Redes Neuronales: Trabajo Final Integrador

Palacio Marina[1]

[1]Facultad de Matemática, Astronomía, Física y Computación, Universidad Nacional de Córdoba, Ciudad Universitaria, 5000 Córdoba, Argentina

Agosto 2025

Resumen

Este trabajo se enfoca el entrenaminto de redes neuronales convolucionales (CNN), especialmente en un autoencoder y un clasificador de imágenes. Amplicados a la base de datos Fashion-MNIST, que clasifica diferentes prendas de vestir. Nota: archivo de gráficos para ver en mas detalle: link.

1. Introducción

Las redes neuronales convoluciones (CNN) son ampliamente utilizadas debido a su buen rendimiento en tareas de reconocimiento de imágenes. Su arquitectura ayuda a reconocer patrones, permitiendo extraer características visuales. La capacidad de identificar objetos tiene aplicaciones en diversas áreas, tales como: diagnóstico médico [1, 2, 3, 4], detección de anomalías en imágenes satelitales [5] y conducción autónoma de vehículos [6]. Esto impulso su integración en sistemas de apoyo a la toma de decisiones, donde la precisión y rapidez de procesamiento son importantes. Es una área con mucho potencian para explorar y con muchas posibles ventajas que fomentan su estudio.

2. Metodología

En este trabajo se utilizo como base de datos Fashion-MNIST [7], que contiene imágenes etiquetadas en 10 categorías de ropa. Sobre este conjunto se programaron y entrenaron redes neuronales convolucionales (CNN), un autoencoder y dos clasificadores.

El autoencoder identifica patrones en las imágenes y comprime la información para luego reconstruir la imagen inicial. La primera parte del autoencoder, encargada de comprimir la información, se utilizo luego como base para un clasificador. De esta forma, se utilizaron las representaciones aprendidas, que capturan las características más relevantes de cada prenda, para luego ser utilizada en un clasificador. Esta primera parte se deja fija y solo se entrena la segunda parte que clasifica el tipo de ropa utilizada. Se comparo este modelo con otro con la misma arquitectura pero sin el entrenamiento previo. Los modelos se implementaron en Python mediante PyTorch. Los pseudocodigos se encuentran disponibles online en el siguiente link.

En general, los modelos implementados se basan en capas convolucionales combinadas con dropout, el cual se aplica en todas las etapas con el fin de mitigar el sobreajuste, utilizando una tasa ajustable. En el caso del autoencoder, la arquitectura está compuesta por un encoder y un decoder diseñados de manera simétrica para comprimir y reconstruir las imágenes. El encoder utiliza dos bloques $Conv2D–ReLU–MaxPooling$ que extraen características relevantes y reducen la resolución de la imagen de entrada (28×28 px en escala de grises) hasta una capa de dimensión configurable (n). El decoder expande capa mediante una capa totalmente conectada y operaciones de ConvTranspose2D, restaurando la resolución original. La última capa emplea una activación Sigmoid, que normaliza los valores de salida al rango [0, 1]. Por su parte, el modelo clasificador combina un encoder convolucional con un clasificador totalmente conectado. El encoder comparte la misma estructura que el utilizado en el autoencoder, mientras que el clasificador consta de una capa lineal intermedia, seguida de la capa de salida ReLU con 10 neuronas que son el número de clases en el dataset Fashion-MNIST.

Se estudió los distintos modelos y se exploraron los distintos parámetros que afectan su precisión y capacidad de generalización. Se probaron variaciones en el tamaño de la capa oculta, el tamaño del batch, el valor de dropout (p) y la tasa de aprendizaje (learning rate, lr). También se experimentó con distintos optimizadores y tipos de capa de salida. En general se utilizaron como parámetros $n = 64$, $lr = 1{,}10^{-3}$, $p = 0{,}2$, $batch - size = 100$, como optimizador se uso ADAM. Para el autoencoder se uso como función perdida el Error Cuadrático Medio (ECM) y para el Clasificador Cross Entropy Loss (CEL).

3. Resultados

3.1. Autoencoder

En la figura 1 se muestra la evolución de la función de pérdida con las épocas de entrenamiento para el

autoencoder. Se usa el modelo entrenado en 30 épocas. Nota: los datos de test fueron graficados con el dropout mientras entrenaba por lo cual la curva de test no es la adecuada, lo correcto seria hacerlo una vez terminado el entrenamiento apagando el dropout.

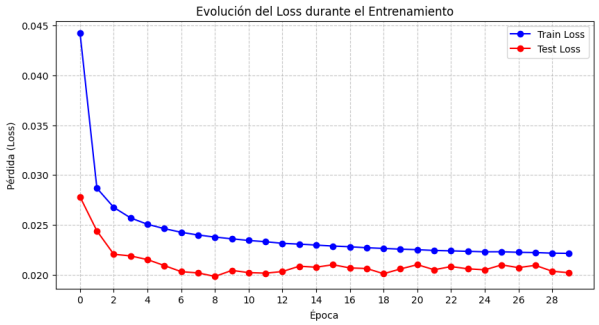


Figura 1: Función perdida con las épocas de entrenamiento para el autoencoder

3.1.1. Exploración de hiperparámetros

En la figura 2 se muestra la evolución de la función de pérdida con las épocas de entrenamiento al cambiar la ultima capa del autoencoder, se utiliza una capa ReLU y una Sigmoid. Se observa que para la primera el entrenamiento no mejora la función perdida mientras que para la segunda si converge, esto se debe a que la imagen de a función RelU es $(0 : \infty)$ mientras que para Sigmoid su imagen es $(0 : \infty)$ lo cual se adapta mejor a los valores de los pixeles que van entre 0 y 1.

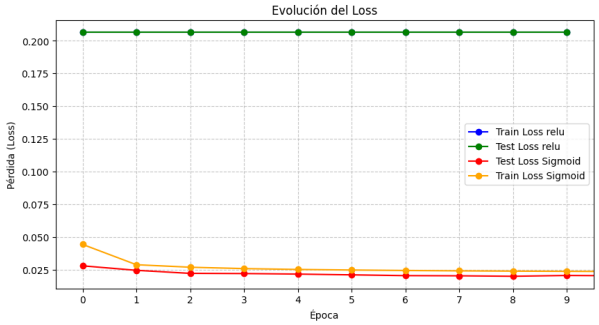


Figura 2: Función de pérdida con las épocas de entrenamiento para dos salida del autoencoder, ReLU y una Sigmoid.

En la figura 3 se muestra la función perdida en función de las épocas de entrenamiento para dos optimizadores distintos, SGD y ADAM. Se observa que para el primera el entrenamiento no mejora la función perdida mientras que para la segunda si converge, esto se debe a que ADAM escapa más fácil de los puntos de inflexión de la función perdida, es mas adaptativo.

El autoencoder se basa en eliminar información redundante y destacar lo esencial. Entre la capa de entrada y la de salida se encuentra una capa oculta, donde los datos se comprimen, forzando a la red a capturar la estructura subyacente del conjunto de datos. El número

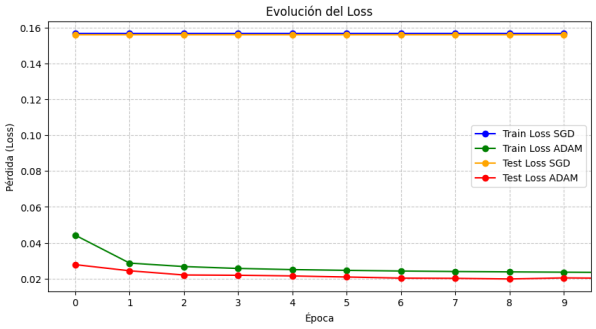


Figura 3: Función perdida con las épocas de entrenamiento para dos optimizadores distintos, SGD y ADAM..

de neuronas en esta capa, representado por el parámetro n, es importante, ya que determina la capacidad de la red para representar la información de manera eficiente. En la figura 4 se muestra la función perdida en variando de las épocas de entrenamiento para distintos número de neuronas de la capa oculta, n. Se observa que con $n = 16$ no se obtiene un buen modelo, no alcanza para representar la complejidad de las imágenes. Al aumentar n se obtiene una menor función perdida, teniendo en cuenta que no utilizar un valor demasiado grande ya que de utilizar $n = 784$ utilizas se copiaría la entrada (seria del mismo tamaño).

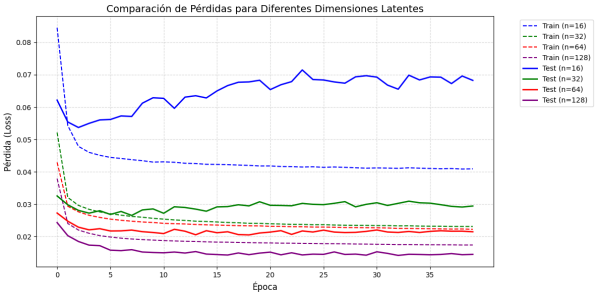


Figura 4: Función perdida con las épocas de entrenamiento para distintos número de neuronas de la capa oculta.

En la figura 5 se muestra la función perdida para distintos valores de dropout. Se observa que menor dropout dan menores valores de la función perdida, esto sugiere utilizar un dropout bajo en el modelo, se mantiene el valor $p = 0{,}2$. Ademas notar que nuestra base de datos es grande (50 mil ejemplos), no se necesita un valor muy alto pero se prefiere que sea no nulo para reducir overfitting.

En la figura 6 se muestra la función perdida para distintos valores del learning rate. Este factor es sumamente importante. Un valor muy grande puede hacer que nunca converga y uno demasiado chico se necesiten mas épocas de entrenamiento.

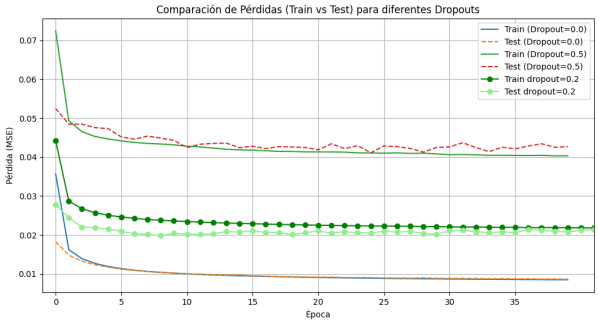


Figura 5: Función perdida con las épocas de entrenamiento para distintos valores de dropout.

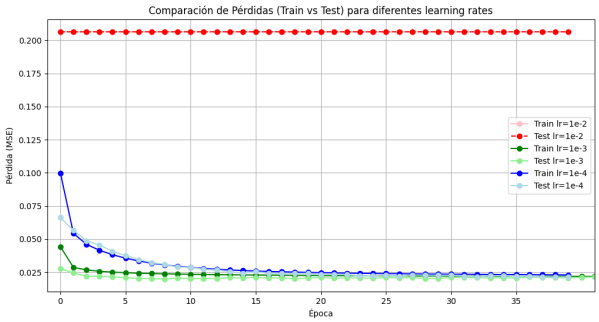


Figura 6: Función perdida con las épocas de entrenamiento para distintos valores del learning rate.

3.2. Clasificador

3.2.1. Con y sin preentrenamiento

En la figura 7 se muestra la función perdida variando las épocas de entrenamiento. Se observa que sin preentrenamiento se obtiene un menor valor final de la función perdida y un mejor valor final de precisión. Esto se debe a que con preentrenamiento el modelo recolecta información para reconstruir la imagen, no para reconocer patrones que ayuden a clasificar que tipo de prenda es.

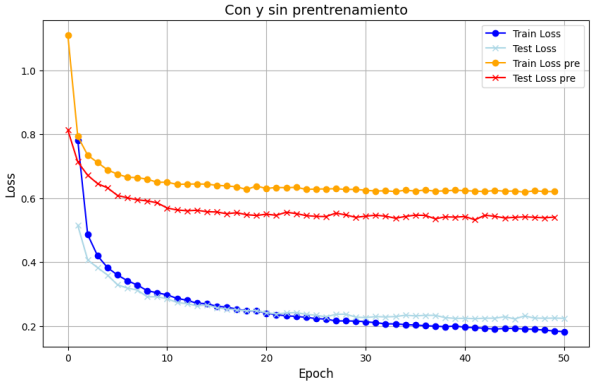


Figura 7: Función perdida con las épocas de entrenamiento.

En la figura 8 se muestra la función perdida variando las épocas de entrenamiento.

En la figura 9 se muestra la matriz confusión para los clasificadores con y sin preentrenamiento. Se observa una matriz con menos confusión en la clasificación para el modelo sin preentrenamiento. Las mayores confusiones son entre shirt, T-shirt, pullover y coat.

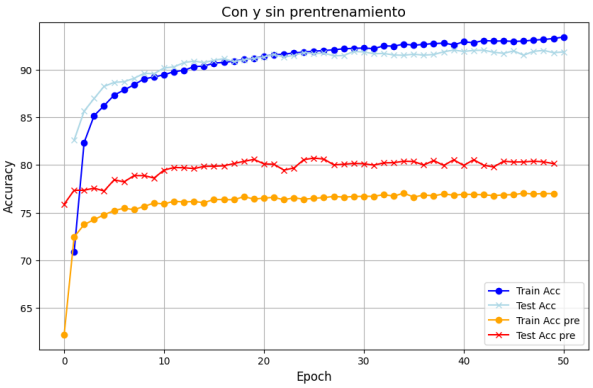
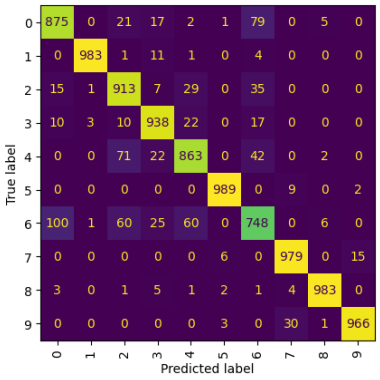
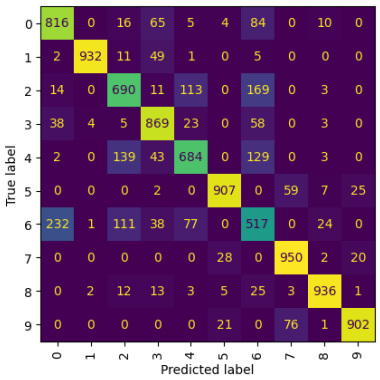


Figura 8: Función perdida con las épocas de entrenamiento.



(a) Sin entrenamiento.



(b) Con entrenamiento.

Figura 9: Comparación de la matriz confusión para el clasificador con y sin preentrenamiento.

3.2.2. Exploración de hiperparámetros

Al modelo de clasificación con preentrenamiento, hago una exploración de hiperparámetros. En la figura 10 se muestra la función perdida con las épocas de entrenamiento para distintos valores del learning rate, lr. Se observa que para $lr = 1{,}10^{-4}$ la convergenia es muy lenta, conviene usar $lr = 1{,}10^{-3}$.

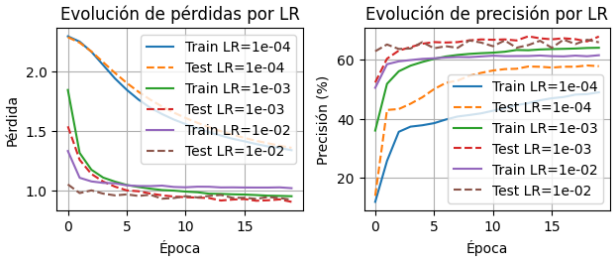


Figura 10: Función perdida con las épocas de entrenamiento para distintos valores del learning rate.

En la figura 11 se muestra la función perdida con las épocas de entrenamiento para distintos valores del número de neuronas de la capa oculta, n. Este valor final se obtiene luego de 20 épocas. Notamos nuevamente que la función perdida disminuye para mayor n, ahora la precisión también aumenta para n mayor.

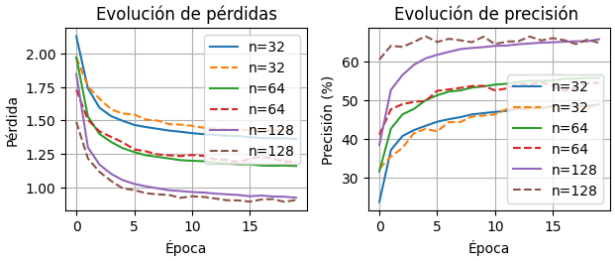


Figura 11: Función perdida con las épocas de entrenamiento para distintos valores del número de neuronas de la capa oculta, n

En la figura 12 se muestra la función perdida con las épocas de entrenamiento para distintos valores del tamaño de batch utilizado en el entrenamiento. Notar que se registra una menor perdida y mayor precisión al disminuir el batch size. Con batch size pequeños la actualización de pesos es mas frecuente lo que permite una mejor búsqueda entre los pesos óptimos.

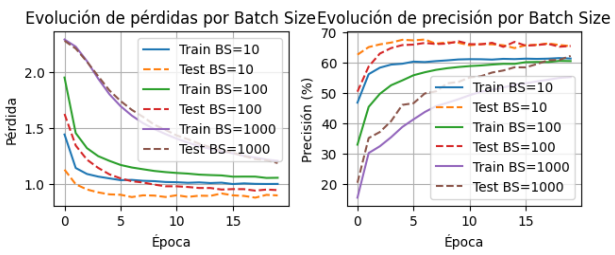


Figura 12: Función perdida con las épocas de entrenamiento para distintos valores del tamaño de batch utilizado.

4. Conclusiones

Sobre la elección de hiperparámetos:

- Learning rate: se elije como valor óptimo ($lr = 10^{-3}$) equilibra velocidad y estabilidad. Valores altos ($> 10^{-2}$) causan divergencia, mientras que valores bajos ($< 10^{-4}$) ralentizan el entrenamiento.
- Capas ocultas: El tamaño n debe ser suficiente para capturar patrones (ej., $n \geq 64$), pero no excesivo para evitar sobreajuste o redundancia $n \ll 784$.
- Dropout: Un valor moderado ($p = 0{,}2$) previene *overfitting* sin sacrificar capacidad de aprendizaje, especialmente en conjuntos de datos grandes como Fashion-MNIST (50k muestras).
- El autoencoder con Sigmoid y ADAM logró la mejor convergencia.

Sobre el modelado en general:

- El clasificador sin preentrenamiento superó al preentrenado en precisión y reducción de pérdida.

Referencias

[1] Thomas F Lüscher, Florian A Wenzl, Fabrizio D'Ascenzo, Paul A Friedman, and Charalambos Antoniades. Artificial intelligence in cardiovascular medicine: clinical applications. *European Heart Journal*, 45(40):4291–4304, 08 2024.

[2] Konstantin Piliuk and Sven Tomforde. Artificial intelligence in emergency medicine. a systematic literature review. *International Journal of Medical Informatics*, 180:105274, 2023.

[3] Shahid Shafi and Anil V. Parwani. Artificial intelligence in diagnostic pathology. *Diagnostic Pathology*, 18:109, 2023.

[4] Zhi-Hua Chen, Lin Lin, Chang-Fu Wu, Chao-Feng Li, Rui-Hua Xu, and Yan Sun. Artificial intelligence for assisting cancer diagnosis and treatment in the era of precision medicine. *Cancer Communications*, 41(11):1100–1115, 2021.

[5] Rebekah Esmaili, David Daniel, Haibing Sun, and Krish Narasimhan. Anomaly detection in satellite imagery using convolutional neural networks. 03 2025.

[6] I Sonata, Y Heryadi, L Lukas, and A Wibowo. Autonomous car using cnn deep learning algorithm. *Journal of Physics: Conference Series*, 1869(1):012071, apr 2021.

[7] Han Xiao; Kashif Rasul; and R. Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *Arxiv*, 2017.