Deep Learning y reconocimineto de patrones

Experimentación y análisis del entrenamiento y funcionamiento de Redes neuronales convolucionales

Juan David Bernal Vesga
Ciencias de la computación
Universidad Nacional de Colombia
Bogotá D.C., Colombia
jubernalv@unal.edu.co

Oscar David Ordóñez Bolaños
Ciencias de la computación
Universidad Nacional de Colombia
Bogotá D.C., Colombia
oordonezb@unal.edu.co

Juan Pablo Urrutia Parrado
Ciencias de la computación
Universidad Nacional de Colombia
Bogotá D.C., Colombia
jurrutia@unal.edu.co

***Abstract*—Se muestra una solución óptima para el problema de reconocimiento de dígitos escritos a mano, usando un modelo de aprendizaje de máquina supervisado, redes neuronales profundas y redes convolucionales. Se discuten distintos resultados y soluciones al problema del *overfitting*.**

***Index Terms*—Aprendizaje profundo, Redes neuronales, Redes convolucionales, Reconocimiento de patrones, Visión por computadora, Reconocimiento de imágenes**

I. Introducción

El aprendizaje de máquina, y dentro de este ámbito, el *Deep Learning* y las redes neuronales, ha estado en el centro de atención en una gran cantidad de ámbitos, tomando en cuenta la capacidad de estas para abstraer y procesar eficientemente información, de manera secuencial y autónoma.

Tomando esto en consideración, es de nuestro interés poner en practica una implementación particular de estos algoritmos, como por ejemplo las *Redes Neurales Convolucionales*; que son conocidas por su eficiencia al momento de identificar patrones a través de imágenes, y así poder clasificarlas dados ciertos parámetros.

En el transcurso de este articulo hemos de mostrar la construcción de una Red Convolucional, y su entrenamiento para que pueda desempeñar la tarea simple de identificar números de un dígito escritos a mano. Ahondaremos en el comportamiento de estos algoritmos teniendo en cuenta diferentes modelos sobre los que serán construidos y gradualmente obtendremos una red que logre de manera efectiva, y con un razonable nivel de éxito, desarrollarse para cumplir su objetivo.

Si el lector esta interesado, por otro lado, en entrar más a detalle acerca de la fundamentación y diferentes usos del Deep Learning, puede encontrarlo de interés la referencia [1], en particular el capitulo 19 y 22.

Por lo que a continuación vamos a tomar camino a través del diseño y la elaboración de una red neuronal simple, los desafíos que se afrontan y como hemos logrado solucionarlos, para así obtener finalmente una red capaz de identificar números de un dígito escritos a mano.

II. Metodología

En esta sección vamos a ahondar en las bases sobre la que hemos construido el agente, las diferentes consideraciones y modificaciones que gradualmente se le han hecho a este, con el objetivo de maximizar el grado de éxito del algoritmo para entrenarse considerando una base de datos dada, e identificar imágenes nuevas que se le proporcionan tras esta etapa de entrenamiento.

A. Materiales

Este agente ha sido programado en Python, y en particular hemos usado Keras, un wrapper de TensorFlow, la cual es una plataforma desarrollada por Google, enfocada en el aprendizaje de maquina y que nos permite construir y entrenar de manera sencilla redes neuronales. Si desea conocer más acerca de esta lo invitamos a visitar su sitio web en [2], donde podrá encontrar información a fondo de ese proyecto, la gran cantidad de funcionalidades que ofrecen al usuario, y la documentación de esta biblioteca en Python, por si desea aprender más de cerca lo que hemos trabajado en nuestro código.

Por otro lado, la base de datos utilizada para entrenar nuestro agente es la MNIST, la cual es una gran base de datos de dígitos escritos a mano ofrecida por el *National Institute of Standards and Technology* (NIST). Puede aprender mas acerca de esta en [3].

B. Estructura de los datos

Como ya se ha mencionado en la subsección anterior, la base de datos utilizada para entrenar las redes es el dataset MNIST. En particular utilizamos 60000 imagenes de entrenamiento y 10000 de prueba y validación.

Particularmente estas imagenes son siempre de 28x28 y el número se encuentra centrado; lo cual se verá que es un factor de importancia más adelante cuando hablemos de las limitaciones que encontramos en el reconocimiento.

A continuación podemos ver unos ejemplos de la base de datos, podemos ver como en conjunto a la imagen viene el dato de la predicción esperada con respecto a cada imagen, lo cual permite que estos agentes trabajen basados en el aprendizaje supervisado y de refuerzo.

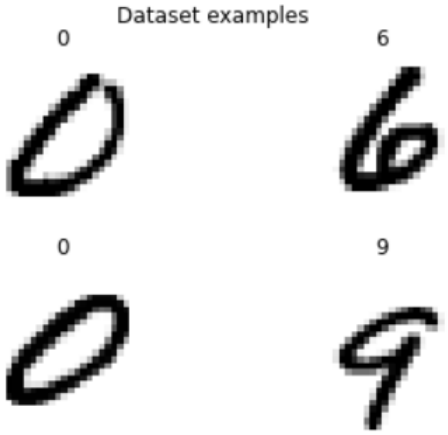

Fig. 1: Ejemplo del dataset

C. Modelo basico de red convolucional

Una red convolucional, como ya hemos venido mencionando, es un modelo de Deep Learning que esta basado en redes neuronales de varias capas. De las más importantes de estas son las capas de Convolución, de donde toma su nombre el modelo; estas capas de neuronas se encargaran de tomar ciertas características clave que ha logrado identificar en las imagenes con las que se entrenó, llamados kernels, que luego va a intentar identificar en pequeñas porciones de la imagen que estará intentando clasificar.

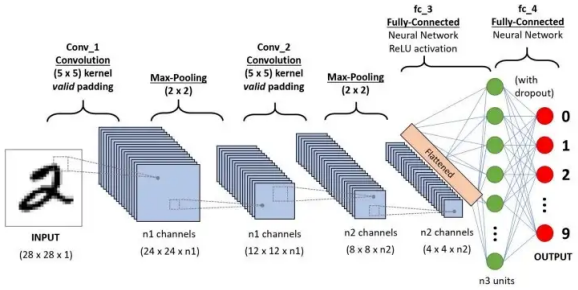

Fig. 2: Estructura CNN para reconocer dígitos

En la Figura 2 podemos ver un ejemplo de arquitectura para una red con el mismo objetivo que la nuestra. A partir de esta capas de neuronas el modelo es capaz de procesar la información que encuentre en la imagen input, para luego así lograr una clasificación de esta de una manera acertada.

La parte más interesante de estas es que uno las puede entrenar para que ellas mismas logren escoger los kernels y otros parametros significativos, con el objetivo de lograr clasificar con mayor certeza y que este proceso sea más simple para el programador.

Si se desea ahondar más acerca de redes convolucionales o en otros modelos de aprendizaje de maquina se invita a revisar las referencias [1] o [4].

D. Modelo Inicial

Para este modelo inicial se considera crear:

- 2 capas de convolución para abstraer la información, una de 32 y otra de 64 neuronas.
- 1 capa Dropout, con probabilidad del 25%. Estas capas tienen como objetivo desestimar aleatoriamente una parte de las salidas que nos otorgan las capas de convolución, lo cual permite evitar el Overfit, tema en el que ahondaremos más adelante.
- 1 capa oculta de procesamiento, con 128 neuronas.
- 1 capa Dropout extra, una vez más con probabilidad del 25%.
- 1 capa final con las 10 categorias esperadas, para que se pueda realizar la clasificación.
- Se compila este modelo tomando ADAM como optimizador y categorical_crossentropy como la función de perdida.

Este modelo permite abstraer y procesar la información de una imagen de entrada, para que en la capa de categorías se pueda realizar una ”votación” con el objetivo de determinar cual de los 10 dígitos es el más apto para categorizar la imagen dada.

Cabe mencionar que en las capas de convolución se esta usando la función de activación RELU, y que se esta usando tras las dos capas de convolución una de maxpooling para procesar los datos.

Puede entrar más a detalle acerca de funciones de activación y Pooling, junto con su importancia en la referencia [4].

E. Entrenamiento

Al momento de entrenar nuestro modelo, lo hacemos en 20 corridas, cada una con 128 datos de entrenamiento. Al final de cada una se ajusta los parámetros de la red con el algoritmo de Backpropagation, que permite considerando el nivel de éxito de una salida de entrenamiento, revisar y ajustar neurona pro neurona desde la ultima capa con el objetivo de lograr mejores resultados. Puede aprender más acerca de este algoritmo en [6].

F. Modificación del modelo y Overfitting

A pesar de las grandes capacidades que poseen los modelos de aprendizaje de maquina, los algoritmos no pueden hacer todo el trabajo. El modelo que se expuso en esta sección no tiene un funcionamiento del todo óptimo, por lo que se le tuvieron que hacer grandes ajustes para llegar a la red que si posee la capacidad de identificar satisfactoriamente las imagenes.

En la sección de Resultados se expone esta red funcional, en la sección de Discusión y conclusiones ahondamos en el Overfitting y como lo superamos.

RESULTADOS

Tras crear, ajustar y entrenar este agente, obtenemos una red neuronal capaz de identificar dígitos escritos de manera satisfactoria. Esta identifica números del 0 al 9 de manera efectiva, aunque con algunas limitaciones que se exponen en la siguiente sección. El funcionamiento correcto de esta se puede ver en la referencia [7], o de otra manera visitar el

repositorio correspondiente a este proyecto en la referencia [8]; donde también se adentra más en el código en el Notebook correspondiente e incluso el usuario puede entrenar y probar el agente por si mismo.

Fig. 3: Programa en funcionamiento

Cabe mencionar que para que la experiencia sea más interesante para el usuario, se implemento un tablero en la aplicación para que se pueda escribir el dígito por uno mismo, para que el agente lo identifique.

Encontramos muy buenos resultados en las pruebas para nuestro modelo final, habiendo un éxito total al identificar las imagenes de mas del 99%, ahondamos más de esto en la siguiente sección donde se analiza y grafica la tasa de éxito de cada uno de los modelos y en el Notebook de la referencia [8], donde se ven las pruebas realizadas con el codigo.

Discusión y conclusiones

Para llegar al modelo expuesto anteriormente, hubo que superar el problema de *Overfitting*. Esto es, cuando el modelo queda sobre-entrenado sobre un dataset, tanto que es incapaz de reconocer dígitos de manera generalizada. Esto se puede detectar encontrando que a medida que se entrena el modelo, el loss (porcentaje de error) sobre el dataset de entreno baja, mientras que el loss sobre el dataset de validación se mantiene o sube. Para la primera implementación del modelo, se implementan 'Dropout layers'. Estas son capas diseñadas para desestimar aleatoriamente la salida de las convoluciones, reduciendo así un posible overfit, como veremos más adelante. Sin embargo, parece no ser suficiente para resolver el problema. Aunque no es muy grave, se puede ver como el loss sobre el set de validación deja de disminuir en el octavo entrenamiento, y luego sube.

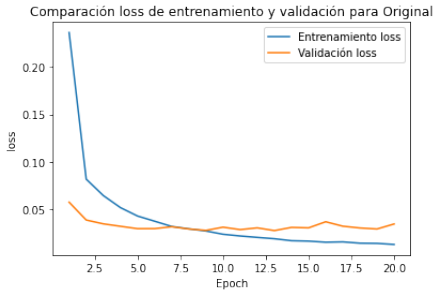

Fig. 4: Porcentaje de pérdida en el modelo con capas Dropout

Para atacar este problema, se plantean algunas estrategias:

- **Reducir el número de neuronas de la capa de segunda convolucion de 64 a 32, y de la capa oculta de 128 a 64.**
  De este manera se busca reducir la capacidad de la red y que se vea obligada a aprender los patrones que importan o que minimizan la pérdida. Con esta estrategia se obtienen los siguientes resultados:

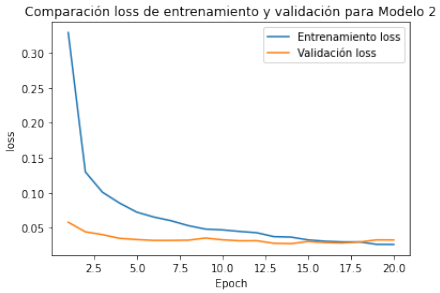

Fig. 5: Porcentaje de pérdida en el modelo con capas reducidas

- **Aplicar regularización a la segunda capa de convolución.**
  Esto añade un coste a la función de pérdida del modelo para los parámetros grandes. Como resultado, se obtiene un modelo más simple que se verá obligado a aprender sólo los patrones relevantes de los datos de entrenamiento. La regularización L2 añade un coste con respecto al valor cuadrado de los parámetros. Esto da lugar a pesos más pequeños para cada neurona.
  Con esta estrategia se obtienen los siguientes resultados:

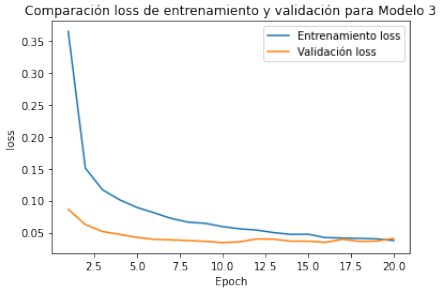

Fig. 6: Porcentaje de pérdida en el modelo con regresores sobre la segunda convolución

Después de estas modificaciones, el modelo resultante presenta las siguientes pérdidas:

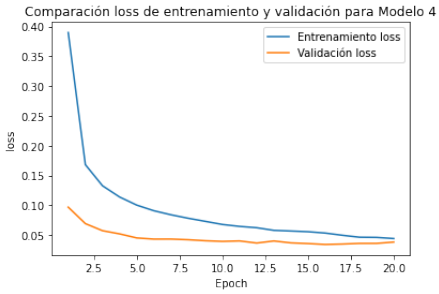

Fig. 7: Porcentaje de pérdida en el modelo con regresores sobre la segunda convolución y capas reducidas

Vemos que se logra una reducción significativa en comparación al modelo inicial, ya que el modelo no presenta 'overfitting'. Luego está listo para ser usado por el usuario final.

Otro problema que presenta el modelo es la incapacidad de reconocer números que no sean dibujados en el centro del recuadro. Por ejemplo,

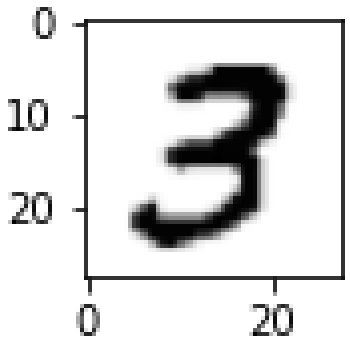

Fig. 8: Imagen de entrenamiento

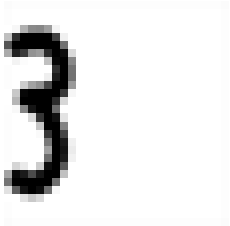

Fig. 9: Imagen defectuosa

Como el modelo es entrenado con imágenes centradas en donde el número ocupa todo el espacio, el presentarle imágenes como la Figura 9. induce al modelo a fallar. Una posible solución al problema es centrar y agrandar los números dibujados por el usuario, de manera que la imagen final quede con las especificaciones de las imágenes del dataset.

En conclusión, después de abordar las problemáticas anteriormente descritas, se obtiene un modelo capaz de satisfacer las necesidades iniciales del problema. Además, este permite el estudio sobre los usos y dificultades de trabajar con redes neuronales convolucionales.

REFERENCES

[1] Stuart J. Russell, Peter Norvig, Artificial Intelligence - A Modern Approach, 4th ed., Global ed., Pearson, 2021.

[2] "Tensorflow," TensorFlow. [Online]. Available: http://www.tensorflow.org/.

[3] "The mnist database," MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges. [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[4] S. Saha, "A comprehensive guide to Convolutional Neural Networks" Medium, 16-Nov-2022. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[5] J. Gupta, "Going beyond 99%-mnist handwritten digits recognition," Medium, 04-May-2020. [Online]. Available: https://towardsdatascience.com/going-beyond-99-mnist-handwritten-digits-recognition-cfff96337392

[6] DalpMaths, "Backpropagation: (parte 1)," YouTube, 19-Jan-2020. [Online]. Available: https://youtu.be/wWV2qroGUMo.

[7] O. Ordoñez, "Digit recognition / proyecto final inteligencia artificial," YouTube, 03-Dec-2022. [Online]. Available: https://youtu.be/pyRzm-rxStc.

[8] J. P. Urrutia Parrado, O. D. Ordóñez Bolaños, and J. D. Bernal Vesga, "JURRUTIAP/digit-recognition: UNAL - Introducción a la inteligencia artificial. Este Proyecto implementa UN Sistema de Reconocimiento de Dígitos escritos a Mano, USANDO deep-learning y redes convolucionales.," GitHub. [Online]. Available: https://github.com/jurrutiap/digit-recognition.git.