**Actividad 4.1 - Ejercicio de clasificación con Keras**

**Introducción**

Keras es una biblioteca de código abierto escrita en Python que tiene por propósito la generación y experimentación con modelos de aprendizaje profundo (deep learning). Su interfase de alto nivel permite que el usuario se enfoque en la parte productiva del ciclo de desarrollo de Ciencia de Datos; y no necesariamente en los detalles técnicos y complejidades subyacentes de la construcción de algoritmos de aprendizaje profundo [1]. A su vez, esto acelera el ciclo de análisis, pues los científicos de datos pueden pasar más tiempo en la interpretación de resultados y en la aplicación de los modelos, en lugar de pasar tiempo en la codificación y depuración de los algoritmos de bajo nivel [2].

Esta biblioteca fue desarrollada con el enfoque de permitir la experimentación rápida y el prototipado, con la idea de poder pasar de la idea al resultado con la menor demora posible. Keras se puede ejecutar en varios motores de cálculo de aprendizaje profundo, siendo los más notables TensorFlow, Microsoft Cognitive Toolkit y Theano [3].

Algunas de las principales características de Keras incluyen:

1. Soporte para redes neuronales convolucionales (CNNs), redes neuronales recurrentes (RNNs), y una combinación de ambas.
2. Soporte para arquitecturas de redes de múltiples entradas y múltiples salidas.
3. Herramientas para trabajar con imágenes y texto, lo que facilita la experimentación con estas formas de datos.
4. Funcionalidades para manejar la regularización, la normalización, las optimizaciones, y mucho más.

Cabe señalar que Keras fue oficialmente adoptado por TensorFlow (una biblioteca de aprendizaje automático de Google) como su interfaz de alto nivel, lo que significa que Keras está integrado en TensorFlow y se beneficia de todas las funciones y mejoras de esta biblioteca [4].

En este caso, se llevó a cabo la construcción de un modelo de Red Neuronal Convolucional (CNN, por sus siglas en inglés) para la detección de peatones utilizando imágenes. El uso de CNNs se ha vuelto muy popular en la detección de objetos en imágenes, gracias a su capacidad para extraer características importantes a través de diferentes niveles de abstracción [5].

**Ejecución de un modelo de red neuronal convolucional para detección de peatones**

El proceso inició con el preprocesamiento de las imágenes, que implicó la lectura, redimensionamiento y normalización de las imágenes (Figura 1). Esta etapa es crucial ya que prepara los datos para la alimentación de la red neuronal. Posteriormente, las imágenes preprocesadas fueron divididas en dos categorías, una etiquetada como 'peatón' y la otra como 'no peatón', creando así un conjunto de datos etiquetados que facilita el entrenamiento supervisado. Este último paso se hizo de acuerdo al juego de datos que se utilizó: <https://github.com/RashadGarayev/PersonDetection> .

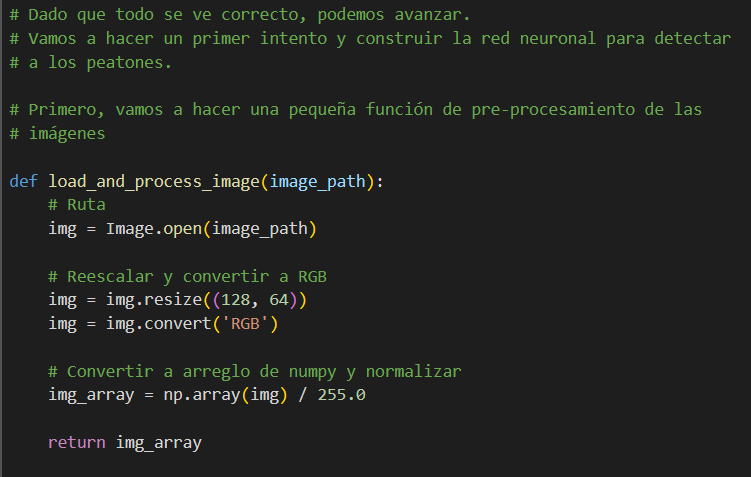


Figura 1. Función de preprocesamiento de las imágenes. Esta función ajusta el tamaño a (128, 64) y luego la transforma a escala de grises.

El diseño de la CNN incluyó varias capas convolucionales intercaladas con capas de agrupamiento o "pooling". Las capas convolucionales son responsables de la detección de características visuales en las imágenes, mientras que las capas de pooling reducen la dimensionalidad de los datos, lo que ahorra tiempo de cálculo y ayuda a evitar el sobreajuste (Figura 2).

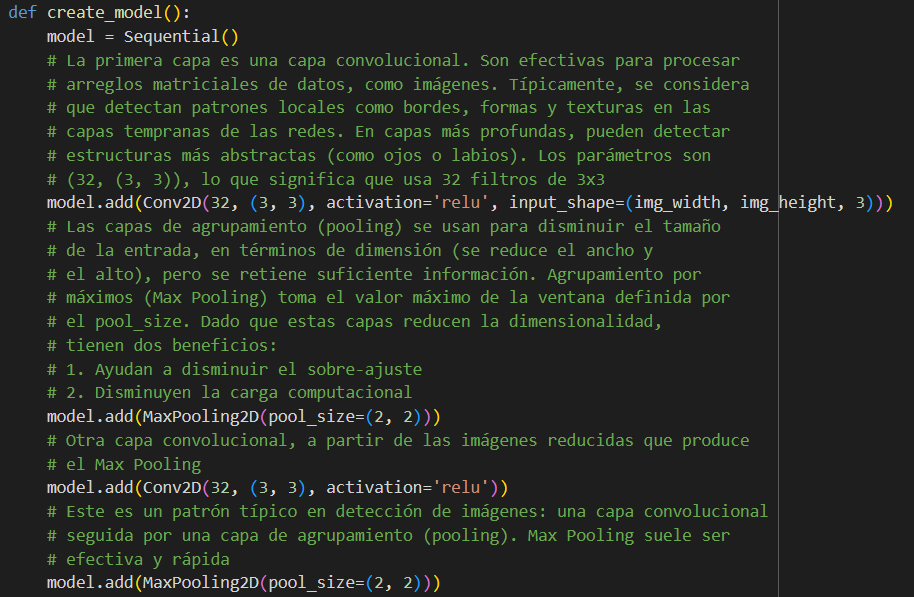


Figura 2. Capas de convolución, seguidas de capas de agrupamiento. Esta secuencia es típica de los modelos convolucionales tradicionales de visión computacional. Las capas convolucionales detectan detalles cada vez más abstractos cuantas más capas se agregan; y las capas de agrupamiento ayudan a mantener el tamaño de la red pequeño, además de que disminuyen el sobre ajuste.

Después de las capas de convolución y pooling, la entrada se aplanó para poder ser alimentada a las capas densas, que son responsables de la clasificación de las características aprendidas en las capas previas. Se incorporó una capa de abandono o "dropout" después de las capas densas para mejorar la generalización del modelo y minimizar el sobreajuste. Finalmente, la capa final, una capa densa con una sola neurona y una función de activación sigmoide, generó la salida del modelo, proporcionando la probabilidad de que la imagen contenga un peatón (Figura 3.

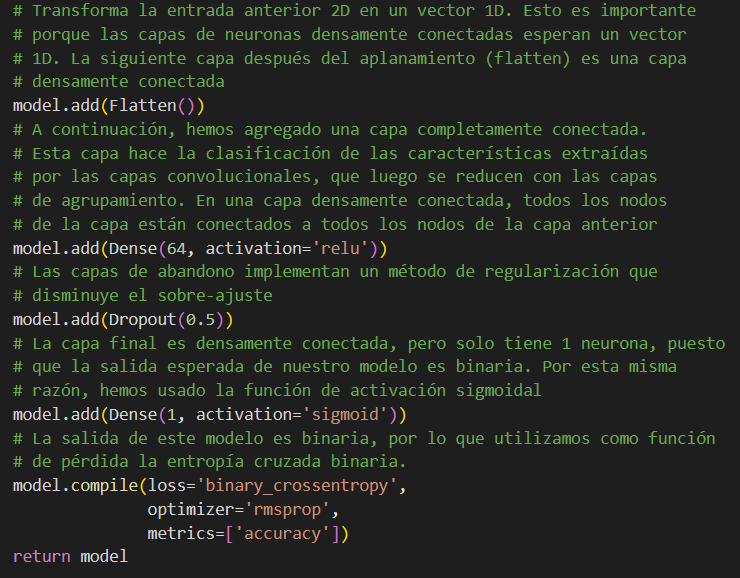


Figura 3. Capas de aplanamiento, densas y de abandono. Las capas de aplanamiento le dan la dimensión correcta (1D) al resultado de las capas de agrupamiento (2D), para que estas puedan ser conectadas con las densamente conectadas. La capa de abandono la agregamos para disminuir el sobre ajuste. Finalmente la capa de salida contiene solo una neurona que determina si la salida es “peatón” o “no peatón”, por lo que se utilizó una activación sigmoidal.

El modelo se compiló con una función de pérdida de entropía cruzada binaria, dado que el problema de detección de peatones es un problema de clasificación binaria, y se utilizó el optimizador 'rmsprop'.

Se realizó una validación cruzada estratificada de 5 divisiones para evaluar la eficacia del modelo, proporcionando una visión más precisa de cómo se desempeñará el modelo en datos no vistos.

Los resultados de la validación cruzada, representados en forma de gráficos de precisión y pérdida para cada pliegue, ofrecieron un medio visual para evaluar el rendimiento del modelo a lo largo de las épocas, tanto en el conjunto de entrenamiento como en el de validación. Al final, se calculó la precisión promedio y la desviación estándar de los 5 pliegues, proporcionando una medida general del rendimiento del modelo (Figura 4).

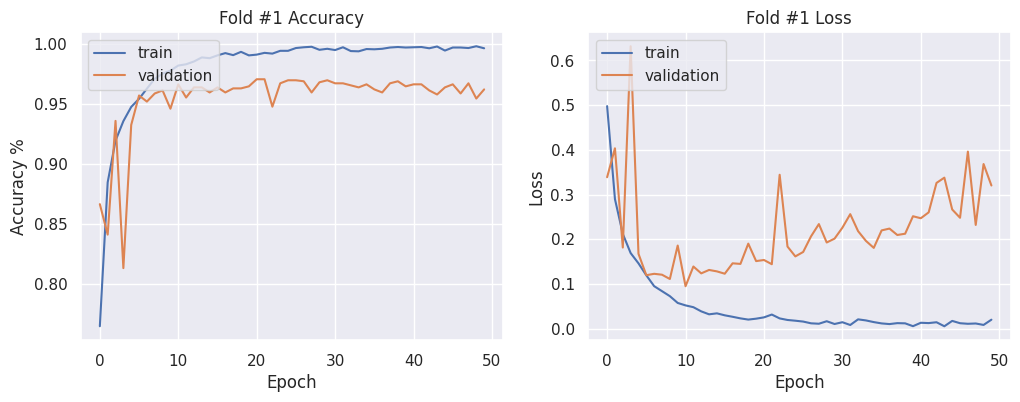


Figura 4. Ejemplo de gráficos de precisión y pérdida para el primer pliegue, durante la primera aproximación que utilizamos. El resto de los pliegues se pueden ver en el notebook.

En resumen, la CNN demostró ser una herramienta eficaz para la detección de peatones. El uso de la validación cruzada proporcionó una evaluación robusta del modelo, y la inclusión de capas de dropout y pooling ayudó a minimizar el sobreajuste.

A partir de esta primera aproximación, decidimos probar esta misma arquitectura, pero con menos ciclos de entrenamiento. Los resultados fueron igualmente satisfactorios (95.6% de precisión combinada entre los 5 pliegues.

Cuando entrenamos esta arquitectura para el modelo final, logramos 96.3% de precisión con nuestro conjunto de pruebas (Figura 5).

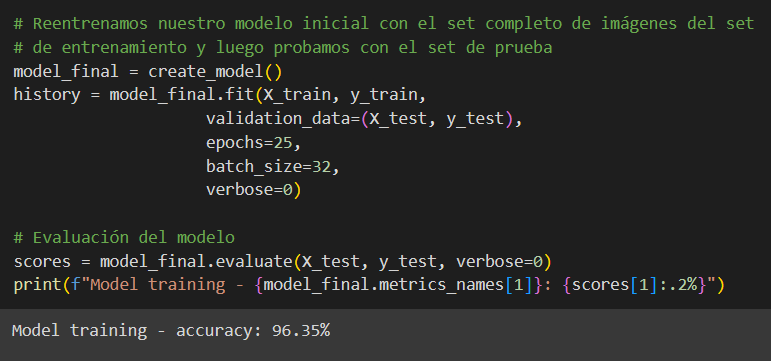


Figura . El modelo final cumple con las expectativas que se solicitan, con más de 90% de precisión.

**Referencias**

[1] Manaswi, N. K., & Manaswi, N. K. (2018). Understanding and working with Keras. *Deep learning with applications using Python: Chatbots and face, object, and speech recognition with TensorFlow and Keras*, 31-43.

[2] Soh, J., Singh, P., Soh, J., & Singh, P. (2020). Machine learning operations. *Data Science Solutions on Azure: Tools and Techniques Using Databricks and MLOps*, 259-279.

[3] Lakhani, P., Gray, D. L., Pett, C. R., Nagy, P., & Shih, G. (2018). Hello world deep learning in medical imaging. *Journal of digital imaging*, *31*, 283-289.

[4] Kienzler, R., & Kyas, H. (2020, January). Tensorflow 2.0 and Kubeflow for Scalable and Reproducable Enterprise AI. In *CS & IT Conference Proceedings* (Vol. 10, No. 1). CS & IT Conference Proceedings.

[5] Ostrovsky, M., Barrett, C., & Katz, G. (2022, October). An abstraction-refinement approach to verifying convolutional neural networks. In *Automated Technology for Verification and Analysis: 20th International Symposium, ATVA 2022, Virtual Event, October 25–28, 2022, Proceedings* (pp. 391-396). Cham: Springer International Publishing.