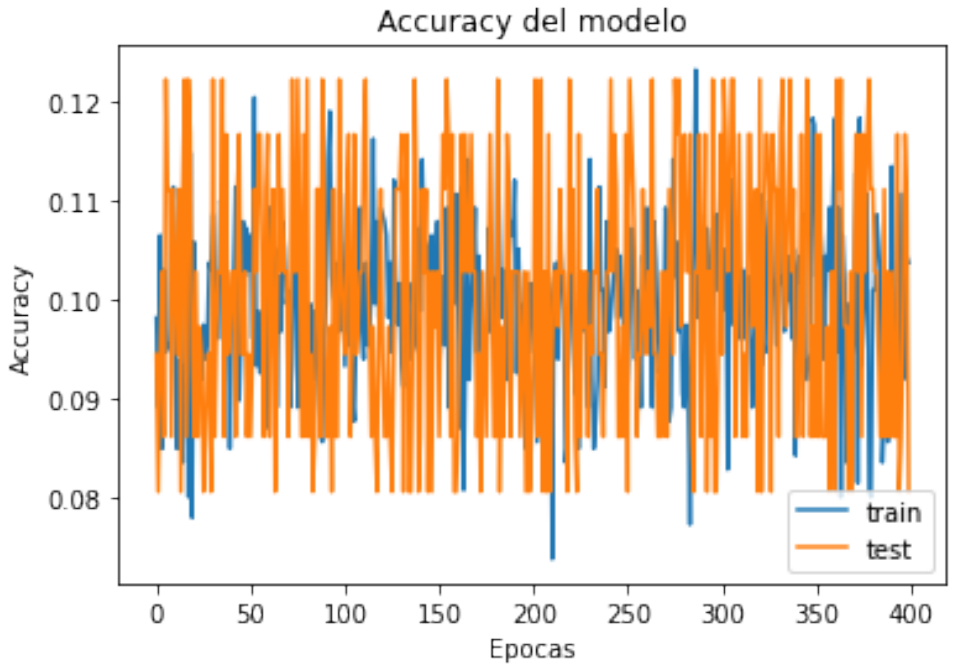

Figura 6: learning-rate: 10, Máxima exactitud: 0.12222222238779068, Época de máxima exactitud: 378

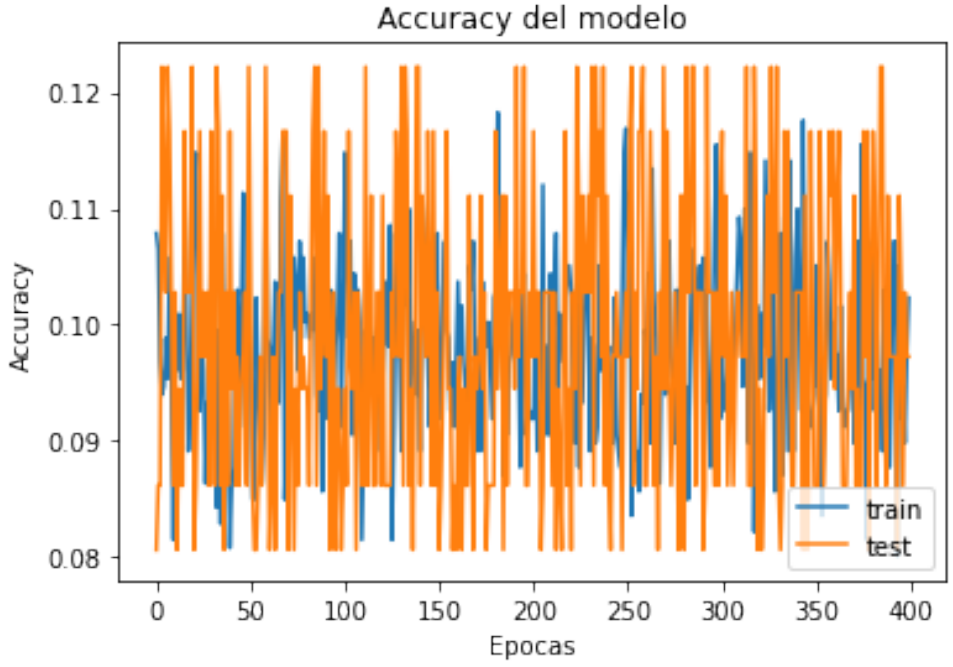

Figura 7: learning-rate: 1, Máxima exactitud: 0.12222222238779068, Época de máxima exactitud: 385

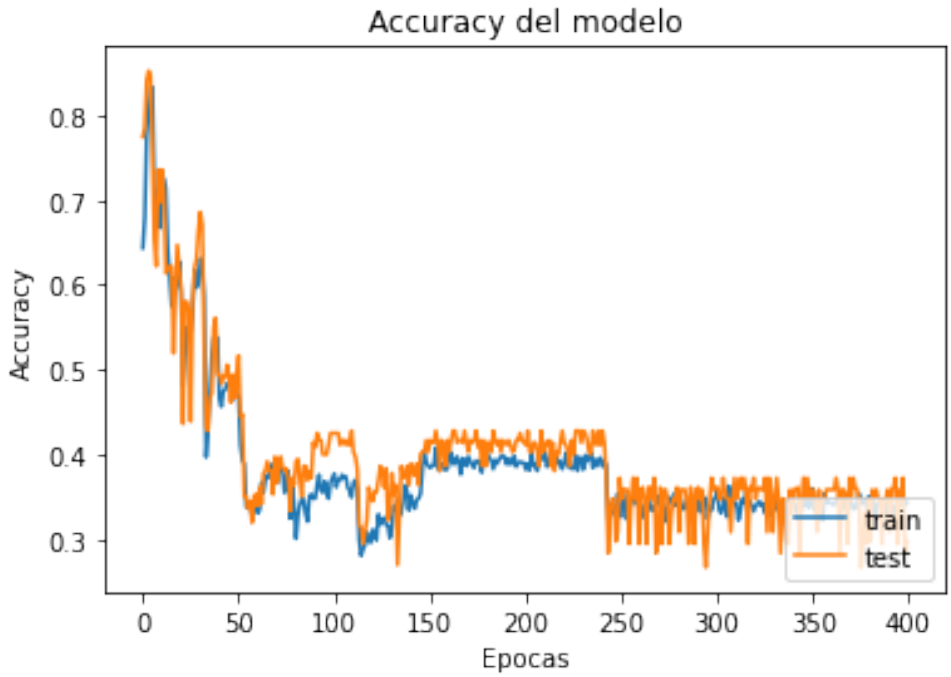

Figura 8: learning-rate: 0.1, Máxima exactitud: 0.8527777791023254, Época de máxima exactitud: 3

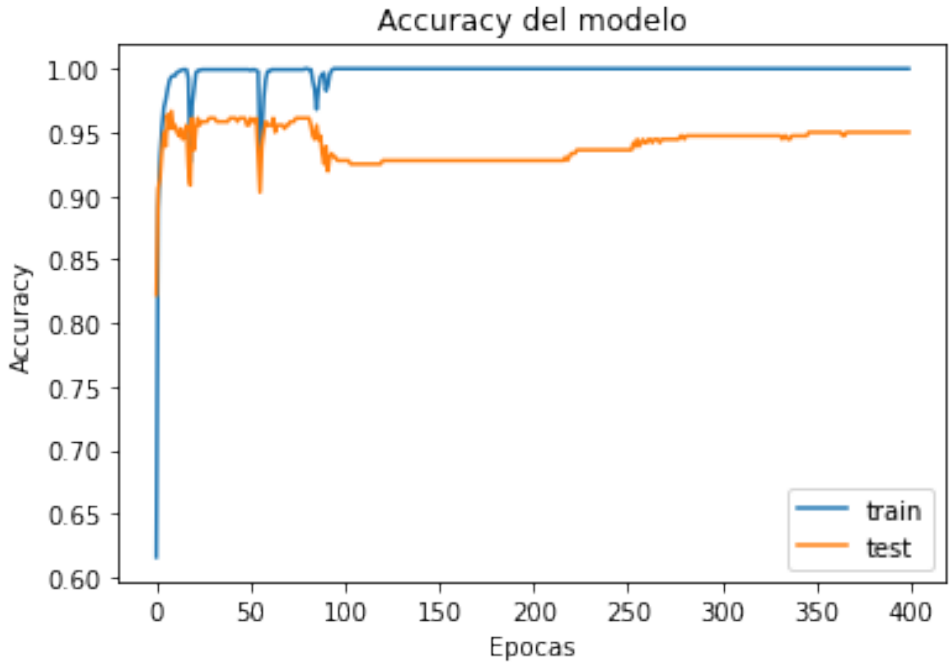

Figura 9: learning-rate: 0.01, Máxima exactitud: 0.9666666388511658, Época de máxima exactitud: 8

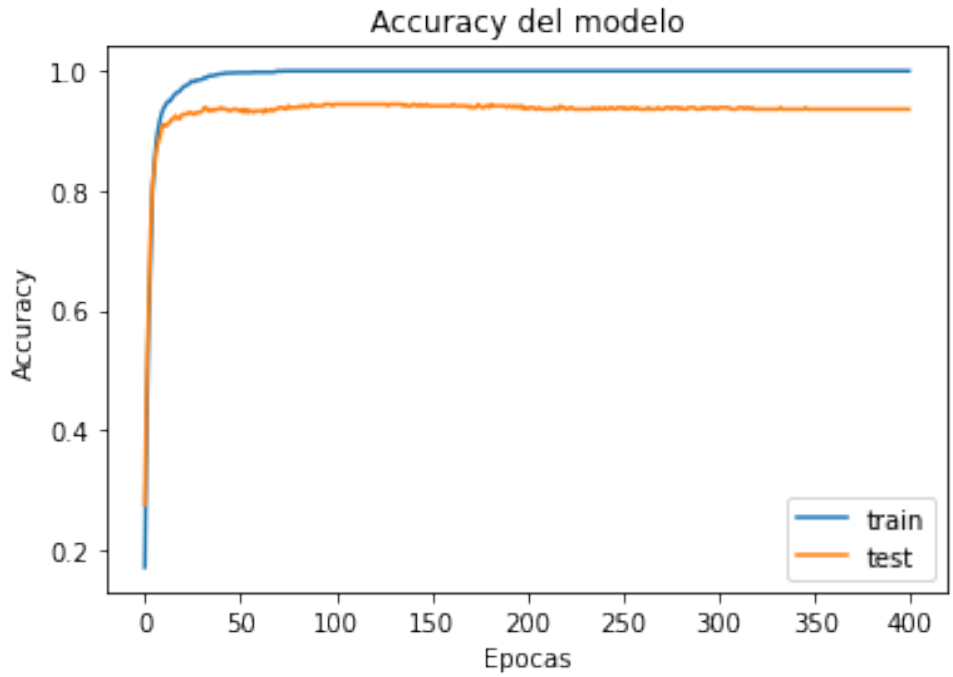

Figura 10: learning-rate: 0.001, Máxima exactitud: 0.9444444179534912, Época de máxima exactitud: 181

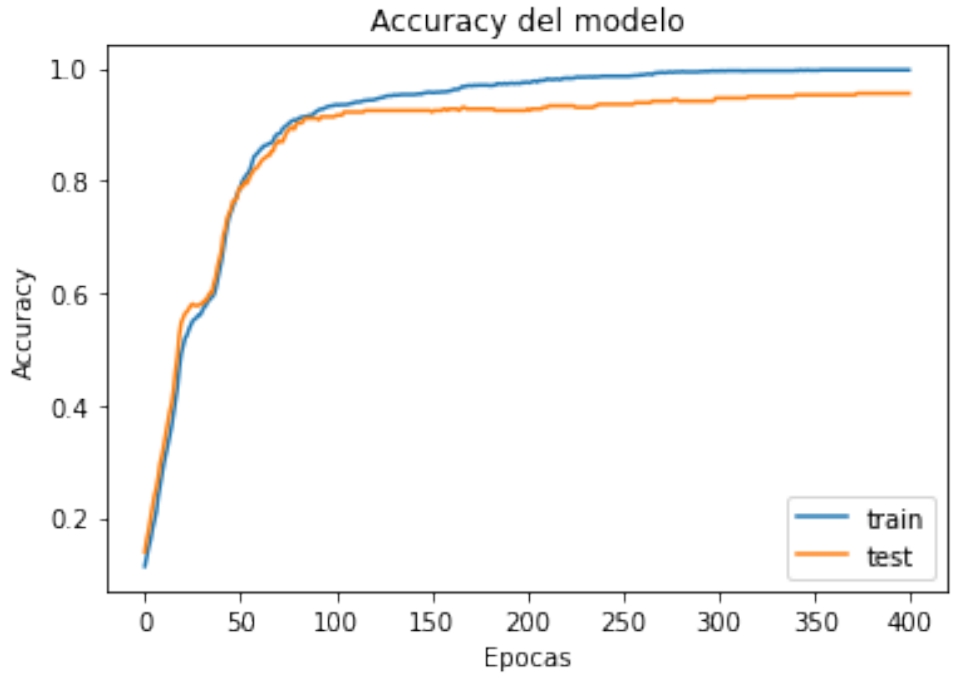

Figura 11: learning-rate: 0.0001, Máxima exactitud: 0.9555555582046509, Época de máxima exactitud: 399

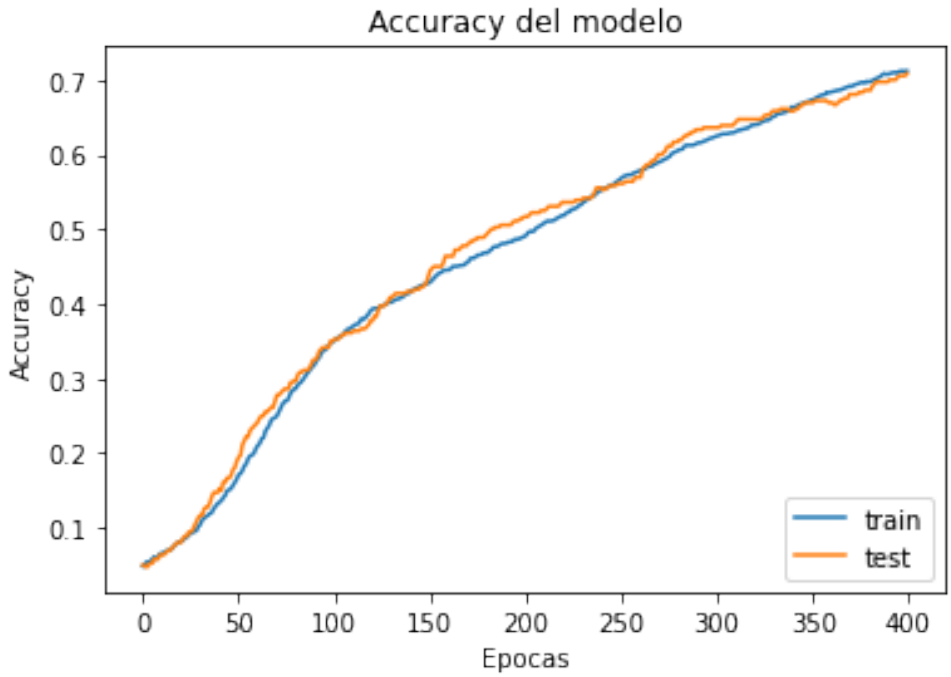

Figura 12: learning-rate: 0.00001, Máxima exactitud: 0.70833331346511, Época de máxima exactitud: 399

3.2.1. Conclusiones

Podemos ver que los valores de "learning-rate" 10, 1 y 0.1 son malas elecciones debido a su inestabilidad, donde el modelo no converge y por lo tanto no aprende dados los datos de entrenamiento. Esto se puede ver en su gráfica, donde los valores de exactitud se estabilizan y oscilan (debido al alto "learning-rate") sobre un valor bajo, luego de haber perdido exactitud a través de las épocas, o ya empezando con una valor de exactitud muy bajo.

De aquí en adelante, al ir disminuyendo los "learning-rates"se intercambia tiempo de entrenamiento (por ejemplo la figura 7 con un "learning-rate" de 0.01 muestra un buen valor de exactitud con tan solo 8 épocas, aunque desde un punto en adelante sobre-entrena el modelo) por una mayor estabilidad en el crecimiento de la exactitud pero con la necesidad de una mayor cantidad de épocas (los "learning-rates" 0.0001 y 0.00001 de las figuras 9 y 10 muestran como aún luego de 400 épocas no llegaron a converger, ya que su exactitud sigue creciendo hacia el final de la gráficas, por lo que se beneficiarían de una mayor cantidad de épocas para seguir entrenando la red).

Finalmente pudimos ver como dado un valor de "learning-rate" aceptable, la red neuronal crea un modelo mucho más preciso que un árbol de decisión (sea o no necesario corregir el sobre-entrenamiento), con la penalización de un mayor tiempo de entrenamiento. Por otro lado, los árboles nos permiten justificar la clasificación de cada imagen lo cual podría ayudar a la hora de elegir los mejores hiper-parámetros; por ejemplo al tener hojas con más valores no permitía que el árbol se metiera en casos muy específicos (lo que se traduce en un menor delta entre las exactitudes, dado que el árbol no sobre-entrena). Por su parte, las redes neuronales representan una especie caja negra en la cual es difícil entender los procesos de entrenamiento y como la red cambiará dada una modificación de sus hiper-parámetros (por ejemplo, el "learning-rate" fue prueba y error, donde no se podía predecir el resultado fácilmente). A la vez que no sabemos exactamente el por qué de la clasificación.

Llegado este punto, continuando con las conclusiones sobre el "learning-rate", dada la idea de usar una red neuronal, será decisión del programador seleccionar que valor de "learning-rate" será el óptimo para la red, siempre que esté dentro de los permisibles (en este caso los valores 10, 1 y 0.1 no se deberían

ser candidatos), teniendo en cuenta los recursos de entrenamiento disponibles y el tiempo que se puede dedicar a entrenar la red.