# Trabajo Integrador - Autoencoder y Clasificador con PyTorch

Kevin Gaston Mansilla*

(Dated: December 1, 2024)

## I. INTRODUCCIÓN

Tomando como punto de partida la base de datos de Fashion MNIST [1], el objetivo de este trabajo es implementar un autoencoder convolucional usando PyTorch [2] que es básicamente un sistema de codificación y decodificación de imágenes.

Luego de entrenarlo, realizaremos experimentos variando los hiperparámetros de la red para ver cómo afectan al error de reconstrucción. Por otro lado, implementaremos y entrenaremos un clasificador convolucional reutilizando el encoder del autoencoder previamente definido.

Por último, probaremos como se comporta solo reentrenando los parámetros de la capa clasificadora, dejando los parámetros de la capa codificadora tal como vienen entrenados del autoencoder convolucional inicial.

## II. AUTOENCODER CONVOLUCIONAL

Se puede intuir que el propósito de un autoencoder es aprender una aproximación de la función identidad, por medio de la reducción de la dimensionalidad, es decir, comprimir la información de entrada en una representación de menor dimensión.

Como se muestra en la figura 1 consta de dos partes, un encoder que comprime los datos de entrada (Compressed Representation), con el objetivo de obtener las características principales de dichos datos y un decoder que reconstruye la entrada a partir de las características encontradas en el encoder.

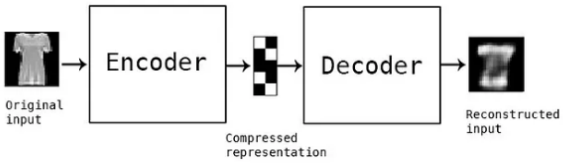


FIG. 1: Autoencoder convolucional

Entrenaremos con los siguientes hiperparámetros (modelo base):

- $n = 64$ (número de neuronas en la capa lineal).
- $lr = 0.001$.
- $p = 0.2$ (probabilidad de dropout).
- $epochs = 15$.
- $batch\_size = 100$.
- $loss = MSELoss$.
- $optimizer = Adam$.

La arquitectura del encoder es la siguiente:

- Una capa convolucional 2D compuesta por:
  - un capa $Conv2d$ de entrada de dimensiones $(1, 28, 28)$, de salida $(16, 26, 26)$ y kernel $(3, 3)$.
  - una capa de activación $ReLU$.
  - una capa $Dropout$.
  - una capa $MaxPool2d$, la cual mapea dimensiones $(16, 26, 26)$ a $(16, 13, 13)$ y kernel $(2, 2)$.
- Otra capa convolucional 2D compuesta por:
  - un capa $Conv2d$ de entrada de dimensiones $(16, 13, 13)$, de salida $(32, 11, 11)$ y kernel $(3, 3)$.
  - una capa de activación $ReLU$.
  - una capa $Dropout$.
  - una capa $MaxPool2d$, la cual mapea dimensiones $(32, 11, 11)$ a $(32, 5, 5)$ y kernel $(2, 2)$.
- Una capa lineal compuesta por:
  - una capa $Flatten$ que mapea dimensiones $(32, 5, 5)$ a $(32 * 5 * 5)$.
  - una capa $Linear$ de entrada $(32 * 5 * 5)$ y salida $n$.
  - una capa de activación $ReLU$.
  - una capa $Dropout$.

El decoder con las siguientes capas:

- Una capa lineal compuesta por:
  - una capa $Linear$ de entrada $n$ y salida $(32 * 5 * 5)$.
  - una capa de activación $ReLU$.
  - una capa $Dropout$.
  - una capa $Unflatten$ que mapea dimensiones $(32 * 5 * 5)$ a $(32, 5, 5)$.
- Una capa convolucional 2D compuesta por:
  - un capa $ConvTranspose2d$ que mapea dimensiones $(32, 5, 5)$ a $(16, 13, 13)$, kelnel_size $(4, 4)$, stride $(2, 2)$ y un output_padding de $(1, 1)$.
  - una capa de activación $ReLU$.
  - una capa $Dropout$.
- Otra capa convolucional 2D compuesta por:
  - un capa $ConvTranspose2d$ que mapea dimensione $(16, 13, 13)$ a $(1, 28, 28)$, kernel_size $(4, 4)$, stride $(2, 2)$ y un output_padding de $(1, 1)$.
  - una capa de activación $Sigmoid$.

No le agregamos capa de $Dropout$ en la capa de salida del decoder, ya que este genera la salida reconstruida, que idealmente debería ser lo más similar posible a los datos de entrada originales, si le agregáramos esta capa introduciría ruido que afectaría directamente la calidad de reconstrucción.

Luego variaremos los hiperparámetros de la red para ver cómo afecta su desempeño. Primero variamos la probabilidad de dropout a 0.1, luego cambiaremos el número de neuronas en la capa lineal a 256 con el mismo dropout y finalmente cambiaremos el optimizador por SGD.

## III. CLASIFICADOR CONVOLUCIONAL

Una vez entrenado el autoencoder, reutilizaremos el encoder para entrenar un clasificador convolucional. Que tendra la siguiente arquitectura:

- Una capa lineal compuesta por:
  - una capa *Linear* de entrada $n$ y salida 10.
  - una capa de activación *ReLU*.

El clasificador será entrenado de dos formas. Primero reentrenaremos todos los parámetros de la red y luego solo reentrenaremos la capa clasificadora, dejando los parámetros de la capa codificadora tal como vienen entrenados del autoencoder convolucional base, con el fin de comparar los resultados obtenidos.

## IV. RESULTADOS

### A. Modelo base sin entrenar

En la figura 2 se muestra la imagen a predecir y las imágenes reconstruidas por el modelo sin entrenar. Se puede observar que el modelo no es capaz de reconstruir la imagen original.

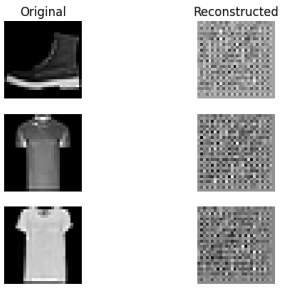


FIG. 2: Imágenes a predecir vs imágenes reconstruidas por el modelo sin entrenar

### B. Modelo base entrenado

Los resultados para el modelo base entrenado se presentan en la figura 3. Las curvas muestran una disminución inicial en las funciones de pérdida a media que avanzan las épocas, sin embargo, a partir de la época 5, las curvas experimentan un leve aumento seguidas de una leve oscilación, lo que podría ser un signo de que el modelo no va a tener un buen desempeño.

Por otro lado, en la figura 4 se aprecia que el modelo logra reconstruir la imagen original, pero con una calidad no muy buena, lo cual confirma lo observado en la función de pérdida.

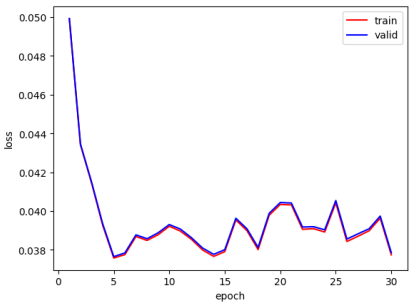


FIG. 3: Función de pérdida del modelo base entrenado

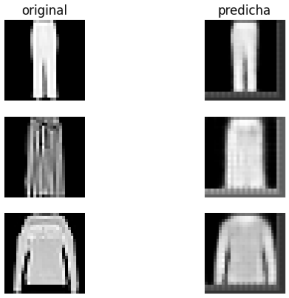


FIG. 4: Imágenes a predecir vs imágenes reconstruidas por el modelo base entrenado

### C. Modelo con probabilidad de dropout 0.1, $n = 64$

En la figura 5 se muestra la función de pérdida del modelo variando solo la probabilidad de dropout a 0.1. Identificamos un comportamiento similar al modelo base, pero con una oscilación no tan pronunciada, por lo que el cambio de este hiperparámetro no afecta significativamente el desempeño del modelo.

A su vez, en la figura 6 vemos que el modelo es capaz de captar los patrones generales de las imágenes, pero pierde información de alto nivel, como los bordes o los detalles más finos.

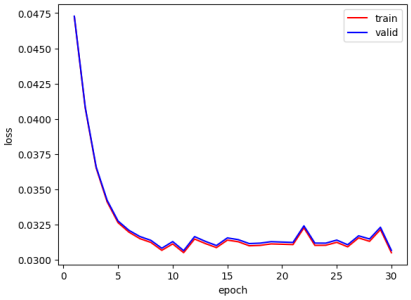


FIG. 5: Función de pérdida del modelo con probabilidad de dropout 0.1

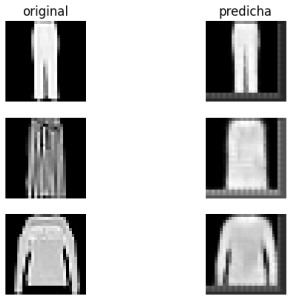


FIG. 6: Imágenes a predecir vs imágenes reconstruidas por el modelo con probabilidad de dropout 0.1

### D. Modelo con probabilidad de dropout 0.1 y $n = 256$

Ahora, con un dropout de 0.1 y $n = 256$ como vemos en el gráfico 7 las funciones de pérdida de ambos conjuntos de datos disminuyen constantemente, lo que indica que el autoencoder está logrando comprimir y reconstruir los datos de entrada de manera eficiente.

La proximidad entre las curvas es casi idéntica que en los casos anteriores, pero al ver el eje de loss vemos que los valores son mucho mejores, lo cual podría ser un indicio de que el modelo está aprendiendo correctamente.

En la figura 8 aunque no son perfectas, el modelo es capaz de capturar más detalles de las imágenes originales.

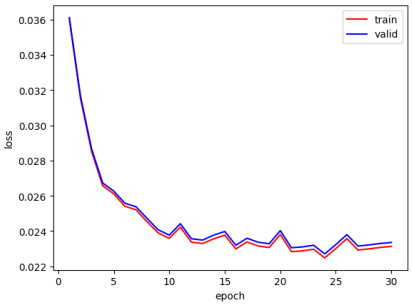


FIG. 7: Función de pérdida del modelo con probabilidad de dropout 0.1 y $n = 256$

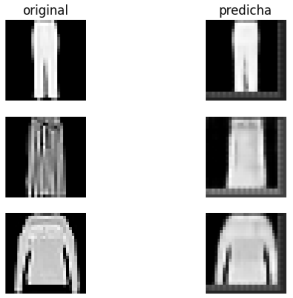


FIG. 8: Imágenea a predir vs imágenes reconstruidas por el modelo con probabilidad de dropout 0.1 y $n = 256$

### E. Modelo con optimizador SGD

Los resultados para el modelo con optimizador SGD se presentan en el gráfico 9 a simple vista vemos como las funciones de perdida de ambos conjuntos de datos disminuyen durante todas las épocas, esto podría indicar que el modelo está aprendiendo correctamente. Sin embargo, al observar el error de esas curvas (eje loss) vemos que los valores son mucho mayores que en los modelos anteriores, lo cual indica que el modelo no está aprendiendo correctamente.

Confirmando lo mencionado en el párrafo previo en 10. Se puede observar que el modelo no es capaz de reconstruir la imagen original.

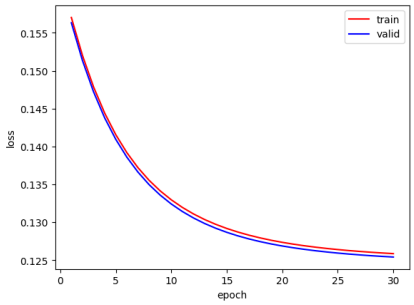


FIG. 9: Función de pérdida del modelo con optimizador SGD

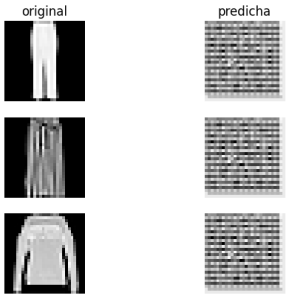


FIG. 10: Imagenea a predir vs imagenes reconstruidas por el modelo SDG

## V. CLASIFICADOR CONVOLUCIONAL

En esta sección se presentan los resultados obtenidos para el clasificador convolucional que reutiliza el encoder del autoencoder previamente entrenado y en este caso se reentrenan todos los parámetros de la red por completo.

En 11 se muestran las funciones de perdidas. Las curvas disminuyen tanto en el conjunto de entrenamiento como en el de validación a medida que avanzan las épocas con un error relativamente bajo, lo cual indica que el modelo mejora su capacidad de ajuste al avanzar las épocas.

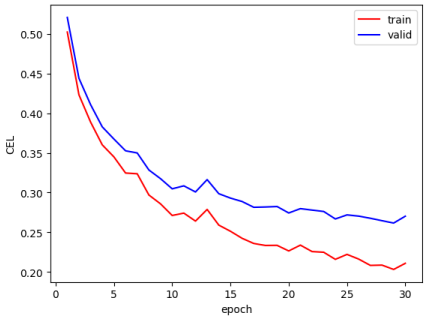


FIG. 11: Función de pérdida del clasificador

La precisión aumenta constantemente en ambos conjuntos de datos según se observa en 12, lo cual indica que el modelo es capaz de clasificar correctamente las imágenes de entrada.

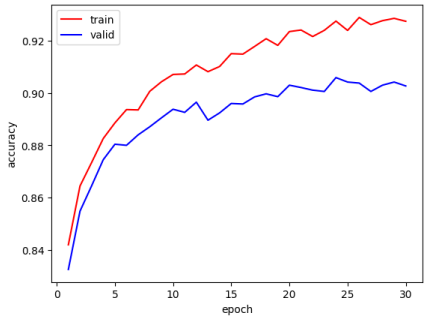


FIG. 12: Presición del clasificador

Y la matriz de confusión expuesta en 13, muestra un buen desempeño del clasificador. Ya que en la diagonal principal presenta valores altos basándonos en la escala de colores de la derecha de la figura, indicando que la mayoría de las predicciones son correctas.

Las clases que más destacan por su desempaño con trouser, sandal, sneaker, bag y ankle boot. Y tiene dificultades en distinguir entre shirt y coat, por ejemplo tomando la fila shirt se ve que solo acierta 713 veces y 142 veces se equivoca prediciendo t-shirt y 60 como pullover, algo similar ocurre con coat, esto se debe a la similitud entre las categorías, ya que todas son prendas de vestir superiores.

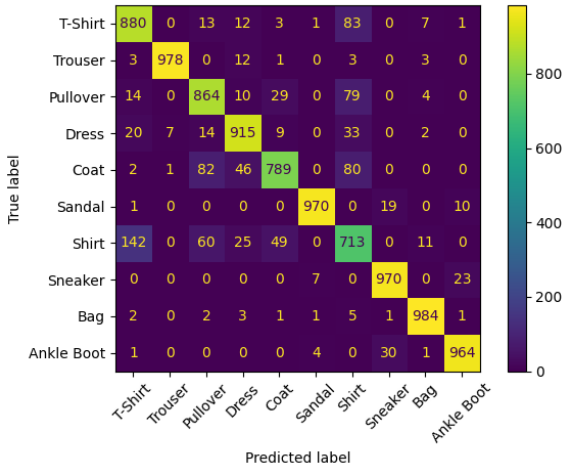


FIG. 13: Matriz de confusión del clasificador

### A. Clasificador solo entrenando la capa clasificadora

Ahora analizamos los resultados obtenidos para el clasificador convolucional reutilizando el encoder del autoencoder previamente entrenado en el modelo base y, en este caso, solo reentrenando los parámetros de la capa clasificadora. Es decir, que tomando como punto de partida el modelo base, solo se reentrenan los parámetros de la capa clasificadora y se dejan los parámetros de la capa codificadora tal como vienen entrenados del autoencoder convolucional.

En 14 se muestran las funciones de pérdidas de ambos conjuntos de datos y presentan una disminución bastante pronunciada en ambos casos. El comportamiento es similar al modelo anterior, pero con un error mucho mayor.

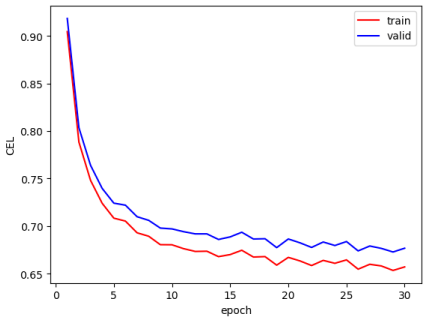


FIG. 14: Función de pérdida del clasificador - solo reentrenando la capa clasificadora

Por otro lado, en 15 el comportamiento de la precisión es diferente al modelo anterior, en este caso las curvas aumentan, pero luego se estabilizan y comienzan a oscilar.

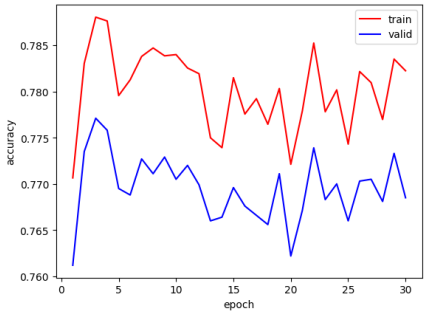


FIG. 15: Presición del clasificador - solo reentrenando la capa clasificadora

Todo esto indica que el modelo podría llegar a tener un desempeño peor que el modelo anterior. A fin de comprobarlo, se muestra en 16 la matriz de confusión donde efectivamente descubrimos que el desempeño es mucho peor. Ya que los valores de la diagonal principal son menores en todas las categorías, indicando un mayor número de predicciones incorrectas.

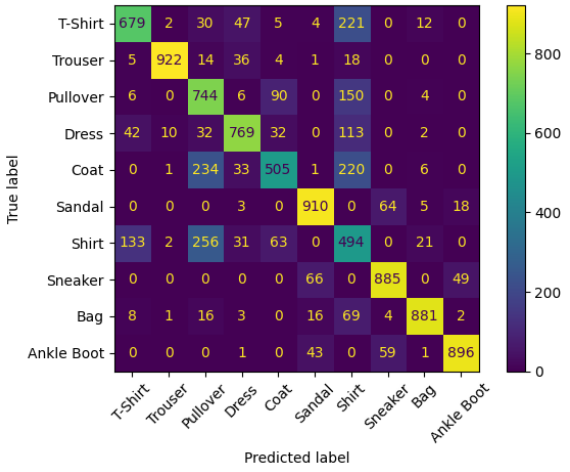


FIG. 16: matrix de confusión - solo reentrenando la capa clasificadora

Entonces concluimos que el modelo donde se reentrenan todos los parámetros de la tiene un mejor desempeño.

## VI. CONCLUSIONES

A lo largo de este trabajo se fueron entrenando diferentes modelos de redes neuronales convolucionales. Inicialmente, se desarrolló un autoencoder convolucional base sobre el cual variamos los hiperparámetros, posteriormente se reutilizó el encoder para entrenar un clasificador convolucional completo y, finalmente, se evaluaron los resultados obtenidos al reentrenar solo la capa clasificadora.

A medida que se incrementó la complejidad de los modelos, los resultados mejoraron. Sin embargo, se observó un aumento en la separación entre las curvas de entrenamiento y validación, lo que señala posibles signos de un desempeño menos robusto.

El modelo base mostró un desempeño limitado en la reconstrucción de imágenes, a pesar de registrar un error relativamente bajo. Al modificar la probabilidad de dropout a 0.1, no se evidenciaron cambios significativos en su rendimiento. En contraste, al incrementar el número de neuronas en la capa lineal a 256 con un dropout de 0.1, se observó una mejora en la calidad de las imágenes reconstruidas. Y el modelo entrenado con el optimizador SGD presentó un desempeño deficiente, con un error de reconstrucción notablemente alto.

Por otro lado, el clasificador convolucional donde se reentrenaron todos los parámetros de la red obtuvo buenos resultados, como se refleja en la matriz de confusión, donde la mayoría de las predicciones fueron correctas. Sin embargo, el clasificador que únicamente reentrenó la capa clasificadora mostró un rendimiento inferior al modelo anterior.

Es importante destacar que encontrar una combinación óptima de hiperparámetros representa un desafío considerable. Este proceso requiere una combinación de tiempo, recursos y experiencia para identificar los valores que maximicen el desempeño del modelo.

* kevin.mansilla@mi.unc.edu.ar

[1] Https://github.com/zalandoresearch/fashion-mnist.
[2] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in *Advances in Neural Information Processing Systems 32* (Curran Associates, Inc., 2019) pp. 8024–8035.