Creación de una red neuronal convolucional genérica

Adrián Cobo Merino
Universidad Rey Juan Carlos
Madrid, España
la.cobo.2020@alumnos.urjc.es

Daniel Alejandro Quinga López
Universidad Rey Juan Carlos
Madrid, España
da.quinga.2020@alumnos.urjc.es

David Duro Aragonés
Universidad Rey Juan Carlos
Madrid, España
d.duro.2020@alumnos.urjc.es

I. Introducción

Este proyecto trata del aprendizaje automático con redes neuronales. El objetivo fundamental es probar cómo aprende nuestra red neuronal formada mediante capas lineales y convolucionales ante distintos datasets de imágenes.

Las capas lineales, también conocidas como capas completamente conectadas, desempeñan un papel crucial en el diseño de redes neuronales. Estas capas realizan transformaciones lineales en los datos de entrada, aplicando pesos y sesgos a cada elemento. A través de esta operación, las capas lineales pueden aprender patrones complejos y relaciones no lineales en los datos.

Las capas convolucionales constituyen una herramienta invaluable cuando se trata de analizar datos que poseen estructuras espaciales. Inspiradas en la capacidad del cerebro humano para reconocer patrones visuales, estas capas aplican filtros a regiones específicas de la entrada, permitiendo la detección de características locales.

II. Funcionamiento redes neuronales

Las redes neuronales son modelos computacionales inspirados en el funcionamiento del cerebro humano, diseñados para aprender y realizar tareas específicas a través de la adaptación a patrones en los datos. En este proyecto, nos centramos en dos tipos de capas fundamentales: capas lineales y capas convolucionales.

A. Capas Lineales: Entendiendo la Transformación Lineal

Las capas lineales, también conocidas como capas densas o completamente conectadas, son esenciales en la arquitectura de las redes neuronales. Cada neurona en una capa lineal está conectada a todas las neuronas de la capa anterior, aplicando una transformación lineal a los datos de entrada. Esta transformación implica multiplicar cada entrada por un peso correspondiente y sumar un sesgo. Matemáticamente, la operación de una capa lineal se expresa como:

$$y=Wx+b$$

Donde:

- y es la salida.
- W es la matriz de pesos.
- x es el vector de entrada.
- b es el sesgo.

La operación de la capa lineal permite aprender relaciones lineales y no lineales entre las características de entrada, facilitando la adaptación del modelo a la complejidad de los datos.

B. Capas Convolucionales: Descifrando Estructuras Espaciales

Las capas convolucionales son especialmente poderosas para el procesamiento de datos bidimensionales, como imágenes. Estas capas aplican filtros a regiones locales de la entrada, permitiendo la detección de patrones visuales como bordes, texturas y formas. La operación de convolución se realiza deslizando el filtro sobre la entrada y calculando productos escalares en cada posición.

Matemáticamente, la operación de convolución se expresa como:

$$S(i,j)=(I * K)(i,j)=\sum_m \sum_n I(m,n)K(i-m,j-n)$$

Donde:

- S(i,j) es el valor en la posición (i,j) de la salida.
- I es la entrada.
- K es el kernel o filtro.

Esta operación captura características locales, permitiendo a la red aprender representaciones más abstractas y complejas a medida que se profundiza en las capas convolucionales.

III. Implementación en python

Para llevar a cabo este estudio, se ha utilizado Python como lenguaje de programación, aprovechando la biblioteca PyTorch. Estas herramientas proporcionan una interfaz eficiente y flexible para construir y entrenar redes neuronales, permitiendo experimentar con diversas arquitecturas y conjuntos de datos.

IV. Optimizaciones

A. Función de Activación ReLU

La función ReLU (Rectified Linear Unit) introduce no linealidades al permitir que solo los valores positivos pasen a través de la red. Esto facilita la convergencia del modelo y mejora su capacidad para aprender patrones complejos en los datos.

B. Capa de Pooling

La operación de pooling se utiliza para reducir la dimensionalidad de las representaciones espaciales, preservando las características más importantes. En nuestro caso, aplicamos pooling máximo (MaxPooling) para seleccionar el valor máximo en regiones específicas de la entrada, ayudando así a identificar características clave.

C. Capa de Regularización Dropout

La regularización es fundamental para prevenir el sobreajuste del modelo durante el entrenamiento. La capa Dropout apaga aleatoriamente un porcentaje de las neuronas durante cada iteración, lo que promueve una representación más robusta y generalizable.

D. Función de Pérdida CrossEntropy (CrossEntropyLoss)

La función de pérdida CrossEntropy es crucial en problemas de clasificación, como el nuestro. Mide la discrepancia entre las predicciones del modelo y las etiquetas reales, proporcionando una señal de retroalimentación para ajustar los pesos y mejorar el rendimiento de la red.

E. Optimizador Adam

Adam es un algoritmo de optimización que adapta las tasas de aprendizaje de cada parámetro de la red de manera individual. Su capacidad para ajustarse automáticamente a diferentes tasas de aprendizaje en función de la magnitud de los gradientes contribuye a una convergencia eficiente del modelo.

V. DATASETS PROBADOS

Para poner a prueba la red neuronal se utilizaron tres distintos datasets, cada uno dividido en datos de entrenamiento, datos de validación y datos de test:

A. Razas de perros

En este dataset se utilizan las imágenes de 3 razas distintas de perros: Golden Retriever, Staffordshire Bull Terrier y Toy Terrier.
La dificultad de este dataset reside en el parecido de todos los perros, es decir, todos los perros comparten muchas características que son difíciles de diferenciar incluso para una persona.

Las pruebas comenzaron con dos razas de perros bastante diferentes, los Golden Retriever y los Toy Terrier, dos razas de distinto tamaño y de distinto color. Los resultados obtenidos fueron bastantes buenos obteniendo un 94% de precisión con datos que la red no había visto nunca.
Para comprobar visualmente qué tan bien funciona la red utilizamos la siguiente matriz de confusión que nos permite ver cuantas veces y con qué clase se ha confundido.

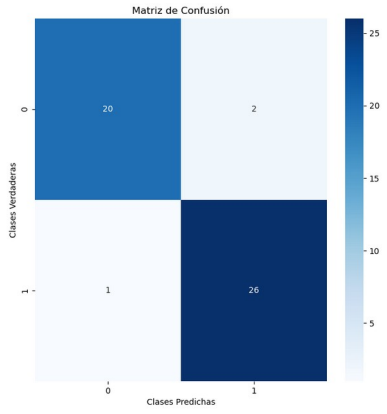

Fig. 1. Matriz de confusión con dos razas de perros

Tras obtener tan buenos resultados después de la primera prueba, se quiso ir incrementando el número de razas, sin embargo, la red no conseguía encontrar las suficientes características en las imágenes para poder predecir con exactitud de qué raza se trataba, pudiendo alcanzar únicamente con 3 razas de perros, las dos anteriores más los Staffordshire Bull Terrier, un 73% de precisión en sus predicciones.

B. Modelos de coches

El contenido de este dataset son 9 modelos distintos de coche, entre ellos se encuentran los Hyundai i10, Seat Ibiza, o Nemo Citroen.

Las imágenes de este dataset estaban más “simplificadas” que las del anterior, es decir, había algunas imágenes en las que aparecía únicamente el coche con un fondo totalmente en negro. Este tipo de imágenes facilitaban el trabajo a la red ya que podía centrarse únicamente en el coche sin importar lo que tuviera alrededor, siendo más “fácil” que encontrara características con las que poder diferenciar de qué modelo de coche se trataba.

Como en el anterior caso, se utiliza la matriz de confusión para ver la precisión de la red neuronal con este tipo de dataset.

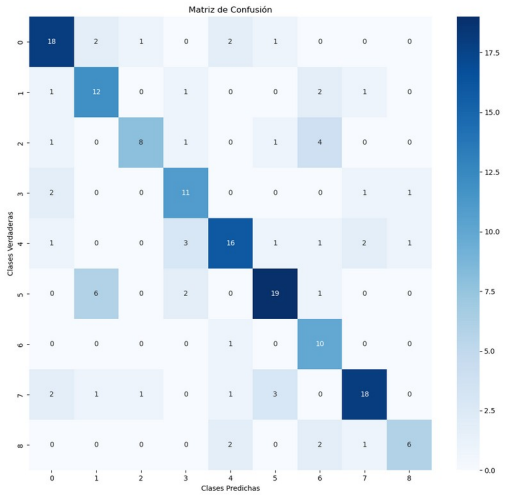

Fig. 2. Matriz de confusión con 9 modelos de coches

C. Pokemons

Este dataset contenía imágenes de 102 clases de pokemons distintas. Como solo teníamos una imagen de cada pokemon y queríamos probar que nuestro modelo funcionaba independientemente del número de canales de la imagen decidimos binarizar la imagen y aumentar nuestro dataset rotando cada imagen tres grados hasta llegar a 360 para que el modelo fuese robusto a rotaciones.

Al binarizar la imagen el número de datos de entrada que definía cada clase era 3 veces menor, observando que el modelo podía encontrar características para discernir entre las distintas clases de forma mucho más eficiente obteniendo un resultado del 89% de tasa de acierto para los datos de validación.

VI. Conclusiones

Tras los resultados obtenidos de la red neuronal frente a los distintos datasets podemos destacar tres puntos importantes:

a) La complejidad para el modelo aumenta al intentar clasificar elementos muy parecidos, tanto en forma, color, tamaño, etc.

b) La arquitectura es robusta independientemente del dataset (siempre que los datos de entrada sean matrices 2D con n canales, es decir datos de mxnxc dimensiones)

c) El desempeño de la red es similar al que tendría un ser humano debido al aumento del error con imágenes que incluyen oclusiones, similitudes entre clases, etc.

Por estas razones creemos que la arquitectura de red neuronal empleada en este estudio es recomendable para cualquier problema de clasificación de imágenes.

Referencias

[1] *Documentacion Pytorch https://pytorch.org/docs/stable/index.html*.

[2] "Deep Learning" de Ian Goodfellow, Yoshua Bengio y Aaron Courville https://www.deeplearningbook.org/

[3] Deep Learning and Neulal networks with Python and Pytorch de Santex https://pythonprogramming.net/introduction-deep-learning-neural-network-pytorch/

[4] *Dogs Datasets https://www.kaggle.com/datasets/jessicali9530/stanford-dogs-dataset*

[5] *Cars Datasets https://www.kaggle.com/datasets/boulahchichenadir/algerian-used-cars*