Jose Antonio López A01610805

Implementación de Clasificador de piezas de LEGO utilizando XCeption

Introducción

El objetivo de este proyecto es implementar una Red Neuronal Convolutiva (CNN, por sus siglas en inglés), utilizando el principio de “Transfer Learning” con la herramienta “XCeption” para la categorización de imágenes de piezas de LEGO.

Se utilizó un Dataset *[1]* de 6,414 elementos (imágenes .PNG) divididos en 16 categorías, que corresponden a 16 tipos diferentes de piezas LEGO. La separación de datos para el entrenamiento y prueba del modelo fue de 80%/20%, considerando que 1280 imágenes serán suficientes para validar el correcto funcionamiento del modelo, y aprovechando el resto para un entrenamiento robusto y completo. Dentro de este porcentaje de entrenamiento, se separó otro 20% para las pruebas de validación cruzada, que se llevan a cabo durante cada epoch del entrenamiento.

A continuación se describe el proceso por el que se pasó para llegar a la versión final del modelo, incluyendo una versión muy poco funcional del mismo.

Escalamiento de Imágenes

Antes del entrenamiento del modelo como tal, las imágenes fueron preprocesadas mediante diferentes técnicas de escalamiento. Luego de un análisis del dataset, observando el tipo de imágenes con que se contaba, se decidió utilizar las siguientes técnicas:

***Zoom*:** Como su nombre lo indica, es la función encargada de hacer zoom a las imágenes, de forma aleatoria dentro del rango establecido del 0 al 30%. Esto sirve para que el modelo no se quede con un solo tamaño de la pieza, y sea capaz de identificarlas sin importar que tan cercana o lejana ha sido tomada la imagen.
***width_shift_range*:** Se refiere al cambio en el rango de ancho de la imagen. En otras palabras, "mueve" la imagen hacia la izquierda o derecha, en este caso dentro de un rango del 0 al 20% de su tamaño hacia cada lado, para que el modelo sea capaz de identificar piezas de LEGO no solo centradas en la imagen, sino ubicadas en distintas zonas de la misma.
height_shift_range: Es la función encargada de "mover" la imagen, al igual que *width_shift_range*, pero de forma horizontal, en un rango del 0 al 20% tanto hacia arriba como hacia abajo. Su objetivo es muy similar, el de ubicar las piezas de LEGO en distintas áreas dentro de la matriz de la imagen, para que el modelo no busque objetos únicamente en el centro de la imagen.
Se decidió no utilizar otras funciones como la ***rotación***, o el ***flip horizontal***, debido a que el dataset se encuentra ya muy completo en cuanto a los ángulos en que fueron

tomadas las imágenes, por lo que dichas técnicas serían de poca utilidad al entrenar el modelo.

Es importante mencionar que el escalamiento de imágenes se lleva a cabo como parte del proceso de entrenamiento, con la misma RAM que se utiliza para correr la red. Las imágenes generadas en ningún momento se almacenan en el disco, sino que la función es utilizada únicamente una vez que se comienza a entrenar el modelo.

Por último, una vez que declaramos el batch de imágenes, tanto para entrenamiento como para validación, éstos pasan por un proceso de "**Shuffle**", para no cesgar el entrenamiento del modelo, ni desbalancear la cantidad de imágenes que recibe de cada categoría.

Red Neuronal Convolutiva

Versión 1

Se utilizó inicialmente como referencia la implementación propuesta por ***Alex Krizhevsky, Ilya Sutskever y Geoffrey E. Hinton*** en su paper **"ImageNet Classification with Deep Convolutional Neural Networks"** *[2]***,** y se adaptó a las capacidades de cómputo con que se disponía para la ejecución de este proyecto, así como las diferencias en dimensiones entre el dataset utilizado para su red *(1.2 millones de imágenes de alta resolución, divididas en 1,000 clases),* y el nuestro. A continuación se describe en detalle el diseño de la red inicialmente propuesta.

Se decidió implementar un ***modelo secuencial***, es decir, una red en la que la salida de una capa se convierte directamente en la entrada de la siguiente. El orden de las capas del modelo es el siguiente:

Input --> Conv2D --> Conv2D --> Conv2D --> Conv2D --> Conv2D --> Flatten --> Dense --> Dense --> --> Dense --> Output

- 5 capas convolutivas de 2 dimensiones, en las que el tamaño de los filtros se va reduciendo, pero la cantidad de filtros a aplicar va aumentando proporcionalmente, y se aplica la función de activación ReLU (Rectified Linear Unit) en las 5.
  La implementación que se utilizó de guía disminuye de igual forma el tamaño de los filtros a aplicar, pero a una escala bastante mayor *(aplicando 96 filtros de 11 x 11 x 3, y llegando hasta 256 filtros de 3 x 3 x 192)*. Como se mencionó anteriormente, en nuestro modelo se redujo el tamaño de dichos filtros, y la cantidad de los mismos, para una correcta adaptación al poder de cómputo con que se cuenta para el entrenamiento.

- Una capa “Flatten”, para “aplanar”, o convertir la salida de la última capa convolutiva 2D, en un vector de una sola dimensión. Hacemos esto para que la siguiente capa (densa) funcione correctamente.
- 1 capa densa de 128 neuronas. Esta capa es también conocida como “completamente conectada” ya que todas las neuronas se conectan entre ellas.
- otra capa densa, pero esta de 64 neuronas.
- Y por último, una capa densa con activación *softmax,* adecuada para la categorización de imágenes, y 16 neuronas, una para cada categoría del dataset con el que se está trabajando.

Sin embargo, la implementación de este modelo fue poco exitosa, ya que el porcentaje de precisión dentro del entrenamiento en ningún momento pasó el 35%, y al probarlo, no llegó siquiera al 10% de presición (se muestran los resultados más a detalle en "Testing y Métricas").

```
Epoch 1/30
80/80 [==============================] - 692s 9s/step - loss: 2.1565 - acc: 0.2269
Epoch 2/30
80/80 [==============================] - 734s 9s/step - loss: 2.0924 - acc: 0.2562
Epoch 3/30
80/80 [==============================] - 699s 9s/step - loss: 2.0374 - acc: 0.2674
Epoch 4/30
80/80 [==============================] - 702s 9s/step - loss: 1.9550 - acc: 0.2848
Epoch 5/30
80/80 [==============================] - 699s 9s/step - loss: 1.9113 - acc: 0.3012
Epoch 6/30
80/80 [==============================] - 704s 9s/step - loss: 1.8624 - acc: 0.3114
Epoch 7/30
80/80 [==============================] - 698s 9s/step - loss: 1.7889 - acc: 0.3402
Epoch 8/30
80/80 [==============================] - 704s 9s/step - loss: 1.8072 - acc: 0.3443
Epoch 9/30
80/80 [==============================] - 699s 9s/step - loss: 1.7609 - acc: 0.3416
Epoch 10/30
80/80 [==============================] - 697s 9s/step - loss: 1.7581 - acc: 0.3449
```

Versión 2 (Xception)

Una vez que se dio por deprecado el modelo inicialmente propuesto, se tomó la decisión de trabajar utilizando el principio de ***Transfer Learning***, el cual utiliza un modelo previamente entrenado como base previo al entrenamiento.

Basado en el paper "An Xception Based Convolutional Neural Network for Scene Image Classification with Transfer Learning" *[3]*, se utilizó como base el modelo de “imagenet”, uno de los datasets públicos más grandes y más utilizados para la categorización de imágenes. A este modelo se le agregaron varias capas, para hacerlo más completo y adaptarlo a la solución de nuestro problema. La estructura completa, quedó de la siguiente manera:

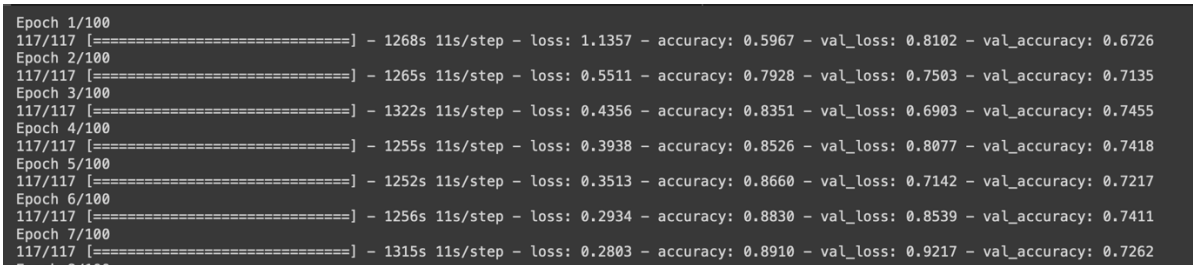
Base Model --> GlobalAvgPooling2D --> Dense --> Dense --> Dropout --> Output

- Una capa **densa**, de 1024 neuronas
- Otra capa **densa,** pero esta de 256 neuronas
- Una capa de **Dropout** para evitar el overfitting *(el sobre-entrenamiento del modelo, donde memoriza en lugar de aprender)*
- Y la capa **densa de salida** con 16 neuronas (una por cada categoría).

Además, se agregó un paso de **validación** *(cross-validation)* dentro del entrenamiento, para conocer los avances reales del entrenamiento incluso antes del proceso de pruebas.

El paper *[3]* sugiere que además se declaren como entrenables las capas del modelo base, sin embargo, por cuestiones de tiempo y falta de poder computacional, se decidió entrenar únicamente las capas agregadas que se mencionaron anteriormente.

El modelo anteriormente mencionado logró un 89.1% de accuracy en training y un 72.6% en validación

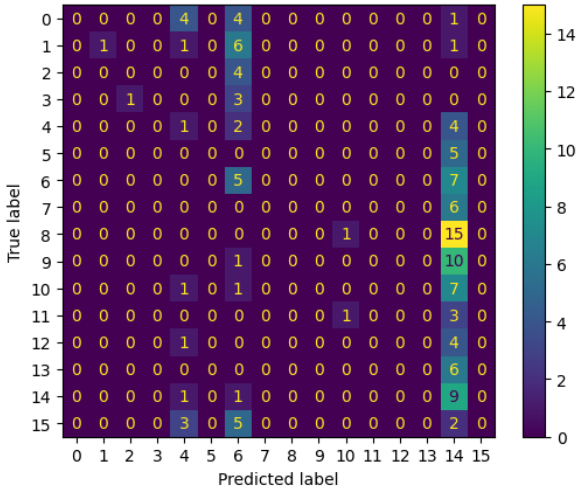


Testing y Métricas

Versión 1

Luego de haber entrenado el modelo hasta su estancamiento en el 34% de *accuracy* dentro del set de entrenamiento, se procedió a probar con el set de Testing, obteniendo un 9% de accuracy, apenas 3% por encima de lo que se obtendría si se categorizaran de forma aleatoria.

Haciendo un análisis más profundo, buscando el porqué de estos resultados, se obtuvieron datos interesantes, muy útiles para dar los siguientes pasos en el mejoramiento del modelo. Al construir una matriz de confusión con los datos obtenidos en las pruebas, se observó que la gran mayoría de las predicciones que hace el modelo, son de las categorías 14, 6 y 4. Es decir, está "confundiendo" varias categorías de piezas y simplificando su clasificación casi en su totalidad, a estas 3 categorías.



Se podría decir que es un problema de *overfitting*, interpretando que el modelo "memorizó" ciertos patrones. Sin embargo, al tener un porcentaje tan bajo de accuracy en el entrenamiento, es muy poco probable que este sea el caso.

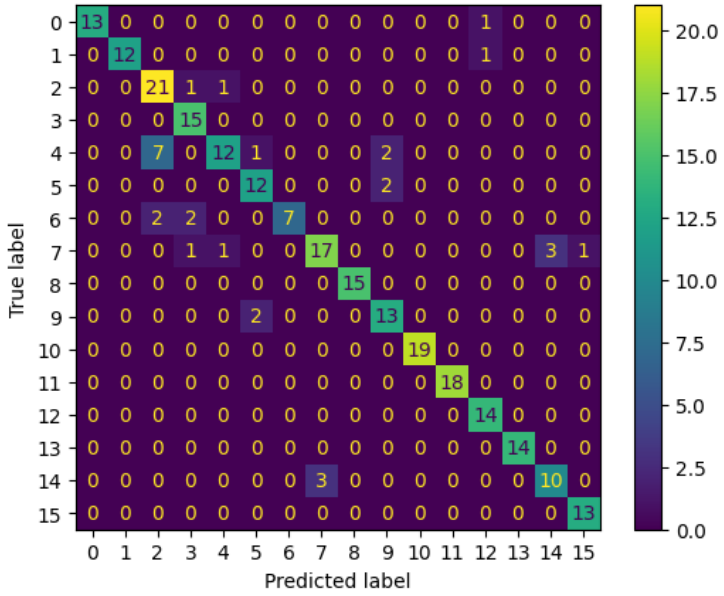
Versión 2

Luego de haber entrenado el modelo de Xception hasta el 88% de *accuracy* en training (valor muy similar al que se presenta en el paper *[3]*), se corrieron pruebas que resultaron en un 87.5% de *accuracy* que, si bien es un modelo aún con posibles mejorasno es un modelo

```
test acc :
0.875
```

perfecto, logró una mejora de casi 1,000% (o 10x) con respecto a la primer versión presentada.

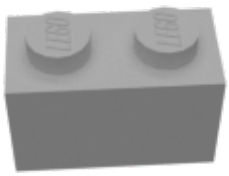
A continuación se muestra la matriz de confusión de dichas pruebas:



Podemos observar que, en su mayoría, el modelo está acertando al pronosticar la categoría a la que pertenece una pieza de LEGO.

Destaca el error de la categoría 4, habiendo “confundido” 7 veces esta categoría con la #2. Esto llega a hacer más sentido si observamos una imagen de ambas piezas:

Categoría #2



Categoría #4

Si queremos un análisis más profundo de los datos obtenidos, a continuación se muestran los resultados de precisión, recall, y f1 por categoría:

	precision	recall	f1-score	support
11214	1.00	0.93	0.96	14
18651	1.00	0.92	0.96	13
2357	0.70	0.91	0.79	23
3003	0.79	1.00	0.88	15
3004	0.86	0.55	0.67	22
3005	0.80	0.86	0.83	14
3022	1.00	0.64	0.78	11
3023	0.85	0.74	0.79	23
3024	1.00	1.00	1.00	15
3040	0.76	0.87	0.81	15
3069	1.00	1.00	1.00	19
32123	1.00	1.00	1.00	18
3673	0.88	1.00	0.93	14
3713	1.00	1.00	1.00	14
3794	0.77	0.77	0.77	13
6632	0.93	1.00	0.96	13
accuracy			0.88	256
macro avg	0.90	0.89	0.88	256
weighted avg	0.89	0.88	0.88	256

Conclusión

Luego de haber entrenado y probado ambos modelos, podemos llegar a la conclusión de que la implementación de una CNN pre-entrenada fue la solución más adecuada para el problema planteado.

Referencias

***[1]* - Dataset -** [Joost Hazelzet]. ([2021]). [Images of LEGO Bricks], [Version 4]. Retrieved [May 15th] from [https://www.kaggle.com/datasets/joosthazelzet/lego-brick-images].

***[2]* - Paper de referencia –** A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, 2012, pp. 1097-1105.

***[3]* - Paper Xception -** X. Wu, R. Liu, H. Yang and Z. Chen, "An Xception Based Convolutional Neural Network for Scene Image Classification with Transfer Learning," 2020 2nd International Conference on Information Technology and Computer Application (ITCA), Guangzhou, China, 2020, pp. 262-267.

Link a Google Drive del Dataset:
https://drive.google.com/drive/folders/1N3XUCOKy_GJDuUURXu-Upi_2kAdC32ds?usp=sharing

Link a Notebook del primer modelo:
https://colab.research.google.com/drive/1Ot38XblgfjPriJ4eujBNpSJjBmvzH5Om?usp=sharing

Link a Notebook del modelo con Xception:
https://colab.research.google.com/drive/1ThbQHjBURHzVIor1GoqaGVfs58MS9pbm?usp=sharing