TRABAJO FINAL INTEGRADOR

Paula Pacheco*
Facultad de Matemática, Astronomía, Física y Computación,
Universidad Nacional de Córdoba, Ciudad Universitaria, 5000 Córdoba, Argentina
(Dated: Febrero 2023)

En este trabajo se implementará una red neuronal autoencoder y clasificador convolucional, entrenada con la base de datos Fashion-MNIST, y se estudiará la forma de mejorar su rendimiento.

I. INTRODUCCIÓN

Este trabajo tuvo como objetivo implementar redes neuronales con distintas arquitecturas para aprender a clasificar las imágenes de la base de datos Fashion-MNIST y analizar las ventajas y desventajas de cada uno de los modelos. La base utilizada consistió en un conjunto de imágenes de 28x28 píxeles en escala de grises de prendas y calzados clasificados en 10 categorías, con 60000 imágenes en el conjunto de entrenamiento y 10000 en el de test.

La primer red implementada es una red neuronal autoencoder convolucional, entrenada para aprender la función identidad usando la base de datos antes mencionada.
El *encoder* cuenta con las siguientes capas:

- una capa convolucional 2D compuesta por:
  - una capa `Conv2d` de entrada de dimensiones $(1, 28, 28)$ y de salida de dimensiones $(16, 26, 26)$;
  - una capa `ReLU`;
  - una capa `Dropout`;
  - una capa `MaxPool`, la cual mapea dimensiones $(16, 26, 26)$ a dimensiones $(16, 13, 13)$.
- otra capa convolucional 2D compuesta por:
  - una capa `Conv2d` de entrada de dimensiones $(16, 13, 13)$ y de salida de dimensiones $(32, 11, 11)$;
  - una capa `ReLU`;
  - una capa `Dropout`;
  - una capa `MaxPool`, la cual mapea dimensiones $(32, 11, 11)$ a dimensiones $(32, 5, 5)$.
- una capa lineal compuesta por:
  - una capa `Flatten` que mapea dimensiones $(32, 5, 5)$ a dimensión $(32 * 5 * 5)$;
  - una capa `Linear` que mapea dimensión $(32 * 5 * 5)$ a dimensión n;
  - una capa `ReLU`;
  - una capa `Dropout`.

El *decoder* cuenta con las siguientes capas:

- una capa lineal compuesta por:
  - una capa `Linear` que mapea dimensión n a dimensión $(32 * 5 * 5)$;
  - una capa `ReLU`;
  - una capa `Dropout`;
  - una capa `Unflatten` que mapea dimensión $(32 * 5 * 5)$ a dimensiones $(32, 5, 5)$.
- una capa convolucional 2D transpuesta (de la segunda convolucional) compuesta por:
  - una capa `ConvTranspose2d` que mapea dimensiones $(32, 5, 5)$ a dimensiones $(16, 13, 13)$;
  - una capa `ReLU`;
  - una capa `Dropout`.
- otra capa convolucional 2D transpuesta (de la primera convolucional)compuesta por:
  - una capa `ConvTranspose2d` que mapea dimensiones $(16, 13, 13)$ a dimensiones $(1, 28, 28)$;
  - una capa `Sigmoid`;
  - una capa `Dropout`.

Modificaremos los hiperparámetros y compararemos los resultados obtenidos. Para esto, se graficará, en función de las épocas, la función de pérdida *(ECM)* del entrenamiento y de la validación. Además, se mostrarán algunas de las imágenes a predecir vs las imágenes predichas por el modelo.

Por último, se implementará un clasificador convolucional reutilizando el encoder del autoencoder convolucional entrenado anteriormente. El mismo constará de las siguientes capas:

- el *encoder* del *autoencoder* entrenado anteriormente, que mapea dimensiones $(1, 28, 28)$ a dimensión n;
- una capa lineal de clasificación compuesta:
  - una capa `Linear` que mapea dimensiones n a dimensión 10;

– una capa `ReLU`;

– una capa `Dropout`.

De la misma manera que antes, crearemos distintas instancias modificando los hiperparámetros. Graficaremos en función de las épocas la función de pérdida *(CEL)* y la *precisión* de entrenamiento y validación. Además, se mostrará la matriz de confusión de cada modelo sobre el conjunto de validación.

II. RESULTADOS

A. Red neuronal autoencoder convolucional

En el primer modelo entrenamos una *red autoencoder convolucional* con parámetros por default: $n = 64$, dropout $p = 0.2$, optimizador *Adam* con learning rate $\eta = 0.001$.

A continuación se muestra, a modo de comparación, las imágenes a predecir vs las imágenes predichas por el modelo sin entrenar con los parámetros antes mencionados.

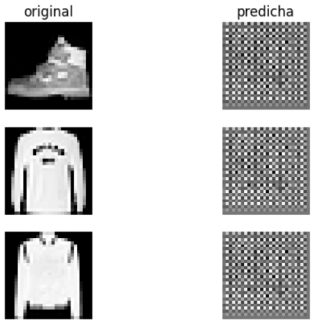
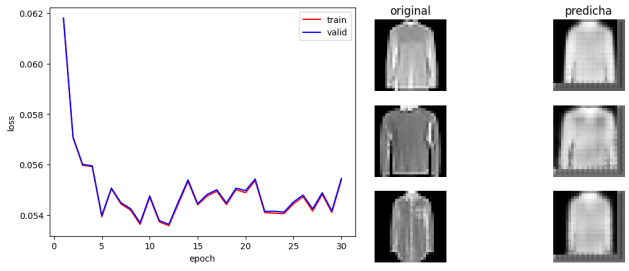


FIG. 1: Imágenes a predecir vs imágenes predichas por el modelo sin entrenar.

Los resultados obtenidos con el modelo entrenado se muestran en la siguiente figura.



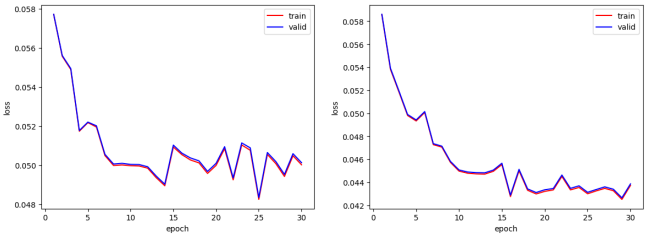
(a) Función de pérdida. (b) Comparación de imágenes

FIG. 2: Resultados para valores por default.

El error cuadrático medio decae hasta la época 5, y se establece en aproximadamente 0.052.

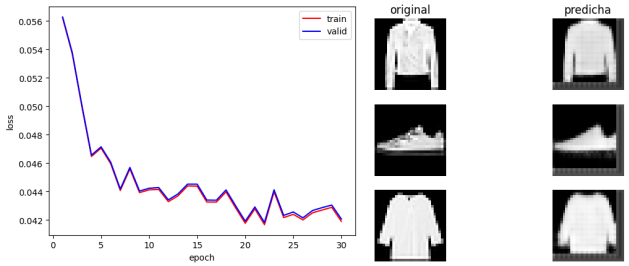
1. *Modificaciones el modelo.*

La primer variación al modelo fue modificar la cantidad de neuronas en la capa oculta, probando con $128, 256$ y 512. Se muestra a continuación la función de pérdida para estas modificaciones.



(a) $n = 128$. (b) $n = 256$.

FIG. 3: Función de pérdida variando n.



(a) Función de pérdida $n = 512$. (b) Comparación de imágenes.

FIG. 4: Resultados para $n = 512$

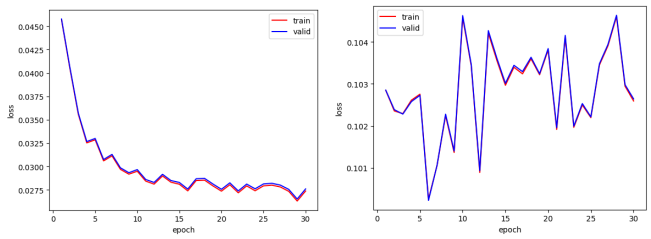
.

Claramente el modelo que obtuvimos con menor error fue el más complejo, con 512 neuronas. No obstante, no presenta mejoras significativas respecto al modelo con 256. Con este último, se obtiene una mejoría notable en la función de pérdida respecto al modelo con 64 neuronas. De igual forma, la elección el modelo con $n = 64$ también resulta ser una buena opción, considerando que su tiempo de entrenamiento es menor.

Para las próximas variaciones del modelo se usará $n = 256$.

En la siguiente modificación, cambiamos el valor del *dropout*. Probamos con $p = 0.1$ y $p = 0.5$. Con el segundo valor, obtuvimos que el error aumentaba a medida que aumentamos las épocas, por lo que no se trata de una buena alternativa. En cambio, en el primer caso, se obtuvo una mejoría en la función de pérdida, disminuyendo su valor promedio a la mitad. Esto es razonable debido a

que se disminuye la cantidad de neuronas que se omiten, obteniendo una red más completa. Si bien esta técnica se utiliza para evitar un sobreajuste de los datos, podemos ver en el gráfico que no hay un *overfiting* considerable.

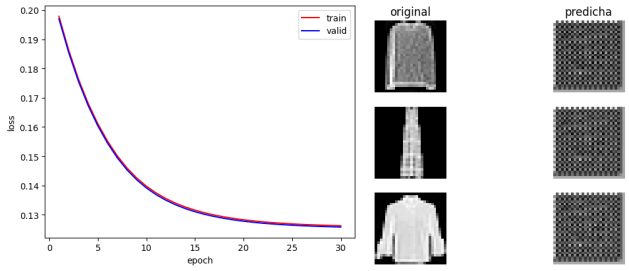


(a) $p = 0.1$ (b) $p = 0.5$

FIG. 5: Función de pérdida utilizando $n = 256$, variando el valor del dropout.

Respecto a otras posibles modificaciones, no logramos mejorar el rendimiento del modelo en comparación a los resultados obtenidos anteriormente. Modificamos el optimizador, el learning rate y el número de ejemplos en cada batch.

A modo de ejemplo, se muestran a continuación los resultados obtenidos utilizando el optimizador *SGD*. En este caso, el error resulta alto, y el modelo no reconstruye las imágenes. Por lo tanto, se continúa usando *Adam*.



(a) Función de pérdida. (b) Comparación de imágenes

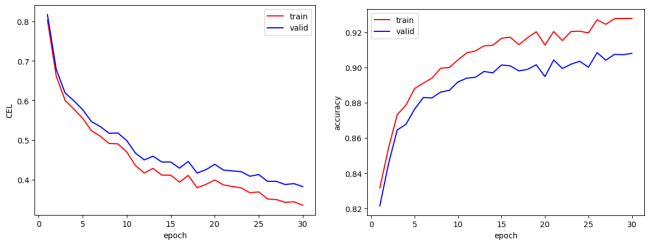
FIG. 6: Modelo con $n = 256$, $p = 0.1$ y optimizador *SGD*.

B. Clasificador convolucional

Se probaron en un inicio los clasificadores en los cuales se utilizaron dos de los encoders entrenados anteriormente:

- modelo con parámetros por default ($n = 64$, $p = 0.2$, optimizador *Adam*, $\eta = 0.001$).
- modelo con modificaciones: $n = 256$ y $p = 0.1$.

Se muestran en Fig. 7 y Fig. 8 los gráficos de función de pérdida, precisión y la matriz de confusión de ambos clasificadores.



(a) Función de pérdida. (b) Precisión.

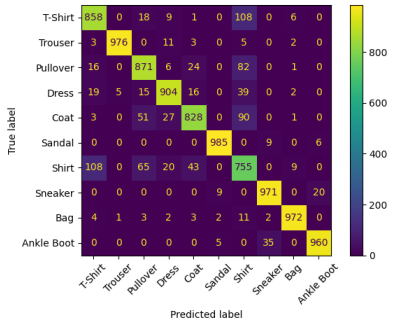
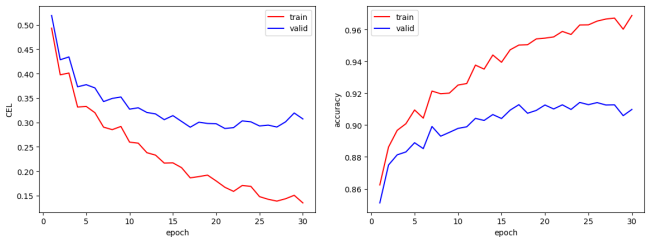


FIG. 7: Clasificador que utiliza el encoder del modelo con valores por default.



(a) Función de pérdida. (b) Precisión.

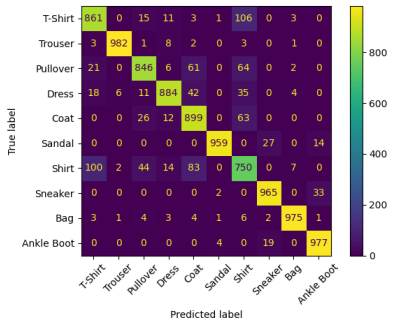


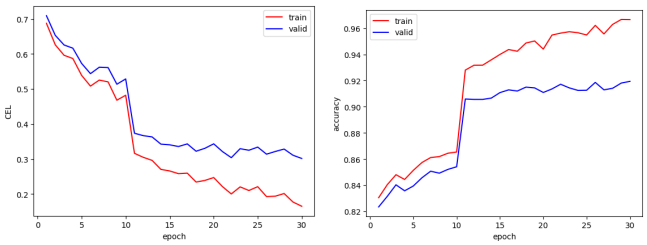
FIG. 8: Clasificador que utiliza el encoder del modelo con $n = 256$ y $p = 0.1$.

Podemos ver que en las matrices de confusión de ambos modelos se obtienen valores similares, y en los dos obtuvimos el valor de *accuracy* de 0.91. Notamos como

disminuyen en general los valores de la función de pérdida en el modelo modificado respecto al modelo original. A su vez, se ve un notable overfiting. Respecto a la precisión, mejora significativamente en el conjunto de entrenamiento pero no en el de validación.

Por lo tanto, para mejorar el modelo debemos reducir el *overfiting*. Para esto, aumentamos el valor del *dropout* a, nuevamente, $p = 0.2$. En la figura Fig. 9 se presentan los resultados.

En efecto, se logró reducir el overfiting a cambio de un aumento en los valores de la función de pérdida en el conjuntro train. Se obtiene una *precisión* de 0.92.



(a) Función de pérdida. (b) Precisión.

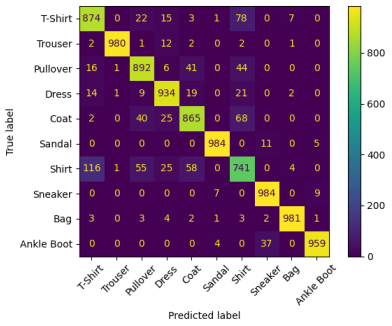
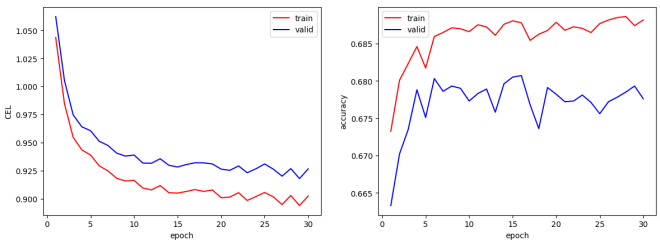


FIG. 9: Clasificador utilizando $n = 256$ y $p = 0.2$

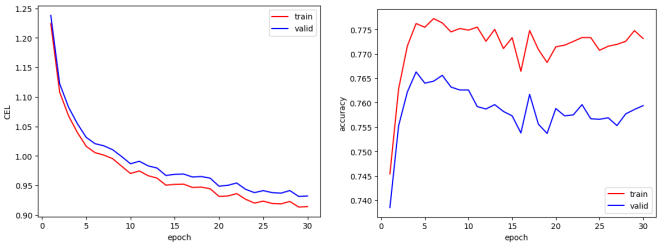
Analizando las matrices de confusión, podemos ver que la categoría con más dificultad para clasificar es *'Shirt'* en todos los modelos, confundiéndola con *'T-Shirt'*, *'Pullover'* y *'Coat'*.



(a) Función de pérdida. (b) Precisión.

FIG. 10: Modelo entrenando solo la capa clasificadora con valores por default.

Finalmente, se probó crear el clasificador dejando los parámetros del encoder ya entrenado sin volver a entrenar, y entrenando sólo la capa clasificadora. Se obtuvieron malos resultados comparados a los obtenidos anteriormente, tanto en el error como en la precisión. En las figuras Fig. 10 y Fig. 11 se muestran los resultados.



(a) Función de pérdida. (b) Precisión.

FIG. 11: Modelo entrenando solo la capa clasificadora con $n = 256$ y $p = 0.1$.

III. CONCLUSIONES

A lo largo de este trabajo se fueron entrenando distintas redes neuronales, se modelaron autoencoder y clasificadores convolucionales. Como se esperaba, a medida que fue aumentando la complejidad de los modelos los errores disminuyeron y las precisiones aumentaron, tanto en el conjunto train como en el de test. Sin embargo, esto también fue generando un aumento en la distancia entre ambas curvas, comenzando a observarse overfitting.

Los valores propuestos por default generaron buenos resultados en ambos modelos. De todas formas, con algunas variaciones de hiperparámetros se lograron mejorar los mismos.

En las redes convolucionales pudo observarse que no es necesario entrenarlas durante muchas épocas, ya que mientras la curva train sigue mejorando, la curva test tiene a estabilizarse.

Se pueden construir nuevas redes con mejores resultados, estudiando los distintos parámetros y arquitecturas posibles, ya que en este trabajo se vieron solo algunas modificaciones posibles. En cualquiera de estos casos, es importante tener en cuenta hasta qué punto complejizar el modelo, estudiando cómo van variando los resultados en el conjunto de entrenamiento y en el de testeo, para de esta manera evaluar la capacidad de aprendizaje y evitar un sobreajuste de la red.