Reconstrucción de imágenes de señalamientos de tránsito utilizando Autoencoders

Emiliano Vásquez Olea – A01707035

1. Introducción

El campo del aprendizaje de máquina suele ser dividido en dos tipos principales, el aprendizaje supervisado y el no supervisado. Normalmente el aprendizaje supervisado se refiere a aquellos modelos o soluciones que son entrenados sobre datos que ya han sido etiquetados, por lo que se pueden realizar tareas como la clasificación y regresión. Por otro lado, el aprendizaje no supervisado consiste en encontrar patrones e información relevante de datos que no han sido etiquetados.

Además de estas áreas principales, existe la técnica del aprendizaje autosupervisado, la cual también se basa en el uso datos sin etiquetar, sin embargo, en la cual se puede llegar a una optimización debido a que los mismos datos son utilizados como resultado. Esta técnica puede ser utilizada para crear nuevas representaciones de la información con la que se esté trabajando, incluyendo imágenes.

El modelo que se presenta en esta entrega entra en esta área del aprendizaje autosupervisado y tiene como objetivo la codificación y decodificación de imágenes de señalamientos de tránsito. El *dataset* utilizado se encuentra en la plataforma *Kaggle* y se puede acceder a el a través de la siguiente liga:

https://www.kaggle.com/datasets/ahemateja19bec1025/traffic-sign-dataset-classification

2. Información del conjunto de datos

El *dataset* original está compuesto por diferentes carpetas e imágenes, con conjuntos de prueba y entrenamiento. Mientras que los datos se encuentran divididos por diferentes categorías de señalamientos de tránsito, al utilizar la técnica de *autoencoder* (aprendizaje autosupervisado) esta información no es necesaria para el entrenamiento y uso del modelo. Es por esto por lo que en la preparación del set de datos se toman todas las imágenes que lo conforman y se envían a una sola carpeta para ser utilizadas. De esta forma se reduce el almacenamiento de la información a lo siguiente:

- data_unlabeled: 6164 imágenes totales.

Mas adelante esta carpeta es dividida en dos conjuntos: *train*, donde se incluyen las 6164 imágenes originales, y *test*, donde se agregan una serie de imágenes adicionales para ser probadas al cargar el modelo. Estas imágenes son redimensionadas a un tamaño de 224 x 224 pixeles y mantienen su color.

Además de las imágenes, el conjunto de datos en *Kaggle* incluye un archivo de tipo CSV, donde se describe el tipo de señalamiento para cada una de las clases. Como fue mencionado anteriormente, esta información no es utilizada para el desarrollo del modelo debido a que no se toman en cuenta etiquetas de los datos.

3. Modelo de clasificación

El modelo utilizado es un *autoencoder*, que es utilizado para realizar operaciones de codificación y decodificación de las imágenes del *dataset*. Los *autoencoders* o autocodificadores permiten realizar tareas como la compresión de datos, extracción de características o la eliminación del ruido en los datos. Este último ejemplo es uno de los aspectos que serán analizados en las secciones de resultados y ajuste de hiperparámetros.

El *autoencoder* es construido como una red neuronal convolutiva, debido a que consiste en una serie de capas de convolución y de *Max Pooling*. Una característica importante de esta red es que es dividida en dos partes principales: el codificador y el decodificador, que pueden ser utilizados de forma separada para realizar su operación de forma individual (codificar o decodificar una imagen). Es importante mencionar que, para el proceso de entrenamiento, ambas partes son ajustadas en conjunto, debido a que la entrada del modelo completo y su salida esperada son la misma imagen.

La primera sección del modelo original, el codificador, está compuesta por una serie de capas convolucionales, cada una seguida por una operación de *Max Pooling*, teniendo como entrada la imagen con dimensiones 224 x 224 x 3. Las capas de convolución utilizan una cantidad de 16, 8 y 4 filtros, cada una con un tamaño de *kernel* de 3 x 3 y una función de activación *ReLU*. Por otro lado, las capas que aplican el *Max Pooling* usan un tamaño de 2 x 2, reduciendo las dimensionas de la imagen a la mitad en cada operación. Estas operaciones permiten extraer información importante de la imagen mientras que esta sigue un proceso de compresión, por lo que al pasar por el codificador se obtiene una representación más pequeña de la imagen original.

El decodificador utiliza una arquitectura similar al codificador, con capas convolutivas de 4, 8 y 16 filtros, sin embargo, se reemplazan los *Max pooling* por la operación de *Up Sampling* con una dimensión de 2 x 2 para incrementar el tamaño de la imagen. Además, se integra una última capa de convolución con 3 filtros, la cual tiene como objetivo regresar a las dimensionas o canal original de la imagen por el color en formato RGB. Esta segunda parte del modelo puede verse como un “reflejo” del codificador, que se encarga de reconstruir las imágenes originales a partir de su representación más compacta.

4. Resultados y Evaluación

El modelo inicial utiliza los siguientes parámetros para el proceso de entrenamiento:

- *optimizer*: “adam”, el optimizador utilizado para el entrenamiento.
- *loss*: “binary_crossentropy”, la función de pérdida usada para los ajustes en el proceso de entrenamiento. Se selecciona esta función debido a que la comparación o error es calculado por los pixeles de la imagen en un rango de 0 a 1.
- epochs: 15, el número de épocas o iteraciones del proceso de entrenamiento.
- steps_per_epoch: 100, número de pasos de ajuste realizados en cada época de entrenamiento.

Debido a que se utiliza un modelo de aprendizaje autosupervisado, la medida utilizada para identificar la precisión del modelo es la función de pérdida dentro del proceso de entrenamiento. Este valor se puede visualizar gracias a la ejecución del método *fit* y pasar por las épocas de entrenamiento:

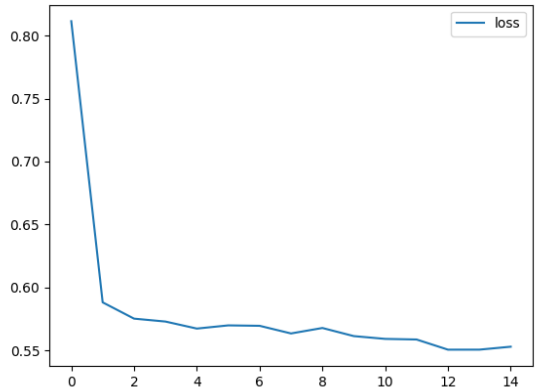


Fig 1. Pérdida a lo largo de las épocas con parámetros iniciales.

Después de las 15 épocas de entrenamiento puede observarse que el valor de la función de pérdida o *loss* llega a 0.5531, con un tiempo de entrenamiento de casi 10 minutos.

Una forma de evaluar el desempeño de este modelo es mediante la comparación de imágenes originales con la reconstrucción generada por el *autoencoder*:

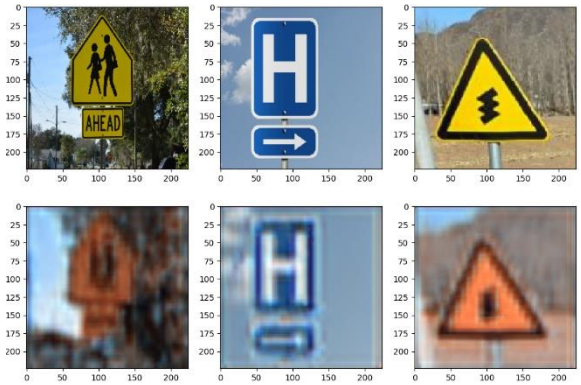


Fig 2. Resultados del uso del modelo con imágenes de prueba.

Aquí podemos ver su desempeño al reconstruir diferentes imágenes de prueba (resultados en imágenes en la segunda fila). De igual forma podemos probar su comportamiento con imágenes que presentan diferentes formas de “ruido” o alteraciones, buscando que la nueva representación logre eliminar estos aspectos:

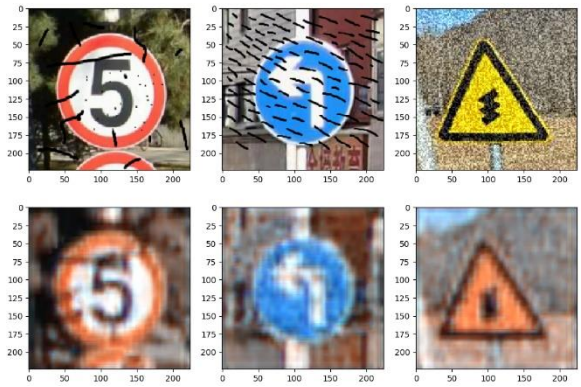


Fig 3. Resultados del uso del modelo con ruido aplicado.

Podemos ver que este modelo inicial es capaz de reducir en cierta medida las alteraciones que siguen un patrón definido dentro de las imágenes, como se puede mostrar con los rayones o cambios de color en la segunda y tercera imagen. Sin embargo, los rayones que son presentados en la primera imagen pueden verse aún presentes después de la reconstrucción. El siguiente paso es realizar una serie de modificaciones a los hiperparámetros del modelo para buscar una mejora o cambio de comportamiento con estas imágenes de prueba.

5. Ajuste de Parámetros

Se probaron diferentes ajustes a los hiperparámetros del modelo con el objetivo de mejorar su desempeño en la reconstrucción de imágenes, por ejemplo, para eliminación del ruido en las mismas. A continuación, se presentan un par de modificaciones junto con sus resultados en los procesos de entrenamiento y al evaluar con imágenes de prueba.

Primero se realiza una modificación a la arquitectura del modelo, agregando una capa convolutiva con 32 filtros a ambas secciones del modelo. Esto quiere decir que hay una capa adicional en el codificador y otra en el decodificador, lo cuál agrega cierta complejidad o peso al modelo. Al terminar el proceso de entrenamiento el valor de la función de pérdida se reduce a 0.5512, sin embargo, el tiempo de entrenamiento se incrementó a alrededor de 25 minutos.

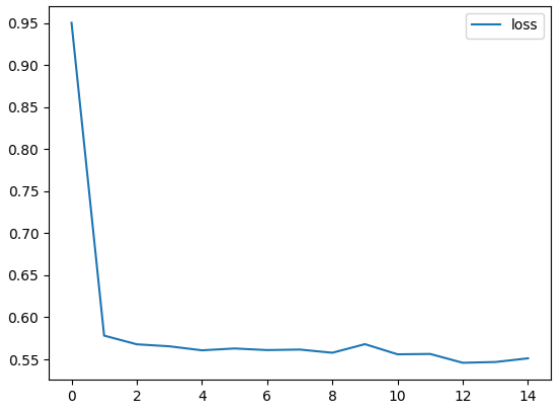


Fig 4. Pérdida a lo largo de las épocas con capas adicionales en la arquitectura.

Mientras tanto, se obtuvieron los siguientes resultados para la reconstrucción de imágenes de prueba:

Fig 5 y 6. Resultados de prueba del modelo con arquitectura modificada.

Se puede observar una ligera mejora en el detalle de la reconstrucción de ciertas imágenes. Sin embargo, en ejemplos

con ruido que sigue un patrón específico, como en la segunda imagen con ruido aplicado, se puede ver claramente el efecto sobre la imagen.

La segunda modificación implementada incluye un cambio al proceso de entrenamiento, reduciendo el tamaño de los *batches* o grupos de imágenes de entrenamiento a 8. Se mantiene el número de épocas usado en los demás modelos y se llega a un valor en la función de pérdida de 0.5498. El tiempo total de entrenamiento llega a alrededor de 11 minutos.

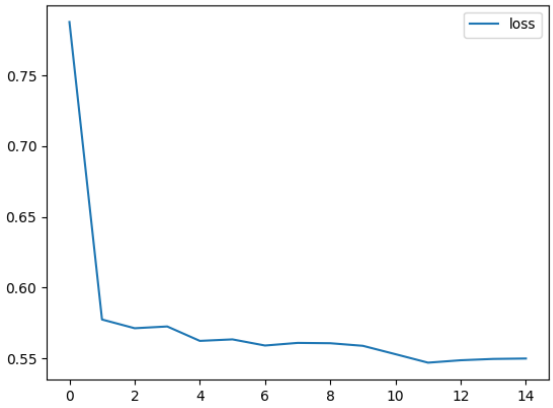
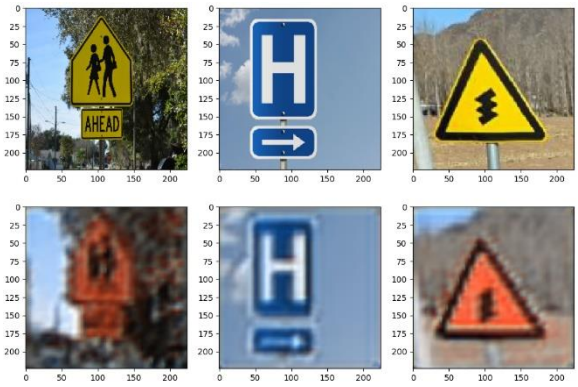


Fig 7. Pérdida a lo largo de las épocas con el modelo utilizando *batches* de menor tamaño.

Para este segundo modelo se obtuvieron los siguientes resultados al probar con las imágenes:



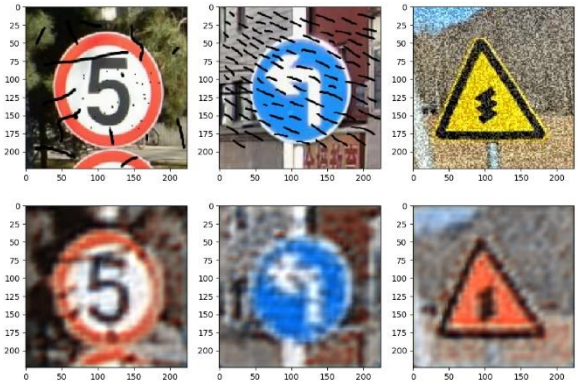


Fig 8 y 9. Resultados de prueba del modelo con *batches* de menor tamaño.

Se ve una leve reducción en la pérdida y mejora en definir ciertos detalles de las imágenes, pero la reducción de ruido tiene un comportamiento similar al modelo inicial.

Los modelos y *scripts* respectivos para su entrenamiento se encuentran en la carpeta de la entrega de este proyecto, donde se puede usar el archivo de Python de despliegue para probarlos con nuevas imágenes.

6. Conclusión y resultados finales

De las diferentes modificaciones probadas, se puede observar una diferencia en el comportamiento al reconstruir cada imagen. Además, al tener cambios en la función de pérdida leves entre cada versión, es difícil utilizar esto como métrica para seleccionar un modelo como óptimo. Es por esto por lo que se puede explorar la posibilidad de integrar los distintos modelos para resaltar o manejar ciertas imágenes o tipos de ruido específicos. De igual forma es importante tomar en cuenta la diferencia en tiempos de entrenamiento entre modelos, donde el modelo con una

modificación en su arquitectura tiene una mayor duración.

En general, esta implementación de *Autoencoders* muestra ser confiable en la compresión y reconstrucción de imágenes de señalamientos, además de que utiliza una arquitectura relativamente ligera. Entre los posibles cambios o mejoras que se pueden implementar al modelo se encuentran: cambios a la arquitectura del codificador y decodificador o un incremento en épocas de entrenamiento.