Deep Learning para la clasificación de imágenes: Un enfoque híbrido con autoencoders y CNNs en Fashion-MNIST

Enzo Nicolás Manolucos
Facultad de Matemática, Astronomía, Física y Computación, Universidad Nacional de Córdoba
enzo.manolucos@mi.unc.edu.ar

Abstract

En este trabajo se presenta el desarrollo y validación de un autoencoder y un clasificador convolucional con PyTorch utilizando el dataset Fashion-MNIST. El proyecto se divide en cuatro partes. En la primera, se diseña un autoencoder compuesto por capas convolucionales en el encoder y capas convolucionales transpuestas en el decoder. La segunda parte abarca el entrenamiento y validación del modelo, analizando variaciones de hiperparámetros para optimizar la reconstrucción de imágenes. En la tercera parte, se reutiliza el encoder para desarrollar un clasificador convolucional, añadiendo una capa de clasificación. Finalmente, se explora el re-entrenamiento selectivo de las capas para mejorar el desempeño. Los resultados validan la eficacia de estas arquitecturas en la reconstrucción y clasificación de imágenes.

1 Introducción

En el procesamiento de imágenes, las redes neuronales cumplen una función muy importante. Para entrenarlas, se necesita un conjunto de datos (dataset) con la cual pueda aprender a identificar características de las mismas. Un dataset muy popular para este tipo de tareas es Fashion-MNIST, el cual contiene imágenes de prendas de vestir.

Con estas imágenes se pueden realizar diversas tareas, como reconstrucción y clasificación. Los autoencoders se utilizan para la reducción de dimensiones y la extracción de características para poder reconstruir imágenes.

Un autoencoder es entrenado para intentar copiar sus entradas a sus salidas y poder reconstruirlas (Goodfellow et al., 2016). La Figura 1 muestra un autoencoder compuesto por dos partes: un codificador (encoder) que comprime la información y un decodificador (decoder) que genera una reconstrucción. Están diseñados de manera que solo puedan copiar de forma aproximada y, solo copiar entradas similares a los datos de entrenamiento.

Tanto el encoder como el decoder están formados por capas convolucionales, que son las encargadas de extraer características al aplicar filtros (*kernels*) sobre las imágenes.

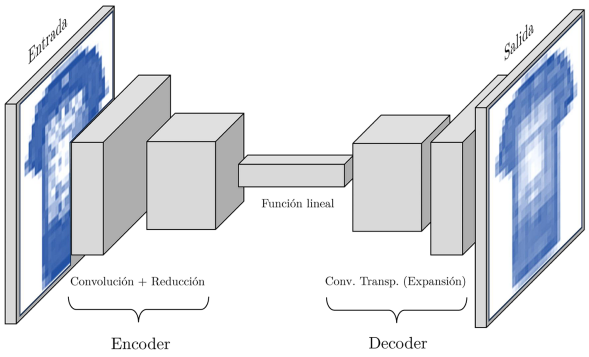

Figura 1: Esquema de un autoencoder convolucional de imágenes. La imagen de entrada se le aplica una convolución y reducción (encoder), seguido de una función lineal y una convolución transpuesta (decoder) para obtener una imagen reconstruida.

A partir de un encoder, se puede construir una red neuronal convolucional (Convolutional Neural Network, CNN), herramientas fundamentales para procesar imágenes. Las CNN son redes neuronales que utilizan la convolución en lugar de la multiplicación matricial en al menos una de sus capas seguida de una capa clasificadora.(Goodfellow et al., 2016). La Figura 2 presenta un ejemplo de CNN compuesto por un encoder y clasificador.

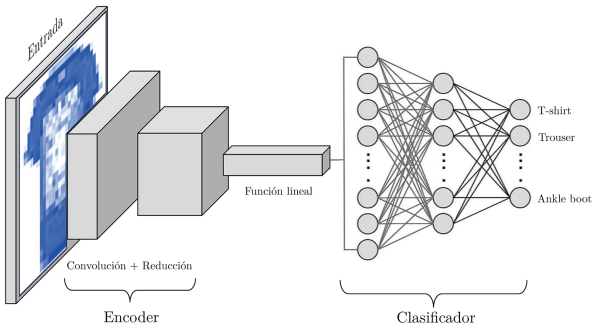

Figura 2: Esquema de un clasificador CNN de imágenes. La imagen de entrada pasa por capas de convolución y reducción (encoder), seguida de capas completamente conectadas (FC) que clasifican la imagen y asignan una clase.

2 Teoría

2.1 Convolución

La convolución es una operación que combina dos funciones para producir una tercera. Un vector de entrada $x[i]$ y un *kernel* $w[i]$ se combinan, de tal manera que

$$y[i] = \sum_{k=-\infty}^{\infty} x[i-k] \cdot w[k],$$

el *kernel* $w[i]$ se invierta y luego se desplace sobre la entrada $x[i]$. La única razón para voltear el *kernel* es para mantener la propiedad conmutativa (Goodfellow et al., 2016).

Muchas librerías, como PyTorch, implementan la correlación cruzada pero la llaman convolución. Esta operación es igual a la convolución pero sin invertir el *kernel*. En este trabajo se llamará convolución a ambas.

Para imágenes, la convolución combina una imagen de entrada con un *kernel* para producir una nueva matriz bidimensional llamada mapa de características. Se compone en su totalidad por los valores de la imagen filtrada. Matemáticamente se define como

$$y[i,j] = \sum_{m=-M}^{M} \sum_{n=-N}^{N} x[i+m], j+n] \cdot w[m,n],$$

donde nuevamente, x es la entrada, w es *kernel* e y es el mapa de características. Al realizar la operación, el *kernel* se desplaza sobre la imagen. En cada posición se multiplican y suman los elementos para producir un único valor. Este valor se coloca en la posición correspondiente del mapa de características.

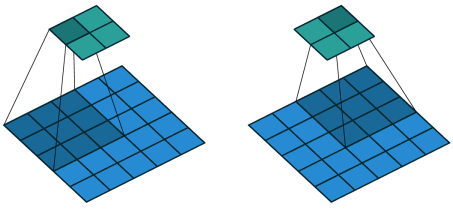

Figura 3: Ejemplo de los primeros dos desplazamientos de una convolución. La operación se realizada entre una entrada 5x5 (azul) y un *kernel* 3x3 (gris), con *stride* = 2 y *padding* = 0, para obtener una salida 2x2 (verde) (Dumoulin and Visin, 2018).

El resultado de la convolución suele ser una matriz más pequeña que la entrada, ya que el *kernel* no puede extenderse más allá de los bordes de la imagen de entrada. Para obtener un resultado que conserve las dimensiones de la entrada se deben tener en cuenta dos conceptos: *stride* y *padding*.

El *stride* define cuantos indices, o pixeles al tratarse imágenes, se mueve el *kernel*. Un *stride* = 1 significa que el *kernel* se mueve pixel por pixel, generando la máxima cantidad de características posibles. Aumentar este valor produce desplazamientos más grandes, reduciendo el tamaño de la salida.

Por su parte, el *padding* consiste en añadir bordes alrededor de la matriz de entrada para controlar el tamaño de salida. La Figura 3 muestra un ejemplo de convolución.

El ancho y alto del mapa de características, luego de la convolución, esta definido por:

$$W_{out} = \left[\frac{W_{in} + 2 \cdot Pad - Kernel}{stride}\right] + 1,$$
$$H_{out} = \left[\frac{H_{in} + 2 \cdot Pad - Kernel}{stride}\right] + 1.$$

2.2 Convolución Transpuesta

La convolución transpuesta surge de la necesidad de realizar el camino opuesto a la convolución. No garantiza recuperar la matriz de entrada, ya que no esta definida como la inversa de una convolución, sino que recupera la matriz con el mismo ancho y alto (Dumoulin and Visin, 2018).

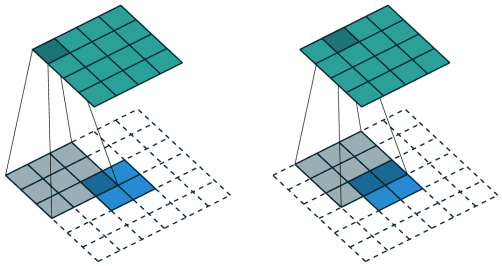

Figura 4: Ejemplo de los primeros dos desplazamientos de una convolución transpuesta. La operación se realizada entre una entrada de 2x2 (azul) y un *kernel* 3x3 (gris), con *stride* = 1 y *padding* = 0 para obtener una salida 4x4 (verde)(Dumoulin and Visin, 2018).

En la convolución transpuesta, un aumento en el tamaño del *kernel* provoca que cada número individual de la matriz de entrada se disperse sobre una región más amplia. Por lo tanto, cuanto mayor sea el tamaño del *kernel*, mayor sera la matriz de características.

Ahora, el *stride* controla el factor de expansión de la matriz de salida, determinando como se distribuyen los valores de la entrada en la salida.

A diferencia de la convolución normal, donde se utiliza el *padding* para expandir la matriz de características a la imagen de entrada, acá se utiliza para ajustar y reducir el tamaño de la salida. La Figura 4 muestra un ejemplo de convolución transpuesta.

El tamaño de a salida de una convolución transpuesta esta dado por

$$H_{\text{out}} = (H_{\text{in}} - 1) \cdot \text{stride} - 2 \cdot \text{padding} + \text{dilation} \cdot \text{kernel} + \text{output padding} + 1.$$

2.3 Autoencoder

Los autoencoders están compuestos por dos redes concatenadas: una red codificadora (encoder) y una red decodificadora (decoder). Para explicar su funcionamiento se utilizaran vectores.

El encoder toma como entrada un vector de dimensión d, asociado con un ejemplo x, es decir, $\mathbf{x}_d$ y lo transforma en un vector z de dimensión p, es decir, $\mathbf{z}_p$. El encoder se encargar de modelar la función

$$z = f(x),$$

el vector codificado z también se conoce como el vector latente (Raschka et al., 2022).

Usualmente, la dimensión del vector latente es menor que la de los ejemplos de entrada. Por lo tanto, podemos decir que el encoder actúa como una función de compresión de datos.

Luego, el decodificador descomprime $\hat{\mathbf{x}}$ a partir del vector latente de menor dimensión, z, donde podemos considerar al decodificador como una función,

$$\hat{\mathbf{x}} = g(\mathbf{z}).$$

El objetivo de un autoencoder es minimizar la diferencia entre la entrada original y la entrada reconstruida, lo que obliga al autoencoder a aprender características útiles en los datos. El autoencoder convolucional es una variante que utiliza capas convolucionales en lugar de capas completamente conectadas, lo que los hace más adecuados para tareas relacionadas con imágenes.

2.4 Red Neuronal Convolucional (CNN)

Una red neuronal convolucional (CNN) es un tipo de red neuronal diseñada para procesar datos, como imágenes. La convolución es el componente principal, identificando patrones. Están compuestas por:

- Convolución
- Agrupamiento (*Pooling*)
- Capas Completamente Conectadas (FC)

Primero, se realizan múltiples convoluciones en paralelo aplicando *kernels*. Al desplazarse por la entrada, se generan un conjunto de activaciones lineales, donde cada activación representa características específicas detectadas por el *kernel*. Posteriormente, estas activaciones son procesadas por una función de activación no lineal como la unidad lineal rectificada (*Rectified Linear Unit*, ReLU), definida como

$$f(x) = \max(0, x),$$

lo que introduce no linealidad al modelo, permitiéndole aprender relaciones complejas y representaciones jerárquicas de los datos.

El *pooling* es una operación de reducción de dimensión aplicada a las activaciones. Su propósito es condensar la información relevante, reducir la cantidad de parámetros y proporcionar robustez frente a pequeñas variaciones en la entrada. Existen dos métodos comunes: el *max pooling*, que selecciona el valor máximo dentro de una región específica, y el *average pooling*, que calcula el promedio. La Figura 5 muestra los tipos de *pooling*.

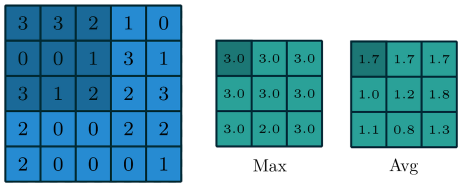

Figura 5: Ejemplo de *max pooling* y *average pooling* sobre una matriz 5x5, con *stride* = 1, generando una salida 3x3.

Por último, las FC operan sobre un vector, creado a partir de la matriz resultante, donde cada elemento está conectado a una neurona. Estas capas permiten combinar las características extraídas para optimizar objetivos. Al conectar todas las neuronas, las capas FC integran la información de alto nivel obtenida a través de las capas anteriores, permitiendo que la red realice predicciones precisas.

3 Resultados

Se fijaron los parámetros descritos en la Tabla 1 para ambos tipos de modelos. Para el entrenamiento y validación se graficaron las perdidas. En el caso del clasificador, se incluyó la precisión, que mide la proporción de predicciones correctas respecto al total de observaciones.

Adicionalmente, se construyó una matriz de confusión para evaluar al clasificador. Por definición, una matriz de confusión C es aquella donde el valor en la posición $C_{i,j}$ representa el número de observaciones que pertenecen a la clase i y que fueron clasificadas como j.

Las imágenes del dataset Fashion-MNIST tienen una resolución de 28x28 píxeles en escala de grises, y cada una está asociada con una de las 10 clases.

Parámetro	Valor
Imagen de entrada	28x28x1
Número de épocas	30
Tamaño de lote	100
Func. de activación	ReLU
Func. de pérdida	MSELoss (Autoencoder) CEL (Clasificador)
Tasa de aprendizaje	1×10^{-3}
Dispositivo	CPU

Tabla 1: Parámetros utilizados para entrenar los modelos.

3.1 Autoencoder

La Tabla 2 presenta los detalles de las capas que conforman el modelo *baseline* del autoencoder. Cada capa convolucional tiene 16 y 32 *kernels* respectivamente. Se utiliza un $Dropout = 0.2$.

Capa	Tipo	Salida	Kernel	Stride	Pad.
1	Conv	26x26	3x3	1	0
2	Max Pool	13x13	2x2	2	0
3	Conv	11x11	3x3	1	0
4	Max Pool	5x5	2x2	2	0
5	ConvT	13x13	5x5	2	2
6	ConvT	28x28	6x6	2	1

Tabla 2: Detalles de las capas que conforman el modelo *baseline* del autoencoder.

La Figura 6 muestra la pérdida del modelo con los cambios realizados.

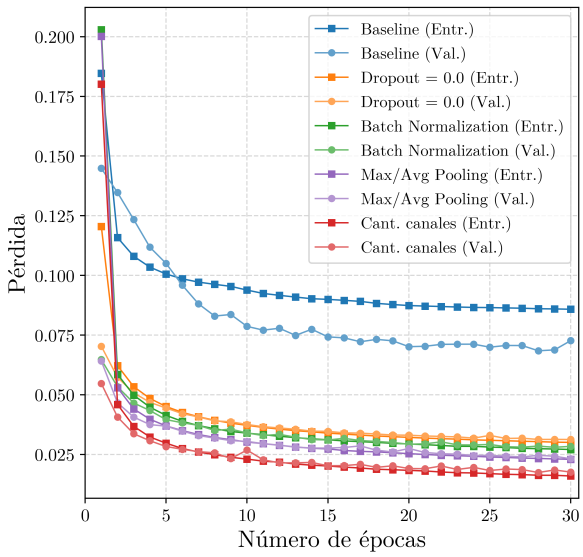

Figura 6: Pérdida del entrenamiento y validación del autoencoder a lo largo de las épocas. Se parte de una modelo *baseline* hasta obtener un modelo que tenga las menores pérdidas posibles.

Primero, se configuró el valor de $Dropout = 0$. Luego, se incorporó *batch normalization* antes de las capas de *pooling*. Esta técnica normaliza las entradas de cada capa para que presenten una media cercana a 0 y una desviación estándar de 1, reduciendo la influencia de valores atípicos en el entrenamiento.

Dado que el modelo incluye dos capas de *pooling*, se modificaron sus configuraciones: la primera utiliza *max pooling* y la segunda, *average pooling* (Gülcü and KUş, 2020).

Finalmente, se aumento el número de *kernels* a 64 y 96, respectivamente.

La Figura 7 presenta tres imágenes seleccionadas aleatoriamente del dataset Fashion-MNIST, junto con sus reconstrucciones generadas por el *baseline* y el modelo optimizado, este último elegido en función de la menor pérdida alcanzada durante el entrenamiento.

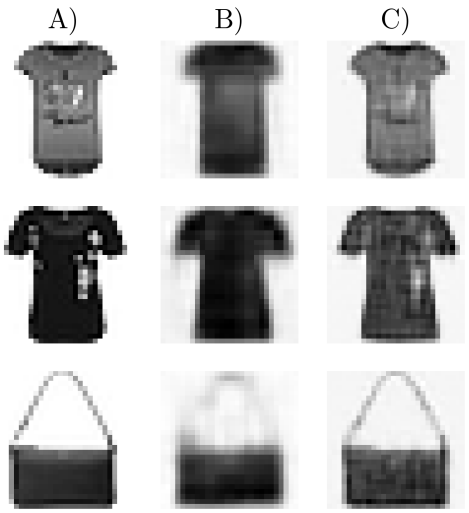

Figura 7: Comparación entre la entrada y las salidas del autoencoder. A) imagen original. B) imagen reconstruida por el modelo *baseline*. C) imagen reconstruida por el modelo optimizado.

3.2 Clasificador CNN

Una vez entrenado el autoencoder, se reutiliza encoder para construir el clasificador. El modelo *baseline* esta formado por el encoder optimizado y una capa FC de 10 salidas, correspondientes a las clases del dataset.

Primero se agrego más neuronas a la FC. Luego se mantuvo la cantidad de neuronas y se cambio el optimizador por descenso del gradiente estocástico (Stochastic gradient descent, SGD). La Figura 8 muestra los resultados obtenidos de la pérdida y la precisión.

La Figura 9 presenta los resultados de la matriz de confusión del clasificador.

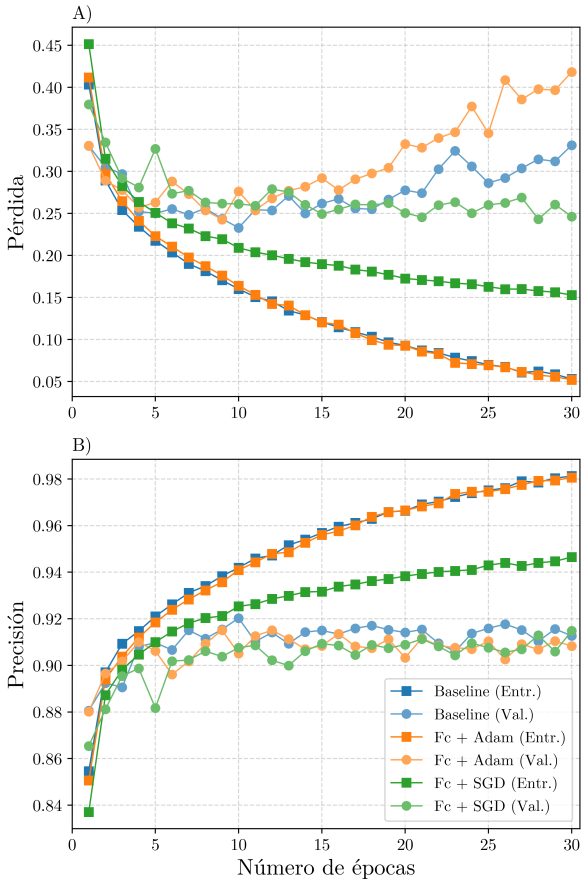

Figura 8: Pérdida y precisión del clasificador CNN durante el entrenamiento y validación. A) Pérdida obtenida del modelo *baseline*, modelo con más neuronas y modelo con SGD. B) Precisión obtenida del modelo *baseline*, modelo con más neuronas y modelo con SGD.

4 Discusión

4.1 Autoencoder

Los resultados obtenidos en la Tabla 3 muestran mejoras cuando hubo un cambios en los hiperparámetros. El modelo *baseline* presenta una pérdida menor al 10%.

Modelo	Entrenamiento	Validación
Baseline	8,58 %	7,26 %
Dropout	3,00 %	3,12 %
Batchnorm	2,70 %	2, 83 %
Max/Avg Pooling	2,29 %	2,31 %
Cant. canales	1,59 %	1,75 %

Tabla 3: Perdidas finales en el autoencoder.

Utilizar $Dropout = 0$ produjo una mejora significativa. Esto se debe a que el *dropout* anula de forma aleatoria neuronas para evitar sobreajuste. Dado que el autoencoder solo tiene dos capas, este parámetro puede llegar a reducir su desempeño.

Al utilizar *batch normalization* entre las capas convolucionale se eliminan los valores atípicos y se estabilizan los valores luego de cada convolución.

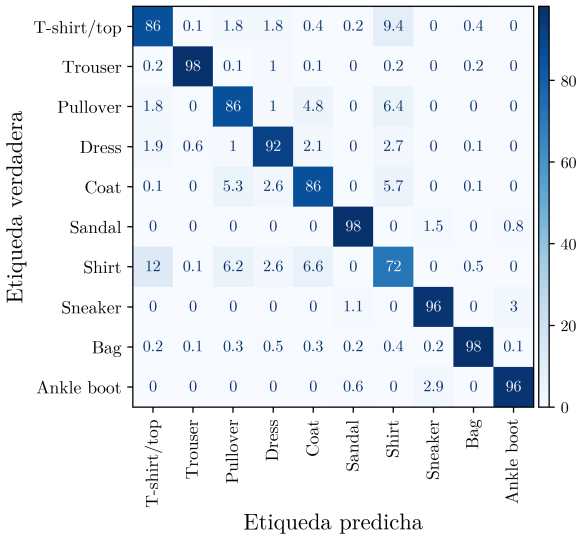

Figura 9: Matriz de confusión para el clasificador CNN con 10 categorías. Cada celda muestra el porcentaje de predicciones correctas e incorrectas de cada clase.

Modelo	Entrenamiento	Validación
Baseline	98,12 %	91,27 %
FC + Adam	98,06 %	90,83 %
FC + SGD	94,65 %	91,48 %

Tabla 4: Precisiones finales en el clasificador.

Cambiar solo la capa de *pooling* más profunda por una que tome los valores promedios mejoró los resultados. Utilizar el promedio en lugar del máximo suaviza los resultados sin generar ese contraste tan alto.

Por último, se incrementa la cantidad de canales en las capas convolucionales, lo que mejora aún más los resultados a costa de tiempos de entrenamiento más largos. Con más *kernels*, se pueden extraer más características.

A partir de la Figura 7 se observa, que el modelo *baseline* realiza una reconstrucción aceptable, pero omite detalles. La correa de la cartera pierde casi todos sus detalles, y las estampas de las remeras son irreconocibles. En cambio, el modelo optimizado captura la mayoría de los detalles aunque aún falla en reproducir los contrastes que otorgan profundidad a las imágenes.

4.2 Clasificador

La Tabla 5 muestra que el *baseline* tiene una precisión superior al 90%. El encoder utilizado ya presentaba un muy buen desempeño, aunque la pérdida empieza a presentar un sobreajuste luego de la mitad de las épocas.

Al aumentar la cantidad de neuronas se logra una pérdida menor y una precisión mayor, pero nuevamente empieza a sobreajustar. Converge más

rápido y alcanza una pérdida menor en el conjunto de entrenamiento, pero la validación sufre de oscilaciones y sobreajuste. Finalmente, con solo cambiar

Modelo	Entrenamiento	Validación
Baseline	5,31 %	33,12 %
FC + Adam	5,20 %	41,82 %
FC + SGD	15,29 %	24,63 %

Tabla 5: Perdidas finales en el clasificador.

la función de pérdida se logra una precisión igual a la obtenida en validación (Tabla 4), sin ningún tipo de sobreajuste. La precisión aumenta de forma progresiva y controlada.

Los resultados sugieren que Adam puede ser más eficiente, ya que en menos épocas se logran los mismo resultados pero es propenso al sobreajuste. En cambio, SGD es más robusto pero con un entrenamiento más lento.

La Figura 9 muestra la matriz de confusión del modelo FC + Adam. Se muestra que en general tiene un muy buen desempeño para clasificar las prendas. También indica que para categoría *shirt*, tiene una muy pequeña correlación con las categorías *T-shirt/top*, *Pullover* y *Coat*. Esto puede indicar que al tratarse de prendas que cubren unicamente el torso, se puede presentar cierta confusión en las imágenes similares. La matriz, a través de su escala de colores, muestra que hay una amplia diferencia entre los valores predichos correctamente y los que no, indican que el clasificador tiene un gran desempeño.

5 Conclusiones

En este trabajo, se analizó el rendimiento de un autoencoder y un clasificador, evaluando diversos ajustes de los hiperparámetros y sus efectos en los resultados.

Para el autoencoder, los experimentos mostraron que los cambios en los hiperparámetros mejoraron significativamente el desempeño del modelo. En particular, la eliminación del *dropout* resultó en una mejora notable. No obstante, en el caso de un autoencoder con solo dos capas, este parámetro podría afectar negativamente el rendimiento.

Además, la aplicación de *batch normalization* estabilizó los valores tras cada convolución, lo que contribuyó a una mayor precisión en la reconstrucción de las imágenes. La sustitución de la capa de *max pooling* por una de *average pooling* también trajo beneficios, suavizando los resultados sin perder demasiados detalles.

Finalmente, el aumento en la cantidad de canales en las capas convolucionales permitió extraer más características, mejorando aún más la reconstrucción, aunque a costa de mayores tiempos de entrenamiento. A pesar de estos avances, el modelo optimizado todavía tuvo dificultades para recuperar los contrastes que dan profundidad a las imágenes, lo que limita la fidelidad en la reconstrucción de algunos detalles.

En cuanto al clasificador, los resultados mostraron una precisión superior al 90%. Sin embargo, al aumentar el número de neuronas, se observó una mejora en la precisión y una menor pérdida en el conjunto de entrenamiento, aunque también se presentó sobreajuste.

La función de pérdida ajustada y la optimización con Adam lograron un aumento controlado en la precisión, sin sobreajuste, a diferencia de otros métodos que causaron oscilaciones en la validación. Los resultados también indicaron que Adam es eficiente al lograr buenos resultados en menos épocas, aunque es propenso al sobreajuste.

Por otro lado, el optimizador SGD, aunque más robusto, mostró un entrenamiento más lento. La matriz de confusión demostró que el clasificador tiene un excelente desempeño general.

Tanto el autoencoder como el clasificador presentaron mejoras sustanciales al modificar los hiperparámetros y técnicas de optimización, con resultados destacables en cuanto a precisión y capacidad de reconstrucción, aunque con algunas limitaciones que aún se pueden optimizar en futuros estudios.

Referencias

Vincent Dumoulin and Francesco Visin. 2018. A guide to convolution arithmetic for deep learning.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Ayla Gülcü and Zeki KUş. 2020. Hyper-parameter selection in convolutional neural networks using microcanonical optimization algorithm. *IEEE Access*, 8:52528–52540.

Sebastian Raschka, Yuxi (Hayden) Liu, and Vahid Mirjalili. 2022. *Machine Learning with PyTorch and Scikit-Learn*. Packt Publishing, Birmingham, UK.