Autoencoder y Clasificador Convolucional sobre Fashion-MNIST

Máximo Caprari*
Facultad de Ciencias Económicas,
Universidad Nacional de Córdoba,
Ciudad Universitaria, 5000 Córdoba, Argentina
(Dated: 12 de Febrero de 2025)

Las Redes Neuronales Profundas incluyen una amplia variedad de arquitecturas, cada una optimizada para resolver problemas específicos o para trabajar con distintos tipos de datos. Entre ellas, destacan las Redes Neuronales Convolucionales (*CNNs*), conocidas por su buen desempeño en el procesamiento de imágenes, y los Autoencoders, usados principalmente para reducir la dimensionalidad y para extraer patrones relevantes en los datos. Este informe analiza estas dos arquitecturas. Además, se incluyen experimentos prácticos para analizar la reutilización de encoders preentrenados, en tareas de clasificación.

I. Introducción

En el ámbito del aprendizaje no supervisado, los autoencoders convolucionales son herramientas clave para la reconstrucción de imágenes y la extracción de características relevantes [3], ya que pueden capturar representaciones espaciales y atributos específicos de los datos [5]. En este estudio, se plantea la utilización de un **Autoencoder Convolucional** para generar representaciones latentes de imágenes, las cuales se emplean posteriormente en el desarrollo de un **Clasificador Convolucional** orientado a la clasificación de imágenes. Para la experimentación, se utiliza el conjunto de datos Fashion-MNIST, compuesto por imágenes en escala de grises de 28×28 píxeles, representando prendas de vestir organizadas en 10 categorías.

Estructura del documento: En la Sección 2, se expone la base teórica relacionada con las **Redes Neuronales Convolucionales** y los **Autoencoders**. La Sección 3 detalla los experimentos llevados a cabo junto con los resultados obtenidos. En la Sección 4, se presenta una discusión sobre los resultados, y en la sección 5 las conclusiones extraídas de este trabajo.

II. Teoría

A. Redes Neuronales Convolucionales

Las **Redes Neuronales Convolucionales** (RNCs) son una clase de redes profundas diseñadas para procesar datos con estructura espacial, como imágenes. Su componente clave es la operación de convolución, que aplica filtros (*kernels*) sobre la entrada para extraer mapas de características (*feature maps*), destacando patrones relevantes [1]. Durante el entrenamiento, los pesos de estos filtros se ajustan para detectar estructuras significativas [2].

Las RNCs incluyen capas de *pooling*, que reducen el tamaño de los mapas de características, mejorando la robustez ante variaciones en la entrada [3]. La combinación de convolución, funciones de activación como ReLU y operaciones de *pooling* permite extraer representaciones jerárquicas, lo que hace a las RNCs ideales para tareas como clasificación de imágenes, detección de objetos y segmentación semántica [4].

B. Autoencoders

Los **Autoencoders** son redes neuronales que aprenden a copiar la entrada en la salida, lo que les permite aprender representaciones internas eficientes de los datos, llamadas **codificaciones** [1]. Aunque su objetivo es replicar la función identidad, las restricciones en su diseño (como un espacio latente de menor dimensión) fuerzan a la red a capturar las propiedades más útiles de los datos [2].

Un Autoencoder consta de dos partes principales: un **encoder** (codificador), que transforma la entrada en una representación latente, y un **decoder** (decodificador), que reconstruye la entrada a partir de esta representación [3]. En este proyecto, utilizamos un **Autoencoder Convolucional**, donde el encoder emplea capas convolucionales y el decoder utiliza **convoluciones transpuestas** para recuperar el tamaño original de la imagen [5].

III. Resultados

Se empleó el dataset Fashion-MNIST [3], realizando experimentos en primer lugar con un autoencoder convolucional y, posteriormente, con un clasificador convolucional. Las imágenes del dataset fueron previamente normalizadas. El objetivo principal de este estudio es evaluar

* maximo.caprari@mi.unc.edu.ar

si la reutilización del encoder preentrenado en el clasificador mejora el rendimiento en comparación con entrenar el clasificador desde cero.

Para el entrenamiento de todas las redes, se utilizó dropout [2] como estrategia para prevenir el sobreajuste.

A lo largo del texto, se denota con η la tasa de aprendizaje y con p la probabilidad de dropout aplicada.

A. Autoencoder Convolucional

1. *Comparación de Arquitecturas de Autoencoders*

En este estudio, se compararon diversas arquitecturas de autoencoders, variando el número de capas convolucionales en el *encoder* y sus parámetros (tamaño del kernel, *stride* y *padding*), así como la inclusión o exclusión de una capa lineal intermedia entre el *encoder* y el *decoder* [2]. Los hiperparámetros de entrenamiento se mantuvieron constantes en todos los experimentos. Las hipótesis planteadas fueron: (1) que una mayor profundidad de la red conduce a mejores resultados [3], y (2) que la presencia de una capa lineal intermedia no afecta significativamente el desempeño del modelo [5].

Cabe aclarar que para el entrenamiento de este tipo de redes, se emplea la función de pérdida **Error Cuadrático Medio**.

Las arquitecturas evaluadas se detallan en las Tablas I, II y III. En estas tablas, las dimensiones de las columnas "entrada" y "salida" se expresan en el formato (profundidad, ancho, largo).

La *Arquitectura I* presenta una estructura simple, con una sola capa convolucional en el *encoder* [1], mientras que en la *Arquitectura II* se introducen dos capas convolucionales, y en la *Arquitectura III*, tres [2]. En cada caso, la transformación de la entrada y las dimensiones intermedias pueden consultarse en los respectivos cuadros. El *decoder* reconstruye la imagen en función de la estructura de cada modelo, empleando una o más capas de convolución transpuesta según corresponda.

La Tabla IV resume la lista de experimentos realizados con el fin de explorar las distintas arquitecturas propuestas, habiendo dejado los hiperparámetros de entrenamiento fijos.

Capa	Entrada	Salida	Kernel	Stride	Padd.
Conv.+ReLU +Dropout	(1,28,28)	(16,28,28)	3×3	1	1
Max Pool	(16,28,28)	(16,14,14)	2×2	2	No
Flatten	(16,14,14)	3136	-	-	-
Lineal+ReLU + Dropout	3136	n	-	-	-
Unflatten	3136	(16,14,14)	-	-	-
Conv. Transp.	(16,14,14)	(1,28,28)	3×3	2	1
Sigmoide	(1,28,28)	(1,28,28)	-	-	-

Cuadro I: Arquitectura de Autoencoder 1: 1 capa convolucional en el encoder cuyo kernel es de 3×3.

Capa	Entrada	Salida	Kernel	Stride	Padd.
Conv.+ReLU +Dropout	(1,28,28)	(16,28,28)	3×3	1	1
Max Pool	(16,28,28)	(16,14,14)	2×2	2	No
Conv.+ReLU +Dropout	(16,14,14)	(32,14,14)	3×3	1	1
Max Pool	(32,14,14)	(32,7,7)	2×2	2	No
Flatten	(32,7,7)	1568	-	-	-
Lineal+ReLU + Dropout	1568	n	-	-	-
Unflatten	1568	(32,7,7)	-	-	-
Conv. Transp +ReLU +Dropout	(32,7,7)	(16,14,14)	3×3	2	1
Conv. Transp.	(16,14,14)	(1,28,28)	3×3	2	1
Sigmoide	(1,28,28)	(1,28,28)	-	-	-

Cuadro II: Arquitectura de Autoencoder 2: 2 capas convolucionales en el encoder, ambas con kernels de 3×3.

Capa	Entrada	Salida	Kernel	Stride	Padd.
Conv.+ReLU +Dropout	(1,28,28)	(16,28,28)	3×3	1	No
Max Pool	(16,28,28)	(16,14,14)	2×2	2	No
Conv.+ReLU +Dropout	(16,14,14)	(32,14,14)	3×3	1	No
Max Pool	(32,14,14)	(32,7,7)	2×2	2	No
Conv.+ReLU +Dropout	(32,7,7)	(64,7,7)	3×3	1	No
Max Pool	(64,7,7)	(64,4,4)	2×2	2	No
Flatten	(64,4,4)	1024	-	-	-
Lineal+ReLU +Dropout	1024	n	-	-	-
Unflatten	1024	(64,4,4)	-	-	-
Conv. Transp. +ReLU +Dropout	(64,4,4)	(32,7,7)	3×3	2	1
Conv. Transp. +ReLU +Dropout	(32,7,7)	(16,14,14)	3×3	2	1
Conv. Transp.	(16,14,14)	(1,28,28)	3×3	2	1
Sigmoide	(1,28,28)	(1,28,28)	-	-	-

Cuadro III: Arquitectura de Autoencoder 3: 3 capas convolucionales en el encoder, todas con kernels de 3×3.

Arq.	Con capa lineal	n	Optim.	η	Tam. de Lote	p	Error Mín. Entr.	Error Mín. Val.
1	Sí	256	Adam	1^{-3}	128	0.2	0.5925	0.5913
1	No	-	Adam	1^{-3}	128	0.2	0.6102	0.6088
2	Sí	256	Adam	1^{-3}	128	0.2	0.5903	0.5876
2	No	-	Adam	1^{-3}	128	0.2	0.5756	0.5726
3	Sí	256	Adam	1^{-3}	128	0.2	0.6816	0.6789
3	No	-	Adam	1^{-3}	128	0.2	0.5852	0.5822

Cuadro IV: Experimentos en donde se varían las arquitecturas del autoencoder, sin variar los hiperparámetros de entrenamiento. En todos los casos, se usó un número de 30 épocas.

A partir de los resultados obtenidos para las distintas arquitecturas, se observa que las pérdidas entre los conjuntos de entrenamiento y validación son muy similares. Además, se puede concluir que la arquitectura que alcanzó los mejores resultados fue la II, la cual no incluye la capa lineal intermedia. Por lo tanto, se ha decidido continuar con esta arquitectura y comenzar a experimentar con distintos hiperparámetros para tratar de mejorar el rendimiento del modelo y, finalmente, seleccionar el Autoencoder que muestre el mejor desempeño.

2. *Variación de hiperparámetros de la Arquitectura 2 sin capa lineal intermedia*

Los diferentes hiperparámetros utilizados junto con los resultados en cada caso se encuentran en la tabla V. En cada fila, se resaltan los hiperparámetros que fueron modificados partiendo desde los base, que corresponden a la primera fila.

Optim.	η	**Tam. de Lote**	p	**Error Mín. Entr.**	**Error Mín. Val.**
Adam	1^{-3}	128	0.2	0.5756	0.5726
SGD	1^{-3}	128	0.2	0.6629	0.6598
Adam	1^{-3}	128	**0.4**	0.5866	0.5828
Adam	$\mathbf{1^{-4}}$	**256**	0.2	0.5884	0.5851
Adam	$\mathbf{1^{-4}}$	**256**	**0.4**	0.5943	0.5924

Cuadro V: Experimentos en donde se varían los hiperparámetros de la arquitectura 2 sin capa lineal del Autoencoder. En todos los casos, se usó un número de 30 épocas.

Haber variado los hiperparámetros no trajo ninguna mejora significativa, y los valores de la tabla reflejan que los óptimos para la arquitectura seleccionada son los de la configuración base: optimizador Adam, tasa de aprendizaje 1^{-3}, tamaño de lote 128, y dropout de 0.2.

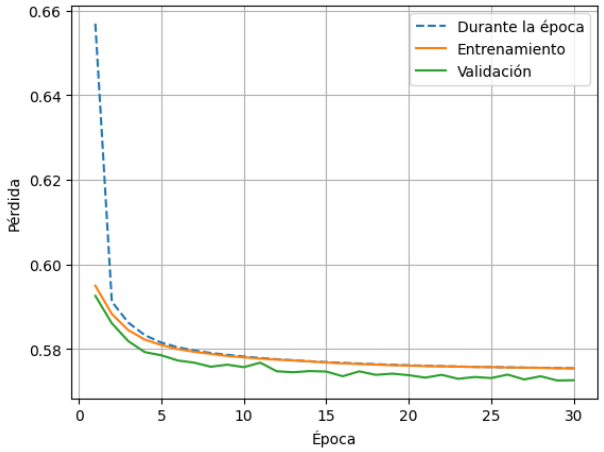


Figura 1: Pérdida promedio por lote del Autoencoder no lineal.

3. *Análisis del Tamaño del Espacio Latente en el Autoencoder*

Por último, se toma el autoencoder (II) y se evalúa el impacto de variar el tamaño del *espacio latente*, el cual determina el nivel de compresión de la información, afectando directamente la capacidad del modelo para reconstruir los datos de entrada. Un *espacio latente* de mayor dimensión preserva una mayor cantidad de información, aunque puede limitar la abstracción y generalización del modelo [1]. En contraste, un *espacio latente* más pequeño obliga al modelo a capturar únicamente las características más esenciales, lo que puede conducir a una pérdida significativa de detalles [3].

Los resultados correspondientes a la tabla VI muestran que, en este caso, un tamaño intermedio del *espacio latente* ofrece la mejor reconstrucción, manteniendo un nivel adecuado de compresión.

Arq.	n	**Optim.**	η	**Tam. de Lote**	p	**Error Mín. Entr.**	**Error Mín. Val.**
1	784	Adam	1^{-3}	128	0.2	0.5925	0.5913
1	256	Adam	1^{-3}	128	0.2	0.5870	0.5848
1	32	Adam	1^{-3}	128	0.2	0.6037	0.6016

Cuadro VI: Experimentos en donde se varía el tamaño del bottleneck en la arquitectura II, sin variar el resto de hiperparámetros de entrenamiento. En todos los casos, se usó un número de 30 épocas.

B. Clasificador Convolucional

En este problema de clasificación, hemos optado por utilizar la función de pérdida de **Entropía Cruzada**, complementada con la métrica de *accuracy* para evaluar el desempeño del modelo.

La arquitectura empleada para el Clasificador se describe en la tabla VII. Cabe destacar que las capas convolucionales utilizadas son las mismas que las del encoder del Autoencoder que se presentan en la tabla II.

1. *Clasificador con encoder preentrenado vs. Clasificador entrenado desde cero*

La primera comparación que realizamos consistió en evaluar el rendimiento de un clasificador utilizando el encoder preentrenado del Autoencoder seleccionado, entrenando únicamente la parte clasificadora, frente a entrenar todo el Clasificador desde cero, manteniendo constantes los hiperparámetros de entrenamiento en ambas configuraciones. Los resultados de las métricas obtenidas se presentan en la tabla VIII. Destacando que se utilizó Adam como optimizador, y se tomó $n_1 = 128$, $n_2 = 64$, una tasa de aprendizaje de 1^{-3}, un dropout de 0.2, y un tamaño de lote de 128.

Capa	Entrada	Salida	Kernel	Stride	Padd.
Conv.+ReLU +Dropout	(1,28,28)	(16,28,28)	3×3	1	1
Max Pool	(16,28,28)	(16,14,14)	2×2	2	No
Conv.+ReLU +Dropout	(16,14,14)	(32,14,14)	3×3	1	1
Max Pool	(32,14,14)	(32,7,7)	2×2	2	No
Flatten	(32,7,7)	1568	-	-	-
Lineal+ReLU + Dropout	1568	n_1	-	-	-
Lineal+ReLU + Dropout	n_1	n_2	-	-	-
Lineal	n_2	10	-	-	-

Cuadro VII: Arquitectura del Clasificador Convolucional: las primeras capas corresponden al encoder del Autoencoder seleccionado (tabla II). Se añaden tres capas para llevar a cabo la clasificación. Aparecen dos nuevos hiperparámetros: n_1 y n_2.

Encoder	Entrenamiento		Validación	
	Acc.	Error	Acc.	Error
Preentrenado	91.98 %	0.2329	89.30 %	0.2981
Sin entrenar	97.15 %	0.0878	91.99 %	0.2199

Cuadro VIII: Resultados obtenidos utilizando el encoder preentrenado y entrenando el clasificador desde cero.

Los resultados obtenidos revelan, en primer lugar, señales de sobreajuste, ya que las pérdidas son considerablemente más bajas en el conjunto de entrenamiento que en el de validación en ambos escenarios. Además, se observa que entrenar todas las capas del Clasificador desde cero produce mejores resultados que reutilizar el encoder preentrenado.

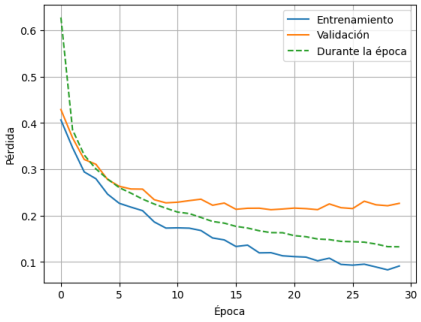
2. *Variación de hiperparámetros del Clasificador entrenado desde cero*

Una vez elegido el clasificador entrenado desde cero como el modelo óptimo, llevamos a cabo varios experimentos ajustando diferentes hiperparámetros para mejorar las métricas alcanzadas previamente. Los resultados obtenidos, junto con los hiperparámetros correspondientes, se presentan en la tabla IX.

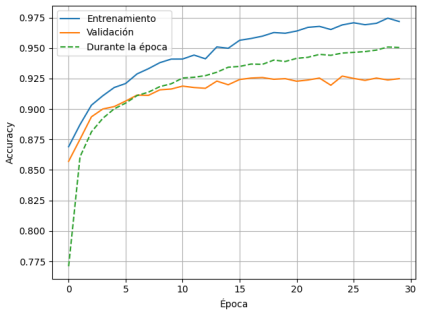
n_1	n_2	Optim.	η	Tam. de Lote	p	Acc. Máx. Val.	Error Mín. Val.
128	64	Adam	1^{-3}	128	0.2	91.99 %	0.2199
128	64	Adam	1^{-3}	**256**	0.2	92.53 %	0.2164
128	64	Adam	1^{-3}	**512**	0.2	91.92 %	0.2213
128	64	Adam	$\mathbf{1^{-4}}$	128	**0.4**	91.48 %	0.2411
128	**128**	**SGD**	1^{-3}	128	0.2	88.61 %	0.3334

Cuadro IX: Experimentos en donde se varían los hiperparámetros del clasificador sin preentrenar. En todos los casos, se usó un número de 30 épocas.

Los resultados indican que el modelo con los hiperparámetros de la segunda fila es el que alcanza el mejor desempeño, diferenciándose únicamente en el tamaño del lote. Las figuras 2a y 2b muestran las gráficas de la pérdida promedio por lote en cada época y el accuracy promedio por lote en cada época, respectivamente. La distancia entre las curvas y los valores de pérdida de cada conjunto reflejan indicios de sobreajuste en el modelo.



(a) Pérdida promedio por lote.



(b) Accuracy promedio por lote.

Figura 2: Métricas del Clasificador entrenado desde cero.

Finalmente, en la figura 3 se presenta la matriz de confusión[2] del modelo. Se observa que las confusiones ocurren principalmente entre prendas similares, como distintas prendas superiores (camisas, remeras, pulóveres) o entre tipos de calzado (zapatillas y botas).

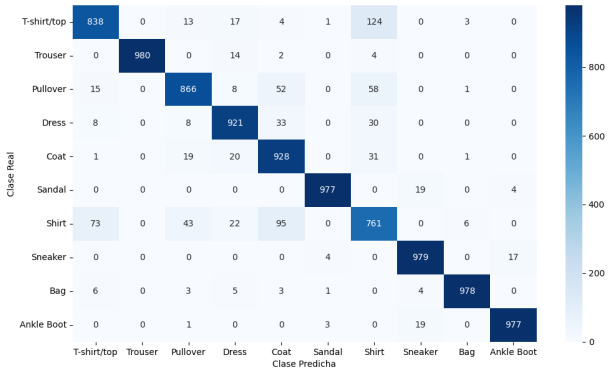


Figura 3: Matriz de confusión obtenida con el Clasificador óptimo.

IV. Discusión

En relación al Autoencoder, el análisis de la tabla IV muestra que, para las respectivas arquitecturas y con los hiperparámetros establecidos, las redes con dos capas convolucionales en el encoder obtienen un mejor desempeño en comparación con las una o tres capas. Asimismo, la eliminación de la capa lineal tiende a ofrecer una mejora de rendimiento. Esto sugiere que una complejidad moderada en el encoder permite una mejor extracción de características relevantes sin sobreajuste [1].

El ajuste de los hiperparámetros de entrenamiento tiene un impacto limitado (Tabla V), lo que sugiere que la arquitectura de la red es el factor dominante. Un tamaño intermedio del espacio latente (Tabla VI) logra un buen equilibrio entre compresión y reconstrucción [1]. Un espacio latente muy pequeño puede perder información, mientras que uno muy grande puede no generalizar bien [1]. Los autoencoders, al comprimir y reconstruir datos, aprenden representaciones útiles, y el espacio latente actúa como el punto de información más simplificada [1].

En cuanto al clasificador, y de forma contraintuitiva, el uso del encoder preentrenado resultó ser menos eficiente que entrenarlo desde cero. A pesar de que la lógica dictaría que un encoder preentrenado, al haber aprendido representaciones útiles de los datos, mejoraría el rendimiento del clasificador, se observó lo contrario. Como se puede observar en la figura 2, se presenta un sobreajuste, a pesar de que las métricas obtenidas en el conjunto de validación, mostradas en la tabla IX, son favorables.

V. Conclusiones

El presente trabajo exploró la sinergia entre Redes Neuronales Convolucionales (RNCs) [3] y Autoencoders [5], demostrando cómo esta combinación puede ser utilizada para construir un Clasificador Convolucional de imágenes. Se presentaron los elementos fundamentales de ambas arquitecturas, destacando sus aplicaciones principales y su relevancia en el campo del aprendizaje automático.

Utilizando el dataset Fashion-MNIST, se llevó a cabo una evaluación exhaustiva de diferentes arquitecturas de Autoencoders Convolucionales [3], con el objetivo de identificar la configuración óptima para la reconstrucción de imágenes. Se observó que un diseño cuidadoso de la arquitectura y una selección adecuada de los hiperparámetros son esenciales para lograr una reconstrucción precisa y eficiente.

Posteriormente, se diseñó un Clasificador Convolucional y se compararon dos estrategias de entrenamiento: el preentrenamiento de la parte convolucional de la red mediante un Autoencoder y el entrenamiento desde cero. Los resultados obtenidos revelaron que, en este caso particular, el preentrenamiento no condujo a una mejora significativa en el rendimiento del clasificador.

En resumen, los experimentos realizados permitieron extraer conclusiones valiosas sobre la importancia de la arquitectura y los hiperparámetros en el desempeño de los Autoencoders Convolucionales [1], y resaltando la necesidad de una cuidadosa consideración de la tarea objetivo al diseñar y entrenar clasificadores basados en representaciones aprendidas.

VI. Agradecimientos

Agradecemos a la cátedra de Redes Neuronales por el apoyo y la motivación constante a lo largo del semestre, también a FAMAF por brindar un espacio de aprendizaje de semejante calidad.

[1] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. Capítulo 9, Çonvolutional Networks".

[2] Géron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media.

[3] LeCun, Y., Bengio, Y., & Hinton, G. (2015). *Deep Learning*. Nature, 521(7553), 436-444.

[4] Stanford University. *CS231n: Convolutional Neural Networks for Visual Recognition*. Disponible en: `http://cs231n.stanford.edu/`.

[5] Chollet, F. (2017). *Deep Learning with Python*. Manning Publications.