UNIVERSIDAD DE GRANADA

Práctica 1. Reconocimiento óptico de caracteres MNIST mediante redes neuronales.

Inteligencia Computacional

Máster Profesional en Ingeniería Informática

Curso académico 2022/2023

Autor

José Alberto Gómez García

Índice de contenidos

1. Introducción.

Una red neuronal artificial es un tipo de modelo computacional inspirado en la estructura y funcionamiento de las redes de neuronas en el cerebro de los seres vivos. Estas redes se componen de multitud de unidades computacionales, llamadas neuronas, que se conectan entre sí en diferentes capas. Las redes neuronales se utilizan ampliamente en el aprendizaje automático y el procesamiento del lenguaje natural, entre otros campos.

Una de las aplicaciones de las redes neuronales artificiales es el reconocimiento y clasificación de objetos en imágenes. En el marco de esta práctica, especificaremos, entrenaremos y evaluaremos diversos modelos de redes neuronales cuyo objetivo será el reconocimiento de dígitos numéricos manuscritos. El objetivo final, como puede uno imaginar, es proponer un modelo capaz de predecir con la mayor precisión posible el número contenido en una determinada imagen.

Concretamente, para evaluar el rendimiento de las diversas redes neuronales propuestas utilizaremos la base de datos MNIST. Este conjunto de datos está compuesto de 70.000 imágenes en blanco y negro y resolución de 28x28 píxeles habiendo en cada una de ellas un dígito manuscrito. Aunque siempre es así, se suelen usar 60.000 de estas imágenes para el entrenamiento de las redes y las 10.000 imágenes restantes para las pruebas.

En la imagen que se adjunta a continuación se muestran algunos de los dígitos contenidos en la base de datos MNIST. Debajo de los mismos, se muestra la etiqueta correspondiente.

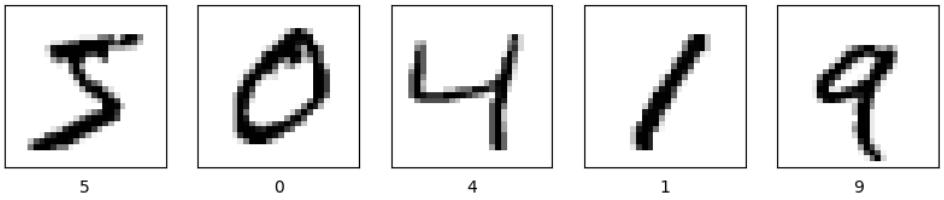


Imagen 1. 5 dígitos de la base de datos MNIST junto a sus etiquetas.

2. Herramientas utilizadas.

2.1. Hardware

Para el entrenamiento y ejecución de las distintas redes neuronales se ha usado un equipo de sobremesa con las siguientes especificaciones hardware:

- **Procesador:** Intel® Core™ i7-8700K CPU 6-Cores / 12-Threads (3.7 – 4.7 Ghz).
- **Memoria RAM:** G.Skill Trident Z RGB 32 GB (4x8GB) DDR4 a 3200 Mhz.
- **Tarjeta gráfica:** NVIDIA (Gigabyte) GeForce® GTX 1080 G1 Gaming 8G.
- **Almacenamiento:** Samsung 970 Evo Plus NVMe M.2 SSD 1TB.

2.2. Software

Para desarrollar el software necesario para la realización de esta práctica, se han utilizado las siguientes herramientas software:

- **Lenguaje de programación:** Python 3.10.6. Se ha creado un entorno virtual para aislar los paquetes que necesitemos de los presentes en la instalación del Python del sistema.
- **Sistema operativo:** Ubuntu 22.04.1 mediante WSL2 sobre Windows 11 Pro.
- **Editor de código:** Visual Studio Code.
- **Librerías de Python:**
    - **Keras (2.11.0).** Es una librería de aprendizaje profundo de código abierto escrita en Python. Está diseñada para facilitar a los desarrolladores construir y entrenar redes neuronales, al ofrecer una interfaz sencilla y fácil de usar, que puede ser empleada por encima de otras bibliotecas, como Tensorflow, ya que actúa como "front-end" de dichas bibliotecas.
    - **Tensorflow (2.11.0)**. Es una biblioteca de código abierto, desarrollada por Google, para entrenar y desplegar modelos de aprendizaje automático. Este framework es usado como "backend" por Keras, al trabajar a un nivel de abstracción más bajo. Su uso se recomienda para proyectos grandes y/o complejos en los que se necesita un control de grano más fino. Tanto esta biblioteca como Keras se ha instalado haciendo uso del comando "pip install tensorflow-gpu".
    - **Matplotlib (3.6.2)**. Es una biblioteca de código abierto para la generación de gráficos en Python, ampliamente usada en el ámbito científico y del análisis de datos.
    - **Numpy (1.23.5)**. Librería que agrega soporte para arrays y matrices multidimensionales, así como una gran colección de funciones matemáticas. Está diseñada para ser utilizada en el cómputo científico y suele utilizarse en tareas de análisis de datos y aprendizaje automático.
- **Otros:** para acelerar los cálculos mediante la tarjeta gráfica (GPU) se han instalado el software propietario NVIDIA CUDA® en su versión 11.2. Por otra parte, para acelerar el entrenamiento e inferencia de las redes neuronales mediante GPU, se ha instalado el software propietario NVIDIA cuDNN® en su versión 8.2. Para más detalles relativos al proceso de descarga e instalación de dicho software, consulte [2] y [3].

3. Implementaciones realizadas.

3.1. Detalles comunes.

El programa realizado incluye todos los modelos de redes neuronales que se detallarán en apartados posteriores, de manera que el usuario puede seleccionar el entrenamiento y evaluación de cada uno de ellos mediante un menú textual.

Nada más arrancar el programa, se comprueba si existe una tarjeta gráfica que pueda usar. De haberla, muestra su modelo y las versiones de NVIDIA CUDA® y NVIDIA cuDNN® instaladas. En caso contrario, se indica que se hará uso de la CPU, el cual conllevará tiempos de ejecuciones más largos, los cuales pueden llegar a ser prohibitivos en el caso de los modelos que hacen uso de redes convolutivas.

Acto seguido, se cargan las imágenes de entrenamiento y prueba, junto a sus etiquetas, del conjunto MNIST, el cual viene integrado en Tensorflow.

Si se analiza rápidamente el código adjunto en esta práctica, se puede apreciar que en ningún momento creamos capas de entrada (de tipo “Input”) explícitamente ya que con el parámetro “input_shape” de las capas “Flatten” y “Conv2D” se puede indicar la dimensión de los datos de entrada, de manera que Keras crea automáticamente la capa de entrada.

3.2. Red neuronal simple. Perceptrón.

Dado que la práctica sigue un enfoque evolutivo, en el que primeramente probamos aproximaciones más sencillas, comencemos por el modelo más simple, un perceptrón.

Este modelo solamente cuenta con dos capas, una de entrada y una de salida.

- La capa de entrada tiene como tarea el convertir una imagen, matriz bidimensional, de 28x28 píxeles en un vector de 784 píxeles. Para ello, añadimos una capa de tipo “Flatten”, que como su nombre indica, “aplana” la imagen para convertirla en una estructura de datos unidimensional. Deberemos hacer esto antes de pasar la imagen a una capa totalmente conectada.
- En la capa de salida tendremos 10 neuronas, una por cada tipo de dígito que queremos que la red clasifique, cuya función de activación es sigmoidal. Las neuronas de esta capa se encuentran totalmente conectadas con la capa anterior. En Keras, las capas totalmente conectadas reciben el nombre de “Dense layers”.

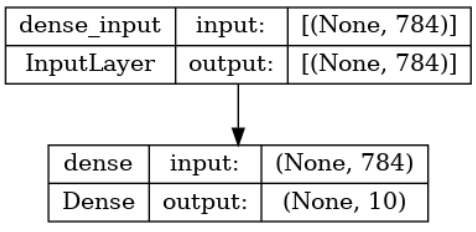


Imagen 2. Modelo del perceptrón.

En la imagen adjunta a la izquierda, se puede ver una representación gráfica del modelo del perceptrón implementado. Esta se ha obtenido haciendo uso de la función “plot_model” incorporada en Keras.

Para compilar el modelo se ha hecho uso del optimizador “Adam”, dado que es el que mejores resultados parece ofrece en la práctica tras una serie de pruebas, lo cual concuerda con lo visto en clase de teoría. Como función de pérdida se ha hecho uso de “sparse categorical crossentropy”, al ser la que mejores resultados parece ofrecer de entre aquellas disponibles en Keras.

Como veremos posteriormente, esta es la implementación que peores resultados prácticos ofrece, lo cual era de esperar debido a su sencillez. Sin embargo, el porcentaje de acierto, de un **92.81%** sobre el conjunto de prueba tras 10 épocas de entrenamiento, puede ser más alto de lo que uno esperaba inicialmente. Este modelo, aparentemente tan simple, cuenta con 7.850 parámetros ajustables.

3.3. Red neuronal multicapa con 256 neuronas en la capa oculta.

Como segundo modelo, se uso el perceptrón simple y se le añadió una capa oculta con 256 neuronas, de forma que pasamos a tener una red neuronal artificial multicapa. Esta capa está totalmente conectada a las capas anteriores y posteriores. Además, la capa de salida pasa a utilizar una función de activación de tipo “softmax”.

En este caso, añadimos una decisión de diseño más, cuantas neuronas emplear en la capa oculta. Dado que tenemos 10 clases por clasificar e imágenes de 784 píxeles, cualquier número entre ambos valores podría ser una posibilidad. Siguiendo las recomendaciones de algunos sitios web, y del propio guion de la práctica, se probó a usar 128 y 256 neuronas. Se obtuvieron mejores resultados con la capa que hacía uso de 256 neuronas, así que se decidió mantener este valor. A pesar de tomar esta decisión, cabe destacar que los resultados no se diferencian demasiado, por lo que en modelos más complejos recurriremos a capas ocultas totalmente conectadas de 128 o menos neuronas.

El modelo propuesto, el cual ha sido compilado con el optimizador “Adam” y función de pérdida “sparse categorical crossentropy”, proporciona una precisión sobre el conjunto de prueba del **98.1%** después de 10 épocas de entrenamiento.

En un intento de mejorar los resultados obtenidos, se añadió una capa “Dropout” entre las dos capas totalmente conectadas de la red. Así, desactivamos un determinado porcentaje de las neuronas (de forma aleatoria) durante el entrenamiento, en un intento de evitar el sobre-aprendizaje.

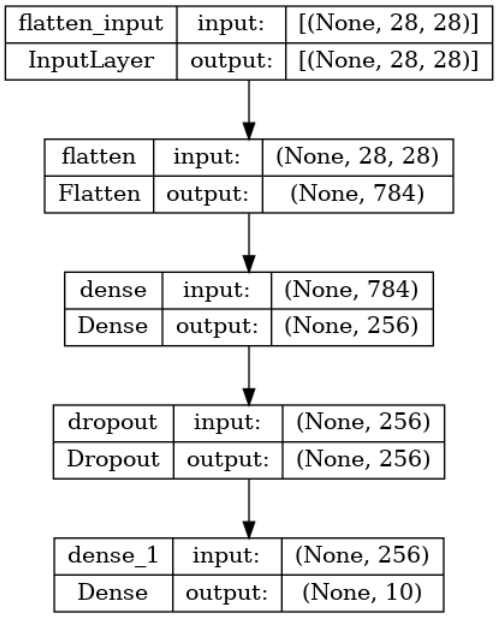


Imagen 3. Modelo de la 1º red neuronal multicapa.

El modelo con la inclusión de este tipo de capa se muestra en la imagen adjunta a la izquierda.

Desactivando un 20% de las neuronas, haciendo uso de lotes de 128 imágenes y un periodo de entrenamiento de 20 épocas, se consiguió una precisión del **98.41%**, un resultado algo superior al anterior, y necesitando apenas un tercio del tiempo de entrenamiento. Como se puede ver, los resultados obtenidos gozan ya de gran precisión. El número de parámetros escala rápidamente, alcanzando los 203.530.

Más detalles de los experimentos realizados se podrán encontrar en el cuarto capítulo de este documento.

3.4. Red neuronal convolutiva (v1) entrenada con gradiente descendente estocástico.

Siguiendo el orden de modelos propuestos en el guion de la práctica, se decide implementar una red neuronal convolutiva entrenada con gradiente descendente estocástico (SGD).

Como es obvio, se mantiene la capa de entrada de las imágenes, pero en esta ocasión, y a diferencia de la anteriores, necesitaremos especificar 3 dimensiones. Esto es debido a que, según la documentación de Keras, los filtros convolutivos se aplican sobre tensores tridimensionales, por lo que necesitamos especificar el número de canales que tiene la imagen. Dado que las imágenes del conjunto de datos MNIST son en blanco y negro, estas tienen un único canal. Así, el “input_shape” será de 28x28x1.

Para aplicar las operaciones de convolución sobre la imagen deberemos añadir capas de tipo “Conv2D”, cuyos parámetros propios más importantes son los siguientes:

- El número de filtros determina el número de canales de salida en la convolución. Esto suele establecerse en una potencia de 2, como 32 o 64.
- El tamaño del “kernel” es el tamaño del filtro de convolución y se suele establecer en un número entero impar pequeño, como 3 o 5. No tiene por qué ser rectangular.
- El valor “stride” indica cuanto se desliza el filtro sobre la imagen de entrada en cada paso. Este valor se suele establecer en uno o dos píxeles.
- El “padding” determina cómo se rellena la imagen antes de aplicar una operación de convolución sobre los extremos.

Aunque no es obligatorio, las operaciones de convolución se suelen acompañar de operaciones que reducen el tamaño de una imagen mediante técnicas de “pooling”. Esto permite mejorar la capacidad de captar patrones en la imagen, al agrupar los píxeles más cercanos en cada subregión de la imagen. En Keras existen dos tipos de estas capas, “MaxPooling” y “AveragePooling”; en función de si asignan a la región el valor máximo o el promedio de la misma.

En el modelo propuesto en este apartado, cuyo resumen a modo gráfico se encuentra a la izquierda, se emplean capas convolucionales con tamaño de “kernel” 3x3 y funciones de activación ReLu. La primera capa convolucional utiliza un tamaño de filtro de 32, mientras que las otras dos utilizan tamaño de filtro 64.

Después de cada una de las dos primeras capas convolutivas, se emplea una capa “MaxPooling2D” que agrupa en un único valor una región de 2x2 píxeles. Después de la tercera capa convolutiva no se utiliza capa de “pooling”.

Posteriormente, convertimos la imagen resultante en una estructura unidimensional mediante la capa “Flatten”, ya que es en este momento cuando pasaremos a utilizar capas totalmente conectadas.

Como anticipábamos en el apartado anterior, utilizamos una capa oculta totalmente conectada de 128 neuronas. Al aplicar convoluciones y “pooling” sobre la imagen esta ha pasado de tener un tamaño de 784 píxeles a 576 píxeles, por lo que reducimos el número de neuronas de esta capa oculta a 128 para que cada entrada esté conectada a un mínimo de 3 neuronas, al igual que antes. No obstante, se podría haber optado por mantener las 256 neuronas de apartados anteriores; aunque usando 128 neuronas parecemos obtener mejores resultados.

Aplicamos una capa “Dropout” en la que desactivamos aleatoriamente el 50% de las neuronas, con el objetivo de evitar sobre-aprendizaje. Se probó a desactivar entre un 20 y 50% de las neuronas, siendo el 50% el valor que mejor resultados proporcionó.

Finalmente, y al igual que en todos los modelos, tendremos una capa de salida totalmente con 10 neuronas, una por posible clase a clasificar. En este modelo, al igual que en los siguientes, esta capa tiene una función de activación de tipo “softmax”.

Curiosamente, a pesar de que esta red es aparentemente más compleja que la anterior, cuenta con menos de la mitad de los parámetros modificables durante el entrenamiento. En esta ocasión, son 93.322 (en lugar de más de 200.000).

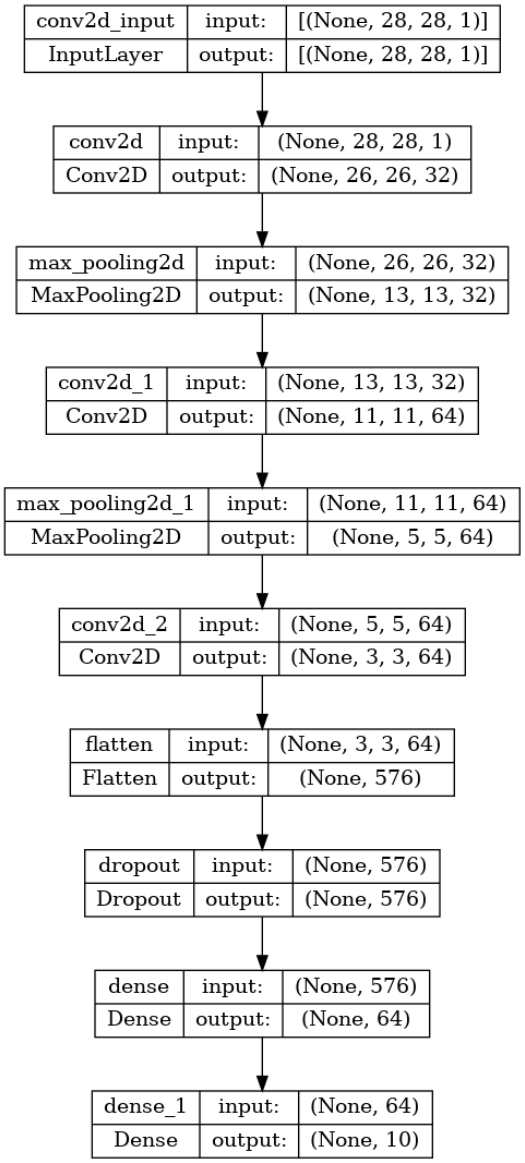


Imagen 4. Modelo de la 1º red neuronal convolutiva.

A pesar de que "Adam" es un optimizador basado en la técnica del gradiente descendente estocástico (SGD), decidimos utilizar el optimizador SGD base, cuya implementación está disponible en Keras.

En esta ocasión, tuvimos que aplicar "One-hot encoding" a las etiquetas, de forma que estas pasaran a ser vectores de largo N (donde N es el número de clases posibles, 10 en nuestro caso), con un único valor verdadero. Si no, el modelo devolvía un error. En consecuencia, la función de pérdida utilizada pasa a ser "categorical crossentropy".

Tras un entrenamiento de 10 épocas con tamaño de lote de 32 imágenes y "Dropout" del 50% se consiguió alcanzar una precisión sobre el conjunto de prueba del **98.88%.** El tiempo de entrenamiento necesario fue de 113 segundos aproximadamente.

Una precisión prácticamente idéntica (del **98.90%**) puede ser alcanzada si se utilizan 20 épocas, tamaño de lote de 32 imágenes y se prescinde de la capa de "Dropout", pero el tiempo de entrenamiento aumenta hasta los 296 segundos.

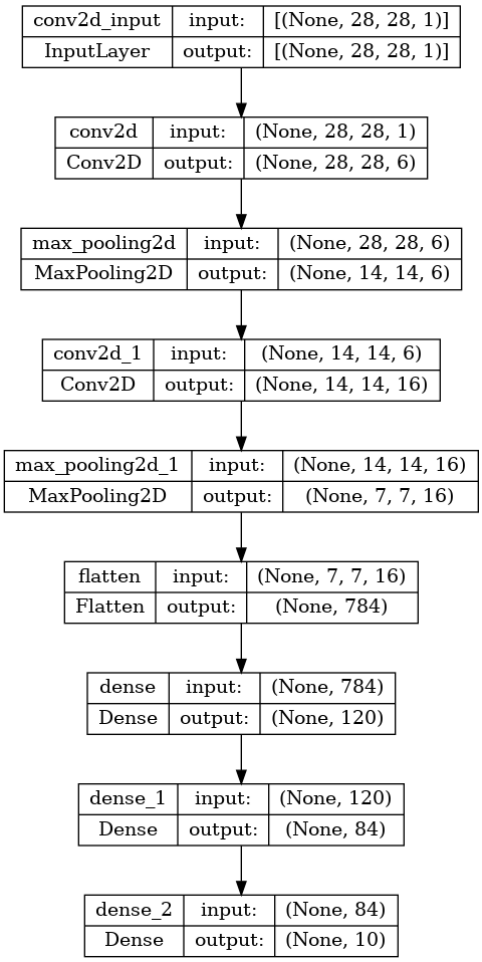
Como se puede apreciar, los resultados conseguidos al hacer de redes neuronales convolutivas relativamente simples son bastante buenos, de manera que el modelo apenas falla en 110 imágenes de las 10.000 del banco de prueba. Puede que al lector le parezcan muchas, pero como veremos en apartados posteriores, hay algunas imágenes difíciles de clasificar hasta para un humano.

3.5. LeNet-5 (con optimizador Adam).

Siguiendo con los experimentos en el campo de las redes neuronales convolutivas, he decidido implementar la red *LeNet-5* [9]. Esta red fue desarrollada originalmente por Yann LeCun, Léon Bottou, Yoshua Bengio y Patrick Haffner en 1998, y promete proporcionar una precisión entre el **99.05 y 99.2%** sobre el conjunto de datos MNIST.

En un primer vistazo, la arquitectura de la red puede parecer similar a la utilizada en el apartado anterior, pero existen algunas diferencias clave. Enumerémoslas:

- Únicamente se emplean dos capas convolucionales. La primera de ellas tiene un tamaño de filtro 6, mientras que en la segunda el tamaño del filtro es de 16. En el modelo anterior, los filtros eran de tamaño 32 y 64 en las dos primeras capas. Adicionalmente, el tamaño del "kernel" empleado en *LeNet-5* es de 5x5, mientras que en la red anterior los "kernels" tenían dimensiones 3x3.
- No existe capa de regularización "Dropout".
- En lugar de tener una capa oculta totalmente conectada de 128 (o 256) neuronas y una capa de salida con 10 neuronas, *LeNet-5* hace uso de dos capas ocultas totalmente conectadas de 120 y 84 neuronas respectivamente, previas a la capa de salida, la cual permanece sin cambios.



La implementación original propuesta en 1998 de esta red neuronal artificial hacia uso del algoritmo de optimización SGD "base u original". En un intento de mejorar los resultados obtenidos, haremos uso del optimizador "Adam", una versión mejorada del SGD "original". Como función de pérdida haremos uso una vez más de "categorical crossentropy".

Si entrenamos la red durante 20 épocas con un tamaño de lote de 128 imágenes, conseguimos obtener una precisión del **98.99%** sobre el conjunto de prueba tras 88.5 segundos de ejecución del programa.

La precisión obtenida se encuentra cerca, aunque ligeramente por debajo, de la que obtuvo el equipo de Yann LeCun en su día. La ganancia de precisión obtenida respecto de la red convolucional presentada en el apartado anterior apenas es del **0.09%.**

Imagen 5. Modelo de la implementación de LeNet-5.

Este modelo de red neuronal cuenta con 107.786 parámetros, un 10% más que el modelo anterior, aproximadamente.

3.6. Red neuronal convolutiva con 2 y 3 capas convoluciones consecutivas, "dropout" y "AveragePooling".

En un intento de bajar del 1% de tasa de error se buscó en Internet modelos más precisos que los usados hasta el momento. En el marco de los ejemplos de la documentación de Keras [8] [13], el equipo de desarrollo del framework propone una red neuronal que hace uso de dos capas convolutivas consecutivas, la primera de ellas con tamaño del filtro 32 y tamaño de "kernel" 3x3, mientras que la segunda tiene tamaño de filtro 64 y tamaño de "kernel" 3x3.

Además, la red cuenta con una capa de "pooling", una capa de "dropout" al 25%, una capa totalmente conectada con 128 neuronas con función de activación ReLu, una segunda capa "dropout" (al 50% en esta ocasión) y la habitual capa de salida con 10 neuronas. Una vez más, el modelo fue compilado haciendo uso del optimizador "Adam" y la función de pérdida "categorical crossentropy". Un resumen del modelo utilizado, el cual entrena 1.199.882 parámetros (casi 1.180.000 corresponden a la capa totalmente conectada oculta) puede consultarse en la imagen adjunta a continuación.

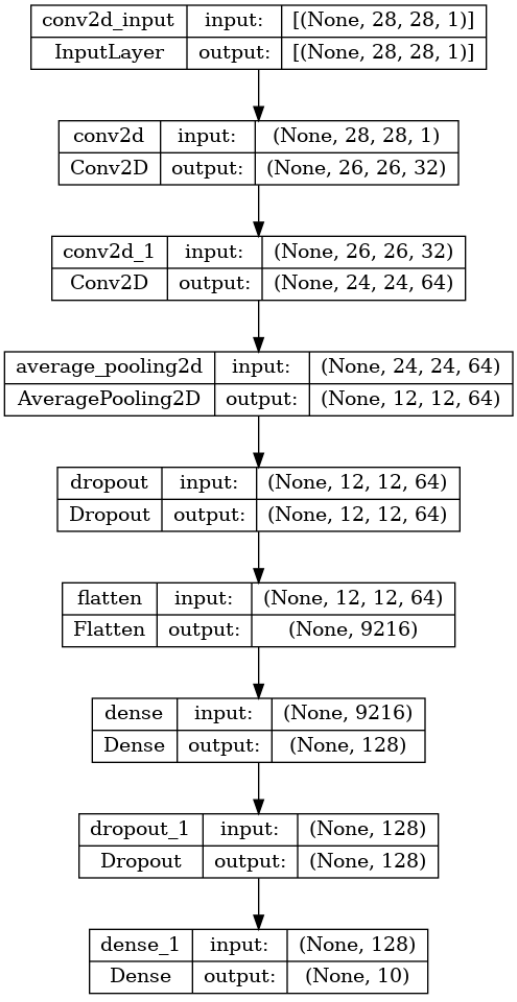


Imagen 6. Modelo de la red neuronal convolutiva.

En un primer momento, intenté mejorar los resultados obtenidos de base con esta arquitectura (con precisión **99.25%** sobre el conjunto de prueba) alterando los tamaños de los "kernels" de las capas convolutivas, y después ajustando el tamaño del filtro. Ambos ajustes, cada uno por separado y combinados redujeron ligeramente la precisión, por lo que en posteriores experimentos mantuve los valores originales.

Posteriormente, probé a variar los valores de las capas de "dropout" al 20 y 40% respectivamente; y a cambiar la técnica de "pooling" por "AveragePooling" con tamaño del muestreo 2x2. De esta manera, la precisión se incrementó muy ligeramente hasta alcanzar el **99.31%.** El entrenamiento consistió en 20 épocas y no se utilizó procesamiento por mini lotes, demorándose la ejecución 226.56 segundos.

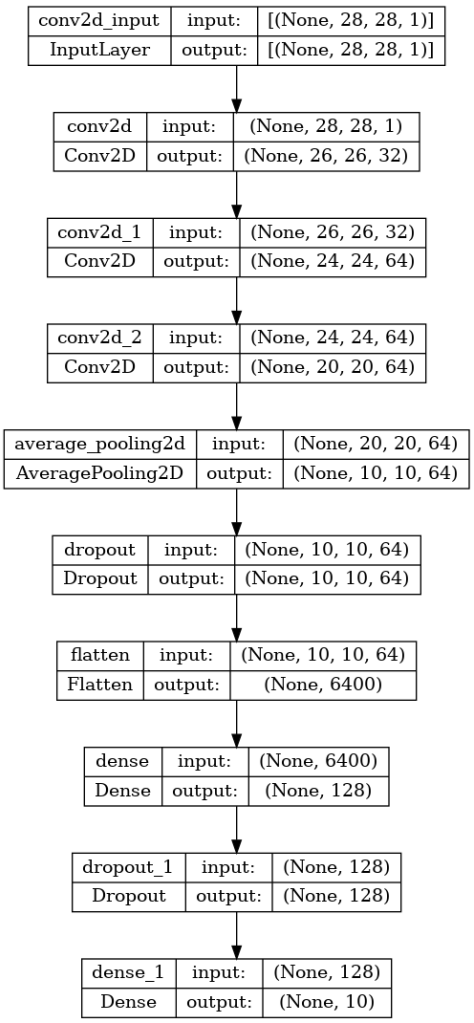


Imagen 7. Modelo de la red neuronal convolutiva.

En un nuevo intento de mejorar la precisión, se decidió probar a incorporar una tercera capa convolutiva. Esta capa tiene un tamaño de filtro 64 y un "kernel" de convolución 5x5, de manera que no es exactamente igual que la capa convolutiva que le precede. Se mantienen las capas "dropout" al 20% y 40%, así como la capa de "pooling" 2x2 por media.

La arquitectura se muestra en la imagen adjunta a la izquierda. Entrenando este modelo durante 15 épocas (193.0389 segundos) sin emplear procesamiento de imágenes por lotes se consiguió mejorar la precisión hasta el **99.40%.** En esta ocasión, el modelo modifica durante su entrenamiento el valor de 941.898 parámetros distintos.

Como estamos viendo a lo largo de esta sección, cada vez cuesta reducir más la tasa de error de los modelos. Hemos llegado a un punto en el que el clasificador apenas falla en 60 de los 10.000 dígitos del conjunto de prueba.

3.7. Red neuronal convolutiva con 4 capas convolutivas, "batch normalization", "dropout" y "AveragePooling".

Tras consultar los apuntes de la asignatura y alguna página web más, decidí utilizar la técnica de regularización "batch normalization" en un intento de mejorar la precisión del detector de dígitos.

En esta ocasión, la red tendrá cuatro capas convolucionales. Las dos primeras tendrán tamaños de filtro 32, y tamaños de "kernel" 3x3 y 5x5; la segunda capa es la novedad respecto del modelo anterior. Las dos segundas tendrán tamaño de filtro 64 y "kernels" 5x5.

Después de cada capa convolucional aplicaremos "batch normalization", y cada dos capas convolucionales tendremos "dropout" del 40% de las neuronas. Antes de aplanar la imagen, realicemos un "pooling" por media con tamaño de muestreo 2x2. Aplanada la imagen, esta pasará a una capa totalmente conectada de 128 neuronas con activación ReLu. Antes de llegar a la capa de salida, aplicaremos un último "dropout" del 40% de

neuronas. En su conjunto, este modelo cuenta con 673.258 parámetros ajustables y 640 parámetros no ajustables.

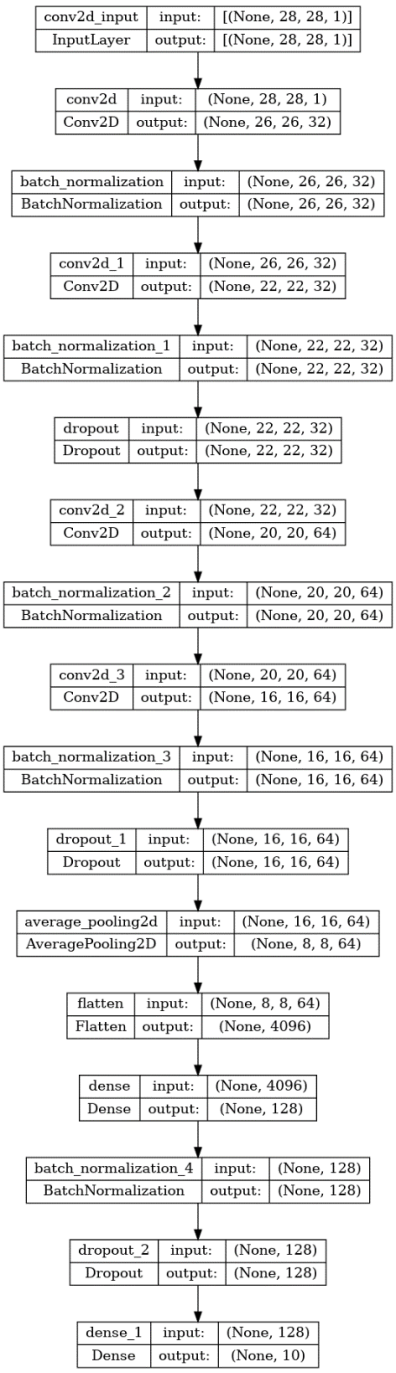


Imagen 8. Modelo de la red neuronal convolutiva que mejores resultados proporciona.

La topología explicada, cuyo resumen se puede ver a la izquierda, es una versión algo distinta (prescinde de dos capas convolutivas y dos capas de regularización e incorpora "AveragePooling") de la presentada en el quinto experimento del artículo de Kaggle [6].

Inicialmente probé dicha versión de Kaggle, que prometía alcanzar una precisión del **99.66%.** Sin embargo, los resultados que obtuve se encontraban en el intervalo **[99.31; 99.47]%.** Dado que la implementación de Kaggle no es propia, no se incluye su análisis en detalle en esta memoria, sin embargo, si que está disponible para su ejecución en el código fuente adjunto. Así pues, decidí modificar dicha red en un intento de mejorar los resultados obtenidos.

Tras ajustar los parámetros y obtener resultados prácticamente idénticos, decidí eliminar 2 de las capas convolucionales y añadir "AveragePooling". De esta manera, tras un entrenamiento de 30 épocas (283.65 segundos) con lotes de 128 imágenes, se consiguió alcanzar la mejor precisión de entre todos los experimentos realizados a lo largo de esta práctica, siendo dicha precisión sobre el conjunto de prueba del **99.54%.**

Llegados a este punto, desistí en mis intentos de mejorar el modelo. Soy consciente de que se puede hacer uso de técnicas de aumento de datos y de modelos como MDCNN (Multi Dimensional Convolutional Neural Network) para alcanzar una precisión de hasta el **99.81%.**

Sin embargo, considero que de los 46 ejemplos en los que falla mi red neuronal, ya hay bastantes de ellos para los cuales un humano podría dudar bastante o incluso equivocarse, por lo que no veo sentido en intentar sobre-ajustar aún más el modelo.

4. Resultados.

En esta sección se exponen algunos de los resultados y conclusiones obtenidos durante la batería de experimentos, a modo de resumen.

Se usará la siguiente leyenda en las tablas:

- E: número de épocas.
- BS: "batch size". Es decir, número de imágenes por lote.
- DP: "dropout". Porcentaje de neuronas deshabilitadas.
- PE y PT. Precisión en entrenamiento y precisión en tests; respectivamente.
- LE y LT. Pérdida en entrenamiento y pérdida en tests; respectivamente.
- CC: capa convolucional.

4.1. Red neuronal simple. Perceptrón.

Para el caso del perceptrón, y dado que era la primera tanda de experimentos, decidí utilizar varias combinaciones de número de épocas y tamaño del lote a procesar en paralelo. Los resultados se resumen en la siguiente tabla:

Parámetros	**PE (%)**	**PT (%)**	**LE**	**LT**	**T (segs)**
E=10; BS=1	93.26	92.66	0.24335	**0.26207**	50.89474
E=10; BS=32	93.29	92.78	0.24454	0.26389	56.24112
E=10; BS=128	92.93	92.68	0.2566	0.26662	**15.11432**
E=20; BS=1	**93.68**	**92.81**	**0.23237**	0.26783	108.36615
E=20; BS=32	93.64	92.77	0.23202	0.26605	99.9477
E=20; BS=128	93.23	92.76	0.24361	0.26334	26.53122
E=30; BS=128	93.51	92.76	0.23616	0.23648	44.712177

Tabla X. Experimentos para el perceptrón.

En este caso, los mejores resultados se obtienen haciendo uso de 20 épocas de entrenamiento y prescindiendo del procesamiento por lotes. Si nos fijamos, la diferencia en precisión no es demasiada si para un mismo número de épocas utilizamos procesamiento por lotes o no, aunque si es cierto que para un tamaño de lote 128 obtenemos mejoras en los tiempos de ejecución bastante considerables. Incrementar el número de épocas de entrenamiento no parece proporcionarnos demasiada mejora, por lo que mantendremos el valor de este hiperparámetro entre 10 y 20. Ocasionalmente realizaremos alguna ejecución con 30 épocas. Soy consciente de que existe "EarlyStopper" en Keras, pero este ejecuta el entrenamiento durante demasiado tiempo y me proporciona peores resultados que con la selección manual de épocas que realizo.

4.2. Red neuronal multicapa con 256 neuronas en la capa oculta.

Para el caso de primera red neuronal multicapa, centramos los experimentos en entrenamientos con 10 y 20 épocas y un tamaño de lote de 128 imágenes. También jugamos

con la cantidad de "dropout" a aplicar (no se muestran todos los experimentos, también se ha probado con 30 y 40%).

Parámetros	PE (%)	PT (%)	LE	LT	T (segs)
E=10; BS=1	**99.79**	98.10	0.00745	0.07855	57.68377
E=10; BS=128	99.74	97.98	0.01291	0.06382	20.38321
E=10; BS=128; DP=20	99.58	98.08	0.1662	0.06111	16.81076
E=10; BS=128; DP=50	99.08	97.91	0.03058	0.06738	**16.74411**
E=20; BS=1	99.39	97.40	0.01818	0.13554	113.17732
E=20; BS=128; DP=20	99.94	**98.41**	**0.00381**	0.06425	31.7661
E=20; BS=128; DP=50	99.67	98.32	0.01262	**0.06036**	32.44129

Tabla X. Experimentos para la primera red neuronal multicapa.

En esta ocasión, los mejores resultados se obtienen con entrenamientos que hacen uso de 20 épocas y tamaños de lote de 128 imágenes. La diferencia entre desactivar un 20, 30, 40 o 50% de las neuronas no es demasiada, obteniéndose los mejores resultados si se desactivan aleatoriamente un 20% de las neuronas de la capa oculta.

Si observamos los porcentajes de error sobre el conjunto de entrenamiento, veremos como casi se alcanza a un aprendizaje perfecto sobre el mismo, aunque luego la tasa de error sea mayor sobre el conjunto de prueba, como era de esperar.

4.3. Red neuronal convolutiva (v1) entrenada con gradiente descendente estocástico.

Dado que esta es la primera ocasión en la que utilizamos redes neuronales convolucionales, y sabiendo que serían las que usaríamos en el resto de la práctica, el número de experimentos ha sido bastante elevado. Así, sabremos que combinaciones dan mejores resultados y podremos reducir los experimentos en secciones posteriores.

Parámetros	PE (%)	PT (%)	LE	LT	T (segs)
E=10; BS=1	98.74	98.62	0.04083	0.04345	93.77367
E=10; BS=32	98.94	98.86	0.03241	0.03902	136.13764
E=10; BS=32; DP=20	98.98	98.78	0.03382	0.03604	114.48358
E=10; BS=32; DP=50	98.87	98.88	0.03733	**0.03507**	112.92865
E=10; BS=128	97.46	97.56	0.08303	0.07639	41.4276
E=10; BS=128; DP=50	97.05	97.26	0.09526	0.08385	**34.0591**
E=20; BS=1	**99.59**	98.83	**0.0138**	0.03523	287.82029
E=20; BS=32	99.46	**98.90**	0.01834	0.0382	295.88198

E=20; BS=128	98.28	98.18	0.05313	0.05555	65.03128
E=20; BS=128; DP=50	98.36	98.49	0.05275	0.04796	68.10074
E=30; BS=128	98.94	98.52	0.03493	0.04197	96.92566

Tabla X. Experimentos para la primera red neuronal convolucional.

En esta ocasión, el mejor resultado sobre el conjunto de prueba ha sido obtenido haciendo uso de 20 épocas de entrenamiento, tamaño de lote de 32 y sin utilizar "dropout". Un resultado prácticamente idéntico ha sido obtenido haciendo uso de 10 épocas de entrenamiento, tamaño de lote 32 y "dropout" del 50% en menos de la mitad de tiempo, por lo que, si tuviéramos que usar este tipo de red, seguramente escogeríamos esta segunda opción. También podemos ver como en la mayoría de los casos los tamaños de lote 128 permiten reducir mucho el tiempo de entrenamiento, a costa de sacrificar algo de precisión, aunque no siempre será así.

4.4. LeNet-5 (con optimizador Adam).

Como se puede observar en la tabla que se adjunta a continuación, los mejores resultados los obtenemos haciendo uso de un entrenamiento de 20 épocas (al igual que en los casos anteriores) y un tamaño de lote de 128 imágenes (por primera vez en un modelo). Cierto es que el entrenamiento en 10 épocas sin procesamiento por lotes proporciona una precisión prácticamente idéntica, pero el tiempo de entrenamiento es considerablemente mayor.

Parámetros	**PE (%)**	**PT (%)**	**LE**	**LT**	**T (segs)**
E=10; BS=1	99.73	98.98	0.00793	**0.03652**	127.67758
E=10; BS=32	99.56	98.86	0.01321	0.049	123.92787
E=10; BS=128	99.51	98.88	0.01463	0.03705	**32.55016**
E=20; BS=1	99.74	98.84	0.0081	0.06926	228.8631
E=20; BS=128	**99.81**	**98.99**	**0.00632**	0.04822	88.58336

Tabla X. Experimentos para la red LeNet-5 (con optimizador Adam).

Si por alguna razón deseáramos entrenar este modelo lo más rápido posible, utilizaríamos 10 épocas y 128 imágenes por lote. De esta manera, solo sacrificaríamos un 0.11% de precisión en la detección, pero conseguiríamos un entrenamiento casi 3 veces más rápido.

4.5. Red neuronal convolutiva con 2 y 3 capas convoluciones consecutivas, "dropout" y "AveragePooling".

Todos los experimentos para este modelo de red tienen un primer "dropout" del 20% y un segundo del 40%, conforme se especifica en la sección 3.6. Así también, el "pooling" es de tipo "average".

Los primeros experimentos se realizaron sobre el modelo con dos capas convolucionales. Los mejores resultados se obtienen con 20 épocas de entrenamiento, pero parece más razonable usar 15 épocas y procesamiento en lotes de 128 imágenes, dado que la precisión apenas varía un 0.05% pero el tiempo de ejecución es unas 3.4 veces menor.

Así pues, los experimentos con el modelo con 3 capas convolucionales se realizaron principalmente haciendo uso de 15 épocas.

Parámetros	PE (%)	PT (%)	LE	LT	T (segs)
E=10; BS=1; CC=2	99.84	99.25	0.00556	0.2626	106.61121
E=15; BS=128; CC=2	99.92	99.26	0.00263	0.2613	**66.63408**
E=20; BS=1; CC=2	**99.99**	99.31	0.00055	0.03529	226.5653
E=20; BS=128; CC=2	99.97	99.28	0.0011	0.0306	87.23179
E=15; BS=1; CC=3	99.95	**99.40**	0.00166	**0.02381**	193.0389
E=15; BS=128; CC=3	99.93	99.35	0.00213	0.2852	102.87128
E=20; BS=128; CC=3	99.97	99.38	**0.00107**	0.02244	130.74435

Tabla X. Experimentos para la red convolutiva con convoluciones consecutivas.

Los mejores resultados se obtienen haciendo uso de 15 épocas de entrenamiento y prescindiendo del procesamiento por lotes, con un 99.40% de precisión. Si incluimos el procesamiento en lotes de 128 imágenes reducimos el tiempo de entrenamiento a la mitad, sacrificando un 0.05% de precisión en las deducciones. Probablemente, la opción más razonable sea utilizar 20 épocas y procesamiento en lotes de 128 imágenes, pues obtiene casi la misma precisión que el mejor caso y necesita menos tiempo de entrenamiento.

4.6. Red neuronal convolutiva con 4 capas convolutivas, "batch" "normalization", "dropout" y "AveragePooling".

Dado que en este modelo utilizamos más operaciones de convolución y un proceso de normalización después de cada convolución, resulta casi obligatorio utilizar procesamiento por lotes (de 128 imágenes) si no deseamos tardar un tiempo excesivo en entrenar los modelos, incluso usando GPU. Por este motivo, el conjunto de pruebas es más reducido en esta ocasión.

Parámetros	PE (%)	PT (%)	LE	LT	T (segs)
E=10; BS=128	99.74	99.36	0.00832	0.02166	**92.55679**
E=20; BS=128	**99.99**	99.53	0.0021	**0.0180**	170.39232
E=30; BS=128	99.98	**99.54**	**0.00076**	0.02009	283.6548

Tabla X. Experimentos para la última red convolutiva.

Como adelantábamos en la sección 3.7, haciendo uso de 30 épocas de entrenamiento y lotes de 128 imágenes obtenemos los mejores resultados de toda la práctica, con una precisión del **99.54%**. Este resultado es casi idéntico que el obtenido al usar 20 épocas. El que uno u otro entrenamiento proporcione mayor o menor precisión depende un tanto de

la inicialización de los números aleatorios que tenga lugar, por lo que depende de cada ejecución concreta.

Llegado a este nivel de precisión, decidí no implementar alternativas mucho más complejas, pero que proporcionan mejores resultados. Si nos fijamos en algunos de los ejemplos en los que falla la red, observamos que hay valores que un humano sí que podría acertar, pero hay un numero considerable en los que se tendrían dudas razonables.

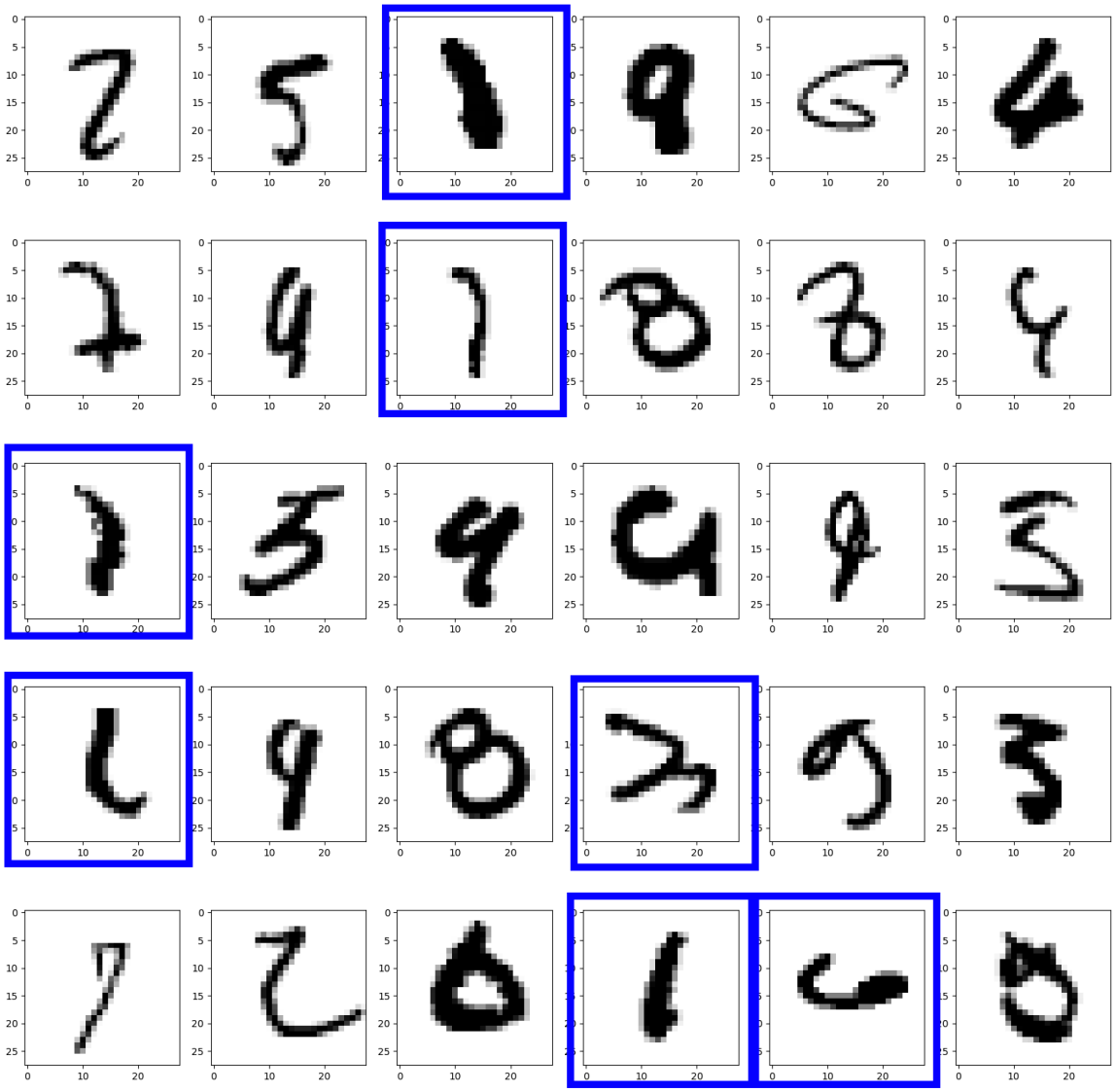


Imagen X. Errores de clasificación de la mejor red neuronal conseguida.

En azul se han marcado aquellos números que un humano podría considerar que no son ni siquiera números. Algunos dígitos 1 parecen más bien manchas de boli y hay un 3 y un 6 que ni parecen números.

A parte de estos, vemos que la red parece tener problemas sobre todo con los 4 y los 9 muy cerrados en la parte superior (véanse ejemplos centrales de la tercera fila); 3 y 5 con los trazos holgados (segundo y sexto ejemplo de la tercera fila y último ejemplo de la cuarta fila) y con los 0, 6 y 9 que tengan trazos muy holgados (véase sobre todo la última fila).

Sin embargo, hay algunos dígitos que un humano sí que parece poder reconocer con facilidad, sobre todo los números de las dos primeras filas y los 8. Puede que con modelos más complejos la red sí que sea capaz de reconocer dichos valores, pero no creo que sea de demasiada utilidad sobre-ajustar aún más la red al conjunto de entrenamiento.

5. Conclusiones.

En la presente práctica se han aplicado diferentes tipos de redes neuronales para introducirnos en el problema de la clasificación de imágenes. Aunque ya había utilizado modelos más complejos en el pasado, como YOLOv4, esta práctica sirve como vehículo para ser consciente de la gran cantidad de opciones que debemos tener en cuenta al diseñar una red neuronal, y como ciertas decisiones de diseño e hiper-parámetros asociados permiten obtener mejores o peores resultados, a pesar de encontrarnos ante "un problema de juguete". Además, esta práctica también nos hace darle el valor que se merecen las bibliotecas de redes neuronales y aprendizaje profundo como Keras y Tensorflow, que facilitan enormemente el desarrollo.

6. Bibliografía.

- [1] Apuntes sobre redes neuronales de la asignatura "Inteligencia Computacional" del Máster en Ingeniería Informática de la Universidad de Granada. *Fernando Berzal.*
- [2] Guía de instalación de NVIDIA CUDA® en WSL2. *NVIDIA Corporation.*
- [3] Guía de instalación de NVIDIA cuDNN®. *NVIDIA Corporation.*
- [4] Building a Neural Network from Scratch: Part 1. *Jonathan Weisberg.*
- [5] How to develop a CNN for MNIST Handwritten Digit Classification. *Jason Brownlee.*
- [6] How to choose CNN Architecture MNIST. *Chris Deotte.*
- [7] MNIST – Deep Neural Network with Keras. *Prashant Banerjee.*
- [8] Example convnet on the MNIST dataset. *AWSLabs repository, forked from Keras repository.*
- [9] Gradient-Based Learning Applied to Document Recognition. (Artículo de LeNet-5). *Yann LeCunn et al.*
- [10] 25 Million Images! [0.99757] MNIST. *Chris Deotte.*
- [11] Multi-column Deep Neural Networks for Image Clasification. *Dan Ciresan, Ueli Meier and Jurgen Schmidhuber.*
- [12] Documentación de la API de Keras. *Keras' Team*
- [13] Guide to CNNs with Data Augmentation (Keras). *Moghazy*