

Práctica 2: Introducción a Keras

Evelyn G. Coronel
Redes Neuronales y Aprendizaje Profundo para Visión Artificial
Instituto Balseiro

(23 de octubre de 2020)

EJERCICIO 1

Considerando los valores de los inmuebles en Boston, se implementa un modelo simple de regresión lineal para predecir los precios, dadas las características de la propiedad. Este conjunto de datos es provisto por `scikit` y las características que tienen los datos son, por ejemplo, índice de criminalidad en el barrio, edad media de los habitantes de la propiedad o los mismos son de clase baja o media para arriba.

Previo al modelo, los datos son preprocesados restando la media y normalizando por la desviación estándar. Luego se implementa una red con una función de activación lineal, con el uso de bias y la función de costo MSE. La pérdida de la red para los datos de entrenamiento y validación se muestra en la Fig. 1, utilizando 'SGD' con una tasa de aprendizaje del 0.001.

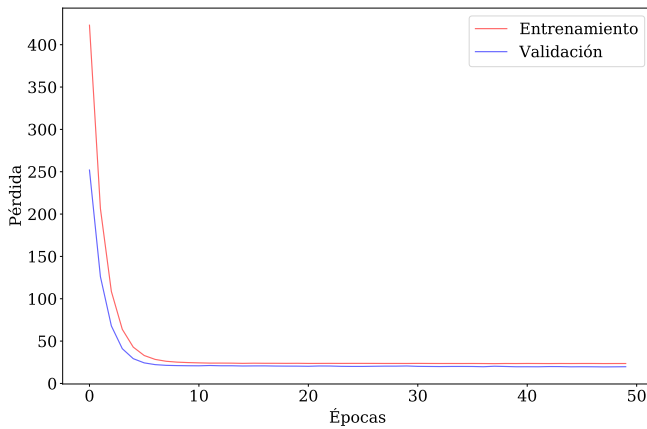


Fig. 1: Pérdida en función de las épocas para el ejercicio 1.

Para dimensionar la efectividad de la red, en la Fig. 2 se muestra una comparación entre los precios reales de los inmuebles con los precios predichos por la red para los datos de validación. La línea de referencia marca los puntos donde los precios reales y predichos son iguales, se observa que la red tiene una dispersión con respecto a la referencia.

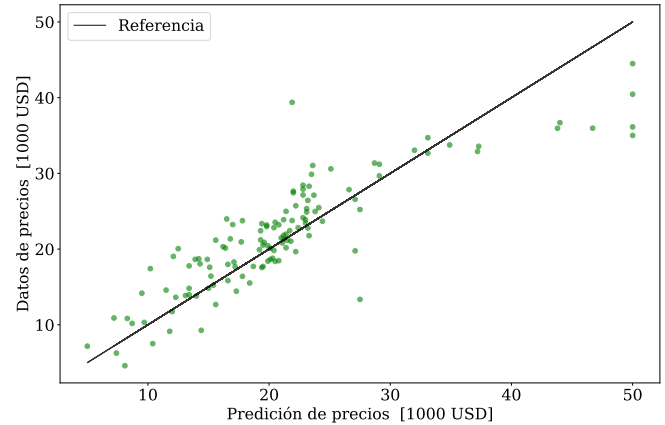


Fig. 2: Comparación de precios reales y predichos en los datos de validación

Otra forma de dimensionar la efectividad de la red, es comprobar si la red capta la correlación entre una característica y el precio del inmueble. Por ejemplo, en el Fig. 3 se muestran el precio de los inmuebles en función del porcentaje de personas en la clase baja, los datos en Boston dicen que los inmuebles en las zonas más pobres tienden a tener un menor precio, esta correlación también se ve reflejado en los datos predichos por la red.

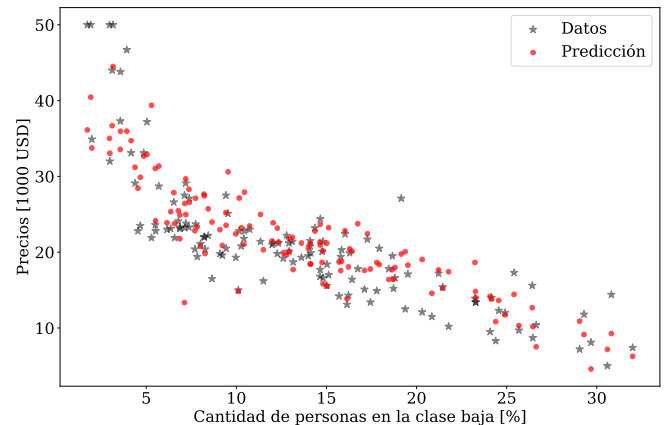


Fig. 3: Precio reales y predichos de los inmuebles en función del porcentaje de personas en la clase baja

EJERCICIO 2

Ejercicio 2-3

En este apartado se plantea clasificar el CIFAR-10 con la misma arquitectura que el problema 3 de la práctica anterior. Se usaron los mismos parámetros para comparar los resultados de **Keras** con la implementación de la práctica anterior. La arquitectura es la siguiente:

1. Capa de entrada
2. Capa Densa de 100 neuronas con bias y función de activación sigmoideal. Se utilizó el regularizador L2 con un parámetro de 10^{-4} .
3. Capa de salida con 10 neuronas con activación lineal y regularizador L2 con un parámetro de 10^{-4} .

El modelo usa una función de pérdida MSE y el optimizador SGD con una tasa de aprendizaje 0.003. Las curvas de precisión y pérdida se muestran en las Figs.4 y 5, se observa que los resultados obtenidos con los mismos parámetros en **Keras** superan a la implementación realizada en la práctica anterior.

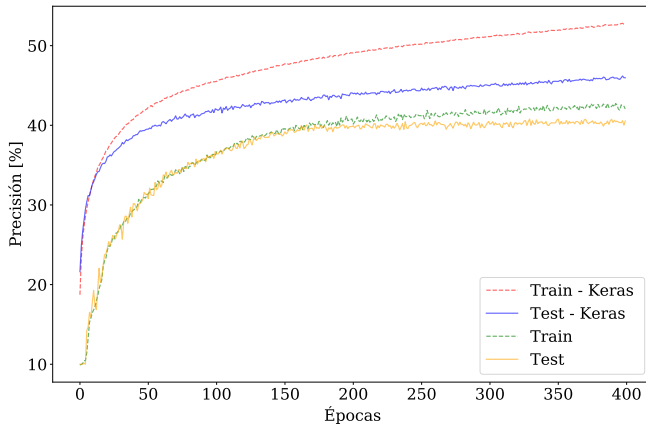


Fig. 4: Precisión de la implementación de ejercicio 3 de la práctica anterior y lo obtenido con Keras con los mismos parámetros.

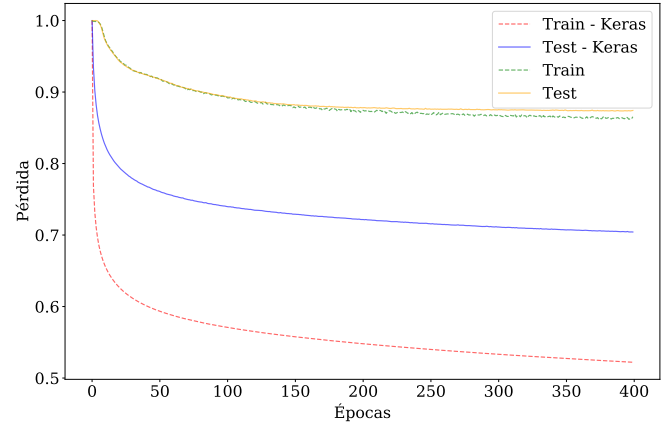


Fig. 5: Pérdida de la implementación de ejercicio 3 de la práctica anterior y lo obtenido con Keras con los mismos parámetros.

Ejercicio 2-4

En este apartado resolvemos el ejercicio 4 de la práctica anterior con la librería **Keras**, con la misma arquitectura que el apartado anterior. En uno de los puntos del ejercicio se pide usar la función de costo CCE, entonces al armar la red se utiliza la opción `from_logits=True` para que internamente pase la salida por una sigmoideal para convertir los datos en probabilidades. El optimizador es SGD con una tasa de aprendizaje de 0.003 y regularizadores L2 en ambas capas con un factor de 0.0001.

En las Fig.6 y 7 se muestran las curvas de precisión y pérdida normalizada del conjunto de validación usando la implementación propia y **Keras**. Se observa que la precisión con Keras es mayor que la implementación de la práctica anterior con los mismos parámetros, aunque se tiene overfitting usando la función de costo CCE. Se probó la CCE aumentando el factor del regularizador a 0.05 y se observa que se evita el overfitting en las primeras 500 épocas de entrenamiento.

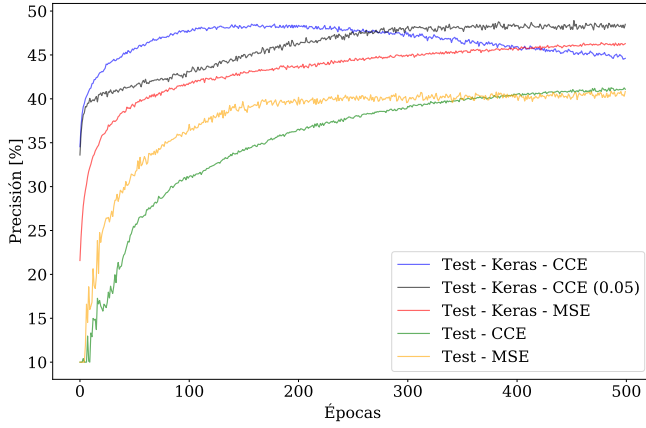


Fig. 6: Precisión de las curvas de validación para Keras y la implementación propia. Se cambió el valor de regularización en una ejecución para mejorar la precisión.

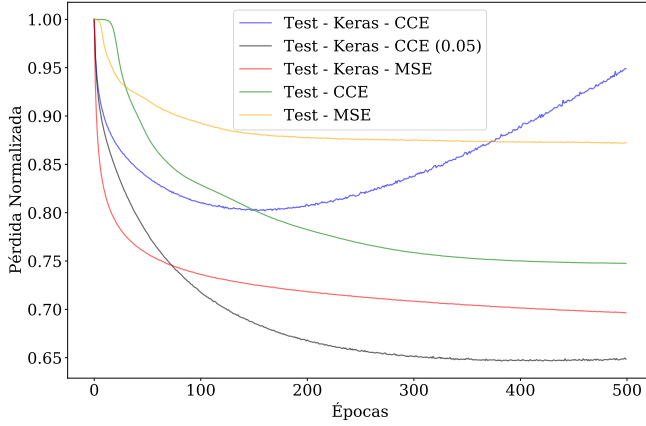


Fig. 7: Pérdida normalizada de las curvas de validación en función de las épocas.

Ejercicio 2-6

En este apartado del ejercicio se plantea resolver nuevamente el XOR de la práctica anterior. Para estos problemas se utilizaron una tasa de aprendizaje de 0.1 y un tamaño de batch de 1, la función de pérdida utilizada fue la función MSE en ambos casos, para utilizar la métrica `BinaryAccuracy` en Keras, los datos de entrenamiento son 0 y 1 en cambio en la implementación propia escrita para la práctica anterior se utilizaron -1 y 1 respectivamente, además de una métrica especializada para una salida binaria entre -1 y 1.

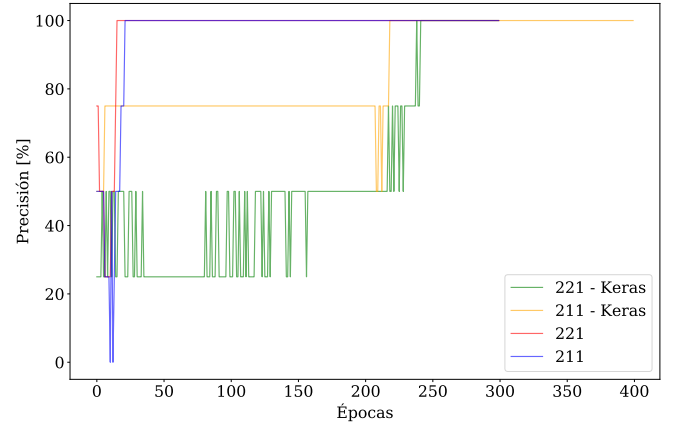


Fig. 8: Precisión en función de la épocas del problema XOR para distintas arquitectura y implementaciones de redes densas.

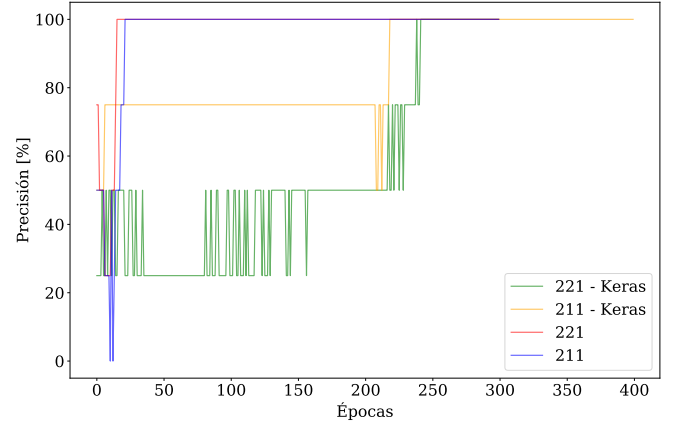


Fig. 9: Pérdida en función de las épocas del problema XOR para distintas arquitectura y implementaciones de redes densas.

MMMEJERCICIO 3

MMMEJERCICIO 4

EJERCICIO 5

El mapeo logístico es una función del tipo $x(t+1) = rx(t)(1-x(t))$, conocida por presentar caos dependiendo del valor de r . Se dice que una función tiene un comportamiento caótico si dos valores iniciales cercanos, la diferencia entre las salidas de los mismo crece exponencialmente con la cantidad de iteraciones. En este ejercicio el mapeo logístico tiene un valor $r = 4$ donde se presenta un comportamiento caótico salvo algunos números.

Podemos hacer una red para los siguientes dos escenarios:

1. Podemos hacer que la red aprenda la función $f(x) = 4x(1 - x) = 4x - 4x^2$
2. Podemos intentar que la red aprenda el comportamiento caótico del mapeo logístico entrenándola con un conjunto de $\{x(0), t\}$ distintos.

Escenario 1

La red tiene como objetivo aprender la función $f(x) = 4x(1 - x) = 4x - 4x^2$, para eso se entrena con números en el rango $[0, 1]$ como entrada y como salida esperada $f([0, 1])$.

En este caso se cambio la cantidad de ejemplos de entrenamiento, donde la cantidad de ejemplos de validación es la mitad de los ejemplos de entrenamiento. En la Fig. ?? se muestra la pérdida normalizada con respecto al mayor valor, para distintas cantidades de ejemplos. Se puede ver que para 10 ejemplos de entrenamiento y 5 de validación, la red tiende a tener overfitting y en cambio para (25, 50) la red tiene una pérdida mayor que (12, 24). En varias ejecuciones, la red entrenada con (25, 50) también presentaba overfitting.

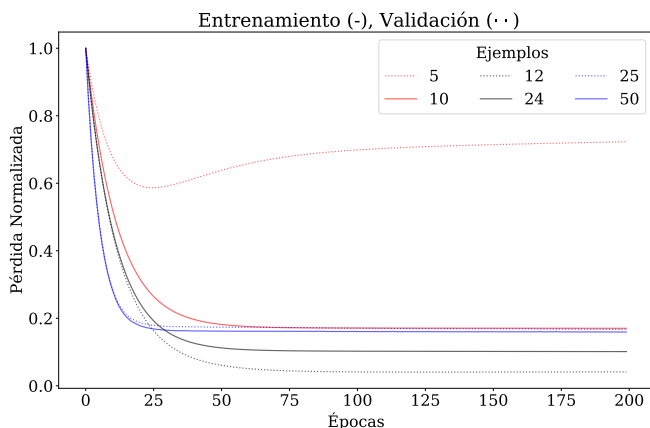


Fig. 10: Pérdida de la validación y entrenamiento para distintas cantidades de ejemplos.

Escenario 2

Este escenario no es realizable con la arquitectura que se pide, ya que la entrada debería tener dos dimensiones, una para $x(0)$ y otra para t . Si entrenáramos a la red con un par $\{x(0), x(t)\}$ con t cuales quiera, por el comportamiento caótico de la función, la red no puede predecir la salida que puede tener $x(0) + \epsilon$.

EJERCICIO 6

En este ejercicio se implementa una red fully-connected para predecir si un paciente puede tener o no diabetes dado su historial médico. La base de datos utilizado es público y fue utilizado por primera vez en el siguiente trabajo [1].

Para algunos pacientes en este conjunto de datos se tiene información incompleta de su historial médico, por ejemplo un dato es la presión arterial que en algunos casos es 0. Dado que solo se tiene información de 768 pacientes, se optó por reemplazar los valores nulos por la media del resto de los pacientes las siguientes columnas: glucosa en sangre, presión arterial, insulina e índice de masa corporal.

La red utilizada tiene la siguiente arquitectura:

- Capa de entrada.
- Capa oculta de 10 neuronas con función de activación lineal y con bias. Primeramente se usó una función de activación del tipo ReLU, pero se cambió por una lineal dado que la ReLU de mejores resultados. Esto puede deberse a que la ReLU no aprende la correlación negativa que pueden tener dos aspectos del historial médico. Se utilizó 10 neuronas para que la red aprenda las posibles correlaciones entre los datos del historial clínico. Para lograr esto la capa oculta tiene que tener al menos 8 neuronas, se optó por 10 porque funcionaba y por encima de 10 la red puede tener parámetros redundantes. También se probó agregando una capa más pero la red no mejoró su rendimiento
- Capa de salida con activación ReLU: Se optó por una ReLU por encima de la lineal porque se espera que la salida sea 0 si la red predice que no va a desarrollar diabetes, y 1 en caso contrario. También se utilizó un función sigmoidal pero la red presentaba un overfitting, incluso con la regularización. Se utilizó el optimizador SGD con una tasa de aprendizaje de 0.001 con la función de pérdida

Para verificar que la red tiene capacidad de generalización dada la cantidad de datos, se usó el método k-fold de la siguiente manera: se toma el conjunto de datos y se separa el conjunto de entrenamiento y validación 5 veces y se entrena la red para cada vez. Esto se implementó con el módulo `model_selection.KFold` de `sklearn`. En cada iteración, el conjunto de entrenamiento y validación tenían entre 614 – 615 y 154 – 153 pacientes respectivamente, y los pesos del entrenamiento anterior no se tienen en cuenta.

El preprocesado de los datos se realizó en cada entrenamiento, el mismo consistió en resta los datos por la media y dividir por la desviación estándar.

En las Figs 11 y 12 se presentan la media de la precisión y pérdida para vez que se entrena la red. Para tener

una referencia de como varían en cada entrenamiento en función de las épocas, se muestra la desviación estándar en cada época con el sombreado alrededor de la línea más oscura. Se observó que no se corrigen los datos nulos la red tiene una precisión menor, alcanzando una precisión media de $\sim 72\%$.

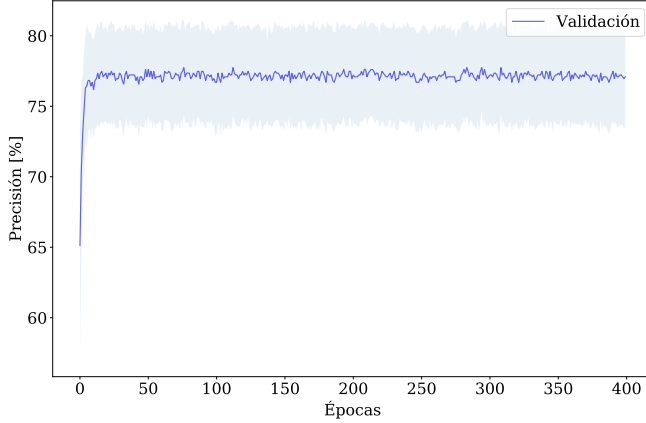


Fig. 11: Precisión de la red en función de las épocas. Se muestra una estimación del error de la precisión por época. La precisión media alcanza un valor de 77.2%

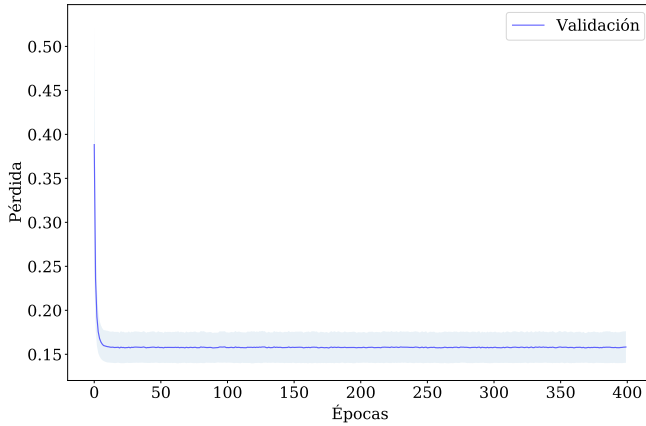


Fig. 12: Pérdida de la red en función de las épocas. Se muestra una estimación del error de la precisión por época.

EJERCICIO 7

En este ejercicio se implementa una red con capas convolucionales para eliminar el ruido de las imágenes de los dígitos del MNIST. Para esto se implementó un autoencoder que toma como entrada la imagen con ruido y como salida se tiene la imagen sin el ruido. Para lograr esto la red tiene dos etapas: *encoding* y *decoding*. El *encoding*

toma la imagen de $28 \times 28 \times 1$ con un único canal de color y la lleva hasta una imagen de $7 \times 7 \times 32$. Mientras que en el *decoding* se lleva la imagen de $7 \times 7 \times 32$ a una imagen de $28 \times 28 \times 1$. Se modificó las dimensiones de la imagen del encoding a 8 y 16 para disminuir la cantidad de parámetros pero no obtuvo la mejor resolución con 32.

El optimizador utilizado fue el *adam* con una tasa de aprendizaje de 0.001, se probó con SGD pero se obtuvo mejores resultados con la primera. La función de pérdida utilizada fue CCE por lo que la capa de salida fue una sigmoideal. La capas ocultas tiene funciones de activación ReLU porque las imágenes tienen un valor entre 0 y 1.

En las Figs. 13 y 14 se observan la precisión y pérdida en función de las épocas. Para mostrar un ejemplo del uso de esta red, en la Fig.15 se muestra un ejemplo de ejemplos de validación con el ruido y luego de ser filtradas por la red entrenada. Se observa que el ruido de la imagen del dígito disminuye notablemente.

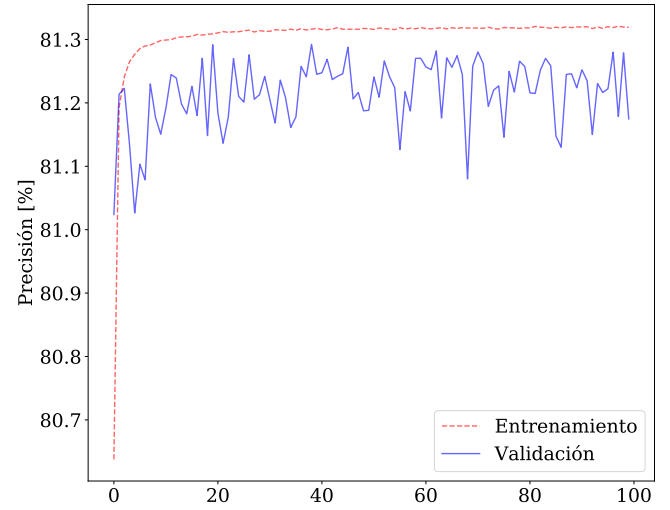


Fig. 13: Precisión de la red en función de las épocas. Se alcanza un precisión del 81.2% con el conjunto de validación

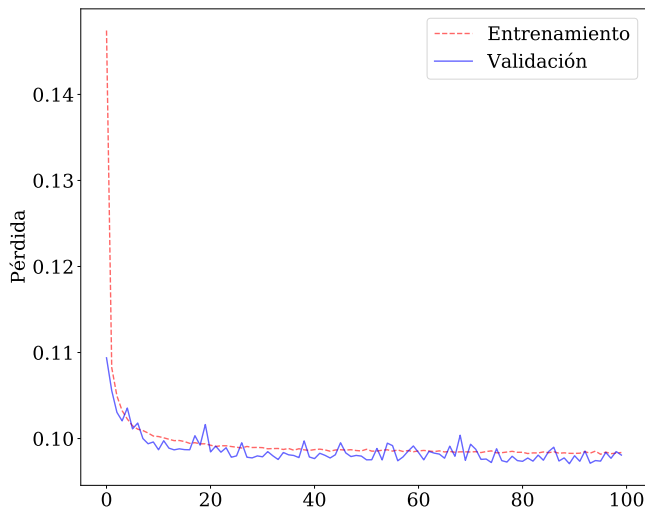


Fig. 14: Pérdida de la red en función de las épocas.

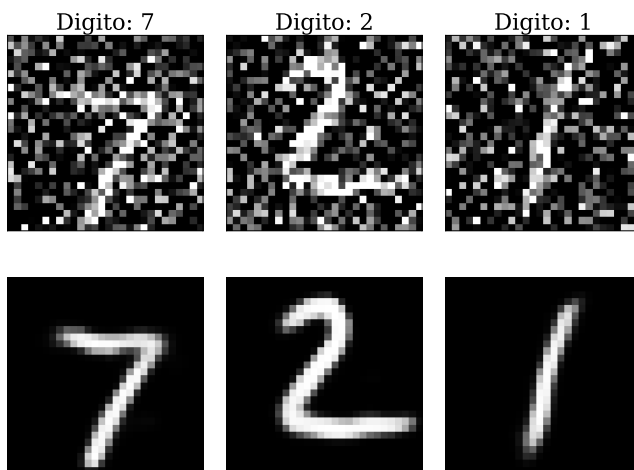


Fig. 15: Ejemplo de entrada y salida de la red posterior al entrenamiento. Se observa que el ruido se reduce y se obtiene una imagen clara del dígito en cuestión.

EJERCICIO 8

En este ejercicio se busca clasificar la base de datos de las imágenes de dígitos del MNIST. Se implementaron dos tipos de redes: densa y convolucional. La red densa tiene una arquitectura similar a la utilizada para clasificar las imágenes de CIFAR-10. Como se muestra a continuación:

Layer	Salida	Parámetros
-------	--------	------------

Dense (ReLU) (100) 78500

Dense (Softmax) (10) 1010

que tiene un total de 79510 parámetros. Para la red convolucional se usaron capas convolucionales y de Max Pooling, además de capa Dropout donde ignoramos un 20 % de la entrada a esa capa. Esta capa es utilizada para aumentar la generalización de la red evitando el overfitting. La lista a continuación tiene más detalles de la red:

Layer	Salida	Parámetros
Conv2D	(24, 24, 30)	780
MaxPooling2D	(12, 12, 30)	0
Conv2D	(10, 10, 10)	2710
MaxPooling2D	(5, 5, 10)	0
Dropout (0.2)	(5, 5, 10)	0
Flatten	(250)	0
Dense (Softmax) (10)		2510
Total parámetros: 6,000		

Se utilizó el optimizador SGD con una tasa de aprendizaje de 0.001, también se probó utilizando el optimizador adam pero la red tenía a tener overfitting en la red densa, incluso con una función de regularizador con parámetro de 0.1. La función de pérdida es CCE y la última capa en ambas redes es densa con una activación softmax.

En las Figs. 16 y 17 se muestran las curvas de precisión y pérdida normalizada para los datos de entrenamiento y validación para las redes mencionadas. La red convolucional tiene una mejor precisión final de 97.9% en el conjunto de validación con respecto a la densa que alcanza un 93.67%. Otra diferencia importante es la cantidad de parámetros de las redes, la red densa utiliza 13 veces más parámetros de la red convolucional.

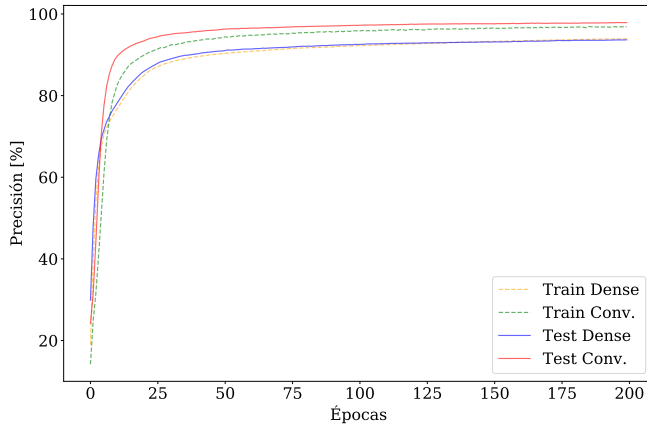


Fig. 16: Precisión de la red en función de las épocas

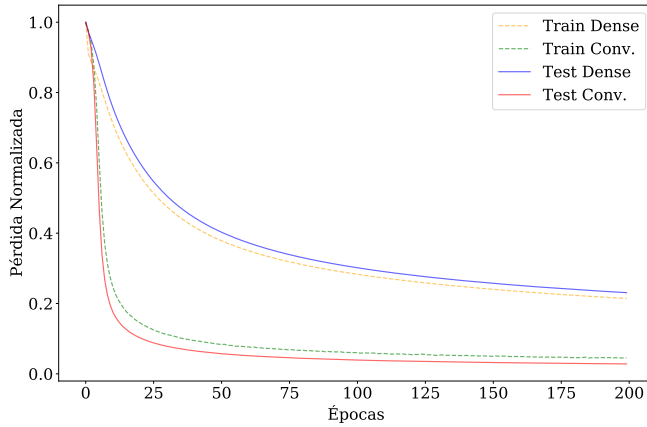


Fig. 17: Pérdida normalizada con respecto al máximo de las redes densa y convolucionales en función de las épocas.

Otro aspecto interesante de la red convolucionales es que también tienen en cuenta la correlación espacial de los puntos en la imagen. En el caso de la red densa, donde perdemos información sobre los píxeles vecinos al hacer que las imágenes sean unidimensionales. Tenemos información en una dirección pero no podemos sacar información de las curvas que puede tener un dígito por ejemplo.

EJERCICIO 9

Utilizando la red convolucional anterior, volvemos a clasificar las imágenes de los dígitos del MNIST, salvo

que ahora hacemos una permutación de los valores de intensidad de los píxeles.

En las Figs. 18 y 19 se muestran las curvas de precisión y pérdida normalizada para la red con los datos originales y permutados. La red entrenada los datos permutados alcanza una precisión de 93.8 %, que es menor con respecto a la red entrenada con los datos originales. Esto se debe a que se pierde información de la correlación especial de los píxeles de un dígito, como el borde, curva y forma. A pesar de esto, la red puede encontrar correlaciones entre los puntos permutados y aprender a identificar los dígitos modificados.

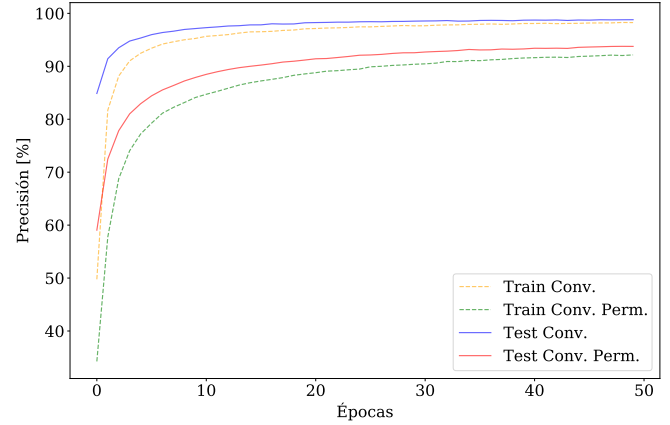


Fig. 18: Precisión de la red en función de las épocas

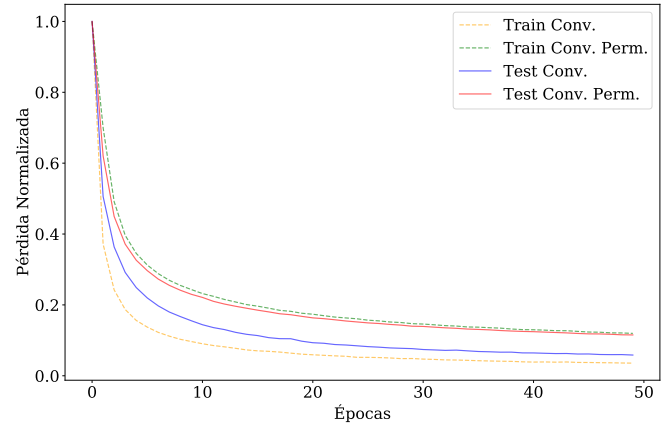


Fig. 19: Pérdida normalizada con respecto al máximo en función de las épocas.

MMMEJERCICIO 10

-
- [1] Smith,J.W., Everhart,J.E., Dickson,W.C., Knowler,W.C., & Johannes,R.S. (1988). Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Symposium on Computer Applications and Medical Care* (pp. 261–265). IEEE Computer Society Press.