




## - Generando el Dataset y Preprocesamiento

Mi primer problema fue encontrar muchas imágenes, ya que para mi problemática el profesor recomendó al menos tener 1000 entradas por clase al momento de entrenar el algoritmo para que reconozca cuál juego le estoy mostrando. Reunir tantas imágenes una por una tomaría horas o incluso días en lo que busco y descargo todo. Por ello busqué una extensión de Chrome y encontré una llamada “Download All Images” (La cuál llamaré DAI a partir de aquí) que, como indica su nombre, descarga todas las imágenes en un sitio web. La extensión me fue de gran utilidad, hasta diría que fue demasiado útil ya que toma un directorio base y se pone a buscar imágenes de la página en la que estás. Además, le puedes indicar el nivel de profundidad que quieres que tenga en su búsqueda, es decir, si se mete a otras páginas y busca imágenes ahí también. DAI también me permitió delimitar las imágenes que obtenía, quitando aquellas que hicieran match con su blacklist.

The screenshot shows the settings for the 'Download All Images' (DAI) extension. At the top, a 'Statistics (debug)' section displays 'Total: 134651, Frames: 1353, Images: 53226, Saving: 1764' with a progress bar. A 'Deep Search' button is set to 'Level 2'. The main settings are organized into several sections: 'File Size (slow)' with min/max values in bytes and an 'If unknown' option (Skip/Save); 'Dimensions (pixels)' with min/max width/height and an 'If unknown' option; 'Image Types' with checkboxes for All images, Selection, JP(E)G images, PNG images, GIF images, BMP images, WebP images, and an option to save as JPG if no extension is present; 'Regular Expressions' with a checkbox for URL-based filtering and a text input; 'Blacklisted Keywords' with a checkbox for keyword exclusion and a text input containing '120px, 122px, 144px, 180px, SSB, SM64, SMO, S'; 'Misc' with checkboxes for domain/server filtering, cross-origin frames, same-origin frames, identical image exclusion, separate downloads, and README generation; and 'File Mask' with a text input for renaming files and a 'Keywords' field. At the bottom, there are buttons for 'Reset', 'Restart', 'Stop', 'Gallery', 'Copy', 'Save DIR', and 'Save'.

Aún después de usar este método 3 veces para agarrar imágenes de *Super Mario Galaxy*, *Super Mario Galaxy 2*, y *Super Mario Odyssey* desde el sitio Mario Wiki, tardé aproximadamente 15 horas en reunir todas las imágenes que usé en mi Dataset. Los resultados de las búsquedas de Galaxy me regresaron aproximadamente 5000 imágenes en total, de las cuales había varias coincidencias. El set de Odyssey me regresó unas 2800 imágenes por sí misma.

Al final, mis tres búsquedas pesaban alrededor de 300MB en archivos ZIP. Después de combinar los resultados de ambos Galaxy y hacer limpieza de manera manual de los resultados (quitar toda imagen que no fuera del juego para el que se hizo la búsqueda), la clase de *Super Mario Galaxy* tenía 2369 imágenes, mientras que el de *Odyssey* tenía 1662 imágenes. Al principio no pensé mucho de esto ya que técnicamente estoy comparando dos juegos a uno, la razón se debe a que originalmente *Galaxy 2* iba a ser una expansión del primero, pero eventualmente se hizo otro juego debido a que era muy grande. No fue hasta que hice el modelo que me di cuenta de que posiblemente tener datos considerablemente desbalanceados afectó en las predicciones del modelo.

 SMG.zip	14/05/2025 10:00 p. m.	Carpeta compri...	101,916 KB
 SMG2.zip	16/05/2025 09:14 p. m.	Carpeta compri...	102,801 KB
 SMO.zip	18/05/2025 06:42 p. m.	Carpeta compri...	108,444 KB

La división de las clases también fue manual. Hice la división de la cantidad de imágenes en mi clase divida sobre 5 para obtener lo que sería el 20% de mi clase. Después ordené mis archivos por tamaño y busqué una letra al azar que me diera una cantidad suficiente, seleccionando los resultados y poniendo estos en una carpeta aparte. Así genere mis datos de entrenamiento y evaluación. Más adelante aprendí a hacer el split directamente en mi modelo para hacer la validación, aunque a veces pienso que hacer esto hizo que el split dividiera más la cantidad de datos de entrenamiento. Además de esto, me di cuenta de que para tomar el 20% de mi dataset no debí configurar el split como 20%, sino como el 25% gracias a que el programa no sabe que el split ya estaba hecho, entonces se toma mi 80% como un 100%, causando que mi porcentaje de validación en realidad sea alrededor de un 16% del dataset, mientras que el entrenamiento es 64%.

```
Found 2630 images belonging to 2 classes.  
Found 656 images belonging to 2 classes.
```

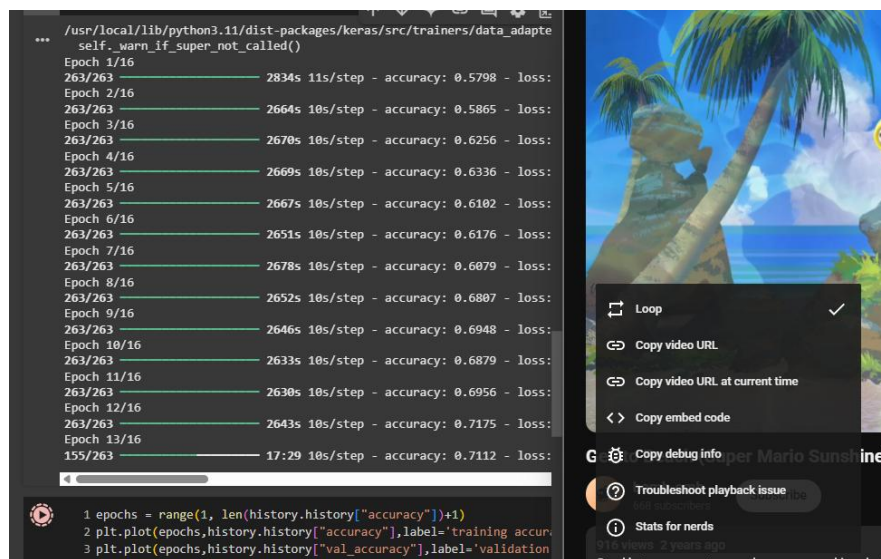
Por último, el preprocesamiento que apliqué a mis datos antes de entrenar el modelo no fue muy complejo. Hice shifts, rotaciones, cambios de brillo, etc. También habilité la función de voltear la imagen sobre su eje vertical para que reconociera personajes u objetos miran a un lado distinto del que viene en su dataset. Finalmente, toda imagen es escalada a tener un tamaño de 255 x 255 pixeles, debido a que pensé que sería de mayor utilidad si se tiene una imagen más grande (por tan poco que sea el incremento) para analizar. En retrospectiva, este puede ser uno de los factores que hizo que el entrenamiento tardara tanto.

## - Troubleshooting

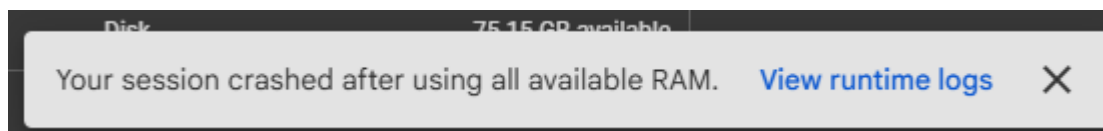
Antes de siquiera empezar a programar, debía instalar las dependencias, sin embargo, mi computadora se rehusaba a instalar tensorflow, por lo que el código se hizo en Google Colab, lo cuál fue tanto para bien como para mal.

```
Using cached tensorflow-2.19.0-cp310-cp310-win_amd64.whl (375.7 MB)
Installing collected packages: tensorflow
ERROR: Could not install packages due to an OSError: [Errno 2] No such file or directory: 'C:\\Users\\[redacted]\\AppData\\Local\\Packages\\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\\LocalCache\\local-packages\\Python310\\site-packages\\tensorflow\\include\\external\\com_github_grpc_grpc\\src\\core\\ext\\filters\\client_channel\\lib_policy\\grpcb\\client_load_reporting_filter.h'
HINT: This error might have occurred since this system does not have Windows Long Path support enabled. You can find information on how to enable this at https://pip.pypa.io/warnings/enable-long-paths
```

Podía hacer uso del GPU de Google para hacer el entrenamiento más rápido, pero se tiene que vigilar constantemente el programa para que la sesión no se termine. Aprendí durante uno de los entrenamientos que Colab tiene 2 límites para las sesiones, 90 minutos para idle (es decir, sin recibir acciones o comandos como un simple click) y 12 horas totales. Debido al tamaño de mi modelo, en las ocasiones que no podía usar el GPU el programa llegó a tardar 11 horas y media para pasar por los epochs. Si dejaba la computadora desatendida, se habría apagado y desconectado al entrar en reposo, por lo cual la mejor solución que encontré fue simplemente poner un video de YouTube largo y ponerlo en bucle mientras ocasionalmente movía el Colab para que no se terminara la sesión.



Otra desventaja de usar Colab fue el RAM, no sé si fue por haber pasado mucho tiempo, pero ocasionalmente se perdían variables durante el entrenamiento de los epochs, lo que significó reiniciar el proceso. En estas ocasiones no quedó de otra y simplemente tuve que volver a entrenar el modelo una vez más.



Finalmente, la cuestión del espacio en disco fue algo que me dejó un poco confundido al principio, pero logré entender el porqué después. Con las aproximadamente 4000 imágenes que quedaron en mi dataset el dataset pesaba 170MB, luego a mi modelo le dí 16 epochs para entrenar, cada epoch son 170MB adicionales debido a que el modelo pasa una y otra vez por el dataset para entender lo que está observando. Al terminar el entrenamiento, cada modelo pesaba 3.5GB, lo que dejó atónito a mis compañeros. Este proyecto también significó el fin de mi espacio en Drive del Tec, debido a que el gran tamaño del modelo llenó lo poco que me quedaba, es un poco irónico que después de 8 años de uso, tan solo una clase llenó  $\frac{1}{4}$  de lo que nos dan.

Total params: 476,192,069 (1.77 GB)	Ubicación: Todos en C:\Users\sebas\Documents\Tec\S10\
Trainable params: 476,192,069 (1.77 GB)	Tamaño: 171 MB (179,631,114 bytes)
Non-trainable params: 0 (0.00 B)	Tamaño en disco: 179 MB (187,858,944 bytes)

## - Modelo

He hablado mucho sobre él, pero no he hablado a fondo de su arquitectura. Para ser completamente honesto, la mayor parte del tiempo simplemente experimenté con capas convolutivas (Conv2D) debido a que son recomendadas al trabajar con imágenes. Más adelante encontré un artículo de investigación sobre la arquitectura CNN escrito por Joshua Aboyami (2022) [1] con base al cuál explicaré un poco la arquitectura usada. Usamos varias capas convolutivas para que el modelo vaya identificando cosas en cada una, iniciando desde líneas rectas o curvas y llegando eventualmente a patrones más complejos como caras y objetos. Las capas convolutivas analizan la imagen poco a poco iniciando desde la esquina superior izquierda, moviéndose a la derecha hasta llegar a un punto de quiebre a partir del cuál se va a la siguiente línea; asignan un valor a cada segmento que analizan en una matriz 3D. Por ello usé 3 capas convolutivas, para analizar la imagen (aunque creo que habría sido mejor usar más, pero opté por no hacerlo debido al tiempo de entrenamiento ya extensivo) las cuales aplané para después condensar los resultados a una sola neurona. Las capas usaron activaciones ReLU excepto la última, que usó sigmoid; ReLU convierte valores negativos a 0 y no cambia el tamaño del input, volviéndola una activación útil al momento de entrenar un modelo gracias a que no lo satura de

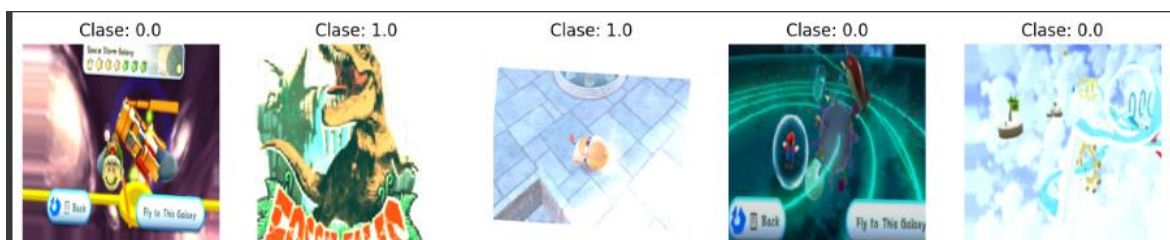
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 253, 253, 10)	370
conv2d_1 (Conv2D)	(None, 251, 251, 20)	1,820
conv2d_2 (Conv2D)	(None, 249, 249, 30)	5,430
flatten (Flatten)	(None, 1860030)	0
dense (Dense)	(None, 250)	476,167,930
dense_1 (Dense)	(None, 64)	16,448
dense_2 (Dense)	(None, 1)	65

información. La última capa del modelo se condensa a la cantidad de clases que hay (es decir, si hay 5 clases, 5 neuronas), a menos que se haga en binario.

Mis modelos hacen predicciones en binario, donde un positivo hace referencia a *Super Mario Odyssey* mientras que un negativo es *Super Mario Galaxy*. Esto facilita conocer la confianza de la predicción, debido a que mientras más se acerca a un valor partiendo desde 0.5, más seguro es.

Un problema con el que me encontré al momento de empezar a hacer el modelo fueron mis imágenes. En clase vimos como trabajar con datos en escala de grises por lo que no consideré lo que haría el trabajar con color. Sin embargo, eso no fue todo, varias de mis imágenes tienen alfas además de color (es decir, valores transparentes), por lo que si se convertían a una escala RGB la imagen quedaría algo deforme debido a que los alfas eran llenados con el color más cercano. Afortunadamente, fue algo fácil de resolver debido a que solo tuve que cambiar el Color Mode a usar RGBA para respetar los alfas.

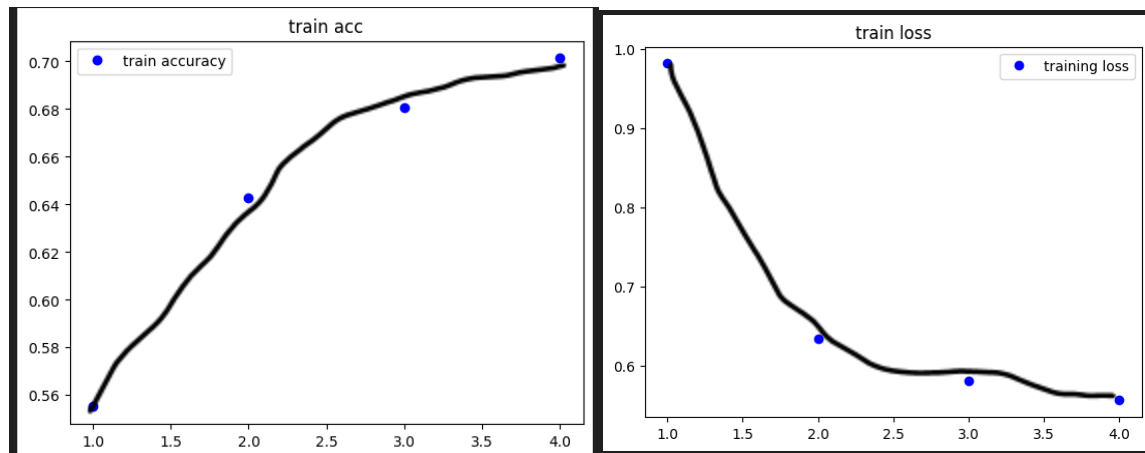
Debido a este cambio, también se cambió el input\_shape que toma el programa, pues en lugar de 3 espectros, ahora toma 4 (rojo, verde, azul, alfa).



- **Resultados**
- **Primer Modelo**

El primer modelo que se hizo fue algo sencillo, mi dataset completo aplicado a una arquitectura hecha a base de experimentación con 4 epochs de entrenamiento. Su accuracy fue de 65% y me dio las graficas que se pueden ver en la siguiente página.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 253, 253, 10)	370
conv2d_1 (Conv2D)	(None, 251, 251, 20)	1,820
dense (Dense)	(None, 251, 251, 256)	5,376
dense_1 (Dense)	(None, 251, 251, 256)	65,792
flatten (Flatten)	(None, 16128256)	0
dense_2 (Dense)	(None, 1)	16,128,257



Desafortunadamente en su momento no calculé los datos de validación, pero dado que el loss en los tests fue muy alto, diría que este primer modelo hacía Overfitting.

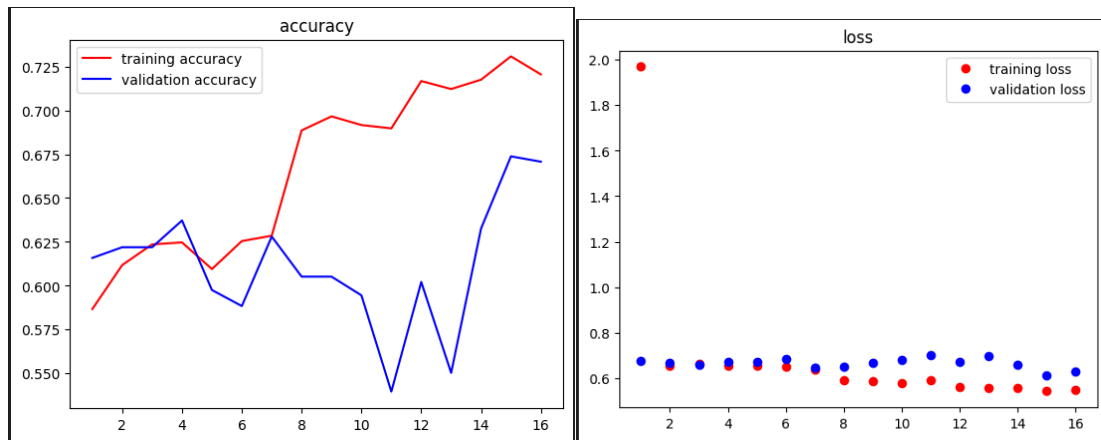
```
25/25 ————— 161s 5s/step - acc: 0.6457 - loss: 119.6239
test acc :
0.6499999761581421
```

## - Segundo Modelo

Insatisfecho con este resultado, decidí cambiar un poco la arquitectura del modelo, ahora con conocimiento de la arquitectura CNN, le agregué una capa convolutiva a mi modelo, aplané los resultados antes de condensar todo, y finalmente llegamos a la arquitectura actual.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 253, 253, 10)	370
conv2d_1 (Conv2D)	(None, 251, 251, 20)	1,820
conv2d_2 (Conv2D)	(None, 249, 249, 30)	5,430
flatten (Flatten)	(None, 1860030)	0
dense (Dense)	(None, 256)	476,167,936
dense_1 (Dense)	(None, 64)	16,448
dense_2 (Dense)	(None, 1)	65

Comparado con el primer modelo que tuvo 16 Millones de parámetros de entrenamiento, este tuvo 476 Millones. Además, decidí darle 16 epochs esta vez para entrenarlo mejor, también opté por usar el optimizador de Adam, que cuenta con un learning rate de aproximadamente 0.001. Supuse que usar un optimizador con varias características haría que el modelo aprendiera mejor que antes de cierta forma sí quedó mejor.



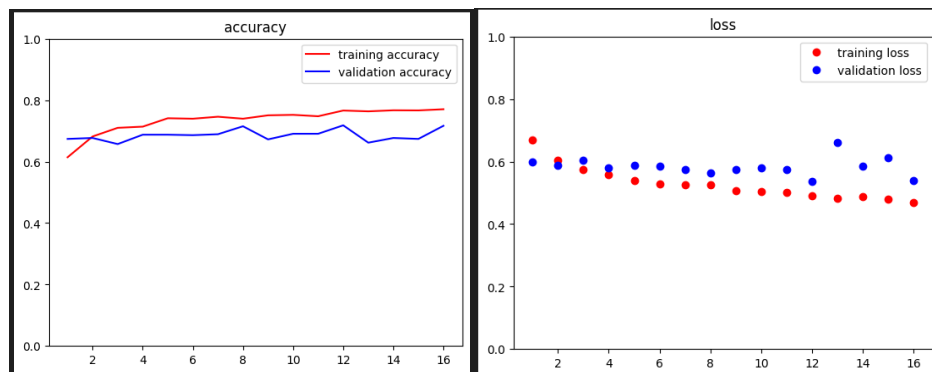
Pese a la dificultad en ver las gráficas debido a que no delimité los ejes correctamente, el modelo mostró ser más cercano a un Fit que Overfit esta vez, sin embargo, el accuracy aún estaba por debajo del 70%

```
25/25 ————— 98s 4s/step - accuracy: 0.6694 - loss: 0.6789
test acc :
0.6919999718666077
```

### - Tercer Modelo

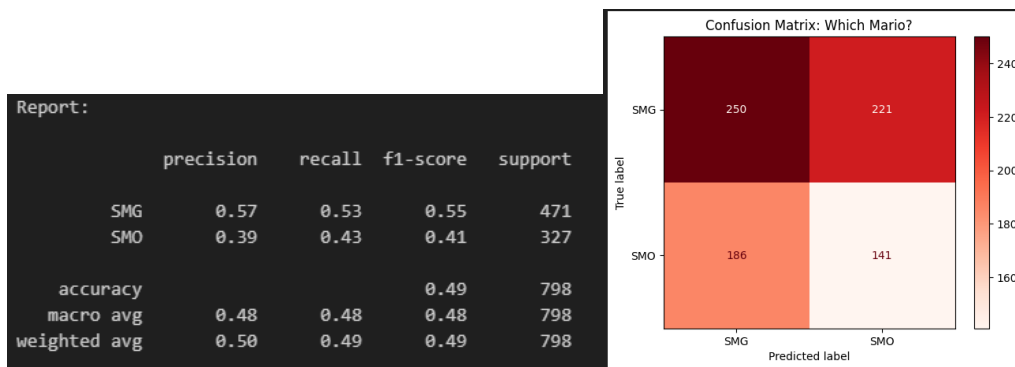
Gracias a que en los últimos 8 años mi percepción de lo que es una calificación aprobatoria a cambiado de 60 para arriba a 70 para arriba, quería mejorar más mi modelo. Esta vez, usé mi learning rate antiguo para entrenar el modelo, no usaría un optimizador preestablecido. Con un rate de  $2e-5$ , y 16 epochs para entrenar, puse a entrenar un último modelo, uno que lo llegó a tener un accuracy de 75%, un resultado con el que estuve más satisfecho. Las graficas de accuracy y loss también estaban niveladas, por lo que identifiqué que hacía fit.

```
Found 798 images belonging to 2 classes.
25/25 ————— 115s 4s/step - accuracy: 0.7761 - loss: 0.5199
test acc :
0.7459999918937683
```



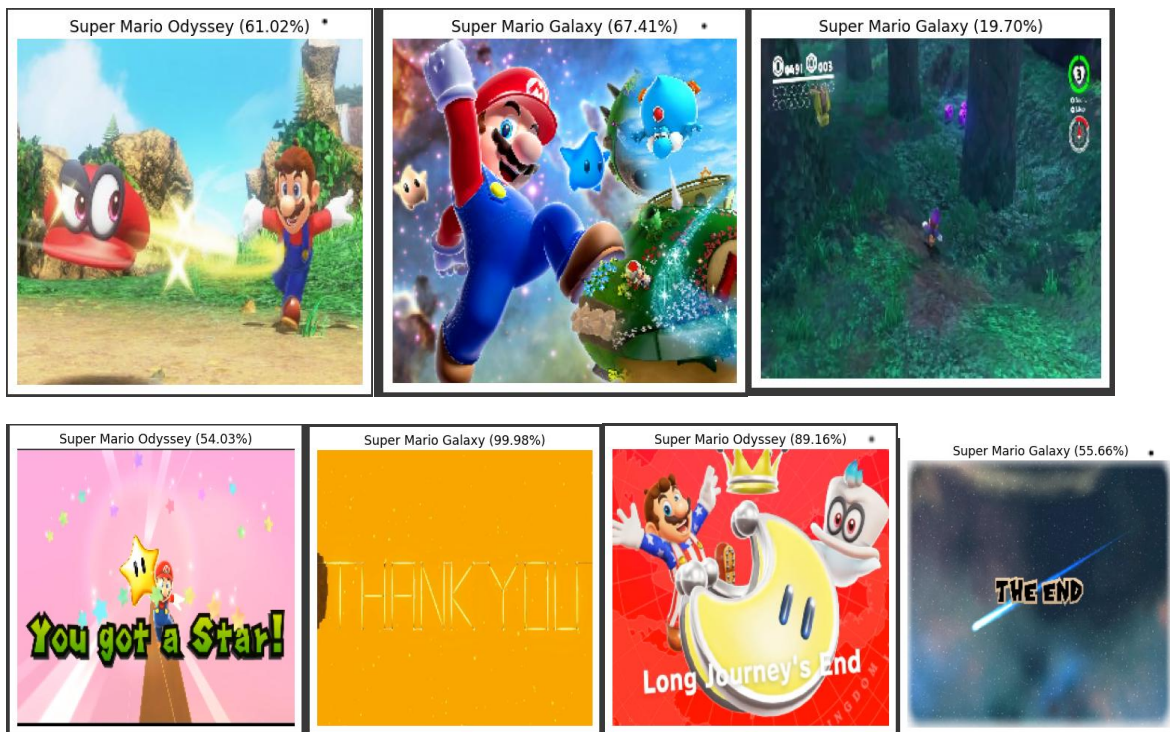


Fue aquí donde observé las consecuencias de tener un dataset desbalanceado en cuanto a la cantidad de datos por clase, el modelo tenía un sesgo a predecir que algo era de *Super Mario Galaxy* más que algo de *Super Mario Odyssey*. Pese a que las graficas muestran un Fit, sería más acertado que al final mi modelo hace Underfit debido a que, gracias al desbalance, hay un sesgo notable.



## - Predicciones

Satisfecho con un accuracy de 75%, decidí poner a prueba el modelo con imágenes que no estaban en el dataset. Me puse a ver unos gameplays de ambos juegos, tomando capturas de vez en cuando para evaluar el modelo. A veces es notable cuando se equivoca, pero al final creo que el modelo sabe lo que ve. A continuación, dejo toda prueba que hice para evaluar el modelo, muchas gracias por todo profesor. Hay un punto en las predicciones que son correctas al lado del porcentaje.





- **Referencia**

[1] J. A. AYENI, "Convolutional Neural Network (CNN): The architecture and applications," *Applied Journal of Physical Science*, vol. 4, no. 4, pp. 42–50, Dec. 2022, doi: <https://doi.org/10.31248/ajps2022.085>.