Proyecto 6

1st Barrios, Alonso
Universidad de Tecnología e Ingeniería
Ciencia de la Computación
Lima, Perú
alonso.barrios@utec.edu.pe

2nd Rodriguez, Mauricio
Universidad de Tecnología e Ingeniería
Ciencia de la Computación
Lima, Perú
mauricio.rodriguez@utec.edu.pe

3rd Tenazoa, Renzo
Universidad de Tecnología e Ingeniería
Ciencia de la Computación
Lima, Perú
renzo.tenazoa@utec.edu.pe

***Abstract*—Este proyecto está desarrollado para poner a prueba lo aprendido acerca de autoencoders. El objetivo del proyecto es crear una red neuronal capaz de transformar imagenes de baja resolución a alta resolución. La red neuronal deberá ser entrenada con un *set* de datos de *training* del mismo. *dataset*.**

***Index Terms*—redes neuronales, cnn, error, autoencoders, pooling, convolution**

I. Introducción

Con el objetivo de poner a prueba la red neuronal se utilizó el *dataset* de Imágenes, el cual contiene un conjunto de 855 imágenes. Se buscó que el modelo pueda mejorar la calidad de las imágenes. Además se mostrarán las gráficas de errores de cada caso.

II. Explicación

Para construir los modelos de *autoencoders* se utilizó la librería *torch* de Python.

Dado el módulo torch.nn que se nombrará solo como nn. En primer lugar, para generar un capa:

- **nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding):** Aplica una convolución 2d a los valores de entrada.
- **nn.ReLU()** Aplica una función de activación RELU sobre el modelo convolucionado.
- **nn.MaxPool2d(kernel_size, stride):** Aplica un *max pooling* 2d para reducir las dimensiones, en este caso las reduce a la mitad.
- **nn.Sequential()** Este método permite generar un módulo, en este caso, una capa que ejecutará de forma secuencial las funciones pasadas como parámetros y guardará los resultados.

Luego de haber generado los capas necesarios aprovechando las funciones mencionadas.

Finalmente, tanto el *encoder* como el *decoder* hacen uso de la función **forward(x)**. Esta función solo recibe x, la data inicial, como parámetro. La función, como el nombre lo dice, se encargará de realizar el proceso de *forwarding* de la red neuronal, el *encoder* se encargará de reducir la imagen, mientras que el *decoder* hace que vuelva a su tamaño original.

III. Experimentos

Se realizaron experimentos con distintas implementaciones de *autoencoders*, cada una con distintas características de forma que se pueda apreciar la utilidad de ellas.

A. CNN 3 capas

Para este modelo se usó CNN de 3 capas tanto en *encoder* como en *decoder*. En cada capa del *encoder* se realizó una convolución seguido de una función ReLU. Para la capa de *decoder* se realizó una convolución transpuesta y luego una función ReLU. Se obtuvo la siguiente tabla de errores:

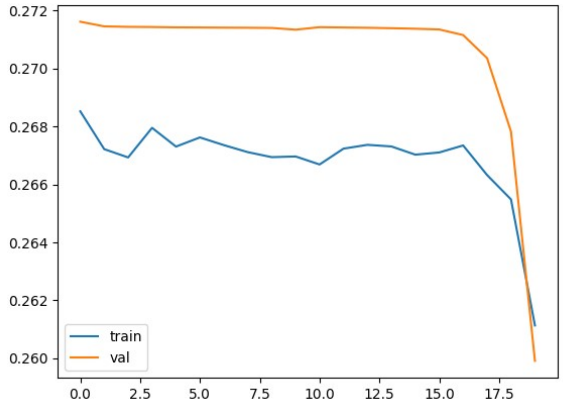

Fig. 1. Error CNN con 3 capas

Como se puede observar al inicio se mantiene un error casi constante, tanto con el *dataset* de entrenamiento como el de validación. Para que posteriormente (aproximadamente 17va época) comienze a bajar. Cabe resaltar que el modelo no presenta *overfitting* ni *underfitting*.

B. CNN 4 capas

Para este modelo se usó un CNN de 4 capas tanto en *encoder* como en *decoder*. Al igual que el modelo anterior, en cada capa del *encoder* se realizó una convolución seguido de una función ReLU. Para la capa de *decoder* se realizó una convolución transpuesta y luego una función ReLU. Se obtuvo la siguiente tabla de errores:

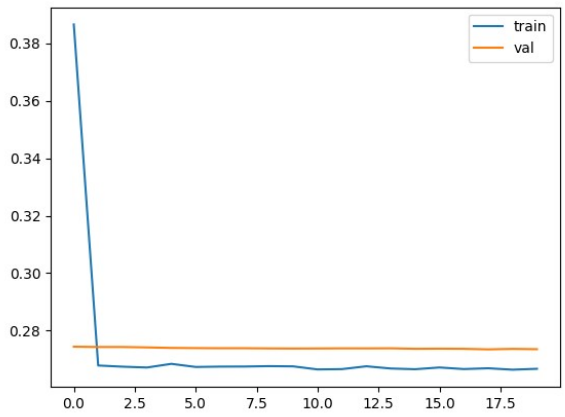

Fig. 2. Error CNN con 4 capas

Como se puede observar en la imagen el error decrece en una de las primeras épocas y este se mantiene constante hasta el final de las épocas. También podemos observar que los errores por parte del *dataset* de validación son casi constantes, esto es raro debido a que nos indica que al modelo le va mejor con datos que nunca a visto. A pesar de todo esto, el modelo no presenta *overfitting* ni *underfitting*.

C. CNN Tanh

Para este modelo se usó el modelo anterior de CNN 4 capas, aquí simplemente se cambiaron las funciones ReLU por Tanh dando las siguientes gráficas de errores:

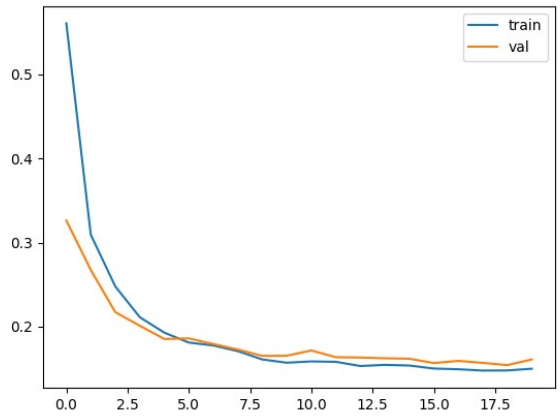

Fig. 3. Error CNN Tanh

Como se ve en la gráfica, este es el modelo que obtuvo un menor error, además el *dataset* de validación se comporta como lo esperado. La gráfica nos indica que no hay presencia de *overfitting* ni *underfitting*.

D. CNN Sigmoid

Para este modelo se usó el modelo anterior de CNN 4 capas, aquí simplemente se cambiaron las funciones ReLU por Sigmoid dando las siguientes gráficas de errores:

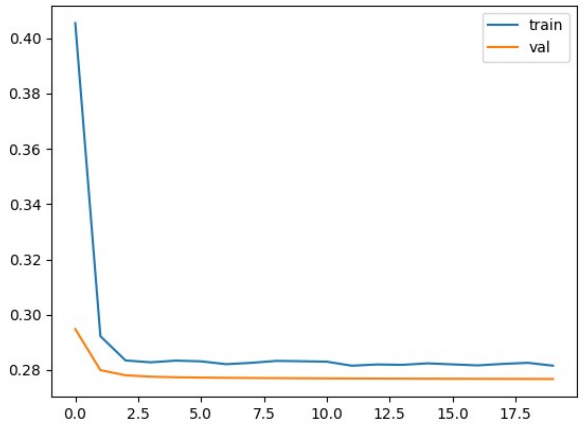

Fig. 4. Error CNN Sigmoid

Como se ve en la gráfica, . La gráfica nos indica que no hay presencia de *overfitting* ni *underfitting*.

CONCLUSIONES

- En los modelos obtenidos se .
- Durante la experimentación se pudo observar que el mejor modelo es el que cuenta con 5 capas, teniendo un *accuracy* de más del 97%. Siendo este uno de los más altos, incluyendo las implementaciones hechas en *kaggle*.
- El uso de VRAM de la GPU para cada modelo fue de aprox. 6GB, a diferencia del modelo de 5 capas que fue solo de 4GB aproximadamente.
- Durante la experimentación se pudo observar que el ”peor” modelo es el que cuenta con *batch normalization* y *dropout* en cada capa.
- El tiempo promedio que tomó la fase de *training* entre todos los modelos es de 3 horas sin contar el modelo de 5 capas.
- El tiempo que tomó el entrenamiento en el modelo de 5 capas fue de menos de 1 hora y usó menos memoria VRAM que los demás modelos.
- Se intentó entrenar el modelo en Khipu, sin embargo el tiempo que tomaba era similar, posiblemente no se estuvieron usando todos los recursos disponibles.

REPOSITORIO

Enlace al repositorio de Github:
https://github.com/alonso804/ai-project-6

REFERENCES

[1] Neural networks: ¿de qué son capaces las redes neuronales artificiales? (2020, 20 octubre). IONOS Digitalguide. https://www.ionos.es/digitalguide/online-marketing/marketing-para-motores-de-busqueda/que-es-una-neural-network/.

[2] Rosebrock, A. (2021). PyTorch: Training your first Convolutional Neural Network (CNN) - PyImageSearch. PyImageSearch. Retrieved 6 December 2021, from https://www.pyimagesearch.com/2021/07/19/pytorch-training-your-first-convolutional-neural-network-cnn/.

[3] torch.nn — PyTorch 1.10.0 documentation. Pytorch.org. (2021). Retrieved 6 December 2021, from https://pytorch.org/docs/stable/nn.html.

[4] Dumoulin, Vincent, and Francesco Visin. “A guide to convolution arithmetic for deep learning.” stat 1050 (2016): 23