Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Trabajo práctico IV: Aprendizaje Automatizado

Introducción a la Inteligencia Artificial

Autores:
Arroyo, Joaquín
Bolzan, Francisco
Montoro, Emiliano

June 22, 2023

Contents

1 Ejercicio 1 **2**
1.1 Introducción . . . 2
1.2 Cuestión 1 . . . 2
1.3 Cuestión 2 . . . 2
1.3.1 Mejor combinación de parámetros . . . 2
1.3.2 Prunning . . . 3
1.4 Cuestión 3 . . . 3

2 Ejercicio 2 **6**
2.1 Introducción+Cuestión 1 . . . 6
2.2 Cuestión 2 . . . 7
2.2.1 Punto a . . . 7
2.2.2 Punto b . . . 9

1 Ejercicio 1

1.1 Introducción

Para la realización de la primera parte de este trabajo se utilizó el dataset *German Credit Data* que contiene información sobre diversas características financieras y personales de solicitantes de crédito, como historial de crédito, ingresos, estado civil, número de dependientes, entre otros. Cada instancia del dataset está etiquetada como "bueno" o "malo" según el riesgo crediticio asociado. El objetivo del conjunto de datos es poder predecir el riesgo crediticio de nuevos solicitantes basándose en las características proporcionadas.

1.2 Cuestión 1

El dataset *German Credit Data* presenta las siguientes caracteristicas: Las clases que intentaremos predecir son: ['good', 'bad']. El formato de la matriz de datos es: (1000, 20). El formato de la matriz de etiquetas es: (1000,)

- Contiene 1000 muestras.
- Las clases a predecir son 2: [bad, good].
- Los atributos de las instancias son los siguientes: [checking_status, duration, credit_history, purpose, credit_amount, savings_status, employment, installment_commitment, personal_status, other_parties, residence_sinc, property_magnitude, age, other_payment_plans, housing, job, num_dependents', own_telephone, foreign_worker].
- La distribución de las muestras por clase es la siguiente:
  - Clase 0: 300
  - Clase 1: 700

1.3 Cuestión 2

Luego de entrenar el árbol de decisión con el dataset y los siguientes parámetros ($criterion = gini$, $max\_depth = None$, $min\_samples\_split = 2$, $min\_samples\_leaf = 1$, $max\_leaf\_nodes = None$), y graficar el árbol obtenido, se oberva un árbol de gran tamaño, con nodos que no tienen muchas muestras, por lo que podríamos estar hablando de un $overfitting$.

Esto último se confirma luego de ejecutar el modelo sobre el dataset de testing, y sobre el dataset de entrenamiento. En el primer caso, obtenemos una $accuracy = 0.66$ y en el segundo caso, una $accuracy = 1.0$, por lo que efectivamente, el árbol esta sobreentrenado.

El $overfitting$ se explica ya que no limitamos ni la altura máxima, ni la cantidad de nodos hoja máximos que el árbol puede alcanzar; y además, tenemos valores bajos para la cantidad mínima de muestras para bifurcar un nodo, y para la cantidad mínima de muestras para los nodos hoja. Por lo anterior, el algorimo que genera el árbol, va a generar tantos nodos como sea posible, para así cubrir todos los casos del dataset de entrenamiento.

1.3.1 Mejor combinación de parámetros

Para obtener la mejor combinación se utilizó la clase $GridSearchCV$ con los siguientes parámetros:

- $max\_depth : [None, 5, 10, 15, 20]$
- $max\_leaf\_nodes : [None, 5, 10, 15, 20, 25]$
- $min\_samples\_leaf : [1, 2, 4, 6, 8, 10]$
- $min\_samples\_split : [2, 4, 6, 8, 10, 12]$

Y el resultado obtenido fue que la mejor combinación es $\{max\_depth : 5, max\_leaf\_nodes : 15, min\_samples_leaf : 1, min\_samples\_split : 2\}$ con una $accuracy$ de 0.75 y sin sobreentrenamiento.

1.3.2 Prunning

Para la poda, se utilizó La función *cost_complexity_pruning_path* de la clase $DecisionTreeClassifier$ de *scikit-learn*, que permite realizar la poda por niveles en un árbol de decisión. Esta función calcula una serie de valores de alpha(parámetro de complejidad) y las impurezas totales correspondientes para cada nivel de poda.

Estos valores de alpha pueden ser utilizados para controlar el nivel de poda del árbol, y se calculan considerando las impurezas totales de los nodos del árbol en diferentes etapas de poda. La impureza total es una medida de la calidad de los nodos y se puede calcular utilizando diferentes criterios, como la entropía o el índice Gini.

Al llamar a *cost_complexity_pruning_path* en un modelo $DecisionTreeClassifier$, se obtiene un objeto que contiene dos atributos principales:

- *ccp_alphas*: Es un array de *numpy* que representa los valores de alpha calculados para cada nivel de poda. Estos valores pueden ser utilizados posteriormente para realizar la poda controlada por el usuario.
- *impurities*: Es un array de *numpy* que representa las impurezas totales correspondientes a cada valor de alpha. Estas impurezas totales proporcionan una medida de la calidad del árbol en cada nivel de poda.

Luego, con cada *alpha*, entrenamos un modelo (parámetros por defecto excepto el *ccp_alpha*) y calculamos su *accuracy* frente a los datasets de entrenamiento y testing.

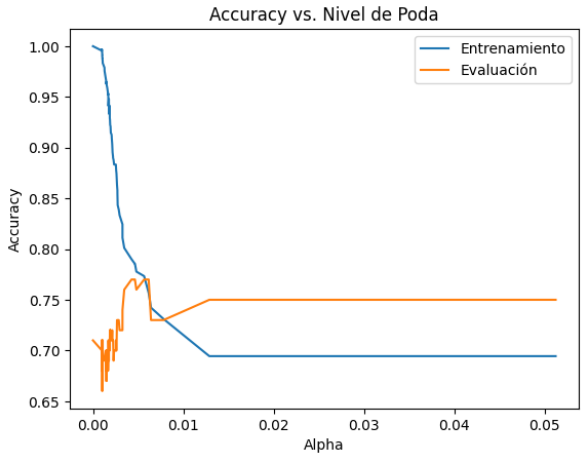

Figure 1: Gráfico que muestra las diferencias entre las *accuracies* del modelo sobre el conjunto de entrenamiento y el conjunto de testeo, a partir de la variación de *alpha*.

A partir de esta información se vió que el ni mejor alpha es aproximadamente 0.00427, con una *accuracy* de 0.77 y sin sobreentrenamiento.

1.4 Cuestión 3

Teniendo en cuenta el contexto del problema que estamos resolviendo, podemos ver que es muy importante considerar los **falsos positivos** (*bueno* que es *malo*) y los **falsos negativos** (*malo* que es *bueno*), ya que estos representan consecuencias financieras significativas para una institución financiera y pérdidas de oportunidades de negocio resprectivamente.

Para tener en cuenta estos casos, no nos alcanza con utilizar la medida *accuracy*, ya que esta solo mide la proporción de predicciones correctas sobre el total de predicciones realizadas. Para el casos de los **falsos negativos** debemos tener en cuenta la medida *recall*, y para los **falsos positivos** la medida $specificity$.

Se pasan a analizar estas medidas sobre los árboles de la cuestión 2, el que tiene la mejor combinación de parámetros, y al cual se le realizó una poda.

Notar que las matrices de confusión tienen la siguientes disposición: [[TN, FP], [FN, TP]]

Árbol con mejor configuración:

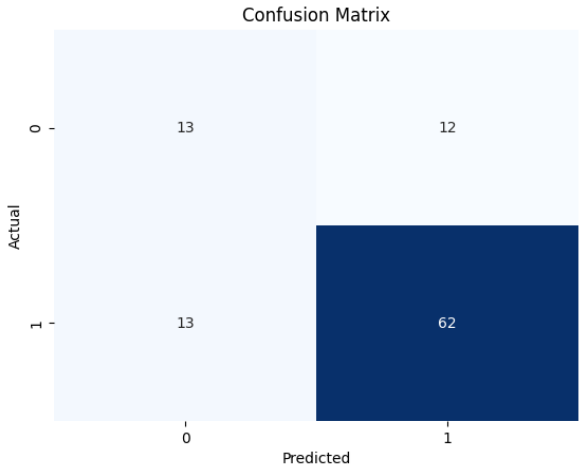

Figure 2: Matriz de confusión del árbol con mejor configuración de parámetros sobre el dataset de testing.

Medidas sobre este árbol:
$recall = 0.82$
$specificity = 0.52$
$accuracy = 0.75$

Árbol podado:

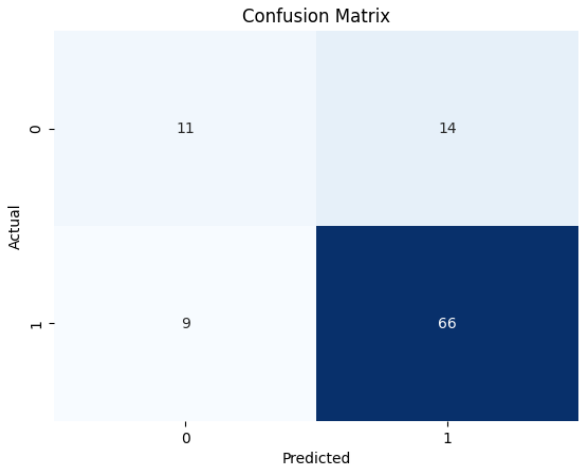

Figure 3: Matriz de confusión del árbol podado sobre el dataset de testing.

Medidas sobre este árbol:
$recall = 0.88$
$specificity = 0.44$
$accuracy = 0.77$

Analizando esta información, vemos que en la medida *accuracy* no tenemos grandes diferencias, pero en las demás dos si.

En la medida $specificity$ el primer modelo tiene una mejor performance, y en la medida *recall* pasa lo opuesto. A partir de esto, vemos que ambos modelos van a clasificar bien de manera general, pero el primero va a clasificar mejor los falsos positivos, y el segundo los falsos negativos.

Podemos pensar que para obtener un modelo que se acerque al comportamiento esperado, es decir, que tenga una performance general y niveles de *recall* y $specificity$ aceptables, se podrían combinar ambas técnicas.

A partir del árbol podado, se calculó la mejor combinación de parámetros y se obtuvo la siguiente configuración:

$\{max\_depth : 5, max\_leaf\_nodes : None, min\_samples\_leaf : 8, min\_samples\_split : 2, ccp\_alpha : 0.00427\}$

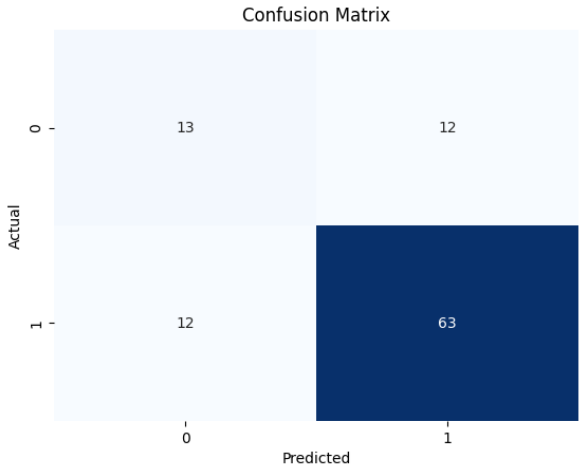

Figure 4: Matriz de confusión del árbol 'combinado' sobre el dataset de testing.

Medidas sobre el árbol con la anterior configuración:

$recall = 0.84$
$specificity = 0.52$
$accuracy = 0.76$

Vemos que con este nuevo modelo, obtenemos en parte las mejores medidas de ambos árboles, por lo nos acercamos al comportamiento esperado y concluímos que el pensamiento sobre la combinación de técnicas, en este caso es correcto.

2 Ejercicio 2

2.1 Introducción+Cuestión 1

Para la segunda parte del trabajo, se utilizo el dataset *Fashion MNIST* el cual contiene 70000 imágenes en escala de grises, divididas en 60000 imágenes de entrenamiento y 10000 imágenes de prueba. Cada imagen tiene una resolución de 28x28 píxeles y está etiquetada con una de las 10 clases de prendas posibles: [Remera, Pantalón largo, Pullover, Vestido, Abrigo, Sandali, Camisa, Zapatilla, Mochila, Botin].

Cada fila cuenta con 784 columnas de atributos con los valores de cada uno de los píxeles de las imágenes. Las etiquetas para cada imagen se encuentran en la última columna de cada archivo.

El dataset de entrenamiento se encuentra dividido equitativamente entre las clases, 6000 imágenes de cada una.

Figure 5: Muesras del dataset *Fashion MNIST*

2.2 Cuestión 2

2.2.1 Punto a

Se presentan tres gráficos comparando la evolución del *accuracy* de cada modelo sobre el dataset de entrenamiento y el dataset de test.

$Modelo_1$: Sin capa oculta.

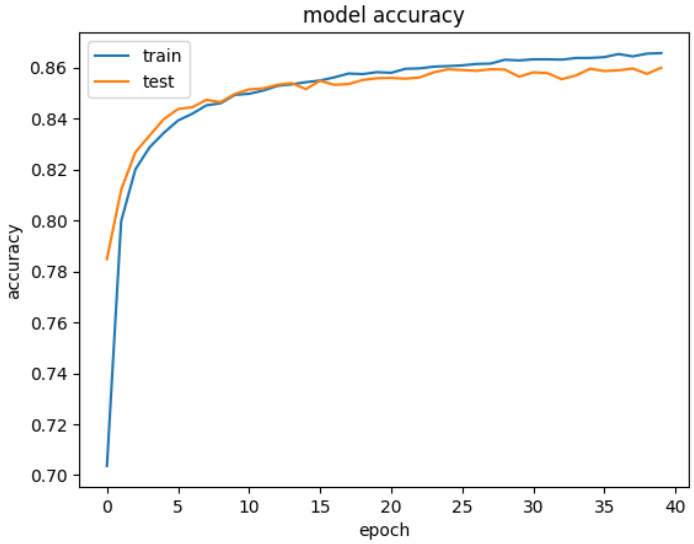

Figure 6: Gráfico que muestra como evoluciona la *accuracy* del $modelo_1$ sobre el dataset de entrenamiento y el dataset de testing.
Resultados finales: test accuracy 0.859 - train accuracy 0.865

$Modelo_2$: Con dos capas ocultas, una de 100 neuronas y otra de 50 neuronas.

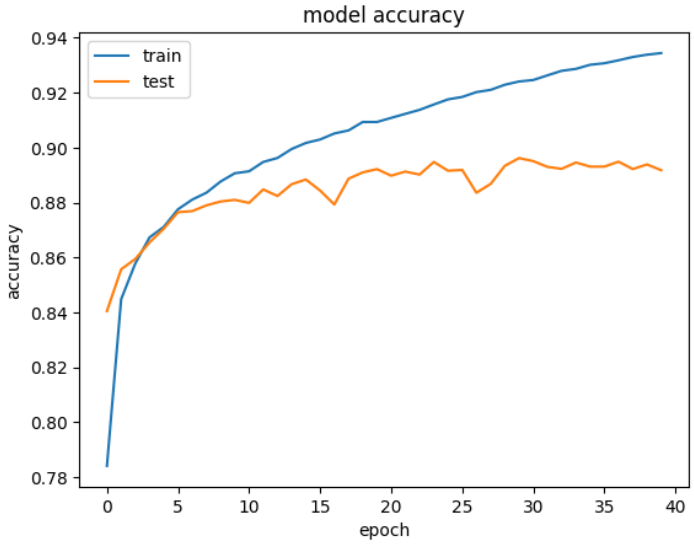

Figure 7: Gráfico que muestra como evoluciona la *accuracy* del $modelo_2$ sobre el dataset de entrenamiento y el dataset de testing.
Resultados finales: test accuracy 0.891 - train accuracy 0.932

$Modelo_3$: Con 6 capas ocultas, tres de 100 neuronas seguidas de otras tres de 50 neuronas.

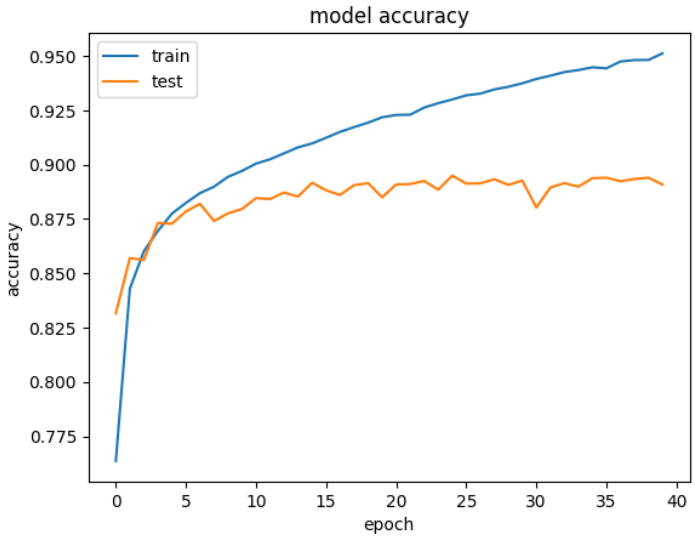

Figure 8: Gráfico que muestra como evoluciona la *accuracy* del $modelo_3$ sobre el dataset de entrenamiento y el dataset de testing.
Resultados finales: test accuracy 0.890 - train accuracy 0.952

Podemos observar en el primer modelo que el valor de accuracy es relativamente bajo, tanto con el conjunto de entrenamiento como el de test, no llegando al 90% en ambos casos. Sin embargo, la diferencia entre el accuracy de test y entrenamiento es negligible. Estas observaciones nos indican que este modelo no esta sobreentrenado, pero tampoco tiene una buena performance; manteniendo el resto de valores es probable que agregar más capas mejore el accuracy del modelo.

El segundo modelo presenta un accuracy mucho mas respetable con el conjunto de entrenamiento, llegando a 93.2%, sin sobreentrenamiento.

El tercer modelo no brinda una mejora de accuracy substancial sobre el segundo, y a su vez confirma que seguir agregando capas al modelo sólo convergerá en un sobreentrenamiento del mismo.

Entre estas observaciones podemos concluir que el número óptimo de capas ocultas se debe encontrar entre 1 y 2, ya que se observó que aumentar la cantidad de capas a partir del modelo 2, solo produce sobreentrenamiento, y no se observa una mejora sustancial en el *accuracy*; y quizás disminuyendo en uno la cantidad de capas del modelo, obtenemos un resultado similar, con menos necesidad de poder de cómputo y probablemente con menos diferencia entre la *accuracy* sobre el dataset de entrenamiento y de testing.

2.2.2 Punto b

Se presentan siete gráficos comparando la evolución del *accuracy* de cada modelo sobre el dataset de entrenamiento y el dataset de test. Todos los modelos cuentan con una única capa oculta de tamaño 50 y con activación *relu*.

$Modelo_1$: Learning Rate = 10

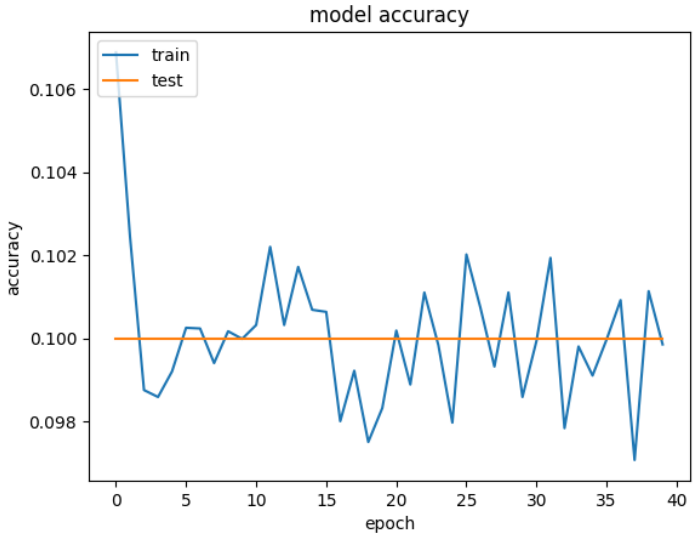

Figure 9: Gráfico que muestra como evoluciona la *accuracy* del $modelo_1$ sobre el dataset de entrenamiento y el dataset de testing.

$Modelo_2$: Learning Rate = 1

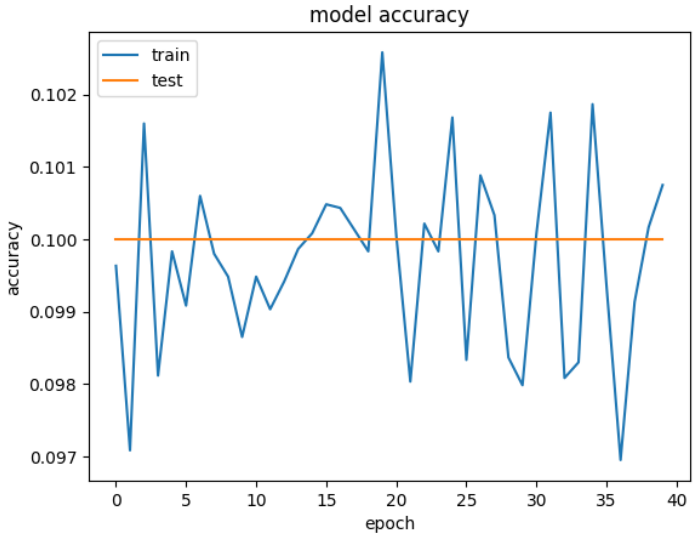

Figure 10: Gráfico que muestra como evoluciona la *accuracy* del $modelo_2$ sobre el dataset de entrenamiento y el dataset de testing.

$Modelo_3$: Learning Rate = 0.1

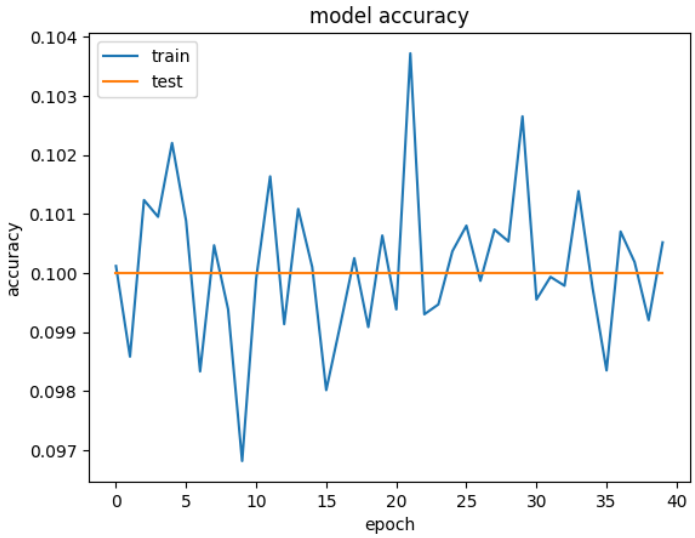

Figure 11: Gráfico que muestra como evoluciona la *accuracy* del $modelo_3$ sobre el dataset de entrenamiento y el dataset de testing.

$Modelo_4$: Learning Rate = 0.01

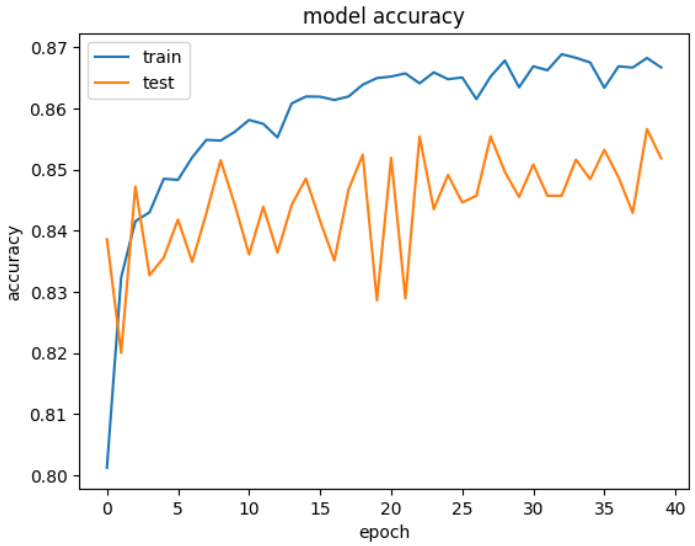

Figure 12: Gráfico que muestra como evoluciona la *accuracy* del $modelo_4$ sobre el dataset de entrenamiento y el dataset de testing.

$Modelo_5$: Learning Rate = 0.001

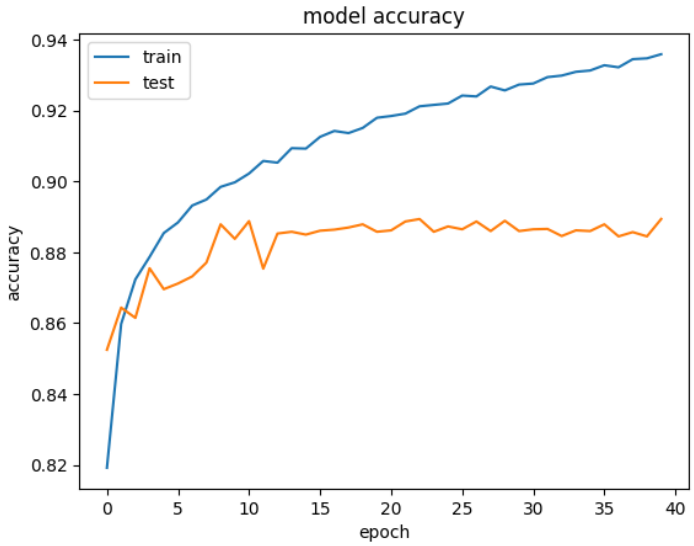

Figure 13: Gráfico que muestra como evoluciona la *accuracy* del $modelo_5$ sobre el dataset de entrenamiento y el dataset de testing.

$Modelo_6$: Learning Rate = 0.0001

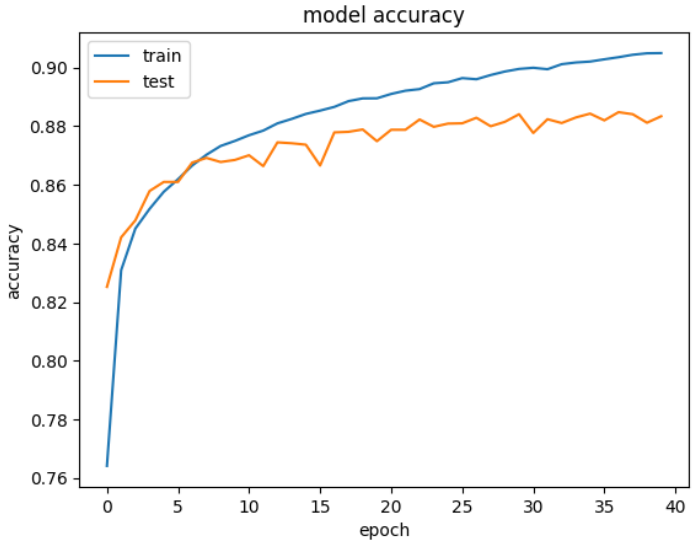

Figure 14: Gráfico que muestra como evoluciona la *accuracy* del $modelo_6$ sobre el dataset de entrenamiento y el dataset de testing.

$Modelo_7$: Learning Rate = 0.00001

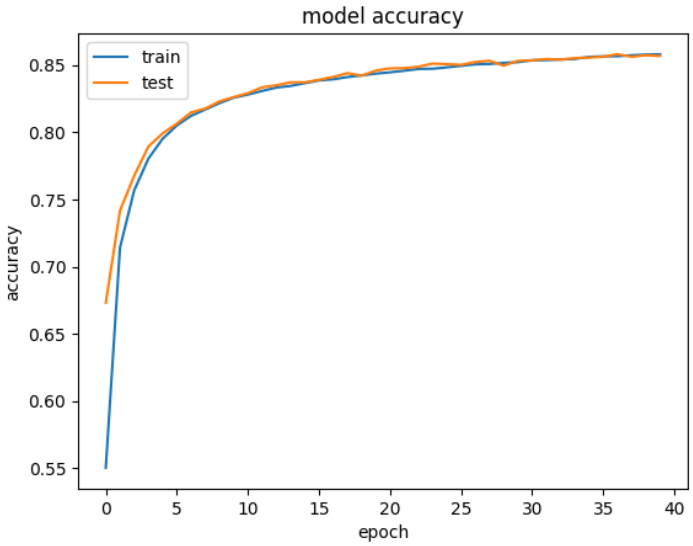

Figure 15: Gráfico que muestra como evoluciona la *accuracy* del $modelo_7$ sobre el dataset de entrenamiento y el dataset de testing.

De las gráficas presentadas anteriormente podemos concluír que el comportamiento del modelo no es el deseado si se utiliza un learning rate mayor o igual que 0.1. Esto es porque con un valor tan alto no lograremos converger correctamente al gradiente de la función de perdida, saltando constantemente sobre el mismo.

Por otro lado, si nuestro learning rate es más pequeño empezamos a ver un funcionamiento más correcto en el modelo, en particular para los valores 0.001 y 0.0001, que presentan valores de accuracy respetables y diferencias tolerables entre el accuracy de test y entrenamiento.

En el caso de learning rate 0.00001 se puede observar qué pasa si vamos al otro extremo con un learning rate demasiado pequeño, en este caso sucede que el ritmo con el que nos acercamos a dicha convergencia baja considerablemente, requiriendo entonces más épocas y por lo tanto más poder de cómputo. También es posible que nos quedemos estancados en mínimos locales.

Finalmente, de esto concluímos que es necesario elegir un learning rate que permita el correcto funcionamiento del modelo pero no alargue demasiado nuestros tiempos de entrenamiento; en particular para nuestro caso los valores 0.001 y 0.0001 son adecuados.

También, combinando esto con las observaciones del ejercicio anterior podemos ver cómo se realiza el proceso de fine-tunning para un modelo de procesamiento de imágenes, primero obteniendo un aproximado del número de capas a utilizar (en este caso se usó 1 capa oculta, como observamos en el ejercicio anterior) y luego afinando el learning rate para el modelo resultante.