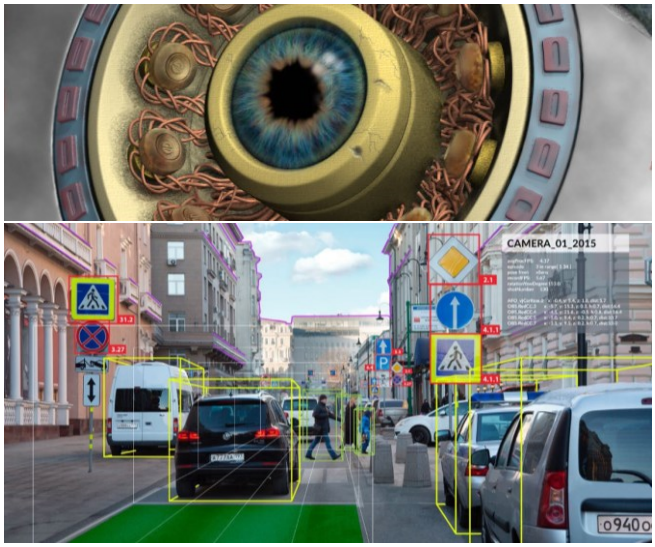


Visión por Computador

Luis Baumela Molina

Máster Universitario en Inteligencia Artificial

Universidad Politécnica de Madrid

Memoria Práctica 4:

Clasificación de imágenes

Adrián Michelena Sanz

Alberto Miño Calero

Índice de contenido

1. Descripción de los datasets 1
   1.1. Traffic Sign Recognition Benchmark 1
   1.2. CIFAR-10 3
2. Proceso para el diseño de las arquitecturas 4
3. Resultados 7
   3.1. Pruebas con redes MLP 7
      3.1.1. MLP sobre TSR 7
      3.1.2. MLP sobre CIFAR-10 8
   3.2. Pruebas con redes convolucionales 9
      3.2.1. CNN sobre TSR 9
      3.2.2. MLP sobre CIFAR-10 11
4. Experiencias sobre Google Colab 13

Referencias 14

Anexos: Configuración y resultados de las arquitecturas estudiadas 15
   A. Pruebas con MLP para el dataset Traffic Sign Recognition 15
      I. Capas ocultas, neuronas por capa, epochs y early stopping 15
      II. Optimizadores: ADAM en lugar de SGD 17
      III. Tamaño del batch 18
      IV. Dropout 19
      V. Weigth decay / batch normalization 20
   B. Pruebas con MLP para el dataset CIFAR-10 22
      I. Capas ocultas y neuronas por capa 22
      II. Optimizadores: ADAM en lugar de SGD 24
      III. Tamaño del batch 25
      IV. Dropout 26
      V. Weigth decay / batch normalization 27
   C. Pruebas con CNN para el dataset Traffic Sign Recognition 30
      I. Capas convolucionales y filtros por capa 30
      II. Optimizadores: SGD en lugar de ADAM 31
      III. Tamaño del batch 33
      IV. Data augmentation 34
      V. Dropout 36
      VI. Weigth decay / batch normalization 37
      VII. Otras combinaciones 38
   D. Pruebas con CNN para el dataset CIFAR-10 40

I. Capas convolucionales y filtros por capa 40
II. Optimizadores: SGD en lugar de ADAM 42
III. Tamaño del batch 43
IV. Data augmentation 45
V. Dropout 48
VI. Weigth decay / batch normalization 50

Índice de Figuras

Figura 1 Imágenes del dataset Traffic Sign Recognition 1
Figura 2 Imágenes preprocesadas del dataset Traffic Sign Recognition 2
Figura 3 Ejemplo de imagen de TSR prrocesada y preparada para entrenar redes de neuronas 2
Figura 4 Imágenes de ejemplo de las clases de CIFAR-10 3

Índice de Tablas

Tabla 1 Resultados de las redes MLP sobre TSR 22
Tabla 2 Resultados de las redes MLP sobre CIFAR-10 29
Tabla 3 Resultados de las redes CNN sobre TSR 39
Tabla 4 Resultados de las redes CNN sobre CIFAR-10 52

Índice de gráficas

Gráfica 1 Resultados de las redes Multi-Layer Perceptron (MLP) con el dataset Traffic Sign Recognition 7
Gráfica 2 Resultados de las redes Multi-Layer Perceptron con el dataset CIFAR-10 8
Gráfica 3 Resultados de las redes CNN con el dataset Traffic Sign Recognition 10
Gráfica 4 Resultados de las redes CNN con el dataset CIFAR-10 12
Gráfica 5 Evolución del entrenamiento de la prueba CNNC10_21: alto bias y bajo variance 48

1. Descripción de los datasets

En esta sección se van a exponer los detalles principales de los datasets que se van a estudiar. En primer lugar, hay que decir que ambos presentan problemas del mismo tipo, puesto que lo que hay que resolver son problemas de clasificación en donde a cada imagen le corresponde una única clase, de entre una multitud de opciones (clasificación multiclase). A partir de ahí, ambos datasets presentan bastantes diferencias.

1.1. Traffic Sign Recognition Benchmark

El dataset German Traffic Sign Recognition Benchmark [1] (TSR) se compone de un conjunto de datos realistas de más de 50.000 imágenes de señales de tráfico a color, cuyo tamaño varía entre 15x15 y 250x250 pixels que no están necesariamente centradas en las señales. Refleja las fuertes variaciones en la apariencia visual de las señales debido a la distancia, la iluminación, las condiciones climáticas, las oclusiones parciales y las rotaciones. El conjunto de datos comprende 43 clases con frecuencias de clase desequilibradas.

Luego, el dataset se ha dividido en 3 grupos, uno de entrenamiento, otro de validación y otro de test, con total de 852 señales para entrenamiento y validación y 361 señales para test. La división entre entrenamiento y validación ha sido de 600 para entrenamiento y 252 para validación.

El dataset contiene también anotaciones en donde se proporcionan las coordenadas, para cada imagen, de las esquinas del cuadrado que rodea la señal, de forma que ésta queda en el centro del cuadrado y llega hasta los bordes de la imagen resultante. En la Figura 1 pueden verse las imágenes originales, mientras que en la Figura 2 se ven las imágenes resultantes de aplicar un proceso de preprocesamiento consistente en recortar las originales mediante el cuadrado en el que está inscrita la señal, cuyas coordenadas están presentes en el dataset. Además, el formato de color será RGB y se normalizan los niveles de los tres canales para que estén entre 0 y 1.

Por último, se ha realizado un procesamiento más de las imágenes antes de utilizarlas para el diseño de redes de neuronas profundas. Al necesitarse una resolución fija para poder establecer un número de neuronas de entrada definido, se han reescalado las imágenes a un tamaño de 224x224. Es decir, que cada imagen recortada ha sido después reescalada a dicha resolución, lo cual añade a las imágenes aun más diversidad en lo que a "definición" se refiere, habiendo algunas señales cuyas características en cuanto a colores y bordes, entre otras, se distinguen más nítidamente y otras en las que menos.

Figura 1 Imágenes del dataset Traffic Sign Recognition

Figura 2 Imágenes preprocesadas del dataset Traffic Sign Recognition

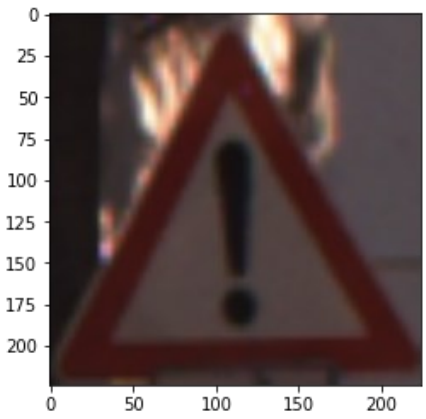


Figura 3 Ejemplo de imagen de TSR prrocesada y preparada para entrenar redes de neuronas

Acerca de este dataset y tras los procesamientos realizados, hay que mencionar algunas cosas importantes. Primero, se van a tener imágenes de señales que estarán centradas en la imagen; luego, los bordes de la imagen coinciden con los del cuadrado en los que las señales están inscritas. Esto significa que, aunque haya diversidad por características como pequeñas rotaciones, la nitidez o la iluminación, ciertas formas y colores características de cada tipo de señal van a encontrarse siempre en las mismas zonas, con cambios menores. Eso implica que, para cada cada tipo de señal, por ejemplo, la que se muestra en la Figura 3, todas las imágenes van a tener una estructura con similitudes muy acentuadas. En este caso, a simple vista se puede detectar que hay un borde triangular con el vértice superior en la parte de arriba y de color rojo, que luego hay un interior de color blanco, y que en la parte central del interior hay un símbolo de color negro. Aunque para una red de neuronas las cosas no son tan simples y es susceptible a diferencias, como las que se ve en la imagen del mismo tipo que hay en la Figura 1, se puede prever que la dificultad del problema no será excepcionalmente elevada, especialmente haciendo uso de métodos de regularización.

1.2. CIFAR-10

El dataset CIFAR-10 [2], por su parte, se compone de 60000 imágenes de resolución 32x32 y a color divididas en 10 clases, con 6000 imágenes por clase. El dataset ya viene dividido en conjuntos de 50000 imágenes para entrenamiento y otras 10000 imágenes para test. Al estar ya dividido en el artículo donde se propone, se ha utilizado la misma división. Además, en lugar de dividir en conjunto de entrenamiento en entrenamiento y validación, se ha decidido utilizar las imágenes de como las de validación, ya que el entrenamiento de la red no las utilizará en cualquiera de los casos para actualizar los pesos, por lo que sigue siendo una estrategia honesta para evaluar su rendimiento.

Como en el caso anterior, todas las clases son excluyentes entre sí, incluyendo por supuesto los casos en los que conceptualmente podría pensarse lo contrario. Por ejemplo, no hay superposición entre automóviles y camiones. "Automóvil" incluye sedanes, SUV y cosas por el estilo. "Camión" incluye solo camiones grandes. Tampoco incluye camionetas.

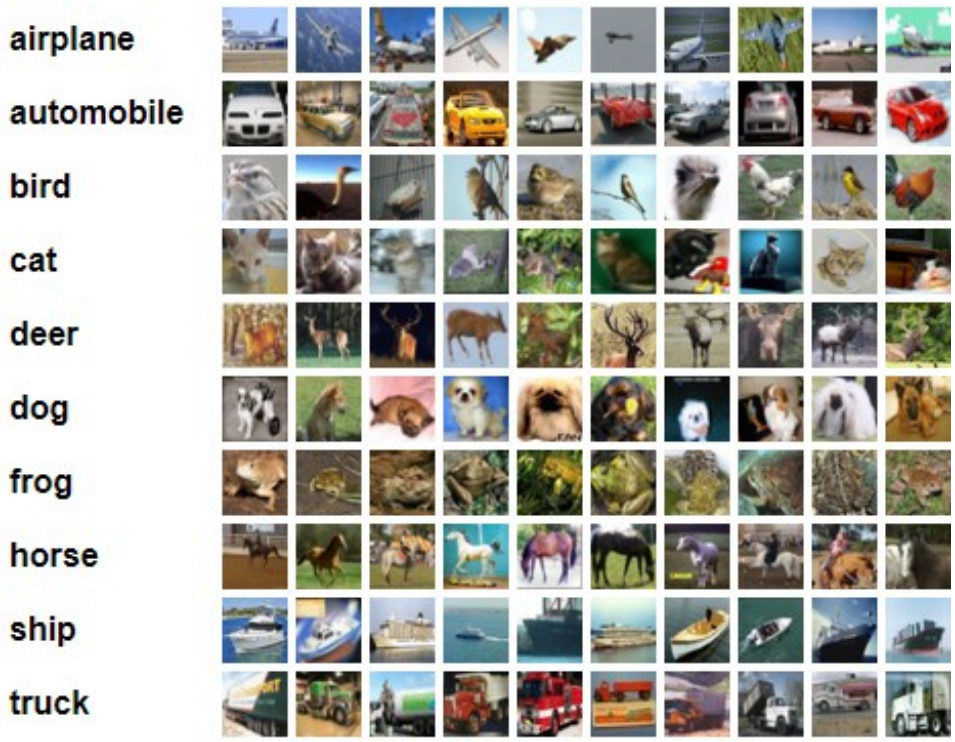


Figura 4 Imágenes de ejemplo de las clases de CIFAR-10

Sobre este dataset, se pueden decir algunas cosas. Las imágenes son de un tamaño muy inferior al de las anteriores, pero la variabilidad de las imágenes es mucho mayor, tanto entre clases distintas como dentro de cada clase, lo que hace pensar que la resolución satisfactoria del problema será más complicada que con el primer dataset, aunque también es cierto que hay más imágenes y aporta más información para el entrenamiento. En cualquier caso, la diversidad dentro de imágenes de la misma clase, que incluye colores, posición de los objetos en la imagen, iluminación y demás, es, hasta cierto punto, bastante acentuada, por lo que es previsible que será necesario construir modelos más profundos si se busca que tengan suficiente capacidad de generalización como para resolver el problema.

2. Proceso para el diseño de las arquitecturas

En este apartado se describe el proceso que se ha seguido para tratar de obtener modelos de redes de neuronas lo más precisos posibles sobre los dos datasets que se han explicado. El proceso tiene como objetivo obtener el mejor balance entre bias y variance, de forma que las redes sean capaces de aprender de los datos extrayendo las características más importantes para clasificarlos ajustándose a los datos de entrenamiento, a la vez que generalizan lo suficiente como para clasificar nuevas instancias con la mayor precisión posible. A continuación, se describe el proceso que se ha seguido describiendo el orden en el que se han diseñado las redes, que incluye capas y neuronas por capa, hiperparámetros como tamaño de batch, learning rate y epochs y técnicas de regularización como dropout, batch normalization y data augmentation.

Para desarrollar las arquitecturas de las redes neuronas, tanto para las Multilayer Perceptron (MLP) como las CNN, se ha seguido un proceso en el que, para cada arquitectura, se han realizado 5 pruebas y se ha registrado aquella con mejores resultados. Como es evidente, algunos elementos son propios para las CNN, y los pasos que impliquen estudiar las variaciones en estos parámetros no se aplican a las MLP. Sobre las arquitecturas de las CNN que se exploran en este trabajo, cabe decir que se ubican dentro de las arquitecturas VGG, cuyas capas están compuestas de capas convolucionales de filtros de 3x3 con stride 1, seguidas de capas maxpool de stride 2. Esto se ha decidido así porque según unas cuantas pruebas preliminares, los resultados obtenidos eran superiores a los conseguidos con CNN convencionales, aunque, eso sí, se han eliminado las últimas capas densamente conectadas típicas de VGG y se han sustituido por una convolucional a la que se le aplica un “aplanamiento” flatten. Y, respecto a esto, hay que decir también que no se han explorado arquitecturas más modernas que incluyan módulos inception y/o redes residuales, pero que, dadas las características de los datasets y su complejidad, no debería ser necesario hacerlo para obtener resultados razonablemente buenos.

En primer lugar, hemos comprobado una serie de combinaciones de distinto número de capas ocultas con distintas neuronas por capa, con el fin de escoger la arquitectura que mejores resultados de y seguir el proceso de desarrollo con ella. Cabe comentar que, dadas las características de los problemas a resolver, siendo ambos de clasificación de imágenes en clases con una clase para cada imagen, y utilizando redes de neuronas profundas, algunos elementos de diseño se han prefijado de la siguiente manera: las funciones de activación de las capas ocultas son ReLU, la de la capa final softmax, y la función de coste o pérdida es *categorical_crossentropy*. El learning rate en todas las pruebas se ha fijado a 0.001, que se trata de un valor recomendado para los dos algoritmos de optimización que se van a explorar, y que además ha mostrado mejores resultados en algunas pruebas preliminares.

A estas arquitecturas se les ha añadido un early stopping y se ha escogido junto a un número de epochs suficientemente elevado como para que el aprendizaje de la red se estabilice, lo cual desemboca en sobreaprendizaje. Esto se ha hecho así para intentar reducir un poco el número de pruebas, sustituyendo el hiperparámetro del número de epochs por un early stopping que se encargará de detener el proceso cuando se observe que comienza a haber sobreaprendizaje, lo cual ocurrirá cuando el valor de la función de coste en validación haya dejado de disminuir tras un número de epochs (parámetro patience para el callback EarlyStopping de keras), que hemos determinado igual a un 20% del número total de epochs escogido. Se han establecido de base un límite de 100 y 200 epochs dependiendo de la profundidad de las redes, para dar a modelos de mayor complejidad más tiempo, ya que, como es lógico, y según unas pruebas preliminares, estos últimos necesitaban más epochs para alcanzar un límite estable en su valor de precisión.

```
EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=int(epochs*0.2))
```

También se han hecho pruebas con variaciones en el algoritmo de optimización para comprobar cuál es el que mejor resultados da. Para ello, se va a probar con ADAM con los parámetros por defecto beta_1=0.9 y beta_2=0.999, y con SGD con los parámetros nerterov=true y momentun=0.9.

Respecto a la inicialización de los pesos, se han hecho algunas pruebas preliminares con inicialización he (he_normal y he_uniform), pero los resultados se han mostrado invariantes respecto a la inicialización por defecto. Esto puede ser porque se está utilizando la función de activación ReLU para las capas ocultas, que no se ve afectada por problemas típicos de las funciones *sigmoid* y *tanh*, y que tiene como ventaja que optimizar el parámetro de inicialización pasa a ser innecesario.

Llegados a este punto, comentar que estas decisiones de ir fijando parámetros de arquitecturas se han tomado para intentar reducir la cantidad de pruebas a realizar, y somos conscientes de que, al ir incorporando técnicas de regularización, tanto en el caso de las MLPs como de las CNNs, cabe la posibilidad de que alguna de las opciones descartadas previamente por haber fijado las capas, las neuronas por capa y el algoritmo diese resultados superiores. Aunque también consideramos que las posibilidades de que estas mejoras sean significativas serán reducidas por haber optimizado esos parámetros mediante pruebas.

En esta fase del proceso, se ha escogido una arquitectura con unos valores fijos, que son el número de capas ocultas, las neuronas por capa, epochs, early stopping, algoritmo de optimización y learning rate. El siguiente paso consistirá en comprobar si mejoran los resultados al modificar el tamaño del batch, de forma que podamos fijar este parámetro.

A continuación, se van a comprobar diferentes combinaciones de las técnicas de regularización que se han visto para optimizar los resultados de la MLP que mejor precisión ha alcanzado. En el caso de las MLP, se van a analizar los efectos del dropout, el weigth decay (a través del regularizador l2) como métodos de regularización, y con batch normalization, en este caso como método de mejora de rendimiento mediante una estandarización de las entradas de las capas. Para el caso de las CNN, también se incluirá la técnica de regularización de data augmentation. Para hacer las pruebas, se comprobarán cada técnica de regularización en el orden anterior y, si se encuentra una arquitectura que mejore los resultados, se continuará con ella en las pruebas sucesivas que se hagan con las técnicas de regularización restantes. Respecto al dropout, hay que comentar que utilizarlo hace necesario aumentar el número máximo de epochs, puesto que el proceso de aprendizaje de los modelos se ralentiza considerablemente. Lo mismo ocurre con la utilización de data augmentation en redes CNN.

Este proceso se seguirá para el desarrollo de cada red neuronal, las MLP y la CNN, para cada dataset *Traffic Sign Recognition* y *CIFAR-10*, aunque dependiendo de la evolución conseguida podrá haber cambios menores. En el anexo se pueden encontrar los parámetros de todas las arquitecturas que aparecen en los resultados. En el caso de las CNN, se ha aprovechado un recurso que hay disponible en el entorno utilizado de Keras, que consiste en almacenar la configuración de los pesos que mejor resultado haya dado en validación durante el proceso de entrenamiento. Para ello se ha utilizado la herramienta de checkpoints de Keras. Para comprobar el rendimiento de las redes, se ha utilizado tanto la configuración final como la óptima, y se ha utilizado para el análisis de resultados la mejor de ambas. Aunque a veces supone una mejora, hay que decir que lo habitual es que, si la hay, sea muy reducida.

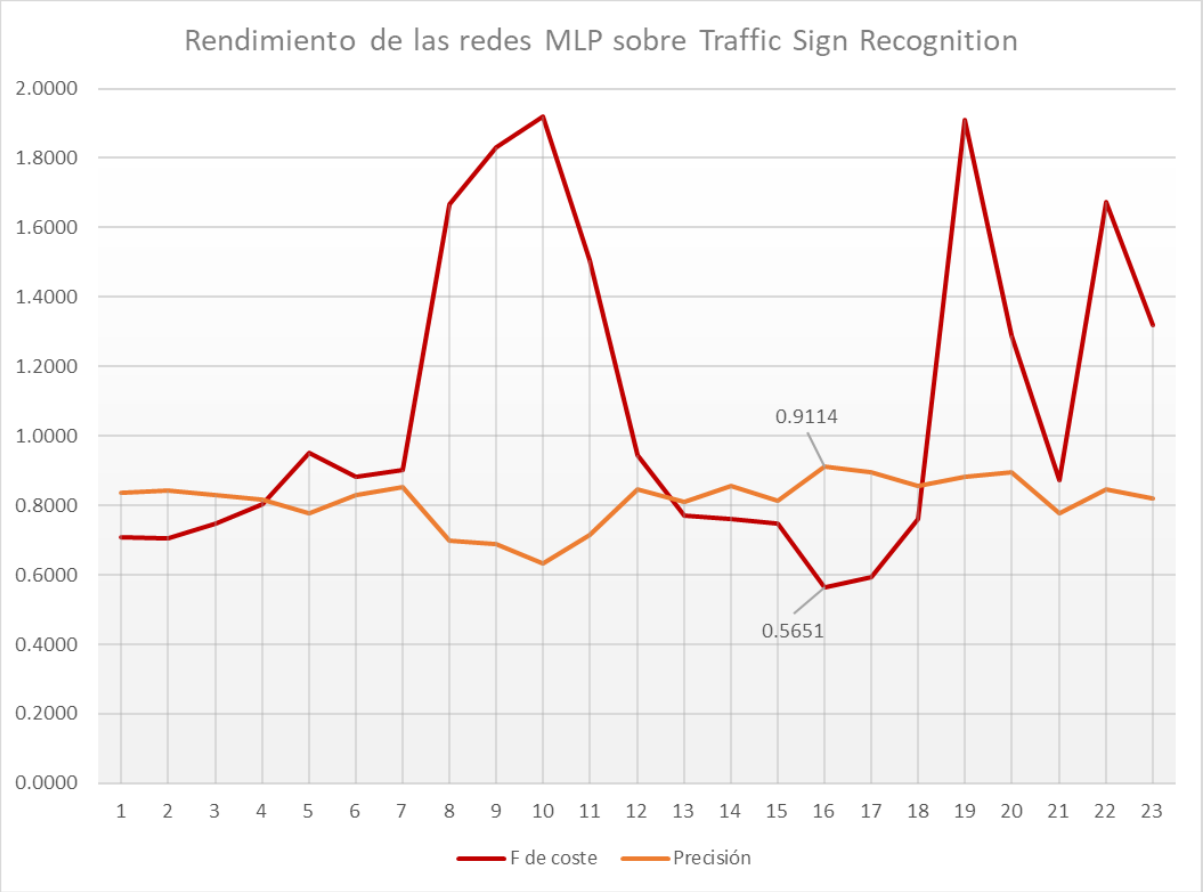
```
checkpoint_callback = ModelCheckpoint(checkpoint_path, monitor='val_acc
uracy', verbose=1, save_weights_only=True,
    # Save weights, every epoch.
    save_freq='epoch',mode='auto',save_best_only=True)
```

3. Resultados

3.1. Pruebas con redes MLP

En esta sección se van a mostrar los resultados de las pruebas realizadas con redes Multilayer Perceptron en cuanto a valores de precisión y función de coste, en la fase de test, de todas las redes arquitecturas que se han probado sobre cada uno de los datasets, así como la configuración de la arquitectura que mejor resultado de precisión ha alcanzado para cada uno. Las configuraciones específicas de cada arquitectura pueden encontrarse al final del documento.

3.1.1. MLP sobre TSR



Gráfica 1 Resultados de las redes Multi-Layer Perceptron (MLP) con el dataset Traffic Sign Recognition

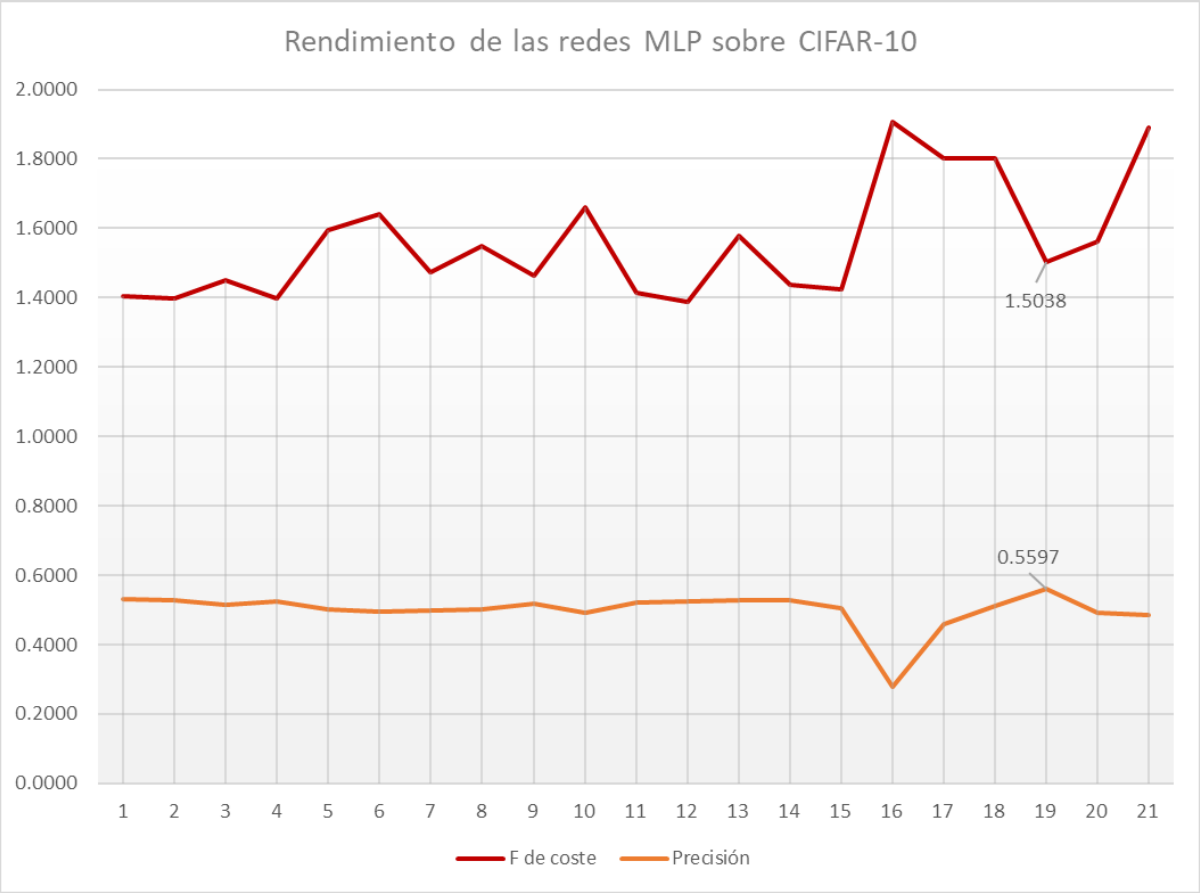
En este primer caso, aunque ha habido variación en los valores de la función de coste, los valores de precisión han sido similares al aplicar regularización. Como puede observarse en la Gráfica 1, la red con mejor rendimiento es la que se corresponde con la arquitectura número 16, es decir, MLPTSR_16, que ha alcanzado una precisión bastante elevada teniendo en cuenta que se trata de un problema de clasificación de imágenes, pero es entendible por lo que se expuso en la sección de la descripción del dataset. Los parámetros de la mejor arquitectura son los siguientes:

- MLPTSR_16
  - Capas ocultas: 4
  - Neuronas por capa: 150, 150, 150, 100
  - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9

- Learning rate: 0.001
- Tamaño del batch: 128
- Epochs: 1000
- Dropout: 0.1 en las capas ocultas
- Early stopping: si
- Weight decay (regularizador l2): no
- Batch normalization: no
    - Valor de función de coste y precisión en test: **función de coste=0.5651**, **precisión=0.9114**

3.1.2. MLP sobre CIFAR-10

En este caso, los resultados de la Gráfica 2 muestran que ninguna de las pruebas ha dado resultados significativamente superiores a las demás. Se trata de un problema de clasificación bastante más complejo que el anterior, puesto que la variabilidad de las imágenes es mucho más grande, a pesar de tener menos clases. Por ello, las redes feed forward MLP que se han explorado en este trabajo no han sido capaces de ofrecer resultados razonables. Es muy posible que no se puedan alcanzar precisiones mucho más altas de los valores en los que se han movido las pruebas aquí presentadas, y la solución al problema pasa por cambiar el modelo de red neuronal a las redes convolucionales (CNN).



Gráfica 2 Resultados de las redes Multi-Layer Perceptron con el dataset CIFAR-10

El mejor resultado se ha alcanzado con la arquitectura MLPC10_19, cuya configuración es la siguiente:

- MLPC10_19
    - Capas ocultas: 5
    - Neuronas por capa: 200, 200, 150, 150,100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 512
    - Epochs: 1000
    - Dropout: 0.2
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: si
        - **Valor de función de coste y precisión en test: función de coste=1.5038, precisión=0.5597**

Algunos detalles más sobre esta evolución:

Ninguna de las arquitecturas de las redes MLP (ambos datasets) ha mostrado resultados superiores a las otras al cambiar SGD por ADAM, y en general tienen un rendimiento bastante deficiente. Sí que parece que, en el caso de SGD, los resultados son un poco mejores. Por ello, se va a seguir con SGD en lugar de ADAM.

Luego, llegados al punto de variar el tamaño de batch, ninguna arquitectura da resultados superiores a las demás, a excepción de la pequeña ventaja de SGD sobre ADAM comentada antes. Después de haber hecho algunas pruebas más con cada arquitectura, queda claro que las variaciones son, más bien, producto de los factores aleatorios en la ordenación y agrupamiento de los datos que hace keras, y no por las arquitecturas. Por ello, se seleccionó la arquitectura más compleja, puesto que al tener más capas debe de tener mayor capacidad de generalización, y quizás en los siguientes pasos alguna de las técnicas de regularización permita a la red mejorar los valores que se han obtenido hasta ahora.

3.2. Pruebas con redes convolucionales

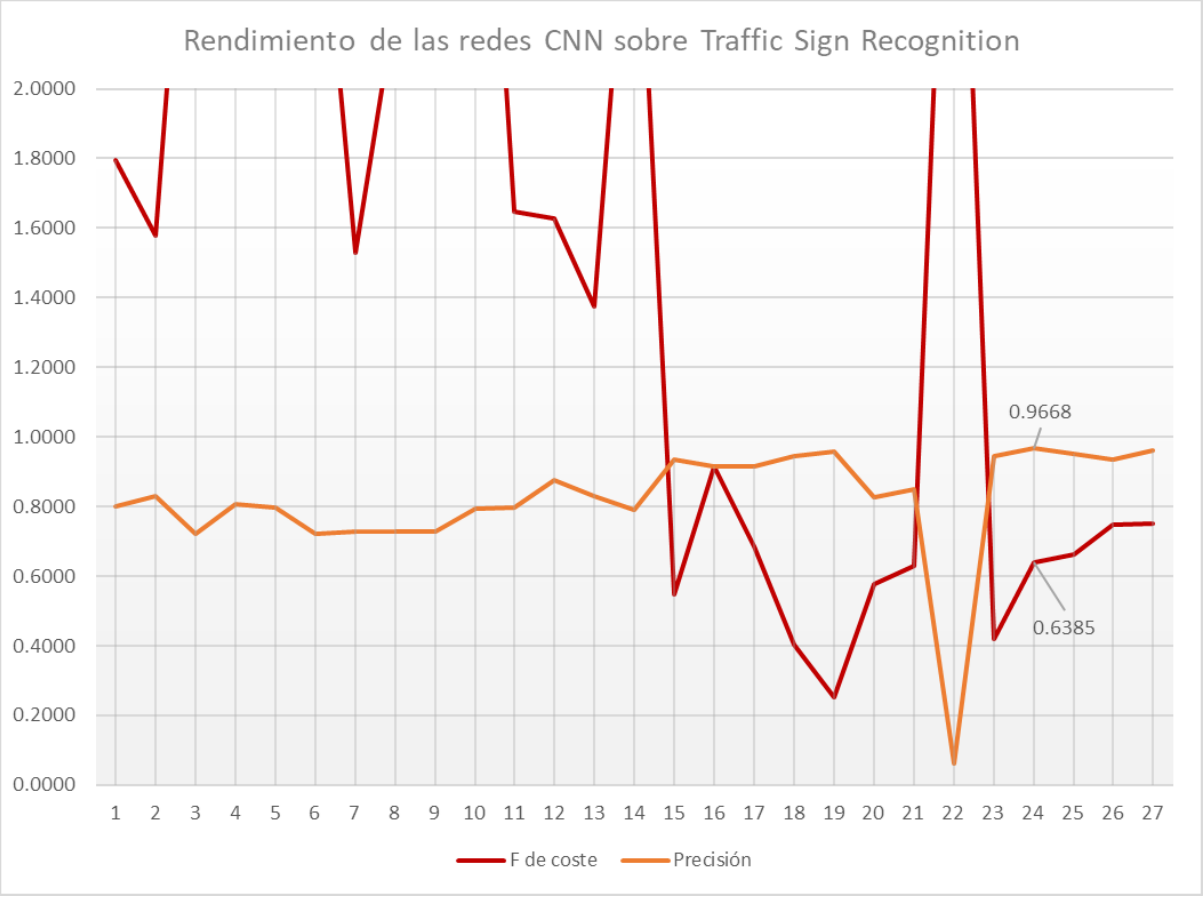
En esta sección se mostrarán los resultados de las pruebas realizadas con redes convolucionales atendiendo a su precisión y función de coste para los dos datasets que se tratan en este trabajo, así como la configuración de la arquitectura que mejor resultado de precisión ha alcanzado para cada uno. Las configuraciones específicas del resto de arquitecturas pueden encontrarse al final del documento.

3.2.1. CNN sobre TSR

En la Gráfica 3 se puede observar como el uso de técnicas de regularización (a partir de la arquitectura 15) ha permitido reducir de forma el valor de la función de coste de las CNN, a la vez ha mejorado la precisión, aunque esto último no de forma especialmente acentuada. El pico de bajada drástico del caso 22 se debe al uso del regularizador l2, que parece ser que impide a la red avanzar en el entrenamiento, posiblemente debido a temas relacionados con las características del dataset y, quizás, al tratamiento previo que se hace de las imágenes. Las técnicas restantes que se han probado han mostrado ser muy efectivas y han permitido mejorar el balance bias-variance, de forma que las redes no sufren apenas de sobreaprendizaje. Para mejorar estos resultados, algunas posibilidades evidentes son el uso de arquitecturas CNN que incluyan módulos de inception y redes residuales.

La arquitectura que mejores resultados ha dado ha sido la CNNC10_24, que obtiene una precisión en test de 0.9668, lo que supone una mejora respecto a la mejor arquitectura MLP de 0.061, un 6%, lo cual es bastante teniendo en cuenta que está en la franja entre el 90% y 100% de precisión.

- CNNTSR_24
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - Epochs: 1000
    - Dropout: 0, 0, 0.2, 0.3, 0.5, 0.5, 0.7, 0
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: si
    - Data augmentation: rotation_range = 10; width_shift_range = 0.1; height_shift_range = 0.1; horizontal_flip = False; shear_range = 0.1
        - **Valor de función de coste y precisión en test: función de coste=0.6385, precisión=0.9668**



Gráfica 3 Resultados de las redes CNN con el dataset Traffic Sign Recognition

Como comentarios adicionales respecto a este experimento, hay que decir que en el caso estos experimentos con data augmentation, mejora los resultados, pero no se aprecian diferencias significativas entre los distintos parámetros de transformación de los datos para el entrenamiento. Hay que señalar también que la opción horizontal_flip activada provoca que la red baje su rendimiento de forma drástica, lo cual es lógico teniendo en cuenta que, en el caso de las señales de tráfico, hay señales que pueden significar cosas diferentes si se las voltea horizontalmente.

Y, por último, que aplicar el regularizador l2 provoca que la red sea incapaz de resolver el problema, por lo que se han omitido el resto de las pruebas que involucran este método. Esto es lo que ocurre con la prueba 22, que como se ve en la Gráfica 3, tiene una subida de la función de coste y la precisión se queda, consecuentemente, entorno al 0.1.

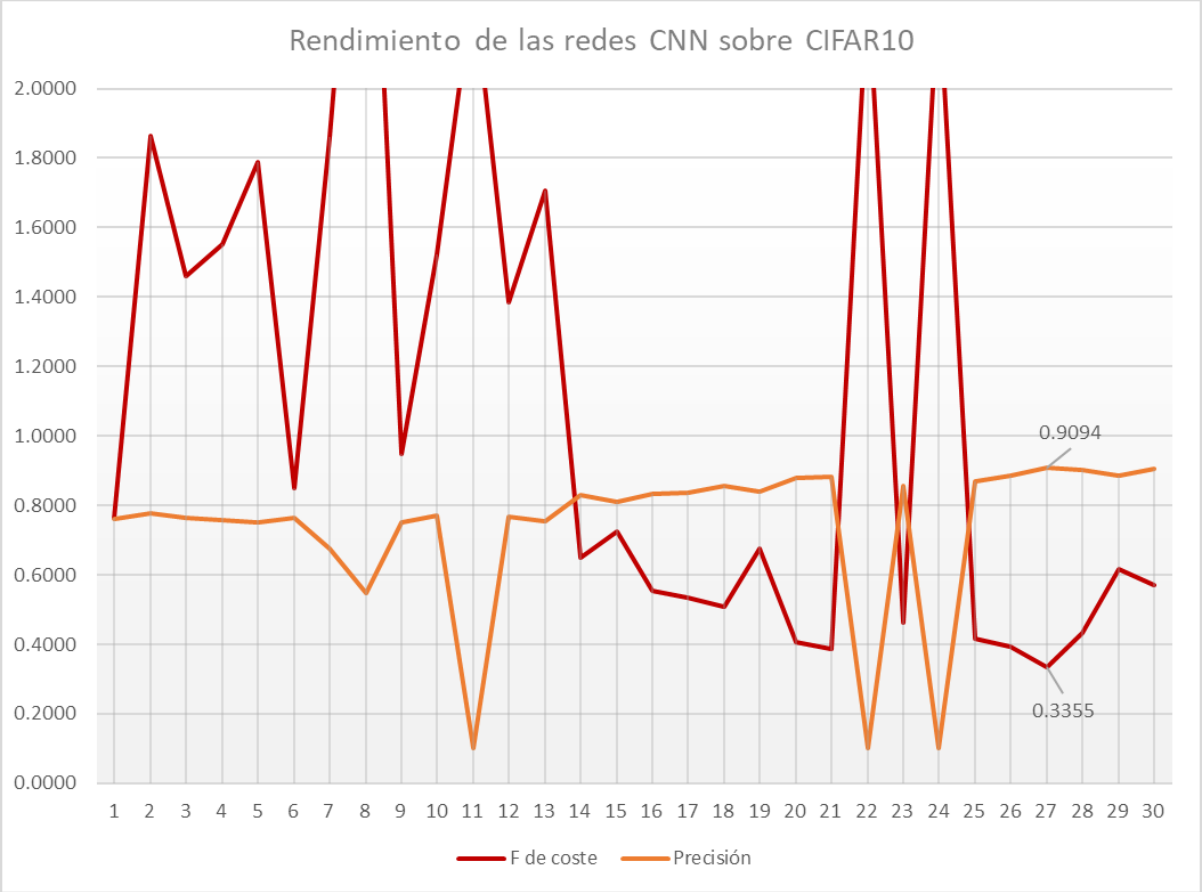
3.2.2. MLP sobre CIFAR-10

En la Gráfica 4 se ve el proceso de evolución de los resultados de las arquitecturas de CNN propuestas sobre el dataset de CIFAR-10. Se puede observar que hay una bajada importante en el valor de la función de coste en el momento en que se incorpora la primera técnica de regularización, data augmentation, en la prueba 15, y como a partir de ahí se mantiene en valores relativamente bajos, con el aumento lógico en los valores de precisión. En este caso ya no ocurre que el regularizador l2 provoque que las redes no puedan aprender, y en lugar de esto lo que pasa es que el exceso de regularización unido a un tamaño de batch reducido provoca, en algunos casos (en las pruebas mostradas, se ve en la 22 y la 24 con regularización, y en la 11 con un tamaño de batch muy bajo).

A pesar de ello, la regularización y el uso de una red CNN de cierta profundidad (11 capas), ha permitido mejorar de forma drástica los resultados, pasando de una precisión del 0.5597 en MLP a un 0.9094 con la mejor arquitectura encontrada, la CNNC10_27, cuyos parámetros se encuentran a continuación. Y, de nuevo, para mejorar estos resultados las principales posibilidades pasan por el uso de arquitecturas CNN que incluyan módulos de inception y redes residuales.

- CNNC10_27
    - Capas convolucionales: 11
    - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 256
    - Epochs: 200
    - *Dropout: 0, 0.2, 0.3, 0.5, 0*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: si
    - Data augmentation: rotation_range = 10; width_shift_range = 0.1; height_shift_range = 0.1; horizontal_flip = True; zoom_range = 0.1; shear_range = 0.1
        - **Valor de función de coste y precisión en test: función de coste=0.3355, precisión=0.9094**

Algunos comentarios más acerca de los resultados y detalles del proceso de diseño para la red CNN que resuelve este problema. Primero, al tratarse de un problema más difícil de resolver para las CNN, se han explorado arquitecturas con varias capas convolucionales entre capas de maxpooling, a diferencia del dataset anterior, donde cada capa convolucional estaba siempre seguida de una de maxpooling (excepto la última, por ser la de salida).



Gráfica 4 Resultados de las redes CNN con el dataset CIFAR-10

Otro detalle es que, en las pruebas iniciales en las que se comparaban arquitecturas de complejidad muy distinta (5 capas vs 11 capas convolucionales) los resultados eran muy parecidos, lo que se debe a la situación de sobreaprendizaje que se alcanzaba todos estos casos por la ausencia de técnicas de regularización. Y, respecto a los optimizadores, funcionabaa considerablemente mejor ADAM respecto a SGD, justo al revés de lo que nos ocurrió con las redes MLP.

Luego, respecto a la regularización, algunos apuntes. Cuando se aplicaba data augmentation, ocurría a veces que la red era incapaz de aprender desde el primer epoch, quedándose estancada. Esto lo pudimos solucionar aumentando el tamaño del batch. Nos ha parecido un detalle de interés, porque en la literatura hemos podido ver que el tamaño de batch 32 se utiliza mucho en la resolución de este tipo de problemas porque se dice que, a tamaños menores, más robusto se genera el modelo. De hecho, pudimos ver que se utilizaba mucho con este dataset específico. Sin embargo, al hacer nosotros uso del mismo y combinarlo con data augmentation, nos vimos obligado a aumentarlo para evitar que se estancase.

Y este problema se trasladó a pruebas con dropout, de forma que ocurría lo mismo. Al final, la solución fue aumentar el tamaño del batch a la vez que se reducían los parámetros tanto de dropout como de rango de variación para el data augmentation. Esto, además, era necesario, puesto que llegamos a situaciones en donde la curva de precisión en validación era superior a la de precisión en entrenamiento. Un ejemplo de esto puede verse en la Gráfica 5, que se encuentra en el anexo del documento que contiene las configuraciones de las pruebas realizadas. En dicha sección del anexo, (Pruebas con CNN para el dataset CIFAR-10), se dan algunos detalles y análisis un poco más detallados respecto a los puntos comentados que se han resumido en este apartado.

4. Experiencias sobre Google Colab

En este apartado vamos a contar algunas de nuestras experiencias con Google Colab, tanto para esta práctica como para las anteriores, con una lista de ventajas e inconvenientes que nos hemos encontrado.

Ventajas:

- No hace falta instalar ni configurar un entorno de programación.
- Tiene todas las librerías que nos han hecho falta instaladas y preparadas para utilizarse.
- Mucha documentación y comunidad activa.
- Ejecuciones rápidas gracias a la posibilidad de utilizar GPUs.
- Utilizar carpetas de Google Drive facilita el trabajo en equipo a través de copias y consultando lo que ha hecho tu compañero.

Inconvenientes:

- No se puede trabajar sobre el mismo archivo dos personas a la vez, hay error de guardado de versiones constantemente, aunque se hagan cambios en celdas de código distintas.
- Las interfaces de matplotlib para Python no funcionan distintas de la *inline* no funcionan.
- El límite de uso de la GPU nos ha dado muchos problemas en esta última práctica al hacer pruebas con las redes CNN y data augmentation, ya que consumen bastante tiempo. Eso nos obligaba a esperar 12 horas para poder volver a hacer pruebas con una GPU.

En resumen:

En general, la experiencia ha sido positiva porque nos hemos evitado, en parte, el problema de tener que preparar un entorno de ejecución a medida para las prácticas, con todo lo que ello conlleva de instalación de librerías, IDEs y demás. Sin embargo, dado que al menos en dos prácticas estábamos obligados a utilizar entornos locales para poder utilizar funcionalidades de tk de Python, otra interfaz para matplotlib (selección de puntos para correspondencias, por ejemplo), en realidad tampoco es que nos hayamos evitado la mayoría del tiempo que supone preparar el entorno.

En el caso de la esta práctica, sí que ha sido particularmente positivo, ya que no hemos tenido que instalar las librerías ni nos ha hecho falta preparar el entorno para que pueda utilizar las GPUs de nuestros equipos, que además son más modestas que las que asigna Google Colab. De hecho, lo intentamos cuando vimos que con CNN nos limitaba el uso, pero no fuimos capaces de hacer funcionar las GPUs, posiblemente por problemas de antigüedad de drivers o de integración entre las versiones de CUDA y TensorFlow. En cualquier caso, para esta práctica ha sido muy útil.

Nos ha parecido que las ventajas superan a las desventajas (las que nosotros hemos considerado), y pensamos que es una buena recomendación para hacer las prácticas, con el único inconveniente de tener que cambiar a un entorno local en ocasiones. Quizás se podrían preparar imágenes o ficheros con los datos necesarios para evitar tener que hacer esos cambios, al menos hasta que Google Colab (confiamos en que es una característica deseable y que en algún momento se añadirá) permita el uso de las interfaces que se han tenido que usar en prácticas anteriores.

Referencias

[1] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, “IEEE International Joint Conference on Neural Networks,” in *The {G}erman {T}raffic {S}ign {R}ecognition {B}enchmark: A multi-class classification competition*, 2011, pp. 1453–1460.

[2] A. Krizhevsky, “Learning Multiple Layers of Features from Tiny Images,” 2009.

Anexos: Configuración y resultados de las arquitecturas estudiadas

En este anexo se incluyen las configuraciones de todas las arquitecturas de las pruebas que se han registrado para los resultados del trabajo.

A. Pruebas con MLP para el dataset Traffic Sign Recognition

I. Capas ocultas, neuronas por capa, epochs y early stopping

- MLPTSR_1 (FeedForward Neural Network Traffic Sign Recognition número 1)
  - *Capas ocultas: 3*
  - *Neuronas por capa: 150, 150, 100*
  - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
  - Learning rate: 0.001
  - Tamaño del batch: 32
  - *Epochs: 100*
  - Dropout: no
  - *Early stopping: no*
  - Weight decay (regularizador *l2*): no
  - Batch normalization: no
    - **Valor de función de coste y precisión en test: función de coste=0.7087, precisión=0.8366**
- MLPTSR_2
  - *Capas ocultas: 3*
  - *Neuronas por capa: 200, 200, 100*
  - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
  - Learning rate: 0.001
  - Tamaño del batch: 32
  - *Epochs: 100*
  - Dropout: no
  - *Early stopping: si*
  - Weight decay (regularizador *l2*): no
  - Batch normalization: no
    - **Valor de función de coste y precisión en test: función de coste=0.7061, precisión=0.8421**
- MLPTSR_3
  - *Capas ocultas: 4*
  - *Neuronas por capa: 150, 150, 150, 100*
  - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
  - Learning rate: 0.001
  - Tamaño del batch: 32
  - *Epochs: 100*
  - Dropout: no
  - *Early stopping: si*
  - Weight decay (regularizador *l2*): no
  - Batch normalization: no

        - **Valor de función de coste y precisión en test: función de coste=0.7482, precisión=0.8283**
- MLPTSR_4
    - *Capas ocultas: 3*
    - *Neuronas por capa: 100, 100, 50*
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - *Epochs: 100*
    - Dropout: no
    - *Early stopping: si*
    - Weight decay (regularizador *l2*): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=0.8033, precisión=0.8171**
- MLPTSR_5
    - *Capas ocultas: 4*
    - *Neuronas por capa: 100, 150, 150, 50*
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - *Epochs: 100*
    - Dropout: no
    - *Early stopping: si*
    - Weight decay (regularizador *l2*): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=0.9507, precisión=0.7783**
- MLPTSR_6
    - *Capas ocultas: 4*
    - *Neuronas por capa: 100, 150, 150, 50*
    - Algoritmo de opa imización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - *Epochs: 200*
    - Dropout: no
    - *Early stopping: si*
    - Weight decay (regularizador *l2*): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=0.8812, precisión=0.8310**
- **MLPTSR_7**
    - *Capas ocultas: 4*
    - *Neuronas por capa: 150, 150, 150, 100*

- Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
- Learning rate: 0.001
- Tamaño del batch: 32
- *Epochs: 200*
- Dropout: no
- *Early stopping: si*
- Weight decay (regularizador *l2*): no
- Batch normalization: no
    - **Valor de función de coste y precisión en test: función de coste=0.9013, precisión=0.8532**

II. Optimizadores: ADAM en lugar de SGD

- MLPTSR_8
    - Capas ocultas: 3
    - Neuronas por capa: 200, 200, 100
    - *Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999*
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - Epochs: 100
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=1.666, precisión=0.6981**
- MLPTSR_9
    - Capas ocultas: 4
    - Neuronas por capa: 150, 150, 150, 100
    - *Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999*
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - Epochs: 100
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=1.8314, precisión=0.6898**
- MLPTSR_10
    - Capas ocultas: 3
    - Neuronas por capa: 100, 100, 50
    - *Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999*
    - Learning rate: 0.001
    - Tamaño del batch: 32

    - Epochs: 100
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador *l2*): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=1.9207, precisión=0.6315**
- MLPTSR_11
    - Capas ocultas: 4
    - Neuronas por capa: 100, 150, 150, 50
    - *Algoritmo de opa optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999*
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador *l2*): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=1.5074, precisión=0.7146**

Ninguna de estas pruebas ha mejorado los resultados de la MLPTSR_7.

III. Tamaño del batch

- MLPTSR_12
    - Capas ocultas: 4
    - Neuronas por capa: 150, 150, 150, 100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - *Tamaño del batch: 16*
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=0.9433, precisión=0.8448**
- MLPTSR_13
    - Capas ocultas: 4
    - Neuronas por capa: 150, 150, 150, 100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - *Tamaño del batch: 64*
    - Epochs: 200
    - Dropout: no

- Early stopping: si
- Weight decay (regularizador l2): no
- Batch normalization: no
    - **Valor de función de coste y precisión en test: función de coste=0.7714, precisión=0.8089**

- **MLPTSR_14**
    - Capas ocultas: 4
    - Neuronas por capa: 150, 150, 150, 100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - *Tamaño del batch: 128*
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador *l2*): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=0.7618, precisión=0.8560**
- MLPTSR_15
    - Capas ocultas: 4
    - Neuronas por capa: 150, 150, 150, 100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - *Tamaño del batch: 256*
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador *l2*): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=0.7472, precisión=0.8144**

IV. Dropout

- MLPTSR_16
    - Capas ocultas: 4
    - Neuronas por capa: 150, 150, 150, 100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - *Epochs: 1000*
    - *Dropout: 0.1 en las capas ocultas*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no

        - Valor de función de coste y precisión en test: **función de coste=0.5651**, **precisión=0.9114**
- MLPTSR_17
    - Capas ocultas: 4
    - Neuronas por capa: 150, 150, 150, 100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - *Epochs: 1000*
    - *Dropout: 0.2 en las capas ocultas*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=0.5949, precisión=0.8947**
- MLPTSR_18
    - Capas ocultas: 4
    - Neuronas por capa: 150, 150, 150, 100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - *Epochs: 2000*
    - *Dropout: 0.3 en las capas ocultas*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=0.7618, precisión=0.8560**

V. Weigth decay / batch normalization

- MLPTSR_19
    - Capas ocultas: 4
    - Neuronas por capa: 150, 150, 150, 100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 1000
    - Dropout: 0.1 en las capas ocultas
    - Early stopping: si
    - *Weight decay (regularizador l2): 0.01*
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=0.1.9091, precisión=0.8836**
- MLPTSR_20
    - Capas ocultas: 4

- Neuronas por capa: 150, 150, 150, 100
- Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
- Learning rate: 0.001
- Tamaño del batch: 128
- Epochs: 1000
- Dropout: 0.1 en las capas ocultas
- Early stopping: si
- *Weight decay (regularizador l2): 0.001*
- Batch normalization: no
    - **Valor de función de coste y precisión en test: función de coste=0. 1.2885, precisión=0.8947**

- MLPTSR_21
    - Capas ocultas: 4
    - Neuronas por capa: 150, 150, 150, 100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 1000
    - Dropout: 0.1 en las capas ocultas
    - Early stopping: si
    - *Weight decay (regularizador l2): no*
    - *Batch normalization: si*
        - **Valor de función de coste y precisión en test: función de coste=0.8732, precisión=0.7784**
- MLPTSR_22
    - Capas ocultas: 4
    - Neuronas por capa: 150, 150, 150, 100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 1000
    - Dropout: 0.1 en las capas ocultas
    - Early stopping: si
    - *Weight decay (regularizador l2): 0.1*
    - *Batch normalization: si*
        - **Valor de función de coste y precisión en test: función de coste=1.6737, precisión=0.8476**
- MLPTSR_23
    - Capas ocultas: 4
    - Neuronas por capa: 150, 150, 150, 100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 1000

- Dropout: 0.1 en las capas ocultas
- Early stopping: si
- *Weight decay (regularizador l2): 0.001*
- *Batch normalization: si*
    - **Valor de función de coste y precisión en test: función de coste=0.1.3199, precisión=0.8199**

Tabla 1 Resultados de las redes MLP sobre TSR

MLPTSR	**F de coste**	**Precisión**
1	0.7087	0.8366
2	0.7061	0.8421
3	0.7482	0.8283
4	0.8033	0.8171
5	0.9507	0.7783
6	0.8812	0.8310
7	0.9013	0.8532
8	1.6660	0.6981
9	1.8314	0.6898
10	1.9207	0.6315
11	1.5074	0.7146
12	0.9433	0.8448
13	0.7714	0.8089
14	0.7618	0.8560
15	0.7472	0.8144
16	0.5651	0.9114
17	0.5949	0.8947
18	0.7618	0.8560
19	1.9091	0.8836
20	1.2885	0.8947
21	0.8732	0.7784
22	1.6737	0.8476
23	1.3199	0.8199

B. Pruebas con MLP para el dataset CIFAR-10

I. Capas ocultas y neuronas por capa

- MLPC10_1 (FeedForward Neural Network CIFAR-10 número 1)
    - *Capas ocultas: 4*
    - *Neuronas por capa: 200, 200, 150, 100*
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 200
    - Dropout: no

- Early stopping: si
- Weight decay (regularizador l2): no
- Batch normalization: no
    - **Valor de función de coste y precisión en test: función de coste=1.4047, precisión=0.5304**

- MLPC10_2
    - *Capas ocultas: 5*
    - *Neuronas por capa: 200, 200, 150, 150,100*
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=1.3987, precisión=0.5265**
- MLPC10_3
    - *Capas ocultas: 4*
    - *Neuronas por capa: 150, 150, 100, 50*
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=1.4497, precisión=0.5142**
- MLPC10_4
    - *Capas ocultas: 5*
    - *Neuronas por capa: 150, 150, 100, 100 ,50*
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=1.3992, precisión=0.5253**

II. Optimizadores: ADAM en lugar de SGD

- MLPC10_5
  - Capas ocultas: 4
  - Neuronas por capa: 200, 200, 150, 100
  - *Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999*
  - Learning rate: 0.001
  - Tamaño del batch: 128
  - Epochs: 200
  - Dropout: no
  - Early stopping: si
  - Weight decay (regularizador l2): no
  - Batch normalization: no
    - **Valor de función de coste y precisión en test: función de coste=1.5950, precisión=0.5010**
- MLPC10_6
  - Capas ocultas: 5
  - Neuronas por capa: 200, 200, 150, 150,100
  - *Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999*
  - Learning rate: 0.001
  - Tamaño del batch: 128
  - Epochs: 200
  - Dropout: no
  - Early stopping: si
  - Weight decay (regularizador l2): no
  - Batch normalization: no
    - **Valor de función de coste y precisión en test: función de coste=1.6394, precisión=0.4935**
- MLPC10_7
  - Capas ocultas: 4
  - Neuronas por capa: 150, 150, 100, 50
  - *Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999*
  - Learning rate: 0.001
  - Tamaño del batch: 128
  - Epochs: 200
  - Dropout: no
  - Early stopping: si
  - Weight decay (regularizador l2): no
  - Batch normalization: no
    - **Valor de función de coste y precisión en test: función de coste=1.4717, precisión=0.4990**
- MLPC10_8
  - Capas ocultas: 5
  - Neuronas por capa: 150, 150, 100, 100 ,50

- *Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999*
- Learning rate: 0.001
- Tamaño del batch: 128
- Epochs: 200
- Dropout: no
- Early stopping: si
- Weight decay (regularizador l2): no
- Batch normalization: no
  - **Valor de función de coste y precisión en test: función de coste=1.5486, precisión=0.5031**

Ninguna de las arquitecturas ha mostrado resultados superiores a las otras, y en general tienen un rendimiento bastante deficiente. Sí que parece que, en el caso de SGD, los resultados son un poco mejores. Por ello, se va a seguir con SGD en lugar de ADAM.

III. Tamaño del batch

- MLPC10_9
  - Capas ocultas: 4
  - Neuronas por capa: 200, 200, 150, 100
  - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
  - Learning rate: 0.001
  - *Tamaño del batch: 256*
  - Epochs: 200
  - Dropout: no
  - Early stopping: si
  - Weight decay (regularizador l2): no
  - Batch normalization: no
    - **Valor de función de coste y precisión en test: función de coste=1.4630, precisión=0.5188**
- MLPC10_10
  - Capas ocultas: 5
  - Neuronas por capa: 200, 200, 150, 150,100
  - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
  - Learning rate: 0.001
  - *Tamaño del batch: 256*
  - Epochs: 200
  - Dropout: no
  - Early stopping: si
  - Weight decay (regularizador l2): no
  - Batch normalization: no
    - **Valor de función de coste y precisión en test: función de coste=1.6594, precisión=0.4932**
- MLPC10_11
  - Capas ocultas: 4
  - Neuronas por capa: 200, 200, 150, 100

    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - *Tamaño del batch: 512*
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=1.4132, precisión=0.5210**
- MLPC10_12
    - Capas ocultas: 5
    - Neuronas por capa: 200, 200, 150, 150,100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - *Tamaño del batch: 512*
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=1.3876, precisión=0.5243**

De nuevo, llegados a este punto ninguna arquitectura da resultados superiores a las demás, a excepción de la pequeña ventaja de SGD sobre ADAM comentada antes. Después de haber hecho algunas pruebas más con cada arquitectura, queda claro que las variaciones son producto de los factores aleatorios en la ordenación y agrupamiento de los datos que hace keras, y no por las arquitecturas. Por ello, vamos a quedarnos con la arquitectura más compleja de estas últimas, puesto que al tener más capas debe de tener mayor capacidad de generalización, y quizás en los siguientes pasos alguna de las técnicas de regularización permita a la red mejorar los valores que se han obtenido hasta ahora.

IV. Dropout

- MLPC10_13
    - Capas ocultas: 5
    - Neuronas por capa: 200, 200, 150, 150,100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 512
    - Epochs: 1000
    - *Dropout: 0.1*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no

        - **Valor de función de coste y precisión en test: función de coste=1.5799, precisión=0.5277**
- MLPC10_14
    - Capas ocultas: 5
    - Neuronas por capa: 200, 200, 150, 150,100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 512
    - Epochs: 1000
    - *Dropout: 0.2*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=1.4366, precisión=0.5277**
- MLPC10_15
    - Capas ocultas: 5
    - Neuronas por capa: 200, 200, 150, 150,100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 512
    - Epochs: 1000
    - *Dropout: 0.3*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=1.4245, precisión=0.5059**
- MLPC10_16
    - Capas ocultas: 5
    - Neuronas por capa: 200, 200, 150, 150,100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 512
    - Epochs: 1000
    - *Dropout: 0.5*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
        - **Valor de función de coste y precisión en test: función de coste=1.9073, precisión=0.2782**

V. Weigth decay / batch normalization

- MLPC10_17
    - Capas ocultas: 5

- Neuronas por capa: 200, 200, 150, 150,100
- Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
- Learning rate: 0.001
- Tamaño del batch: 512
- Epochs: 1000
- Dropout: 0.2
- Early stopping: si
- *Weight decay (regularizador l2): 0.01*
- *Batch normalization: no*
    - **Valor de función de coste y precisión en test: función de coste=1.8006, precisión=0.4593**

- MLPC10_18
    - Capas ocultas: 5
    - Neuronas por capa: 200, 200, 150, 150,100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 512
    - Epochs: 1000
    - Dropout: 0.2
    - Early stopping: si
    - *Weight decay (regularizador l2): 0.001*
    - *Batch normalization: no*
        - **Valor de función de coste y precisión en test: función de coste=1.8004, precisión=0.5120**

- **MLPC10_19**
    - Capas ocultas: 5
    - Neuronas por capa: 200, 200, 150, 150,100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 512
    - Epochs: 1000
    - Dropout: 0.2
    - Early stopping: si
    - *Weight decay (regularizador l2): no*
    - *Batch normalization: si*
        - **Valor de función de coste y precisión en test: función de coste=1.5038, precisión=0.5597**

- MLPC10_20
    - Capas ocultas: 5
    - Neuronas por capa: 200, 200, 150, 150,100
    - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
    - Learning rate: 0.001
    - Tamaño del batch: 512
    - Epochs: 1000

- Dropout: 0.2
- Early stopping: si
- *Weight decay (regularizador l2): 0.01*
- *Batch normalization: si*
  - **Valor de función de coste y precisión en test: función de coste=1.5614, precisión=0.4923**

- MLPC10_21
  - Capas ocultas: 5
  - Neuronas por capa: 200, 200, 150, 150,100
  - Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9
  - Learning rate: 0.001
  - Tamaño del batch: 512
  - Epochs: 1000
  - Dropout: 0.2
  - Early stopping: si
  - *Weight decay (regularizador l2): 0.001*
  - *Batch normalization: si*
    - **Valor de función de coste y precisión en test: función de coste=1.8886, precisión=0.4866**

Tabla 2 Resultados de las redes MLP sobre CIFAR-10

MLPC10	**F de coste**	**Precisión**
1	1.4047	0.5304
2	1.3987	0.5265
3	1.4497	0.5142
4	1.3992	0.5253
5	1.5950	0.5010
6	1.6394	0.4935
7	1.4717	0.4990
8	1.5486	0.5031
9	1.4630	0.5188
10	1.6594	0.4932
11	1.4132	0.5210
12	1.3876	0.5243
13	1.5799	0.5277
14	1.4366	0.5277
15	1.4245	0.5059
16	1.9073	0.2782
17	1.8006	0.4593
18	1.8004	0.5120
19	1.5038	0.5597
20	1.5614	0.4923
21	1.8886	0.4866

C. Pruebas con CNN para el dataset Traffic Sign Recognition

En estas arquitecturas, tras cada capa convolucional exceptuando la última de salida, cuyo número de filtros se indica en "filtros por capa", hay una capa de maxpooling de tamaño de ventana 2x2 y con stride por defecto (tamaño de ventana).

En las capas, aparecen todas las capas en lugar de mostrarse solo las ocultas, incluyendo la de entrada y la de salida, pero tras cada capa convolucional a excepción de la de salida hay una de maxpooling para reducir la dimensionalidad. Por ello, en pruebas donde se aplica dropout con parámetro diferente según la capa, hay tantos valores como capas, aplicándose el dropout después de las capas de maxpooling. Y por ello, hay un 0 al final, puesto que la capa de salida no debe tener dropout.

I. Capas convolucionales y filtros por capa

- CNNTSR_1 (Convolutional Neural Network Traffic Sign Recognition número 1)
    - *Capas convolucionales: 8*
    - *Filtros por capa: 32, 64, 128, 128, 256, 256, 512, 43*
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=1.7958, precisión=0.8006**
- **CNNTSR_2**
    - *Capas convolucionales: 8*
    - *Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43*
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=1.5768, precisión=0.8310**
- CNNTSR_3
    - *Capas convolucionales: 8*
    - *Filtros por capa: 32, 64, 128, 256, 512, 256, 64, 43*
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999

- Learning rate: 0.001
- Tamaño del batch: 128
- Epochs: 200
- Dropout: no
- Early stopping: si
- Weight decay (regularizador l2): no
- Batch normalization: no
- Data augmentation: no
    - **Valor de función de coste y precisión en test: función de coste=3.0684, precisión=0.7229**

- CNNTSR_4
    - *Capas convolucionales: 8*
    - *Filtros por capa: 16, 32, 64, 128, 256, 512, 128, 43*
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=2.5230, precisión=0. 8060**
- CNNTSR_5
    - *Capas convolucionales: 7*
    - *Filtros por capa: 16, 32, 64, 128, 256, 512, 43*
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=2.7485, precisión=0.7977**

II. Optimizadores: SGD en lugar de ADAM

- CNNTSR_6
    - Capas convolucionales: 8
    - Filtros por capa: 32, 64, 128, 128, 256, 256, 512, 43
    - *Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9*
    - Learning rate: 0.001

    - Tamaño del batch: 128
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste= 2.7794, precisión= 0.7229**
- CNNTSR_7
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - *Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9*
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste= 1.5298, precisión= 0.7285**
- CNNTSR_8
    - Capas convolucionales: 8
    - Filtros por capa: 32, 64, 128, 256, 512, 256, 64, 43
    - *Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9*
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste= 2.2424, precisión= 0.7202**
- CNNTSR_9
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 128, 43
    - *Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9*
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 200
    - Dropout: no

- Early stopping: si
- Weight decay (regularizador l2): no
- Batch normalization: no
- Data augmentation: no
    - **Valor de función de coste y precisión en test: función de coste= 2.6409, precisión= 0.7202**

En este caso, el optimizador SGD parece funcionar algo peor que ADAM, al contrario de lo que se ha observado en los casos anteriores con redes MLP.

III. Tamaño del batch

- CNNTSR_10
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - *Tamaño del batch: 8*
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=3.3723, precisión=0.7950**
- CNNTSR_11
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - *Tamaño del batch: 16*
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=1.6482, precisión=0.7978**
- **CNNTSR_12**
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - *Tamaño del batch: 32*

    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=1.6279, precisión=0.8753**
- CNNTSR_13
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - *Tamaño del batch: 64*
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=1.3057, precisión=0.8283**
- CNNTSR_14
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - *Tamaño del batch: 256*
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=2.8346, precisión=0.7895**

En este caso, los resultados han mostrado que es más conveniente utilizar un tamaño de batch menor que el que se utilizó con MLP.

IV. Data augmentation

- CNNTSR_15
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999

    - Learning rate: 0.001
    - Tamaño del batch: 32
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - *Data augmentation: rotation_range = 5; width_shift_range = 0.05; height_shift_range = 0.05; horizontal_flip = False; shear_range = 0.05*
        - **Valor de función de coste y precisión en test: función de coste=0.5478, precisión=0.9362**
- CNNTSR_16
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - *Data augmentation: rotation_range = 10; width_shift_range = 0.1; height_shift_range = 0.1; horizontal_flip = False; shear_range = 0.1*
        - **Valor de función de coste y precisión en test: función de coste=0.9140, precisión=0.9141**
- CNNTSR_17
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - *Data augmentation: rotation_range = 20; width_shift_range = 0.2; height_shift_range = 0.2; horizontal_flip = False; shear_range = 0.2*
        - **Valor de función de coste y precisión en test: función de coste=0.6838, precisión=0.9141**

En el caso estos experimentos con data augmentation, mejora los resultados, pero no se aprecian diferencias significativas entre los distintos parámetros de transformación de los datos para el entrenamiento. Hay que señalar también que la opción horizontal_flip activado provoca que la red

baje su rendimiento de forma drástica, lo cual es lógico teniendo en cuenta que, en el caso de las señales de tráfico, hay señales que pueden significar cosas diferentes si se las voltea horizontalmente.

V. Dropout

- CNNTSR_18
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - Epochs: 500
    - *Dropout: 0, 0, 0.2, 0.2, 0.3, 0.5, 0.5, 0*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: rotation_range = 5; width_shift_range = 0.05; height_shift_range = 0.05; horizontal_flip = False; shear_range = 0.05
        - **Valor de función de coste y precisión en test: función de coste=0.4029, precisión=0.9446**
- **CNNTSR_19**
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - Epochs: 500
    - *Dropout: 0, 0, 0.2, 0.3, 0.5, 0.5, 0.7, 0*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: rotation_range = 5; width_shift_range = 0.05; height_shift_range = 0.05; horizontal_flip = False; shear_range = 0.05
        - **Valor de función de coste y precisión en test: función de coste=0.2510, precisión=0.9585**
- CNNTSR_20
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - Epochs: 1000
    - *Dropout: 0, 0.2, 0.2, 0.5, 0.5, 0.7, 0.7, 0*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no

    - Data augmentation: rotation_range = 5; width_shift_range = 0.05; height_shift_range = 0.05; horizontal_flip = False; shear_range = 0.05
        - **Valor de función de coste y precisión en test: función de coste=0.5770, precisión=0.8255**
- CNNTSR_21
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - Epochs: 1000
    - *Dropout: 0, 0.2, 0.2, 0.5, 0.5, 0.8, 0.8, 0*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: rotation_range = 5; width_shift_range = 0.05; height_shift_range = 0.05; horizontal_flip = False; shear_range = 0.05
        - **Valor de función de coste y precisión en test: función de coste=0.6303, precisión=0.8504**

VI. Weigth decay / batch normalization

- CNNTSR_22
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - Epochs: 500
    - Dropout: 0, 0, 0.2, 0.3, 0.5, 0.5, 0.7, 0
    - Early stopping: si
    - *Weight decay (regularizador l2): 0.01*
    - *Batch normalization: no*
    - Data augmentation: rotation_range = 5; width_shift_range = 0.05; height_shift_range = 0.05; horizontal_flip = False; shear_range = 0.05
        - **Valor de función de coste y precisión en test: función de coste=3.4017, precisión=0.0609**
- CNNTSR_23
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - Epochs: 500
    - Dropout: 0, 0, 0.2, 0.3, 0.5, 0.5, 0.7, 0
    - Early stopping: si

- *Weight decay (regularizador l2): no*
- *Batch normalization: si*
- Data augmentation: rotation_range = 5; width_shift_range = 0.05; height_shift_range = 0.05; horizontal_flip = False; shear_range = 0.05
    - **Valor de función de coste y precisión en test: función de coste=0.4195, precisión=0.9446**

Aplicar el regularizador l2 provoca que la red sea incapaz de resolver el problema, por lo que se han omitido el resto de las pruebas que involucran este método.

VII. Otras combinaciones

- **CNNTSR_24**
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - *Epochs: 1000*
    - Dropout: 0, 0, 0.2, 0.3, 0.5, 0.5, 0.7, 0
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: si
    - *Data augmentation: rotation_range = 10; width_shift_range = 0.1; height_shift_range = 0.1; horizontal_flip = False; shear_range = 0.1*
        - **Valor de función de coste y precisión en test: función de coste=0.6385, precisión=0.9668**
- CNNTSR_25
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 32
    - *Epochs: 500*
    - Dropout: 0, 0, 0.2, 0.3, 0.5, 0.5, 0.7, 0
    - *Early stopping: no*
    - Weight decay (regularizador l2): no
    - Batch normalization: si
    - Data augmentation: rotation_range = 10; width_shift_range = 0.1; height_shift_range = 0.1; horizontal_flip = False; shear_range = 0.1
        - **Valor de función de coste y precisión en test: función de coste=0.6625, precisión=0.9529**
- CNNTSR_26
    - Capas convolucionales: 8
    - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999

- Learning rate: 0.001
- *Tamaño del batch: 64*
- *Epochs: 500*
- Dropout: 0, 0, 0.2, 0.3, 0.5, 0.5, 0.7, 0
- *Early stopping: no*
- Weight decay (regularizador l2): no
- Batch normalization: si
- Data augmentation: rotation_range = 10; width_shift_range = 0.1; height_shift_range = 0.1; horizontal_flip = False; shear_range = 0.1
  - **Valor de función de coste y precisión en test: función de coste=0.7472, precisión=0.9335**

- CNNTSR_27
  - Capas convolucionales: 8
  - Filtros por capa: 16, 32, 64, 128, 256, 512, 512,43
  - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
  - Learning rate: 0.001
  - *Tamaño del batch: 16*
  - *Epochs: 500*
  - Dropout: 0, 0, 0.2, 0.3, 0.5, 0.5, 0.7, 0
  - *Early stopping: no*
  - Weight decay (regularizador l2): no
  - Batch normalization: si
  - Data augmentation: rotation_range = 10; width_shift_range = 0.1; height_shift_range = 0.1; horizontal_flip = False; shear_range = 0.1
    - **Valor de función de coste y precisión en test: función de coste=0.7514, precisión=0.9612**

Tabla 3 Resultados de las redes CNN sobre TSR

CNNTSR	**F de coste**	**Precisión**
1	1.7958	0.8006
2	1.5768	0.8310
3	3.0684	0.7229
4	2.5230	0.8060
5	2.7485	0.7977
6	2.7794	0.7229
7	1.5298	0.7285
8	2.2424	0.7272
9	2.6409	0.7272
10	3.3723	0.7950
11	1.6482	0.7978
12	1.6279	0.8753
13	1.3757	0.8283
14	2.8346	0.7895
15	0.5478	0.9362
16	0.9140	0.9141

17	0.6838	0.9141
18	0.4029	0.9446
19	0.2510	0.9585
20	0.5770	0.8255
21	0.6303	0.8504
22	3.4017	0.0609
23	0.4195	0.9446
24	0.6385	0.9668
25	0.6625	0.9529
26	0.7472	0.9335
27	0.7514	0.9612

D. Pruebas con CNN para el dataset CIFAR-10

De nuevo, en estas arquitecturas, tras cada capa convolucional exceptuando la última de salida, cuyo número de filtros se indica en "filtros por capa", hay una capa de maxpooling de tamaño de ventana 2x2 y con stride por defecto (tamaño de ventana). Se han explorado para este dataset arquitecturas con varias capas convolucionales entre capas de maxpooling; cuando esto ocurra, se indicará entre paréntesis. Por ejemplo, si en el número de filtros por capa aparecieran los valores 10, (20,20), (30,10) eso significaría que hay una capa convolucional de entrada con 10 filtros seguida de una de maxpooling, luego dos capas convolucionales de 20 filtros seguida la última de una de maxpooling, y luego dos capas convolucionales, la primera de 30 filtros y la última (de salida) de 10.

En las capas, aparecen todas las capas en lugar de mostrarse solo las ocultas, incluyendo la de entrada y la de salida. Por ello, en pruebas donde se aplica dropout con parámetro diferente según la capa, siempre habrá un 0 tanto en la primera como en la última posición, puesto que no debe haberlo en las capas de entrada y de salida.

I. Capas convolucionales y filtros por capa

- CNNC10_1 (Convolutional Neural Network CIFAR-10 número 1)
    - *Capas convolucionales: 4*
    - *Filtros por capa: 32, 128, 256, 10*
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - *Epochs: 200*
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=0.7632, precisión=0.7615**
- CNNC10_2
    - *Capas convolucionales: 5*
    - *Filtros por capa: 32, 128, 256, 512, 10*

    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - *Epochs: 200*
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=1.8644, precisión=0.7760**
- CNNC10_3
    - *Capas convolucionales: 6*
    - *Filtros por capa: 64, 128, 256, 512, (512, 10)*
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - *Epochs: 200*
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=1.4606, precisión=0.7643**
- CNNC10_4
    - *Capas convolucionales: 9*
    - *Filtros por capa: 64, 128, (256, 256), (512, 512), (512,512,10)*
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - *Epochs: 200*
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=1.5505, precisión=0.7570**
- CNNC10_5
    - *Capas convolucionales: 10*
    - *Filtros por capa: 32, (64, 64), (128, 128), (256, 256), (512,512,10)*
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001

    - Tamaño del batch: 128
    - *Epochs: 200*
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=1.7883, precisión=0.7508**
- CNNC10_6
    - *Capas convolucionales: 11*
    - *Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)*
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - *Epochs: 200*
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=0.8480, precisión=0.7640**

Ninguna de las pruebas muestra resultados significativamente superiores, lo que parece indicar que con los datos disponibles no es posible mejorar la precisión o reducir la situación de sobreaprendizaje que se está dando. Por ello, se van a promocionar a la siguiente fase dos arquitecturas, una con menos y otra con más capas, de manera que más adelante se podrá comprobar hasta qué punto son arquitecturas adecuadas, cuando se hayan incluido métodos de regularización.

II. Optimizadores: SGD en lugar de ADAM

- CNNC10_7
    - Capas convolucionales: 5
    - Filtros por capa: 32, 128, 256, 512, 10
    - *Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9*
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=1.8608, precisión=0.6769**
- CNNC10_8

- Capas convolucionales: 11
- Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
- *Algoritmo de optimización y parámetros específicos (si los hubiera): SGD con momentum=0.9*
- Learning rate: 0.001
- Tamaño del batch: 128
- Epochs: 200
- Dropout: no
- Early stopping: si
- Weight decay (regularizador l2): no
- Batch normalization: no
- Data augmentation: no
    - **Valor de función de coste y precisión en test: función de coste=3.0998, precisión=0.5474**

Como ocurrió con el primer dataset, el optimizador SGD ha dado peores resultados que Adam.

III. Tamaño del batch

- CNNC10_9
    - Capas convolucionales: 5
    - Filtros por capa: 32, 128, 256, 512, 10
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - *Tamaño del batch: 32*
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=0.9464, precisión=0.7503**
- CNNC10_10
    - Capas convolucionales: 5
    - Filtros por capa: 32, 128, 256, 512, 10
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - *Tamaño del batch: 512*
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=1.5235, precisión=0.7696**

- CNNC10_11
    - Capas convolucionales: 11
    - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - *Tamaño del batch: 32*
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=2.3026, precisión=0.1000**
- CNNC10_12
    - Capas convolucionales: 11
    - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - *Tamaño del batch: 64*
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=1.3857, precisión=0.7663**
- CNNC10_13
    - Capas convolucionales: 11
    - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - *Tamaño del batch: 512*
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: no
        - **Valor de función de coste y precisión en test: función de coste=1.7067, precisión=0.7549**

El tamaño del batch no parece afectar mucho al rendimiento de la red, aunque a tamaños mayores se acelera algo el entrenamiento al reducir el número de actualizaciones de pesos por epoch.

IV. Data augmentation

En las pruebas con la arquitectura de 11 capas ha sido necesario aumentar el tamaño del batch, ya que se ha podido comprobar que con 64, el proceso de entrenamiento se queda la mayoría de las veces estancado y no es capaz de aprender.

- CNNC10_14
    - Capas convolucionales: 5
    - Filtros por capa: 32, 128, 256, 512, 10
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 64
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - *Data augmentation: rotation_range = 10; width_shift_range = 0.1; height_shift_range = 0.1; horizontal_flip = True; zoom_range = 0.1; shear_range = 0.1*
        - **Valor de función de coste y precisión en test: función de coste=0.6502, precisión=0.8288**
- CNNC10_15
    - Capas convolucionales: 5
    - Filtros por capa: 32, 128, 256, 512, 10
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 64
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - *Data augmentation: rotation_range = 5; width_shift_range = 0.05; height_shift_range = 0.05; horizontal_flip = True; zoom_range = 0.05; shear_range = 0.05*
        - **Valor de función de coste y precisión en test: función de coste=0.7248, precisión=0.8116**
- CNNC10_16
    - Capas convolucionales: 5
    - Filtros por capa: 32, 128, 256, 512, 10
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 64
    - Epochs: 200
    - Dropout: no
    - Early stopping: si

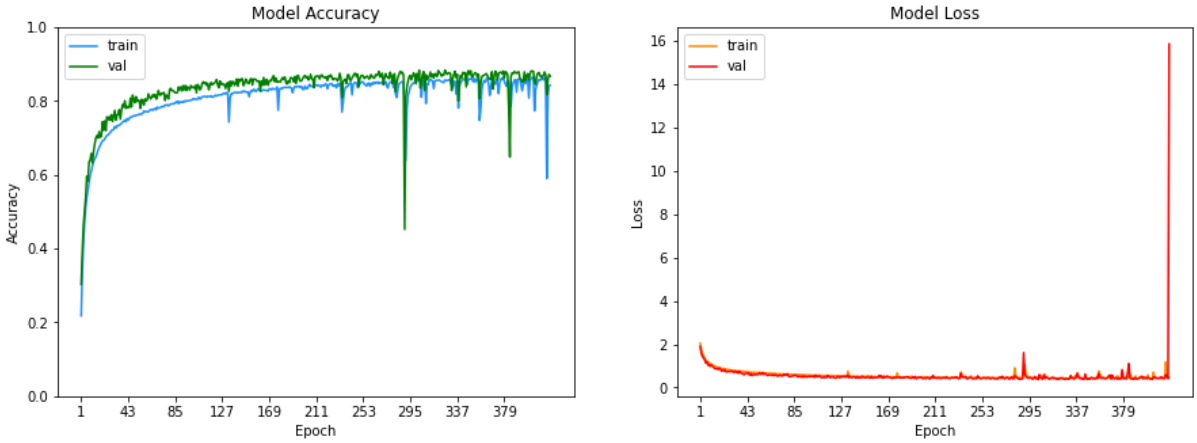
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - *Data augmentation: rotation_range = 20; width_shift_range = 0.2; height_shift_range = 0.2; horizontal_flip = True; zoom_range = 0.2; shear_range = 0.2*
        - **Valor de función de coste y precisión en test: función de coste=0.5555, precisión=0.8317**
- CNNC10_17
    - Capas convolucionales: 5
    - Filtros por capa: 32, 128, 256, 512, 10
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 64
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - *Data augmentation: rotation_range = 30; width_shift_range = 0.3; height_shift_range = 0.3; horizontal_flip = True; zoom_range = 0.3; shear_range = 0.3*
        - **Valor de función de coste y precisión en test: función de coste=0.5341, precisión=0.8364**
- CNNC10_18
    - Capas convolucionales: 11
    - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 200
    - Dropout: no
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - *Data augmentation: rotation_range = 10; width_shift_range = 0.1; height_shift_range = 0.1; horizontal_flip = True; zoom_range = 0.1; shear_range = 0.1*
        - **Valor de función de coste y precisión en test: función de coste=0.5077, precisión=0.8564**
- CNNC10_19
    - Capas convolucionales: 11
    - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 200
    - Dropout: no
    - Early stopping: si

- Weight decay (regularizador l2): no
- Batch normalization: no
- *Data augmentation: rotation_range = 5; width_shift_range = 0.05; height_shift_range = 0.05; horizontal_flip = True; zoom_range = 0.05; shear_range = 0.05*
  - **Valor de función de coste y precisión en test: función de coste=0.6749, precisión=0.8411**

- CNNC10_20
  - Capas convolucionales: 11
  - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
  - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
  - Learning rate: 0.001
  - Tamaño del batch: 128
  - Epochs: 200
  - Dropout: no
  - Early stopping: si
  - Weight decay (regularizador l2): no
  - Batch normalization: no
  - *Data augmentation: rotation_range = 20; width_shift_range = 0.2; height_shift_range = 0.2; horizontal_flip = True; zoom_range = 0.2; shear_range = 0.2*
    - **Valor de función de coste y precisión en test: función de coste=0.4076, precisión=0.8791**
- CNNC10_21
  - Capas convolucionales: 11
  - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
  - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
  - Learning rate: 0.001
  - Tamaño del batch: 128
  - Epochs: 500
  - Dropout: no
  - Early stopping: si
  - Weight decay (regularizador l2): no
  - Batch normalization: no
  - *Data augmentation: rotation_range = 30; width_shift_range = 0.3; height_shift_range = 0.3; horizontal_flip = True; zoom_range = 0.3; shear_range = 0.3*
    - **Valor de función de coste y precisión en test: función de coste=0.3863, precisión=0.8838**

Aunque en la prueba final (CNNC10_21) es donde mejor precisión se ha alcanzado, también se genera una situación de alto bias y bajo variance, y la red no es capaz de ajustar a los datos de entrenamiento por encima del 0.90, como se puede ver en la Gráfica 5. De hecho, los valores de validación, (que en este dataset son los de test) se sitúan sobre e incluso por encima los de entrenamiento. Esto ocurre también, aunque en menor medida, con el caso CNNC10_20. El motivo principal puede estar en que se está aplicando data augmentation de forma algo agresiva, y en tiempo real. Eso implica que en cada epoch se generan aleatoriamente imágenes nuevas a partir de las de entrenamiento, y que estos nuevos datos son demasiado diversos como para que la red sea capaz.

Por ello, es mejor configurar unos parámetros de data augmentation menos agresivos, con lo que, aunque la precisión final en test sea menor, reduzcan el bias para así poder explorar otras técnicas de regularización. De esta forma, se podría seguir buscando la manera de mejorar los resultados por encima de los que se han conseguido en esta prueba. Otra estrategia más recomendable sería aumentar el número de parámetros de la red, por ejemplo, añadiendo más capas convolucionales o densamente conectadas, pero dadas las restricciones por límite de uso del entorno que se está utilizando, Google Colab, hacerlo no es viable. Se van a utilizar las arquitecturas de 11 capas, puesto que han mostrado una capacidad de generalización para resolver el problema más alta que en el caso de las de 5, y unos parámetros de data augmentation intermedios entre los utilizados.



Gráfica 5 Evolución del entrenamiento de la prueba CNNC10_21: alto bias y bajo variance

V. Dropout

En el caso de estas arquitecturas, hay dos capas convolucionales antes de cada capa de maxpooling. Al aplicar el dropout, se ha decidido hacerlo después de esta capa de maxpooling, y por tanto solo aparece un valor de dropout por cada grupo (pareja) de capas convolucionales entre paréntesis, ya que después de cada paréntesis de cierre hay una capa de máxpooling, a excepción de la última que es la de salida. Por tanto, al haber 5 grupos de capas convolucionales, siendo el último el de salida, aparecerán 5 valores, y el último será 0. Otro problema encontrado es que el entrenamiento vuelve a quedarse estancado al volverse a aumentar la regularización con dropout, y de nuevo incrementando el tamaño de batch se puede atajar el problema.

- CNNC10_22
  - Capas convolucionales: 11
  - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
  - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
  - Learning rate: 0.001
  - Tamaño del batch: 128
  - Epochs: 200
  - *Dropout: 0.2, 0.3, 0.4, 0.5, 0*
  - Early stopping: si
  - Weight decay (regularizador l2): no
  - Batch normalization: no
  - Data augmentation: rotation_range = 10; width_shift_range = 0.1; height_shift_range = 0.1; horizontal_flip = True; zoom_range = 0.1; shear_range = 0.1

        - **Valor de función de coste y precisión en test: función de coste=2.3025, precisión=0.1000**
- CNNC10_23
    - Capas convolucionales: 11
    - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 128
    - Epochs: 200
    - *Dropout: 0.2, 0.3, 0.5, 0.7, 0*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: rotation_range = 10; width_shift_range = 0.1; height_shift_range = 0.1; horizontal_flip = True; zoom_range = 0.1; shear_range = 0.1
        - **Valor de función de coste y precisión en test: función de coste=0.4627, precisión=0.8555**
- CNNC10_24
    - Capas convolucionales: 11
    - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 256
    - Epochs: 200
    - *Dropout: 0.2, 0.3, 0.4, 0.5, 0*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: rotation_range = 20; width_shift_range = 0.2; height_shift_range = 0.2; horizontal_flip = True; zoom_range = 0.2; shear_range = 0.2
        - **Valor de función de coste y precisión en test: función de coste= 2.3026, precisión= 0.1000**
- CNNC10_25
    - Capas convolucionales: 11
    - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 256
    - Epochs: 200
    - *Dropout: 0.2, 0.3, 0.5, 0.7, 0*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: rotation_range = 20; width_shift_range = 0.2; height_shift_range = 0.2; horizontal_flip = True; zoom_range = 0.2; shear_range = 0.2

        - **Valor de función de coste y precisión en test: función de coste= 0.4150, precisión= 0.8707**
- CNNC10_26
    - Capas convolucionales: 11
    - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 256
    - Epochs: 200
    - *Dropout: 0, 0.2, 0.3, 0.5, 0*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: no
    - Data augmentation: rotation_range = 10; width_shift_range = 0.1; height_shift_range = 0.1; horizontal_flip = True; zoom_range = 0.1; shear_range = 0.1
        - **Valor de función de coste y precisión en test: función de coste= 0.3922, precisión= 0.8854**

Tal y como se ha podido comprobar, la combinación de data augmentation y dropout con los parámetros estudiados no produce mejoras significativas en los resultados, a la vez que provoca que, en varias ocasiones, el entrenamiento no sea capaz de reducir el valor de la función de coste o el error, y se quede estancado desde el primer momento. Por tanto, se va a utilizar la última de las pruebas, la 26, que tiene parámetros tanto de dropout como de data augmentation un poco menos agresivos, y que además ha mostrado unos resultados mejores, aunque bien puede haber sido producto de los factores aleatorios del proceso. Dado que la regularización en esta prueba es menor, el modelo es capaz de ajustarse de forma un poco más precisa a los datos, aunque sigue estando por debajo del 0.9. Sin embargo, los casos en donde se consigue aumentar este valor en entrenamiento también empeoran los resultados en validación, por lo que es más adecuado continuar por esta vía para intentar maximizar los resultados en validación. La mejora principal vendría dada, como se comentó anteriormente, por la incorporación de más parámetros a la red, especialmente mediante los últimos avances con módulos de inception y redes residuales, aunque esto sería a costa de tiempo de ejecución y más problemas de cuota de GPU con Google Colab.

VI. Weigth decay / batch normalization

Sobre estas pruebas, hay que especificar que el batch normalization se aplica después de cada capa convolucional a excepción de la de salida, y que los valores para el regularizador l2 se han reducido a 0.001 y 0.0001 en base a lo visto en la red anterior, donde con valores 0.01 el proceso de entrenamiento nunca era capaz de reducir el error. Para intentar evitar que vuelva a ocurrir que los procesos no puedan avanzar debido a una excesiva regularización, se van a explorar en algunas pruebas soluciones con parámetros de data augmentation algo más bajos.

- CNNC10_27
    - Capas convolucionales: 11
    - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 256

- Epochs: 200
- *Dropout: 0, 0.2, 0.3, 0.5, 0*
- Early stopping: si
- Weight decay (regularizador l2): no
- Batch normalization: si
- Data augmentation: rotation_range = 10; width_shift_range = 0.1; height_shift_range = 0.1; horizontal_flip = True; zoom_range = 0.1; shear_range = 0.1
    - **Valor de función de coste y precisión en test: función de coste=0.3355, precisión=0.9094**

- CNNC10_28
    - Capas convolucionales: 11
    - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 256
    - Epochs: 200
    - *Dropout: 0, 0.2, 0.3, 0.5, 0*
    - Early stopping: si
    - Weight decay (regularizador l2): no
    - Batch normalization: si
    - Data augmentation: rotation_range = 5; width_shift_range = 0.05; height_shift_range = 0.05; horizontal_flip = True; zoom_range = 0.05; shear_range = 0.05
        - **Valor de función de coste y precisión en test: función de coste=0.4322, precisión=0.9007**
- CNNC10_29
    - Capas convolucionales: 11
    - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 256
    - Epochs: 200
    - *Dropout: 0, 0.2, 0.3, 0.5, 0*
    - Early stopping: si
    - Weight decay (regularizador l2): 0.001
    - Batch normalization: si
    - Data augmentation: rotation_range = 10; width_shift_range = 0.1; height_shift_range = 0.1; horizontal_flip = True; zoom_range = 0.1; shear_range = 0.1
        - **Valor de función de coste y precisión en test: función de coste=0.6162, precisión=0.8857**
- CNNC10_30
    - Capas convolucionales: 11
    - Filtros por capa: (32,32), (64, 64), (128, 128), (256, 256), (512,512,10)
    - Algoritmo de optimización y parámetros específicos (si los hubiera): ADAM con beta_1=0.9 y beta_2=0.999
    - Learning rate: 0.001
    - Tamaño del batch: 256

- Epochs: 200
- *Dropout: 0, 0.2, 0.3, 0.5, 0*
- Early stopping: si
- Weight decay (regularizador l2): 0.0001
- Batch normalization: si
- Data augmentation: rotation_range = 5; width_shift_range = 0.05; height_shift_range = 0.05; horizontal_flip = True; zoom_range = 0.05; shear_range = 0.05
    - **Valor de función de coste y precisión en test: función de coste=0.5707, precisión=0.9038**

Tabla 4 Resultados de las redes CNN sobre CIFAR-10

CNNC10	**F de coste**	**Precisión**
1	0.7632	0.7615
2	1.8644	0.7760
3	1.4606	0.7643
4	1.5505	0.7570
5	1.7883	0.7508
6	0.8480	0.7640
7	1.8608	0.6769
8	3.0998	0.5474
9	0.9464	0.7503
10	1.5235	0.7696
11	2.3026	0.1000
12	1.3857	0.7663
13	1.7067	0.7549
14	0.6502	0.8288
15	0.7248	0.8116
16	0.5555	0.8317
17	0.5341	0.8364
18	0.5077	0.8564
19	0.6749	0.8411
20	0.4076	0.8791
21	0.3863	0.8838
22	2.3025	0.1000
23	0.4627	0.8555
24	2.3026	0.1000
25	0.4150	0.8707
26	0.3922	0.8854
27	0.3355	0.9094
28	0.4322	0.9007
29	0.6162	0.8857
30	0.5707	0.9038