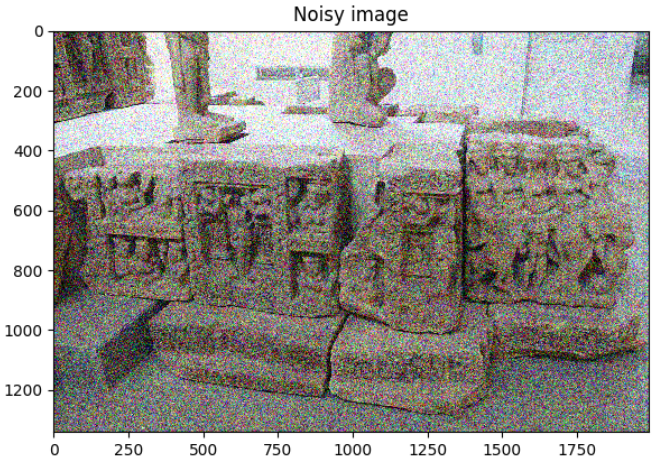
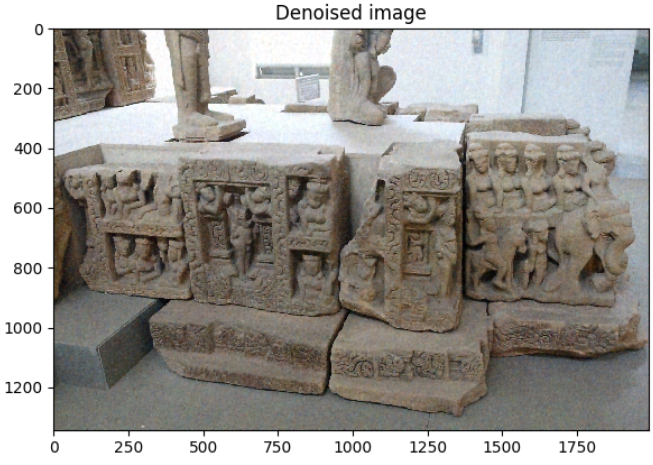
Eliminación de ruido mediante Autoencoders

Julio Castaño Amorós

March 2021





Contents

1 Introducción **3**

2 Objetivos **3**

3 Datasets **3**
3.1 Mnist 3
3.2 FER2013 4
3.3 CIFAR-100 4

4 Experimentación **5**
4.1 Autoencoder a mano junto con dataset Mnist 5
4.2 Experimentación con dataset FER2013 8
4.2.1 Experimentación con arquitectura a mano 9
4.2.2 Experimentación con arquitectura U-NET 11
4.2.3 Structural Similarity Index - SSIM 13
4.3 Experimentación con dataset CIFAR-100 13

5 Conclusiones **15**

1 Introducción

Este proyecto consiste en utilizar un tipo de arquitectura de redes neuronales conocida como "Autoencoders" para eliminar ruido de imágenes.

La resolución de este problema se enfoca desde diferentes niveles de dificultad, empezando con arquitecturas más sencillas y bases de datos simples como Mnist, hasta arquitecturas más complejas como Unet.

Esta forma de implementar el proyecto permitirá hacer un seguimiento de los éxitos y limitaciones de las diferentes arquitecturas aplicadas a diferentes bases de datos.

2 Objetivos

El objetivo principal es profundizar en algún tipo de técnica de aprendizaje profundo que hemos visto en clase. Mi elección ha sido profundar en Autoencoders, ya que es un tipo de arquitectura con la que he intentado realizar algún experimento en otras ocasiones, pero sin éxito.

Por lo tanto, me gustaría conocer mejor esta arquitectura desde un punto de vista de más bajo nivel, con arquitecturas más sencillas, hasta un punto de vista de más alto nivel, utilizando arquitecturas que formen parte del estado del arte.

3 Datasets

3.1 Mnist

Es un dataset de dígitos manuscritos que está incorporado en Keras y Tensorflow. Contiene 60000 imágenes de entrenamiento y 10000 de test, y 10 clases que corresponden con los dígitos del 0 al 9.

Figure 1: Ejemplos de Mnist.

3.2 FER2013

FER2013 es un dataset que está formado por imágenes de tamaño 48x48 de personas con diferentes emociones. Este dataset recoge las 7 expresiones básicas: enfado, asco, miedo, felicidad, tristeza, sorpresa y neutro. Contiene dos conjuntos de datos, uno de train con 28709 imágenes y otro de test con 7178 imágenes de test.

Para este problema no es importante las clases que tenga este dataset, es más importante tener en cuenta que son imágenes más grandes. Este dataset es más complejo ya que estas imágenes contienen un mayor número de características que el de Mnist.

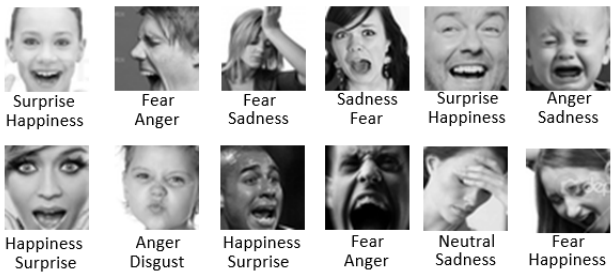


Figure 2: Ejemplos de FER2013.

3.3 CIFAR-100

CIFAR-100 es un dataset que está formado por 600 imágenes de 100 clases diferentes. Como ya se ha comentado, las clases en general no tienen gran importancia en este problema.

Sin embargo, que este dataset tenga 100 clases significa que hay una mayor variedad de imágenes que en el resto de datasets. Este factor hará que eliminar el ruido en las imágenes sea más complicado. Además, las imágenes de este dataset están un poco borrosas.

Al principio, la idea no era utilizar este dataset, sino un dataset con imágenes más grandes que también estuvieran en color. Dos factores han influido para no poder utilizar otro dataset. Por una parte, la memoria en Google Colab no soporta un tamaño de imágenes más grande (400x400 aprox.) con la arquitectura U-NET. Por otra parte, el número de imágenes del dataset utilizado previamente era muy pequeño para una arquitectura como U-NET.

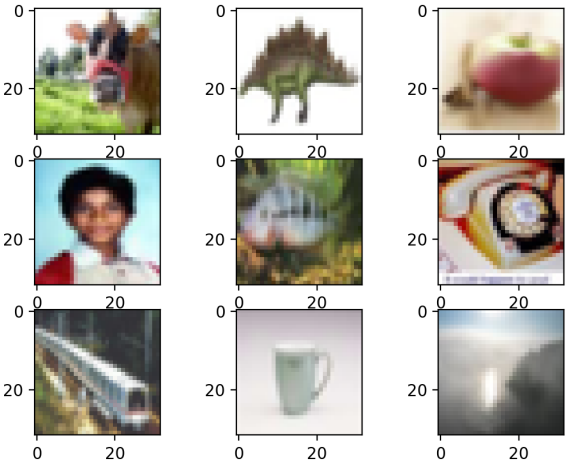


Figure 3: Ejemplos de CIFAR-100.

4 Experimentación

4.1 Autoencoder a mano junto con dataset Mnist

Esta sección corresponde con la primera experimentación realizada para eliminar ruido de imágenes. Puesto que es la primera toma de contacto con el problema, se utiliza una arquitectura sencilla junto con un dataset sencillo como es Mnist.

Como he comentado anteriormente, Mnist contiene 60000 mil imágenes de entrenamiento y 10000 imágenes de test. Sin embargo, para realizar el entrenamiento, he dividido el conjunto de test en dos conjuntos: uno de validación con 7000 imágenes y otro de test más pequeño de sólo 3000 imágenes, al que lo he llamado test_small.

Una vez tengo los datos divididos en tres conjuntos, puedo realizar su preprocesamiento. Básicamente, se convierten las imágenes a float y se dividen entre 255 para normalizar los valores entre 0 y 1, debido a que la red entrena mejor con valores más acotados.

A continuación, se crea un dataset nuevo añadiendo ruido a los datos iniciales. Este ruido es aleatorio y se obtiene de una distribución gaussiana. Después de añadir el ruido, se vuelven a normalizar los datos entre 0 y 1. Entonces, tengo dos conjuntos de datos, los datos de entrenamiento, validación y test con ruido, y los datos de entrenamiento, validación y test sin ruido.

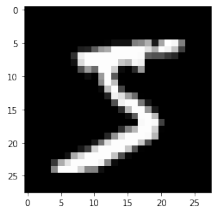


Figure 4: Ejemplo dígito 5 sin ruido.

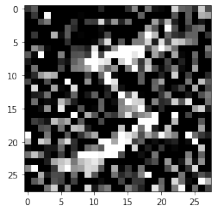


Figure 5: Ejemplo dígito 5 con ruido.

El siguiente paso es crear el autoencoder, la arquitectura es bastante sencilla, pero veremos que es suficiente para eliminar el ruido en este dataset. Básicamente, la arquitectura está formada por un encoder y un decoder:

- **Encoder:** 2 capas convolucionales de 32 filtros con kernel 3x3 y función de activación relu. Entre cada capa de convolución, un capa de MaxPooling de 2x2 para reducir el tamaño de la imagen.
- **Decoder:** 2 capas convolucionales iguales que en el encoder, y entre cada capa, una capa de UpSampling 2x2 para recuperar el tamaño inicial.
- **Salida:** 1 última capa convolucional con un único filtro 3x3 y función de activation sigmoide para que la imagen de salida tenga valores entre 0 y 1.

Una vez tenemos el modelo creado, definimos Adam como optimizador y bynary cross-entropy como función de pérdida. Ejecutamos el entrenamiento utilizando los datos con ruido como datos de train y sus etiquetas son las imágenes sin ruido. Con los datos de validación se hace lo mismo, datos con ruido y los datos sin ruido como etiquetas. De este modo, entrenamos el autoencoder para que, en el UpSampling, aprenda a crear las imágenes sin ruido.

El entrenamiento termina después de 100 épocas con un train loss de 0.0929 y un validation loss de 0.030. Las gráficas de entrenamiento son las siguientes, Figura 6 y Figura 7:

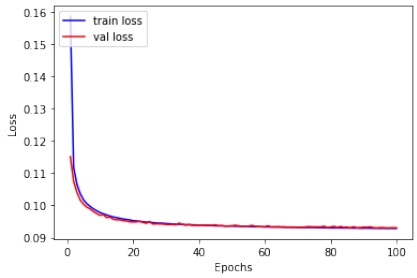


Figure 6: Gráfica de error.

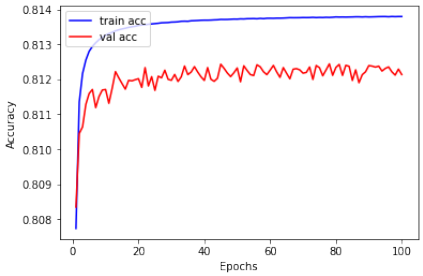


Figure 7: Gráfica de precisión.

Por último en cuanto a esta prueba, se muestran los resultados con el conjunto de test.

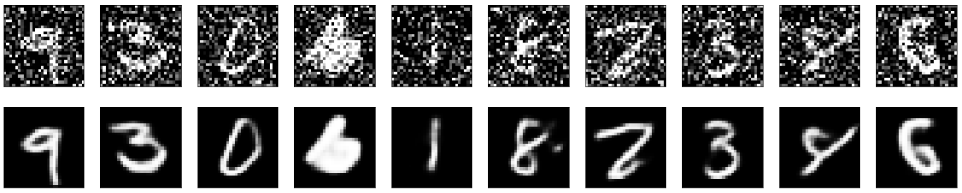


Figure 8: Dígitos con ruido añadido arriba, dígitos predichos abajo.

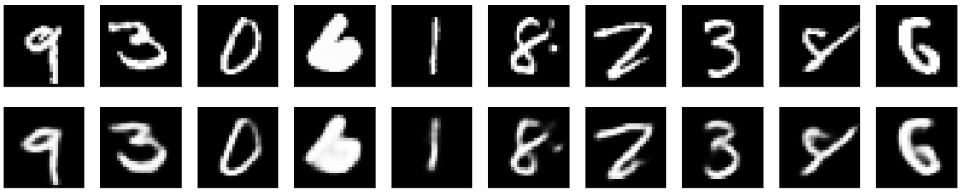


Figure 9: Dígitos originales arriba, dígitos predichos abajo.

Es obvio que los resultados son muy buenos, pero ya hemos comentado que este problema es muy sencillo. A continuación, entrenaremos este autoencoder con un dataset un poco más complejo.

4.2 Experimentación con dataset FER2013

En primer lugar, se describe todo el proceso para cargar los datos y preprocesarlos y, a continuación, se explican en dos secciones separadas las dos experimentaciones: una con la misma arquitectura utilizada con el dataset Mnist, y otra con una arquitectura de UNET.

Para cargar los datos se utiliza la API de Kaggle ya que estos datos forman parte de una antigua competición. Después, se utiliza la librería pandas para cargar los datos debido a que tienen formato csv.

Una vez cargados los datos, tenemos un conjunto de train y uno de test. Puesto a que en el dataset también vienen incluídas las etiquetas, ya que el dataset fue creado para un problema de clasificación, debemos coger solamente los píxeles que representan las imágenes.

Cuando ya tenemos las imágenes, separo el conjunto de train en dos conjuntos, un nuevo conjunto de train con 21709 imágenes y un conjunto de validación con 7000 imágenes. En este momento, ya dispongo de los tres conjuntos: train, validación y test.

Es momento de pasar a preprocesar los datos. El proceso a aplicar es el mismo que en Mnist, pero antes hay que convertir cada valor de píxel de cada imagen en entero debido a que en el dataset están como strings. Después, se normalizan las imágenes entre 0 y 1, y se aplica el reshape, pero en este caso de 48x48.

Las imágenes originales ya están preprocesadas, entonces hay que generar el dataset con ruido. El proceso es el mismo que en Mnist, sólo cambia el factor de ruido que le añadimos. Mientras que en Mnist utilizamos un factor de ruido de 0.5, para estos datos hay que utilizar un valor más pequeño (0.1) sino las imágenes se vuelven totalmente borrosas.

Las siguientes imágenes, Figura 10, Figura 11 y Figura 12, muestran ejemplos del dataset con ruido.

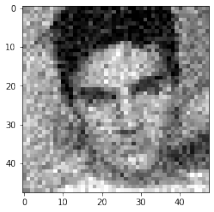


Figure 10: Ejemplo de FER2013 con ruido.

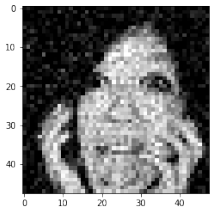


Figure 11: Ejemplo de FER2013 con ruido.

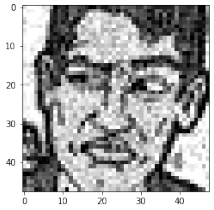


Figure 12: Ejemplo de FER2013 con ruido.

4.2.1 Experimentación con arquitectura a mano

El modelo creado es el mismo que el que se utiliza con Mnist y se entrena con los mismos hiper-parámetros.

Después de un entrenamiento de 100 épocas con batch size de 128, los resultados obtenidos en el entrenamiento son bastante peores que con el dataset anterior. Lo podemos observar en las siguientes imágenes, Figura 13 y Figura 14.

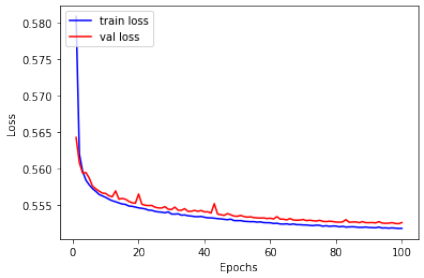


Figure 13: Gráfica de error.

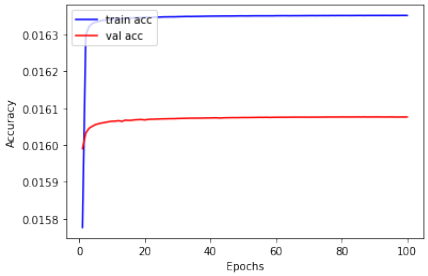


Figure 14: Gráfica de precisión.

Estos resultados se ven mejor en las imágenes de predicción, que aunque no son tan desastrosos como aparece en las gráficas, es obvio que este modelo no es suficiente para eliminar el ruido de las imágenes.

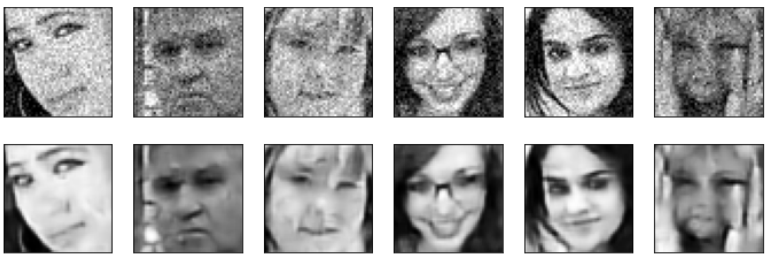


Figure 15: Imágenes con ruido añadido arriba, predichas abajo.

Figure 16: Imágenes con ruido añadido arriba, predichas abajo.

4.2.2 Experimentación con arquitectura U-NET

La arquitectura U-NET es una arquitectura que está considerada estado del arte en diferentes problemas como por ejemplo la segmentación de imágenes.

Este tipo de arquitectura destaca porque concatena los mapas de características extraídos en las partes de MaxPooling con las características que se utilizan para hacer el UpSampling. Esta técnica ayuda a la red para recuperar características y no perder tanto detalle en el UpSampling.

La arquitectura se puede observar mejor en la siguiente imagen, Figura 17.

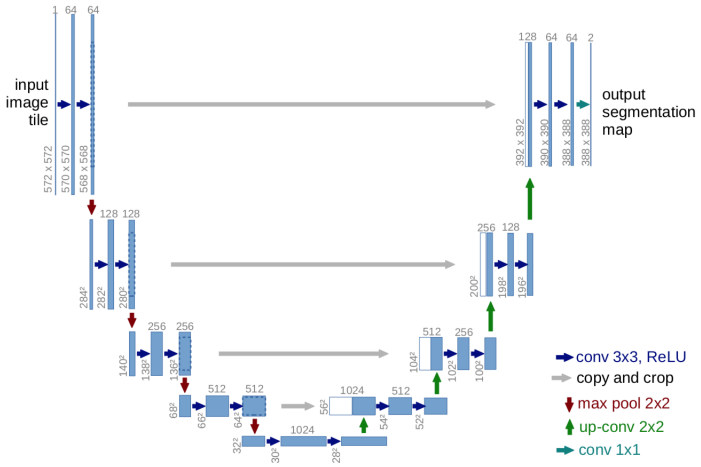


Figure 17: Arquitectura U-NET.

Debido a que es una arquitectura más profunda, el entrenamiento es mucho más lento que el anterior. Se entrena este modelo durante 35 épocas (tiempo justo para poder entrenar en Google Colab) con un batch size de 64. Como se

puede observar en la siguiente gráfica, Figura 18, el error es muy bajo desde las primeras épocas.

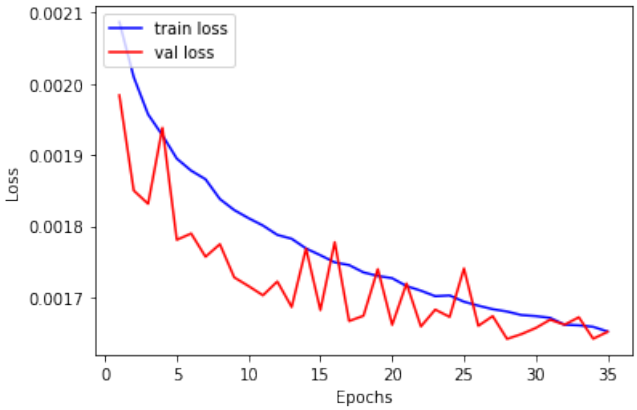


Figure 18: Gráfica de error.

Los resultados obtenidos en el conjunto de test son realmente buenos. Esta arquitectura es capaz de eliminar totalmente el ruido.

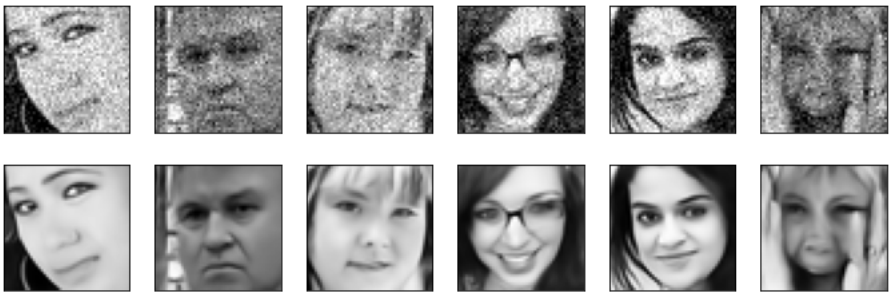


Figure 19: Imágenes con ruido añadido arriba, predichas abajo.

Figure 20: Imágenes con ruido añadido arriba, predichas abajo.

4.2.3 Structural Similarity Index - SSIM

Aunque la comparación entre los resultados de ambas arquitecturas se puede observar visualmente de forma clara, una buena métrica para comparar cuán se parece una imagen a otra es Structural Similarity Index. Esta métrica tiene en cuenta la intensidad, el contraste y la estructura para hacer el cálculo. El valor máximo que puede alcanzar es 1, que es cuando las dos imágenes son la misma.

Arquitectura	SSIM
Simple	0.9691
U-NET	0.9814

Table 1: Comparación de ambas arquitecturas en SSIM.

4.3 Experimentación con dataset CIFAR-100

Debido a que este dataset está incluído en Tensorflow, se puede cargar y utilizar de forma sencilla.

Al cargar los datos, obtenemos dos conjuntos. Un conjunto de entrenamiento con 50000 imágenes y un conjunto de test con 10000 imágenes. Separamos el conjunto de entrenamiento en dos subconjuntos, uno nuevo de entrenamiento con 35000 imágenes y otro de validación con 15000.

Como para el resto de datasets, el preprocesamiento es similar. Se normalizan las imágenes con valores entre 0 y 1, y se convierten las imágenes a tipo float. Después, se crea el dataset con ruido de la misma forma que en las secciones anteriores, pero en este caso, se utiliza un factor de ruido de 0.15.

Una vez los datos están preparados, se crea el modelo con la arquitectura U-NET modificando la capa de entrada tanto para imágenes de 32x32x3 y modificando

la capa de salida para que genere imágenes en color. El entrenamiento dura 35 épocas con un batch size de 128.

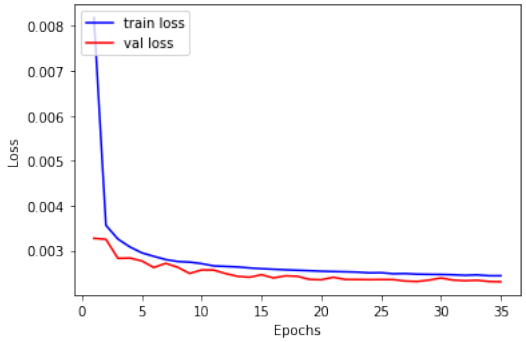


Figure 21: Gráfica de error.

Los resultados obtenidos puede que no impresionen tanto como los anteriores, pero se debe tener en cuenta el estado de las imágenes originales y el nivel de ruido.

Estos resultados se obtienen en un primer entrenamiento sin optimizar hiperparámetros. Los resultados podrían mejorar, pero aquí simplemente se busca una primera toma de contacto con este tipo de problema.

Figure 22: Imágenes originales arriba, predichas abajo.

Figure 23: Dígitos originales arriba, dígitos predichos abajo.

El resultado de SSIM obtenido es 0.9768.

5 Conclusiones

En definitiva, se ha visto como este tipo de arquitecturas son capaces de eliminar ruido de imágenes. Se ha trabajado tanto con arquitecturas y datos más sencillos como con arquitecturas y datos más complejos.

En general, se ha alcanzado el objetivo de este proyecto, que era profundizar en este tipo de arquitecturas aplicadas a la eliminación de ruido en imágenes. Aunque es cierto que me hubiera gustado poder utilizar imágenes más grandes y poder entrenar durante más tiempo, pero por temas de memoria ha sido imposible.