

# Redes Neuronales 2024: Trabajo Práctico Final

Aldecoa Mariana\*

Facultad de Ciencias Económicas, Universidad Nacional de Córdoba, Ciudad Universitaria, 5000 Córdoba, Argentina

(Dated: Febrero 2025)

## ABSTRACT

En este trabajo se implementaron distintos modelos de autoencoders para la reconstrucción de imágenes del conjunto de datos Fashion-MNIST. Se probaron variaciones en los hiperparámetros, como el tamaño de la capa lineal, el dropout, el optimizador, el learning rate y el batch-size. A partir de estos experimentos, se seleccionó el autoencoder convolucional con 64 neuronas en la capa oculta, dropout de 0.1 y optimizador ADAM por su desempeño en la reducción del error cuadrático medio. Posteriormente, el encoder preentrenado se reutilizó en la construcción de un clasificador convolucional, evaluando su rendimiento en distintos escenarios de entrenamiento y preentrenamiento.

## I. INTRODUCCIÓN

El dataset Fashion-MNIST consiste en 70,000 imágenes de 28x28 píxeles distribuidas en 10 clases de prendas de vestir. Este trabajo utiliza autoencoders para comprimir y reconstruir las imágenes, seguido de la incorporación de un clasificador convolucional. La metodología permite evaluar el desempeño del encoder tanto en la tarea de reconstrucción como en la clasificación, analizando la influencia de hiperparámetros clave como el tamaño de capas lineales, el dropout, el optimizador, etc.

Se realizaron experimentos variando hiperparámetros como el número de neuronas en la capa lineal ( $n$ ), la tasa de dropout ( $p$ ), el optimizador (ADAM o SGD), el learning rate y el tamaño del batch. Además, se implementó un enfoque de *transfer learning* mediante el uso de un encoder preentrenado en la construcción de un clasificador.

En la sección II, se presenta una introducción teórica. En la sección III, se exponen los resultados obtenidos en las aplicaciones y, por último, en la sección IV, se ofrece una breve conclusión.

## II. TEORÍA

Un autoencoder es un tipo de red neuronal diseñada para aprender representaciones comprimidas de los datos de entrada. Su arquitectura consta de dos componentes principales: un encoder, que reduce la dimensionalidad de los datos capturando las características más relevantes de los mismos, y un decoder, que intenta reconstruir la entrada original.

Al restringir la cantidad de neuronas en la capa oculta, la red requiere identificar patrones esenciales en los datos. En los autoencoders convolucionales, las capas convolucionales 2D juegan un papel fundamental. Estas aplican filtros (kernel) que recorren la entrada, realizando una operación de convolución en la que los valores del filtro se multiplican con los de una parte específica de la ima-

gen, generando un único valor de salida. (Goodfellow et al., 2016)

Por otra parte, la función de activación ReLU es utilizada para introducir no linealidad en la red. Además, el *max - pooling* es empleado para reducir la dimension espacial de la imagen manteniendo sus características esenciales.

Para evitar sobreajuste, se implementa una capa dropout, que desactiva aleatoriamente un porcentaje de neuronas durante el entrenamiento. Este no se utiliza en la capa de salida ya que el objetivo es generar una reconstrucción precisa de los datos.

El clasificador convolucional, basado en la reutilización del encoder del autoencoder, añade una capa lineal de clasificación que permite asignar etiquetas a los datos de entrada. Este enfoque permite aprovechar las características extraídas por el encoder para mejorar la precisión en tareas de clasificación.

## III. RESULTADOS

### A. Autoencoder

La estructura de los autoencoders utilizados se encuentran en la figura 1.

El autoencoder comienza con un encoder que toma imágenes de entrada de tamaño  $1 \times 28 \times 28$  y las comprime en un espacio latente de dimensión  $n$ .

Primero, una capa convolucional con 16 filtros y un kernel de tamaño 3 reduce la dimensionalidad a  $16 \times 26 \times 26$ . Luego, una capa de *max pooling* de tamaño  $2 \times 2$  disminuye las dimensiones a  $16 \times 13 \times 13$ . Posteriormente, otra capa convolucional con 32 filtros y un kernel de tamaño 3 transforma la representación a  $32 \times 11 \times 11$ , seguida de una segunda capa de *max pooling* que reduce la salida a  $32 \times 5 \times 5$ . Finalmente, la salida es aplanada en un vector de 800 ( $32 \times 5 \times 5$ ) y pasada por una capa completamente conectada (*fully-connected*) que la reduce a un espacio de tamaño  $n$ .

A partir de este punto, el decoder comienza a reconstruir la imagen original aplicando operaciones inversas, utilizando capas transpuestas. Entre las capas hay dropout y se emplea la función de activación ReLU.

```

1 class Autoencoder_Convolucional(nn.Module):
2     def __init__(self, n=n, p=p):
3         super().__init__()
4         self.n = n
5         self.p = p
6
7         self.encoder = nn.Sequential(
8             #Capa Convencional 1
9             nn.Conv2d(in_channels=1, out_channels=16, kernel_size=3),
10            nn.ReLU(),
11            nn.Dropout(self.p),
12            nn.MaxPool2d(kernel_size=2, stride=2),
13            #Capa Convencional 2
14            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=3),
15            nn.ReLU(),
16            nn.Dropout(self.p),
17            nn.MaxPool2d(kernel_size=2, stride=2),
18            #Capa Lineal
19            nn.Flatten(),
20            nn.Linear(32 * 5 * 5, self.n),
21            nn.ReLU(),
22            nn.Dropout(self.p)
23        )
24
25        self.decoder = nn.Sequential(
26            #Capa Lineal
27            nn.Linear(in_features=self.n, out_features=32 * 5 * 5),
28            nn.ReLU(),
29            nn.Dropout(self.p),
30            nn.Unflatten(1, (32, 5, 5)),
31            #Capa Convencional 2 Traspuesta
32            nn.ConvTranspose2d(in_channels=32, out_channels=16, kernel_size=
33            =5, stride=2, padding=2, output_padding=0, dilation=2),
34            nn.ReLU(),
35            nn.Dropout(self.p),
36            #Capa Convencional 2 Traspuesta
37            nn.ConvTranspose2d(in_channels=16, out_channels=1, kernel_size
38            =6, stride=2, padding=1, output_padding=0, dilation=1),
39            nn.Sigmoid()
40        )
41
42        def forward(self, x):
43            x = self.encoder(x)
44            x = self.decoder(x)
45            return x

```

Figura 1: Arquitectura del Autoencoder Convolucional.

Implementamos autoencoders con un ciclo de entrenamiento (train loop) y evaluación (eval loop) para iterar sobre los lotes de datos durante cada época. La función de pérdida empleada fue el Error Cuadrático Medio (ECM) y el número de épocas óptimo fue determinado a través del método *early stopping*, que permite detener el entrenamiento cuando la validación deja de mejorar.

Para analizar el impacto de cada hiperparámetro en el rendimiento del modelo, se tomará como punto de partida el experimento 1, que actúa como caso base con  $n = 64$ ,  $p = 0.2$ , el optimizador ADAM, un learning rate de  $10^{-3}$  y un batch size de 100. A partir de esta configuración, en cada experimento posterior se modificará un único hiperparámetro a la vez, manteniendo los demás constantes.

La figura 2 corresponde al experimento 2, donde se implementan variaciones del tamaño de  $n$  del caso base, probando con valores de  $n = 32$  y  $n = 128$ . Los resultados mostraron que, a medida que  $n$  aumenta, la pérdida tiende a ser menor.

Posteriormente, se analizó el comportamiento de un autoencoder convolucional con variaciones en el dropout ( $p$ ). La figura 3 corresponde al experimento 3 y muestra

Exp.	$n$	$p$	Optimizador	Learning Rate	Batch-Size
1	64	0.2	ADAM	$10^{-3}$	100
2	32, 128	0.2	ADAM	$10^{-3}$	100
3	64	0.1, 0.5	ADAM	$10^{-3}$	100
4	64	0.2	SGD	$10^{-3}$	100
5	64	0.2	ADAM	$10^{-2}, 10^{-4}$	100
6	64	0.2	ADAM	$10^{-3}$	50, 150

Cuadro I: Hiperparámetros utilizados en los experimentos.

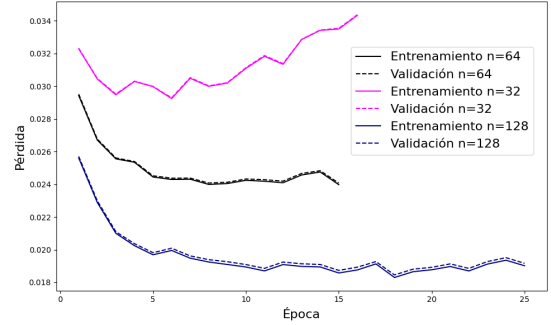


Figura 2: Función de pérdida del Autoencoder Convolucional a lo largo de épocas para diferentes tamaños de las capas lineales ( $n$ ). Se comparan tres configuraciones:  $n = 32$ ,  $n = 64$  y  $n = 128$ . El caso base corresponde a  $n = 64$ , y el experimento analiza el impacto de reducir y aumentar  $n$ .

que, con valores más altos de  $p$ , la función de pérdida tiende a ser mayor en cada época. En ninguno de los tres casos que se ven en el gráfico hay presencia de *over – shooting*.

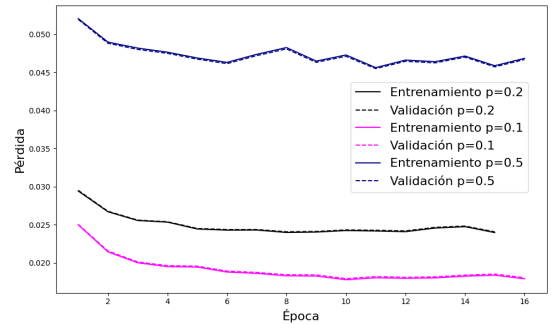


Figura 3: Función de pérdida del Autoencoder Convolucional a lo largo de épocas para diferentes valores de la tasa de dropout ( $p$ ). Se comparan tres configuraciones:  $p = 0.1$ ,  $p = 0.2$  y  $p = 0.5$ . El caso base corresponde a  $p = 0.2$ , y el experimento analiza el impacto de reducir y aumentar  $p$ .

Además, se comparó el impacto del optimizador en el entrenamiento del autoencoder convolucional, que representa el experimento 4. En la figura 4 se evidencia que ADAM presenta una convergencia más rápida, alcanzando una pérdida relativamente menor en solo 15 épocas, momento en el cual el entrenamiento y la validación se detienen debido al criterio de *early stopping*. En con-

traste, el optimizador SGD requiere un mayor número de iteraciones para reducir la pérdida. Esto justifica la elección de ADAM para los experimentos finales.

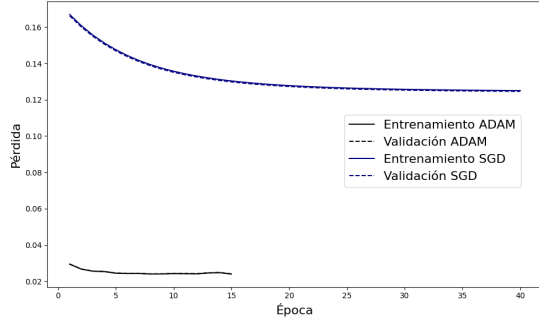


Figura 4: Función de pérdida del Autoencoder Convolutional a lo largo de épocas para diferentes optimizadores. Se comparan dos configuraciones: el caso base con el optimizador Adam y una variante utilizando SGD.

Se llevó a cabo también el experimento 5, donde se aplican variaciones en la tasa de aprendizaje. Se observa que cuando  $lr = 10^{-2}$  la función de pérdida oscila bastante y no parece disminuir a medida que avanzan las épocas. Los resultados se muestran en la figura 5.

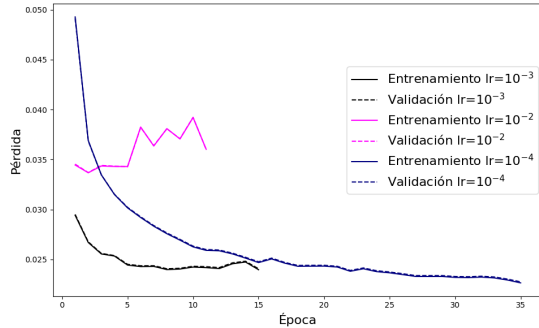


Figura 5: Función de pérdida del Autoencoder Convolutional a lo largo de épocas para diferentes valores de la tasa de aprendizaje (learning rate). Se comparan tres configuraciones:  $10^{-2}$ ,  $10^{-3}$  y  $10^{-4}$ . El caso base corresponde a  $10^{-3}$ , y el experimento analiza el impacto de utilizar una tasa de aprendizaje mayor y una menor.

En la figura 6 se observa la evolución de la función de pérdida para diferentes tamaños de batch (experimento 6). Con un batch size de 50 la pérdida tiende a aumentar en las últimas épocas, lo que sugiere un posible sobreajuste. En cambio, los tamaños de 100 y 150 muestran una reducción más estable, con curvas similares entre sí.

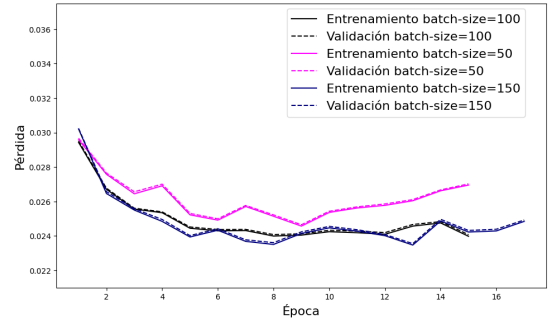


Figura 6: Función de pérdida del Autoencoder Convolutional a lo largo de épocas para diferentes tamaños de batch. Se comparan tres configuraciones: 50, 100 y 150. El caso base corresponde a un batch size de 100, y el experimento analiza el impacto de utilizar un tamaño menor y uno mayor.

## B. Clasificador

En base a los resultados obtenidos en los experimentos previos, se seleccionó el encoder del autoencoder con  $n = 64$  (si bien  $n = 128$  presentó valores menores de la función de pérdida en cada una de las épocas,  $n = 64$  mostró un buen desempeño probablemente con un menor esfuerzo computacional). Se optó por un dropout de  $p = 0,1$ , un batch size de 100, una tasa de aprendizaje de  $10^{-3}$  y el optimizador ADAM. A partir de este encoder preentrenado, se construyó el clasificador, cuyo desempeño será evaluado en los siguientes experimentos, utilizando *Cross Entropy Loss* y Precisión. Compararemos cuatro casos:

- **Caso 1:** Se utiliza el encoder preentrenado y se entrenan el clasificador y el encoder conjuntamente.
- **Caso 2:** Se utiliza el encoder preentrenado, y únicamente se entrenó el clasificador.
- **Caso 3:** Se utiliza el encoder sin preentrenamiento, y se entrenan el encoder y el clasificador conjuntamente.
- **Caso 4:** Se utiliza el encoder sin preentrenamiento, y únicamente se entrena el clasificador.

En la figura 7, se observa la evolución de la función de pérdida de entrenamiento y validación para cada uno de los cuatro casos. En todos los casos, la pérdida disminuye a medida que avanzan las épocas. El caso 4 comienza con una pérdida inicial considerablemente más alta y luego muestra una disminución, pero al finalizar el entrenamiento sigue presentando valores superiores a los otros casos. En contraste, los casos 1 y 3 logran una pérdida menor en las últimas épocas.

La figura 8 muestra la precisión de entrenamiento y validación a lo largo de las épocas. Se observa que todos los casos muestran una mejora progresiva en el desempeño del modelo a medida que avanzan las épocas. El caso 1 alcanza la mayor precisión tanto en entrenamiento como en

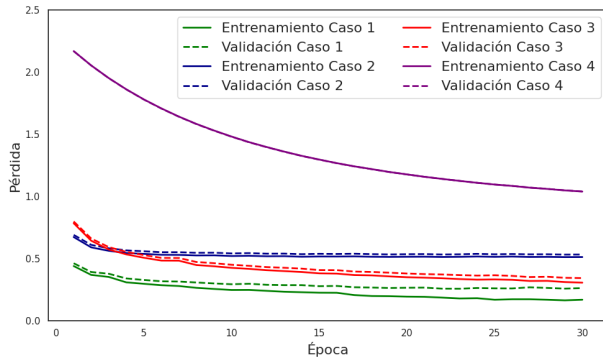


Figura 7: Función de pérdida del conjunto de entrenamiento y validación a lo largo de las épocas para los cuatro casos.

validación, superando el 90 %. El caso 3 también muestra un buen desempeño, con una precisión cercana al 88 %, mientras que el caso 2 mantiene valores estables alrededor del 80 %. Por otro lado, el caso 4, que presentaba la mayor pérdida, es el que alcanza la menor precisión, con valores inferiores al 75 %.

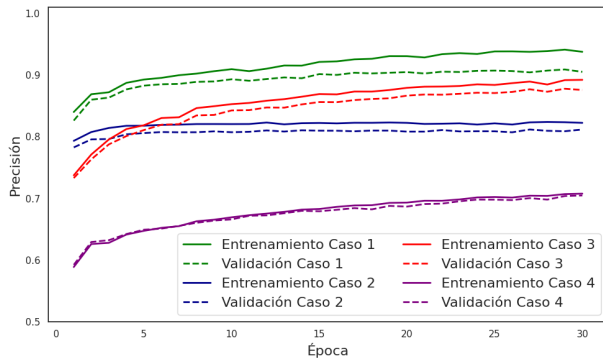


Figura 8: Precisión del conjunto de entrenamiento y validación a lo largo de las épocas para los cuatro casos.

Finalmente, en la figura 9 se presentan las matrices de confusión para cada caso, lo que permite evaluar el desempeño de los modelos en la clasificación. Estas matrices muestran la frecuencia con la que las etiquetas predichas coinciden con las etiquetas verdaderas.

Como era de esperar, el caso 1 tiene un mejor desempeño, mientras que el caso 4 es el que presenta mayor confusión en la clasificación. Sin embargo, el hecho de que el caso 4 aún sea capaz de clasificar correctamente algunas categorías sugiere que el modelo está captando ciertas estructuras en los datos, incluso sin contar con un encoder preentrenado ni entrenado.

Etiqueta Original \ Etiqueta Predicha	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
T-shirt/top	0.86	0.00	0.02	0.01	0.00	0.00	0.10	0.00	0.01	0.00
Trouser	0.00	0.98	0.00	0.01	0.00	0.00	0.01	0.00	0.00	0.00
Pullover	0.01	0.00	0.89	0.01	0.04	0.00	0.05	0.00	0.00	0.00
Dress	0.01	0.01	0.01	0.91	0.01	0.00	0.04	0.00	0.00	0.00
Coat	0.00	0.00	0.06	0.04	0.79	0.00	0.10	0.00	0.00	0.00
Sandal	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.01	0.00	0.00
Shirt	0.10	0.00	0.07	0.02	0.05	0.00	0.75	0.00	0.01	0.00
Sneaker	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.96	0.00	0.03
Bag	0.01	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.97	0.00
Ankle Boot	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.03	0.00	0.96

(a) Caso 1: encoder preentrenado, entrenamiento conjunto.

Etiqueta Original \ Etiqueta Predicha	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
T-shirt/top	0.78	0.00	0.02	0.06	0.00	0.00	0.11	0.00	0.02	0.00
Trouser	0.01	0.94	0.01	0.04	0.00	0.00	0.00	0.00	0.00	0.00
Pullover	0.01	0.00	0.71	0.01	0.14	0.00	0.13	0.00	0.01	0.00
Dress	0.03	0.01	0.01	0.85	0.03	0.00	0.07	0.00	0.00	0.00
Coat	0.00	0.00	0.18	0.04	0.66	0.00	0.10	0.00	0.01	0.00
Sandal	0.00	0.00	0.00	0.00	0.00	0.91	0.00	0.06	0.01	0.03
Shirt	0.18	0.00	0.16	0.04	0.12	0.00	0.47	0.00	0.03	0.00
Sneaker	0.00	0.00	0.00	0.00	0.00	0.03	0.00	0.92	0.00	0.06
Bag	0.00	0.00	0.01	0.01	0.00	0.01	0.03	0.00	0.94	0.00
Ankle Boot	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.05	0.00	0.94

(b) Caso 2: encoder preentrenado, entrenamiento solo del clasificador.

Etiqueta Original \ Etiqueta Predicha	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
T-shirt/top	0.83	0.00	0.04	0.03	0.01	0.00	0.08	0.00	0.01	0.00
Trouser	0.00	0.97	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00
Pullover	0.01	0.00	0.82	0.01	0.07	0.00	0.08	0.00	0.00	0.00
Dress	0.02	0.01	0.02	0.88	0.04	0.00	0.04	0.00	0.00	0.00
Coat	0.00	0.00	0.09	0.02	0.80	0.00	0.08	0.00	0.00	0.00
Sandal	0.00	0.00	0.00	0.00	0.00	0.96	0.00	0.02	0.00	0.01
Shirt	0.15	0.00	0.10	0.02	0.09	0.00	0.62	0.00	0.02	0.00
Sneaker	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.95	0.00	0.03
Bag	0.00	0.00	0.01	0.00	0.01	0.00	0.00	0.01	0.97	0.00
Ankle Boot	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.05	0.00	0.95

(c) Caso 3: encoder sin preentrenar, entrenamiento conjunto.

Etiqueta Original \ Etiqueta Predicha	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle Boot
T-shirt/top	0.74	0.00	0.03	0.10	0.03	0.02	0.06	0.00	0.01	0.00
Trouser	0.01	0.87	0.00	0.07	0.01	0.02	0.03	0.00	0.00	0.00
Pullover	0.03	0.01	0.41	0.02	0.20	0.03	0.27	0.00	0.03	0.00
Dress	0.05	0.02	0.02	0.77	0.03	0.05	0.06	0.00	0.01	0.00
Coat	0.01	0.00	0.12	0.08	0.61	0.01	0.15	0.00	0.01	0.00
Sandal	0.00	0.01	0.00	0.01	0.00	0.73	0.00	0.18	0.01	0.06
Shirt	0.21	0.00	0.14	0.07	0.18	0.05	0.31	0.00	0.03	0.00
Sneaker	0.00	0.00	0.00	0.00	0.00	0.07	0.00	0.83	0.00	0.10
Bag	0.01	0.00	0.01	0.03	0.02	0.03	0.05	0.02	0.83	0.00
Ankle Boot	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.04	0.00	0.94

(d) Caso 4: encoder sin preentrenar, entrenamiento solo del clasificador.

Figura 9: Matrices de confusión para los cuatro casos.

## IV. CONCLUSIÓN

En conclusión, los resultados obtenidos en este trabajo resaltan el potencial del *transfer learning* para mejorar el desempeño de modelos de clasificación. Al preentrenar el encoder mediante un autoencoder convolucional, se lograron extraer características relevantes de las imágenes, las cuales, al ser transferidas al clasificador, permitieron alcanzar una mayor precisión y una menor pérdida.

Es notable que, aunque los modelos sin preentrenamiento (como el caso 4) logran captar cierta información útil, la configuración basada en *transfer learning* (caso

1 y caso 2) muestra una capacidad superior para distinguir entre las categorías para la clasificación. Esto evidencia que aprovechar el conocimiento adquirido en tareas relacionadas, como en el autoencoder, puede ser de gran utilidad para desarrollar modelos de clasificación.

---

\* mariana-aldecoa@unc.edu.ar

## V. REFERENCIAS

Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning*. MIT Press.