

Redes Neuronales: Autoencoder y clasificador convolucional sobre Fashion-MNIST

Tomas Martinez*

(Dated: December 2024)

El aprendizaje automático, y en particular el Deep Learning, ha impulsado avances en la visión artificial. Las Redes Neuronales Convolucionales (CNN) se destacan en tareas como la clasificación de imágenes. Este trabajo se centra en un tipo específico de CNN, el autoencoder convolucional, aplicado a la clasificación de imágenes de prendas de vestir del dataset Fashion-MNIST. Se explora el proceso de entrenamiento de un autoencoder para aprender representaciones latentes y luego se utiliza esa información para entrenar un clasificador.

I. INTRODUCCIÓN

En este trabajo, exploraremos el uso de autoencoders convolucionales para la clasificación de imágenes de prendas de vestir del conjunto de datos Fashion-MNIST. Primero, entrenaremos un autoencoder para que aprenda una representación latente de las imágenes. Luego, utilizaremos esta representación para entrenar un clasificador que pueda predecir la clase de una imagen dada. Este enfoque nos permite aprovechar la capacidad de los autoencoders para extraer características relevantes de las imágenes y mejorar el rendimiento del clasificador. A lo largo del trabajo, se detallarán los pasos de implementación, entrenamiento y evaluación del modelo, junto con una visualización de los resultados obtenidos. [1]

II. TEORÍA

Un autoencoder convolucional es una red neuronal artificial que utiliza capas convolucionales tanto en el codificador como en el decodificador. El codificador reduce la dimensionalidad de la entrada, generando una representación latente, mientras que el decodificador reconstruye la entrada original.

Los autoencoders pueden ser utilizados para la clasificación de imágenes de la siguiente manera. Se entrena un autoencoder convolucional con un conjunto de datos de imágenes sin etiquetar. Esto permite al autoencoder aprender características relevantes de las imágenes, de esto se encarga la primera capa convolucional y seguidamente una segunda capa convolucional traspuesta que se va a encargar de reconstruir estas imágenes en el decoder. Entre estas se incluye una capa lineal que tiene una representación más comprimida de la entrada.

Por último se entrena un clasificador utilizando las características extraídas por el autoencoder y las etiquetas correspondientes. El entrenamiento del autoencoder se realiza creando un conjunto de datos en el que cada elemento tiene la misma entrada y salida. Esto permite que el modelo aprenda una función que se asemeja a la identidad. Para evaluar el rendimiento de la red neuronal, se emplearán diferentes métricas. Para la reconstrucción en el autoencoder, se utiliza el **Error Cuadrático Medio (ECM)**. Para medir el rendimiento del clasificador, se aplica la **Cross Entropy Loss (CEL)**. Por último, se emplea una **Matriz de Confusión** para analizar el desempeño del clasificador. [2]

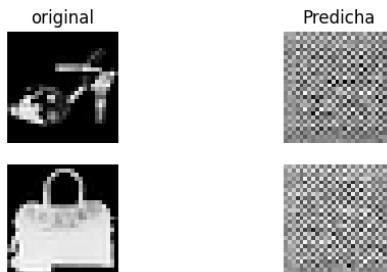


FIG. 1. Comienzo del proyecto, autoencoder no entrenado

III. RESULTADOS

A. Comienzo

Al comienzo de mi trabajo defino mis **hiperparametros** y sus valores iniciales. **Tasa de aprendizaje(lr)=** 0,001, **Dropout(p)=** 0,2, **Tamaño del batch=** 100, **Épocas=** 30, y el **Optimizador=** ADAM. Junto a esto la **aquitectura** de mi trabajo donde forma con dos capas conv, una linear y por ultimo dos transp.

Una vez creado mi autoencoder convolucional lo primero que va a suceder si queremos ver el resultado es, obviamente, totalmente errado(Ver fig. 1). Entonces primero comenzamos con un entrenamiento inicial donde usaremos el valor fijo de mis hiperparametros, donde los primero resultados van a ser los reflejados en la siguiente imagen(Ver fig. 2).



FIG. 2. Resultado de entrenamiento inicial. a) Función de pérdida por Error Cuadrático Medio aprendizaje(lr)= 0,001, Dropout(p)= 0,2, Tamaño del batch= 100, Épocas= 30, y el Optimizador= ADAM. Arquitectura: 2 conv., 1 lineal, 2 transp. Y b) imagen que el modelo Autoencoder ha generado después de procesar la imagen original.

B. Comparando la tasa de aprendizaje

En este punto se evaluó la perdida del autoencoder variando en distintos valores la tasa de aprendizaje, estos valores son 0.01, 0.05 y 0.001.(Ver fig. 3)

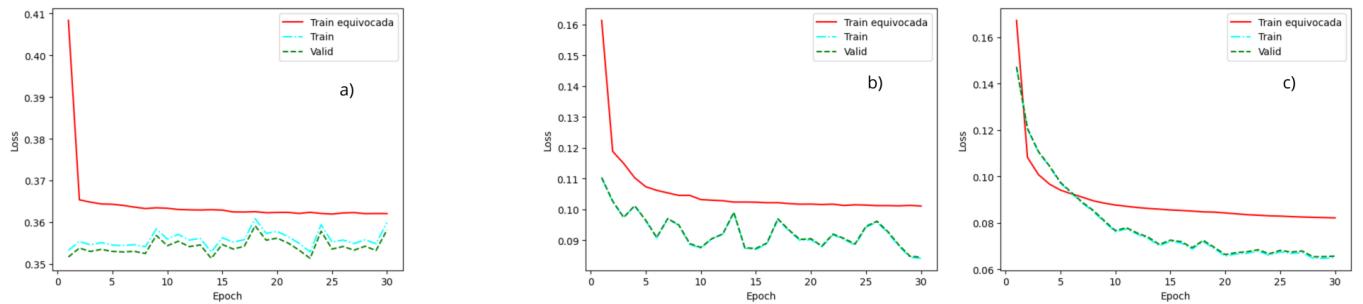


FIG. 3. Función de pérdida por Error Cuadrático Medio. Dropout(p)= 0,2, Tamaño del batch= 100, Épocas= 30, y el Optimizador= ADAM. Arquitectura: 2 conv., 1 lineal, 2 transp. Cuya tasa de aprendizaje es a) 0.01, b) 0.005 y c) 0.001

C. Comparación entre el optimizador ADAM y el SGD

Ahora se realiza una evaluación de los resultados pero esta vez cambiado de optimizador, SGD.(Ver fig. 4)

El análisis que podemos hacer es que con el optimizador ADAM (Fig. 3) se puede notar que alcanza la convergencia de manera significativamente más rápida en comparación con SGD. Además, en ADAM, una tasa de aprendizaje más reducida favorece el rendimiento, mientras que en el caso de SGD, un valor más alto de learning rate facilita la

convergencia. En ambas configuraciones, al igual que en la fase inicial de entrenamiento, la pérdida en el conjunto de validación se mantiene similar a la del conjunto de entrenamiento, lo que indica la ausencia de sobre ajuste.

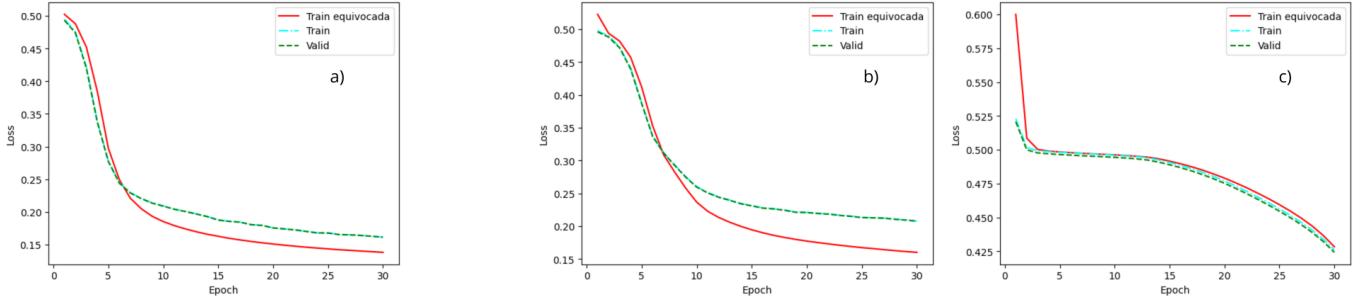


FIG. 4. Función de pérdida por Error Cuadrático Medio. Dropout(p)= 0,2, Tamaño del batch= 100, Épocas= 30, y el Optimizador= SGD. Arquitectura: 2 conv., 1 lineal, 2 transp. Cuya tasa de aprendizaje es a)0.01, b)0.005 y c)0.001

D. Alterando el Dropout

Ahora sabiendo que con el optimizador ADAM tenemos mejores resultados, decidido realizar estos experimentos con este optimizador para ver lo diferentes resultados con el dropout con valores de a) 0,35, b) 0,50 y c) 0,2.(Ver fig. 5)

Podemos observar que en cada uno de los casos, el resultado es bastante similar, sin embargo podemos notar que a medida que van pasando las épocas el autoencoder da mejores resultados con el dropout inicial c) y con el dropout a), pues la perdida es menor y está aprendiendo correctamente. Mientras que con el b) tiene picos mas altos de perdida sin embargo sigue aprendiendo bien.

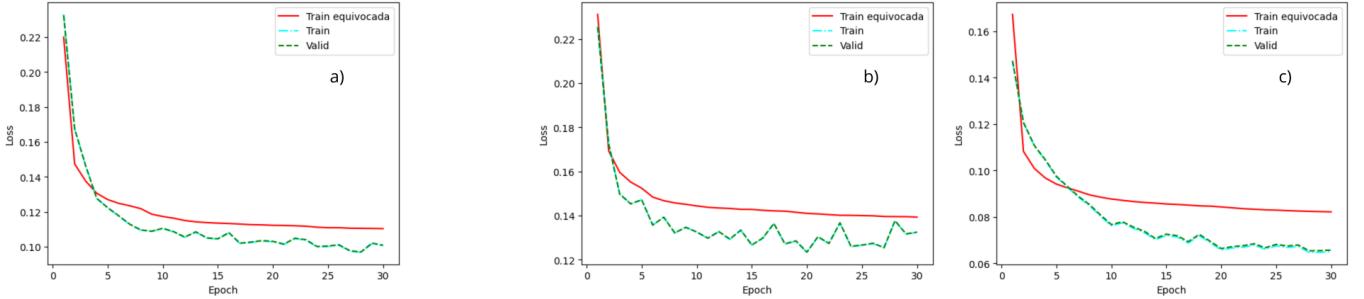


FIG. 5. Función de pérdida por Error Cuadrático Medio. Tasa de aprendizaje(lr)= 0.001, Tamaño del batch= 100, Épocas= 30, y el Optimizador= SGD. Arquitectura: 2 conv., 1 lineal, 2 transp. Cuyo Dropout(p) es a)0.35, b)0.5 y c)0.2

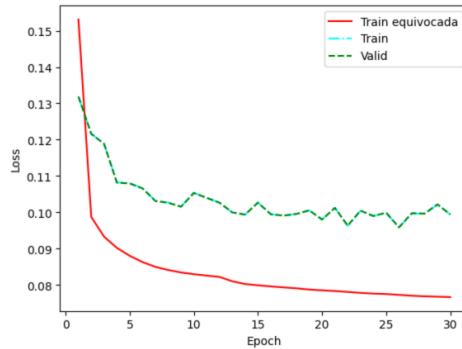


FIG. 6. Función de pérdida por Error Cuadrático Medio aprendizaje(lr)= 0,001, Dropout(p)= 0,2, Tamaño del batch= 100, Épocas= 30, y el Optimizador= ADAM. Arquitectura: 2 conv., 2 transp.

E. Retirando la capa lineal

Ahora analizamos la perdida si se elimina la capa lineal intermedia.(Ver fig. 6)

El autoencoder no logra aprender mucho, pues podemos ver que esta por encima de la linea roja, y esta perdida se mantiene estable.

F. Definición y entrenamiento de un clasificador

En este punto del trabajo definimos un clasificador convolucional y evaluamos los resultados mediante la función de perdida Cross Entropy Loss (CEL) y viendo la precisión de la clasificación de los datos.

Entonces primero vemos los resultados a la hora de re entrenar el encoder y entrenar la capa clasificadora. (Ver fig. 7 y 8)

Luego podemos ver los resultados solamente entrenando la capa clasificador y no re entrenar el encoder. Se observa que el rendimiento se reduce cuando únicamente se entrena la capa clasificadora, pero como hay un sobreajuste en el caso que reentrenamos todo.(Ver figs. 9 y 10)

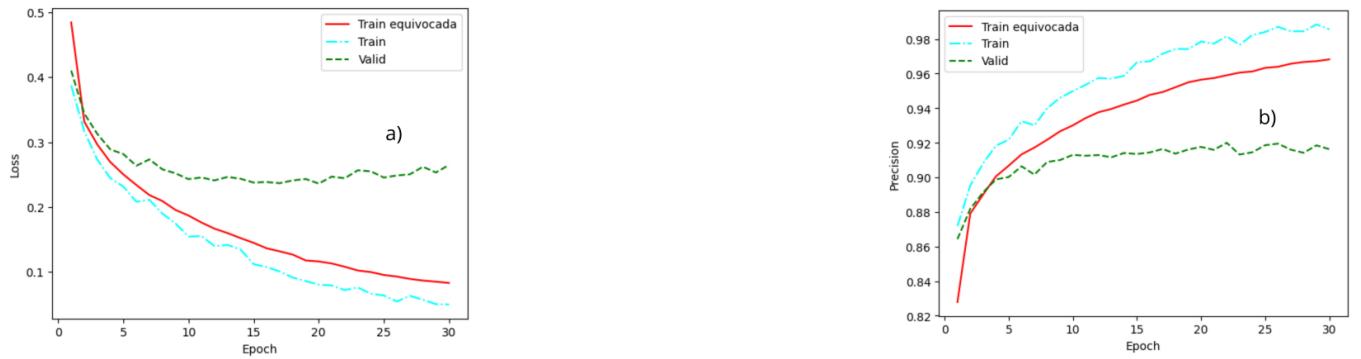


FIG. 7. Encoder re entrenado y entrenando el clasificador. tasa de aprendizaje(lr)= 0,001, Dropout(p)= 0,2, Tamaño del batch= 100 Epochs= 30, y el Optimizador= ADAM. a)función de perdida Cross Entropy Loss (CEL) y b)Precisión



FIG. 8. Matriz de confusión del encoder re entrenado y clasificador entrenado

IV. DISCUSIÓN

A través de los resultados mostrados y analizados podemos ver que a la hora de realizar los ajustes en el autoencoder nos brindan todos un impacto similar, pues al alterar los hiperparametros no podemos ver un resultado imponente en algunos, como por ejemplo en el dropout donde lo obtenido en los tres casos nos brinda casi lo mismo.

En el clasificador estudiamos dos casos donde a la hora de re entrenar el encoder quien ya tenia representaciones muy validas las cuales ayudan para mejorar el clasificador. Donde se puede ver reflejado la diferencia viendo las matrices de confusión (Figs. 8 y 10) cuando solamente entrenamos el clasificador, tiene menor precisión.



FIG. 9. entrenando solamente el clasificador. tasa de aprendizaje(lr)= 0,001, Dropout(p)= 0,2, Tamaño del batch= 100 Epochas= 30, y el Optimizador= ADAM. a)función de perdida Cross Entropy Loss (CEL) y b)Precisión



FIG. 10. Matriz de confusión de solamente el clasificador entrenado

V. CONCLUSIÓN

Lo que podemos sacar de este trabajo es que a la hora de experimentar con el autoencoder, no se va a ver afectado su rendimiento a grandes rasgos modificando sus hiperparametros. En cuanto al clasificador podemos determinar que es mucho mas rentable y muestra mejor desempeño general, re entrenar el encoder. Sin embargo aun perdemos un poco de desempeño entrenar solo la capa clasificadora también brinda buenos resultados y en un menor costo de computación.

Entonces el factor del re entrenamiento brinda una mejor experiencia y mayor eficacia a la hora de un realizar un trabajo de aprendizaje profundo

VI. AGRADECIMIENTOS

Expreso mi agradecimiento a los docentes de la cátedra y a FaMAF por brindarme no solo herramientas para realizar este trabajo, si no que también la posibilidad de cursar esta materia.

* tomasmartinezog@gmail.com

- [1] github resource, redes neuronales (2024), accessed: 2025-11-02.
- [2] Wikipedia contributors, Autoencoder (2024), accessed: 2025-11-02.