Práctica 2.2: Clasificación de imágenes con PM y CNN

Redes de Neuronas Artificiales

Diciembre 2023

Grado en Ingeniería Informática

Pablo Hidalgo Delgado. NIA: 100451225
Marcos Caballero Cortés. NIA: 100451047

ÍNDICE

1. Introducción

El objetivo de la práctica es entrenar diferentes arquitecturas de CNN para estudiar el impacto de los diferentes hiperparámetros de estas redes en la resolución del problema planteado. Se pretende también comparar el resultado de la mejor CNN con el resultado obtenido con un Perceptrón Multicapa (PM) que utilice como entrada los píxeles de la imagen.

2. Análisis exploratorio de datos (EDA)

Primero, realizamos un breve Análisis Exploratorio de Datos (EDA) para resumir y entender la naturaleza de nuestro conjunto de datos. Esto nos permite comprender mejor la naturaleza de nuestros datos. Por medio de este análisis, observamos que el conjunto se compone de imágenes con dimensiones de 256x256 píxeles y 3 canales de color (RGB). Disponemos de un total de 21 clases diferentes, y cada clase cuenta con 100 imágenes, lo que suma un total de 2100 imágenes.

3. Preproceso

Previamente a entrenar los modelos, debemos realizar una transformación de los datos para que puedan ser interpretados por los algoritmos de manera eficiente. Las transformaciones de datos que realizamos son las siguientes:

3.1 División en datos de train, validation y test

Dividimos el conjunto de datos en train (80%), y test (20%). Posteriormente, dividimos los datos de entrenamiento en train train (80%) y train validation (20%). Este proceso lo realizamos mediante la función train_test_split() de la librería de scikit-learn. Además, utilizamos el parámetro stratify para realizar una partición estratificada, es decir, mantener la misma proporción de clases en ambos conjuntos. Esta decisión se fundamenta en la limitada cantidad de imágenes disponibles para cada clase. Sin la estratificación, existe la posibilidad de obtener conjuntos de datos con un número escaso de muestras para algunas clases.

3.2 Aleatorización

En nuestro caso, es importante aleatorizar los datos ya que las imágenes vienen ordenadas por clase, por lo que el modelo podría captar patrones relacionados con este orden.
La aleatorización la realizamos a la vez que la división de datos especificando el parámetro shuffle = True en la función train_test_split().

3.3 Normalización

La normalización es crucial para las CNN ya que aborda problemas de estabilidad, mejora la convergencia del modelo y contribuye significativamente a una mejor generalización en datos no vistos. Para llevar a cabo la normalización de datos, incorporamos una capa de BatchNormalization al comienzo de la red. Esta capa se encarga de centrar y escalar las entradas durante el entrenamiento.

4. Perceptrón Multicapa (PM)

4.1 Experimentación

Realizamos varios experimentos con Perceptrón Multicapa para la clasificación de las imágenes modificando algunos de sus hiperparámetros (tasa de aprendizaje, función de activación y capas ocultas). De esta manera, entrenamos y evaluamos un total de 5 modelos, tal y como se especifica en el enunciado de la práctica. Todos ellos tienen una capa de salida con 21 neuronas (21 clases) y con función de activación softmax, por lo que la salida de la red indicará la probabilidad de pertenecer a cada clase.

Hemos de mencionar que utilizamos el optimizador Adam y tratamos de optimizar la función sparse_categorical_crossentropy. Esta función mide la pérdida de entropía entre las clases y las predicciones y se utiliza cuando la salida deseada es un número entero, que indica la clase a la que pertenece el patrón de entrada. Por esta razón, no realizamos una codificación OHE de la variable de salida aun abordando un problema de clasificación multiclase.

Es importante señalar que en todos estos experimentos empleamos un mismo número de épocas, 100. No obstante, utilizamos la técnica de "Early Stopping" con un umbral de paciencia establecido en 20 épocas. Además, configuramos el parámetro restore_best_weights = True, permitiendo que el modelo devuelto sea aquel que ha exhibido el menor error de validación durante el proceso de entrenamiento.

A partir de estos modelos construidos, seleccionaremos aquel que maximice el accuracy y lo entrenaremos con los datos de train y validación conjuntamente para optimizar el rendimiento del modelo y garantizar una generalización adecuada. Al entrenar con ambos conjuntos de datos, el modelo tiene acceso a una variedad más amplia de patrones y características, lo que puede ayudar a mejorar su capacidad para reconocer y adaptarse a diferentes situaciones.

4.2 Resultados obtenidos

pruebas_PM

LR	Hidden Layers	Neuronas	Funcion Activacion	Tiempo Train	Best Epoch	Epochs	Accuracy train	Accuracy val	Accuracy test
0,01	2	[150, 50]	relu	27,753	5	100	0,0528	0,0506	0,0476
0,01	3	[300, 150, 50]	sigmoid	73,007	45	100	0,4152	0,2768	0,2429
0,001	2	[300, 100]	sigmoid	62,079	20	100	0,5818	0,2679	0,2571
0,001	2	[150, 50]	relu	23,764	19	100	0,4792	0,1548	0,0500
0,0005	2	[300, 100]	sigmoid	65,959	37	100	0,7381	0,2887	0,2738

Escogemos el modelo que maximiza el accuracy:

- sigmoide, lr = 0.0005, neuronas = [300, 100]

Lo entrenamos durante 37 épocas (best epoch) con los datos de train y validación conjuntamente. Evaluamos el modelo con los datos de test y los resultados obtenidos son los siguientes:

```
Accuracy: 0.310

Recall: [0.2, 0.1, 0.35, 0.55, 0.3, 0.9, 0.2, 0.9, 0.4, 0.25, 0.65, 0.1, 0.2, 0.1, 0.2, 0.05, 0.45, 0.4, 0.05, 0.05, 0.1]

Confusion Matrix:
[[ 4  0  0  1  0  2  0  7  0  2  0  1  0  0  0  0  1  1  1  0  0]
 [ 1  2  0  4  2  0  0  2  2  1  0  0  0  0  1  0  1  0  1  3  0]
 [ 0  1  7  0  0  0  0  4  0  5  0  0  0  0  0  0  1  0  1  0  1]
 [ 0  0  0 11  0  0  0  0  0  2  0  0  0  1  0  0  0  0  0  4  2]
 [ 0  1  0  0  6  0  0  0  2  3  1  0  0  0  1  0  0  2  2  2  0]
 [ 0  1  0  0  0 18  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  2  0  1  0  1  4  0  1  2  0  0  0  1  0  1  2  1  2  1  1]
 [ 0  0  0  0  0  0  0 18  0  0  1  0  0  1  0  0  0  0  0  0  0]
 [ 0  1  3  1  1  1  0  1  8  1  0  0  0  1  0  0  0  0  0  2  0]
 [ 0  4  2  1  0  0  0  1  0  5  0  0  1  0  0  0  1  2  3  0  0]
 [ 0  0  0  0  0  0  1  2  0  0 13  0  1  0  0  2  0  1  0  0  0]
 [ 0  0  0  3  2  0  0  1  2  1  1  2  2  1  0  1  2  0  0  1  1]
 [ 0  2  0  1  3  1  1  0  0  1  1  1  4  1  0  0  2  1  0  1  0]
 [ 0  0  1  1  1  0  0  1  1  0  3  1  0  2  1  1  3  1  2  0  1]
 [ 0  0  1  1  2  0  2  3  1  0  1  0  0  2  4  0  0  0  0  3  0]
 [ 0  0  0  1  1  2  1  6  0  0  1  1  0  2  0  1  0  2  2  0  0]
 [ 1  0  0  0  0  0  0  7  0  0  0  0  0  0  0  1  9  2  0  0  0]
 [ 0  0  0  0  0  1  0  0  0  0  2  0  0  0  1  0  6  8  0  2  0]
 [ 1  1  0  2  2  0  0  3  3  3  0  0  1  0  1  0  0  0  1  1  1]
 [ 0  0  1  2  3  0  5  1  2  0  1  1  0  0  1  0  1  1  0  1  0]
 [ 1  0  1  1  1  0  0  1  0  3  1  0  0  0  3  0  3  2  1  0  2]]
```

4.3 Análisis de resultados

Observando las pruebas, como podíamos prever inicialmente, no hemos obtenido resultados satisfactorios al emplear el PM para la clasificación de imágenes. Se observa que el accuracy alcanza un máximo de 0.27, un rendimiento que consideramos insuficiente para catalogar el modelo como exitoso. Cabe resaltar que, según nuestros resultados, la función sigmoide presenta un desempeño superior en comparación con la función de activación ReLU en este contexto específico.

En relación con el mejor modelo generado, notamos que el accuracy es ligeramente superior al obtenido en el conjunto de validación. No obstante, sigue siendo un valor bajo (0.31). Al examinar la matriz de confusión y el recall, no podemos afirmar que este resultado sea satisfactorio, aun teniendo en cuenta el considerable número de clases disponibles (21). Para el modelo puede resultar desafiante aprender adecuadamente todas estas clases diversas.

Destacamos el alto recall obtenido para la clase 5 y 7, que se corresponden con imágenes aéreas de un chaparrales y bosques respectivamente. A pesar de este logro puntual, podemos ver fijándonos en el recall y en la matriz de confusión que seguimos enfrentando desafíos significativos en la clasificación de las demás clases.

5. Red Convolucional (CNN)

5.1 Experimentación

En un primer momento, nuestro objetivo era construir numerosos modelos, variando todos los hiperparámetros que componen una red convolucional, tales como la tasa de aprendizaje, el número de capas convolucionales, la cantidad de filtros, el tamaño del kernel, el número de capas ocultas del Procesador de Mapas (PM), el número de neuronas en cada capa oculta, y la función de activación de cada capa oculta del PM. Sin embargo, debido a problemas relacionados con la capacidad de la RAM de la GPU, finalmente

tuvimos que limitar nuestros experimentos y nos vimos obligados a reducir los hiperparámetros que queríamos probar. Además, para no sobrecargar este recurso, reducimos el batch_size a 16. Así, decidimos utilizar una única tasa de aprendizaje, ya que somos conscientes de que el optimizador Adam la ajustará a lo largo de las iteraciones. Después de algunas pruebas iniciales, optamos por una tasa de aprendizaje inicial de 0.001, que observamos que funcionaba bien.
También nos vimos obligados a utilizar un único tamaño de kernel para cada número de capas convolucionales, así como el número de épocas, que lo reducimos a 50 con una paciencia de 20.

Cabe mencionar que en cada capa convolucional incorporamos una capa de MaxPooling para reducir la dimensionalidad de las representaciones espaciales. Esto permite una extracción de características más robusta y eficiente durante el proceso de convolución. La operación de MaxPooling ayuda a preservar las características más relevantes mientras reduce el número de parámetros y la carga computacional en capas subsiguientes del modelo. Las capas de pooling tienen una dimensión de ventana (pool_size) de 2x2 y un stride (número de píxeles en que se desplazará el filtro de pooling) de 2x2.

Después de completar los experimentos iniciales, exploramos el impacto del dropout en los tres mejores modelos obtenidos. Para ello, creamos dos variantes de cada uno de estos modelos, aplicando distintos valores de dropout (0.15 y 0.25) tanto a las capas convolucionales como a las capas del Perceptrón Multicapa (PM). Estos modelos fueron entrenados durante 100 épocas, con un umbral de paciencia de 20 épocas, ya que observamos, tras pruebas preliminares, que los modelos con dropout tienden a requerir más tiempo para converger.

5.2 Resultados obtenidos

pruebas_CNN

LR	Filtros	Kernel	Neurons PM	Funcion Activ PM	Tiempo Train	Best Epoch	Accuracy train	Accuracy val	Accuracy test
0,001	[16, 32]	[[3, 3], [2, 2]]	[300, 100]	relu	93,708	27	1,0000	0,5000	0,4333
0,001	[16, 32]	[[3, 3], [2, 2]]	[300, 100]	sigmoid	88,105	13	1,0000	0,5982	0,5833
0,001	[16, 32]	[[3, 3], [2, 2]]	[300, 150, 50]	relu	60,767	9	1,0000	0,4792	0,3857
0,001	[16, 32]	[[3, 3], [2, 2]]	[300, 150, 50]	sigmoid	96,201	18	1,0000	0,5476	0,5071
0,001	[8, 16, 32]	[[2, 2], [3, 3], [4, 4]]	[300, 100]	relu	39,802	24	1,0000	0,4881	0,4429
0,001	[8, 16, 32]	[[2, 2], [3, 3], [4, 4]]	[300, 100]	sigmoid	45,080	24	1,0000	0,5565	0,5119
0,001	[8, 16, 32]	[[2, 2], [3, 3], [4, 4]]	[300, 150, 50]	relu	38,495	23	0,9978	0,4702	0,3571
0,001	[8, 16, 32]	[[2, 2], [3, 3], [4, 4]]	[300, 150, 50]	sigmoid	48,315	11	0,9993	0,5387	0,5476
0,001	[32, 64]	[[3, 3], [2, 2]]	[300, 100]	relu	119,694	23	1,0000	0,5179	0,5286
0,001	[32, 64]	[[3, 3], [2, 2]]	[300, 100]	sigmoid	263,969	42	0,9940	0,5655	0,5548
0,001	[32, 64]	[[3, 3], [2, 2]]	[300, 150, 50]	relu	103,658	15	1,0000	0,4256	0,4024
0,001	[32, 64]	[[3, 3], [2, 2]]	[300, 150, 50]	sigmoid	163,836	26	1,0000	0,5387	0,5214
0,001	[16, 32, 64]	[[2, 2], [3, 3], [4, 4]]	[300, 100]	relu	56,545	21	1,0000	0,4375	0,2833
0,001	[16, 32, 64]	[[2, 2], [3, 3], [4, 4]]	[300, 100]	sigmoid	81,058	22	1,0000	0,6101	0,5881
0,001	[16, 32, 64]	[[2, 2], [3, 3], [4, 4]]	[300, 150, 50]	relu	58,011	7	0,9963	0,4881	0,3667
0,001	[16, 32, 64]	[[2, 2], [3, 3], [4, 4]]	[300, 150, 50]	sigmoid	77,379	11	0,9993	0,5268	0,5452

Obtenemos los 3 mejores resultados y los entrenamos incluyendo dropout:

- lr = 0.001, filtros = [16,32,64], kernel = [[2,2],[3,3],[4,4]], funcion_activacion_PM = sigmoide, neuronas= [300,100]: Accuracy_validation = 0.6101

- lr = 0.001, filtros = [16,32], kernel = [[3,3],[2,2]], funcion_activacion_PM = sigmoide, neuronas= [300,100]: Accuracy = 0.5833

- lr = 0.001, filtros = [32,64], kernel = [[3,3],[2,2]], funcion_activacion_PM = sigmoide, neuronas= [300,100]: Accuracy_validation = 0.5665

pruebas_CNN_dropout

LR	Dropout	Filtros	Kernel	Neurons PM	Funcion Activ PM	Tiempo Train	Best Epoch	Accuracy train	Accuracy val	Accuracy test
0,001	0,25	[16, 32, 64]	[[2, 2], [3, 3], [4, 4]]	[300, 100]	sigmoid	329,538	99	0,4903	0,497	0,45
0,001	0,25	[16, 32]	[[3, 3], [2, 2]]	[300, 100]	sigmoid	172,122	45	1	0,5655	0,5119
0,001	0,25	[32, 64]	[[3, 3], [2, 2]]	[300, 100]	sigmoid	264,931	39	1	0,5923	0,5524
0,001	0,15	[16, 32, 64]	[[2, 2], [3, 3], [4, 4]]	[300, 100]	sigmoid	131,960	37	1	0,631	0,6143
0,001	0,15	[16, 32]	[[3, 3], [2, 2]]	[300, 100]	sigmoid	123,437	15	1	0,5506	0,5071
0,001	0,15	[32, 64]	[[3, 3], [2, 2]]	[300, 100]	sigmoid	266,311	20	1	0,5506	0,5095

Escogemos el mejor modelo (aquel que tiene un accuracy más alto) y lo evaluamos con los datos de test.

- lr = 0.001, filtros = [16,32,64], kernel = [[2,2],[3,3],[4,4]], funcion_activacion_PM = sigmoide, neuronas_PM = [300,100], dropout = 0.15: Accuracy_validation = 0.6310

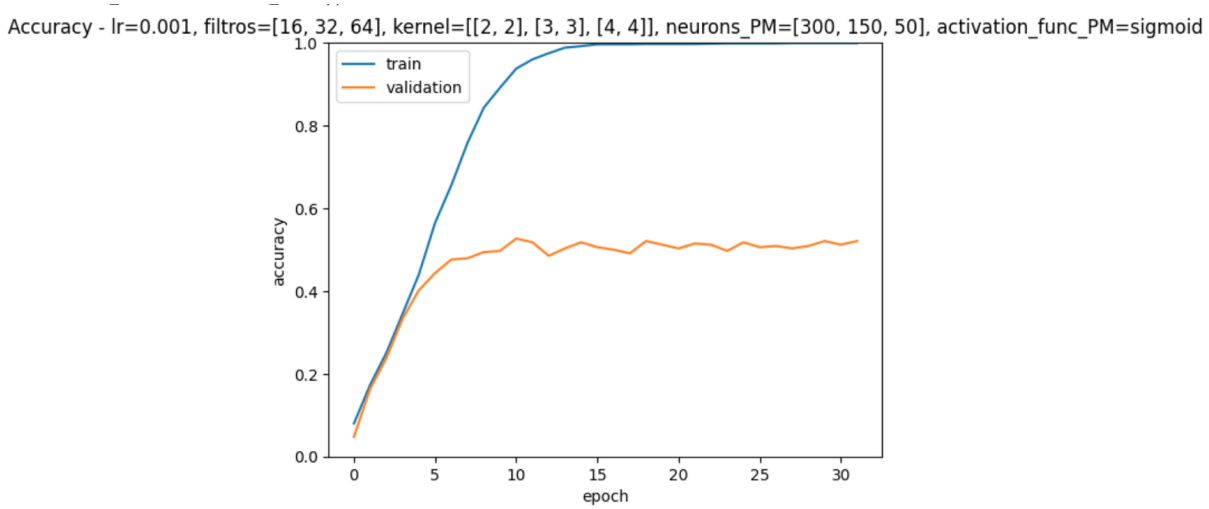
```
Accuracy: 0.507

Recall: [0.75, 0.5, 0.85, 1.0, 0.4, 0.95, 0.35, 0.75, 0.3, 0.4, 0.8, 0.4, 0.25, 0.4, 0.5, 0.4, 0.3, 0.6, 0.5, 0.05, 0.2]

Confusion Matrix:
[[15  0  0  0  0  2  0  0  0  0  0  0  0  0  0  0  0  3  0  0  0]
 [ 0 10  0  0  0  3  0  1  0  1  0  0  0  0  2  0  0  0  0  2  1  0]
 [ 0  0 17  0  0  0  0  0  1  0  0  0  0  0  0  0  1  0  0  0  1]
 [ 0  0  0 20  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  1  0  0  8  0  1  0  0  0  1  0  3  3  0  2  0  0  0  1  0]
 [ 1  0  0  0  0 19  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  1  0  0  3  0  7  0  0  0  0  1  3  2  0  0  0  0  0  3  0]
 [ 2  0  0  0  0  0  0 15  0  1  0  0  0  0  0  0  0  0  1  0  1]
 [ 0  0  0  0  0  1  0  0  6  1  1  0  1  0  2  1  0  3  3  1  0]
 [ 2  0  1  1  0  0  0  1  1  8  0  1  1  0  0  0  1  1  2  0  0]
 [ 0  0  0  0  0  0  0  0  0  0 16  0  0  1  0  2  0  0  1  0  0]
 [ 0  2  0  0  0  0  0  0  0  0  0  8  1  3  1  2  0  1  1  0  1]
 [ 0  0  0  0  1  0  3  0  0  0  2  1  5  5  0  1  0  0  0  0  2]
 [ 0  1  0  0  1  0  4  0  1  0  2  0  0  8  0  3  0  0  0  0  0]
 [ 1  1  0  0  3  0  1  0  0  0  1  1  1  1 10  0  0  0  0  0  0]
 [ 0  0  0  0  2  0  2  0  2  0  3  0  0  2  0  8  0  0  1  0  0]
 [ 0  0  0  0  0  0  0  4  1  4  1  0  1  0  0  0  6  1  0  0  2]
 [ 0  1  0  2  0  0  0  0  0  0  0  0  0  0  1  1  0 12  0  1  2]
 [ 0  6  0  0  0  0  1  0  1  1  0  0  0  0  0  0  0  1 10  0  0]
 [ 0  4  0  0  6  0  0  0  1  1  1  0  2  0  2  0  1  0  1  1  0]
 [ 0  0  2  0  0  0  1  0  3  0  1  1  2  1  0  1  2  0  1  1  4]]
```

5.3 Análisis de resultados

En cuanto a los resultados de nuestras pruebas, hemos observado que la función sigmoide proporciona accuracys de validación más altos en comparación con la función ReLU. También hemos constatado que se obtienen mejores resultados al distribuir las neuronas en [300,100] que al distribuirlas en [300,150,50]. Sin embargo, respecto al número de filtros y al

tamaño de los kernels, no hemos podido identificar un patrón claro, ya que los resultados son dispares.

En líneas generales, hemos notado que los modelos convergen rápidamente, dado que la mayoría alcanza su mejor época en menos de 30 iteraciones, a pesar de tener una paciencia elevada de 20 épocas. Por tanto, las gráficas que representan el accuracy de los modelos siguen un patrón similar, mostrando un aumento inicial seguido de una estabilización o ligera mejora, lo que indica que la red neuronal aprende de manera efectiva durante las primeras épocas y posteriormente alcanza su rendimiento óptimo. Lo vemos mejor en el siguiente gráfico:



En cuanto a los modelos que incluyen dropout, hemos observado que la elección entre un dropout de 0.25 o 0.15 no marca una diferencia significativa. De esta manera, los resultados obtenidos con modelos que incorporan dropout son bastante similares a los modelos sin dropout.

Al analizar nuestro mejor modelo, hemos logrado un accuracy de 0.507. Este valor es ligeramente inferior al obtenido en validación, por lo que evidenciamos que el modelo encuentra ciertas dificultades para generalizar sobre nuevos datos. No obstante, dada la considerable cantidad de clases (21), aunque no califiquemos este resultado como excelente, lo consideramos aceptable. Al observar la matriz de confusión, notamos que la diagonal principal contiene el mayor número de instancias, y en el resto predominan los valores 0, lo que sugiere que nuestro modelo no está tan desacertado.

6. Comparación PM y CNN

Comparando los resultados de los dos mejores modelos seleccionados, observamos la diferencia entre los valores del accuracy, 0.310 para el PM y 0.507 para la CNN. Esto subraya las mejoras que suponen las capas convolucionales a la hora de clasificar imágenes, evidenciando la eficacia de la CNN en la extracción y comprensión de características complejas presentes en las imágenes.
Sin embargo, donde mejor se evidencia la diferencia entre los resultados del PM y la CNN es en el recall, que indica la probabilidad que va a tener el modelo de clasificar bien una

clase determinada. Si observamos esta métrica para ambos modelos, podemos ver como la gran mayoría de valores del recall de la red convolucional para cada clase es mayor que los del perceptrón multicapa.
También fijándonos en esta métrica, podemos identificar más fácilmente las clases que ambas redes clasifican correctamente. Notamos que, en general, las clases con altos valores de recall en el Perceptrón Multicapa (PM) tienden a tener también altos valores en la red convolucional (CNN), y viceversa. Un ejemplo claro es la clase 5, que presenta valores de recall de 0.9 y 0.95 en el PM y la CNN respectivamente. En contraste, la clase 19, asociada a imágenes aéreas de depósitos de almacenamiento, muestra un bajo valor de recall de 0.05 en ambos modelos.

7. Conclusiones

En conclusión, los resultados de nuestros experimentos revelan que la Red Convolucional (CNN) supera significativamente al Perceptrón Multicapa (PM) en la tarea de clasificación de imágenes, evidenciado por un accuracy más alto en el mejor modelo obtenido para cada arquitectura (0.507 frente a 0.310). Además, hemos observado de manera consistente que la función sigmoide se destaca como la opción más efectiva para nuestros datos, desempeñándose bien tanto en el PM como en la CNN.

A pesar de la inclusión del dropout, no hemos observado mejoras sustanciales en el rendimiento general de los modelos. Aunque un modelo con dropout alcanzó el mejor resultado en accuracy de validación, no se tradujo en un aumento significativo en la clasificación general.

Por último, cabe mencionar que hemos tenido especiales dificultades a la hora de ejecutar el código de esta práctica ya que la GPU no siempre estaba disponible en Google Collab. Además, en algunas ocasiones, como hemos comentado anteriormente, la RAM de la GPU llegaba a su límite y se nos generaba un error “ResourceExhaustedError: Graph execution error". Estos inconvenientes afectaron la ejecución de nuestros experimentos.