INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

Inteligencia artificial avanzada para la ciencia de datos II (Gpo 502)

Modulo 2: Deep Learning

“Entregable: Implementación de un modelo de deep learning ”

Jose Pablo Cobos Austria **A01274631**

Profesor:
Dr. Benjamín Valdés Aguirre

Santiago de Querétaro, 25 de noviembre del 2022

Introducción

Descripción del problema

Antes de empezar a generar nuestros modelo de redes neuronales, es necesario comprender correctamente cual es el problema o situación al cual tratamos de dar solución con nuestra propuesta.

“La navidad es una temporada donde la mayoría de los niños reciben regalos, siendo la mayoría juguetes, que van desde muñecas, figuras de acción, bicicletas, etc, y entre todos ellos encontramos los LEGO, que son bloques con los que los niños pueden armar cualquier que se imaginen. Por esa razón lo que se quiere lograr es que se implemente un modelo de reconocimiento de imágenes usando deep learning para identificar qué piezas tienen.“

Desarrollo

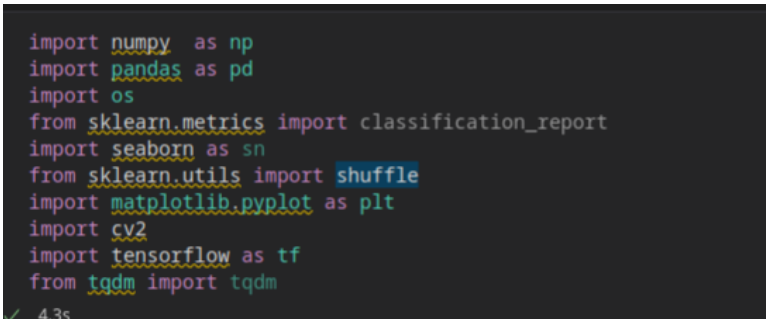
Informacion del Dataset

El dataset con el que se va a trabajar se obtuvo de la página de Kaggle, teniendo originalmente un total de más de 200 piezas de LEGO diferentes, con un total de 4000 imágenes, no obstante por practicidad y utilidad, se recortó el dataset a simplemente 6 piezas de LEGO y cada una con 1500 imagenes para entrenar y 100 de prueba.

Además, para el desarrollo de la actividad se hizo uso de librerías de Tensorflow con Keras para generar las capas de nuestro modelo de red neuronal.

Código

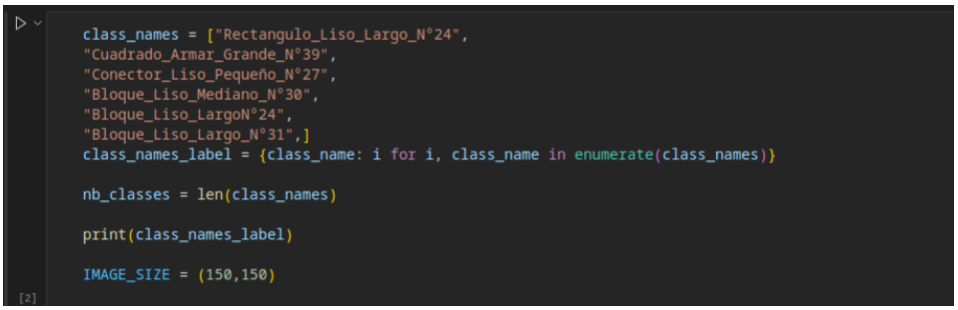
Lo primero que hacemos importar las librerías que utilizamos en el desarrollo de nuestro modelo, con un gran énfasis en la librería de numpy, cv2 y tensorflow, ya que estas tres son la que mayor importancia tienen para la actividad ya que nos ayudarán a analizar las imágenes, realizarles ajustes, pasar a arreglos, manipular estos mismos y finalmente para poder generar nuestro modelo de red neuronal



```
import numpy  as np
import pandas as pd
import os
from sklearn.metrics import classification_report
import seaborn as sn
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from tqdm import tqdm
```

Imagen 1. *Importar librerías*

Después de eso, lo que realizamos fue primero crear un arreglo con los nombres de cada uno de los bloques que habíamos seleccionado como categoría, después de eso fueron enumerados y finalmente imprimimos cada categoría con su debida etiqueta.



```
class_names = ["Rectangulo_Liso_Largo_N°24",
"Cuadrado_Armar_Grande_N°39",
"Conector_Liso_Pequeño_N°27",
"Bloque_Liso_Mediano_N°30",
"Bloque_Liso_LargoN°24",
"Bloque_Liso_Largo_N°31",]
class_names_label = {class_name: i for i, class_name in enumerate(class_names)}

nb_classes = len(class_names)

print(class_names_label)

IMAGE_SIZE = (150,150)
```

Imagen 2. *Enumerar categorías*

Para la carga de los datos, lo que se realizó fue una función, donde lo primero que hacía era cargar la dirección donde se encontraba la carpeta de los datos de entrenamiento y los datos de prueba, y después en un ciclo lo que se iba haciendo era entrar a cada uno de las carpetas anteriormente mencionadas e ir entrando a las subcarpetas de las categorías que

tenían las imágenes, y estas mismas eran leías y se formateaban, se les cambiaba de color y el tamaño con el fin de qué todos tuvieran el mismo estándar, finalmente esto se agregaban a un arreglo donde estaba la imagen en formato de arreglo de imagenes e etiquetas . Al terminar la función lo que te regresaba era un arreglo que tenía tanto las imágenes en forma de arreglo, como sus etiquetas.

```
def load_data():
    DIRECTORY = r"/home/josecobos/Documentos/Escuela/IA_Parte2/Modulo_2/data/data2"
    CATEGORY = ["seg_train","seg_test"]
    output = []

    for category in CATEGORY:
        path = os.path.join(DIRECTORY, category)
        #print(path)
        images = []
        labels = []

        print("Loading {}".format(category))

        #print(os.listdir(path))

        for folder in os.listdir(path):
            print(folder)

            label = class_names_label[folder]
```

Imagen 3. *Enumerar categorías*

Ahora para la parte de análisis de nuestros datos, lo primero que hacemos con las imágenes y las etiquetas entrenamiento lo que hacemos es mezclarlas, haciendo un shuffle. Y posteriormente imprimimos cuantas muestras había de entrenamiento y cuánto sería de prueba además de qué tamaño tenía cada una de las imágenes.

```
train_images,train_labels = shuffle(train_images,train_labels, random_state=25)


n_train = train_labels.shape[0]
n_test = test_labels.shape[0]

print ("Number of training examples: {}".format(n_train))
print ("Number of testing examples: {}".format(n_test))
print ("Each image is of size: {}".format(IMAGE_SIZE))
```

```
Number of training examples: 9000
Number of testing examples: 600
Each image is of size: (150, 150)
```

Imagen 4. *Análisis de datos*

Aparte de eso gráfica vamos de forma breve un poco cómo es que está la distribución de nuestros samples, y podemos observar que en efecto las imágenes de entrenamiento son de 1500 y las imágenes de prueba son solamente 100 por cada una de las categorías.

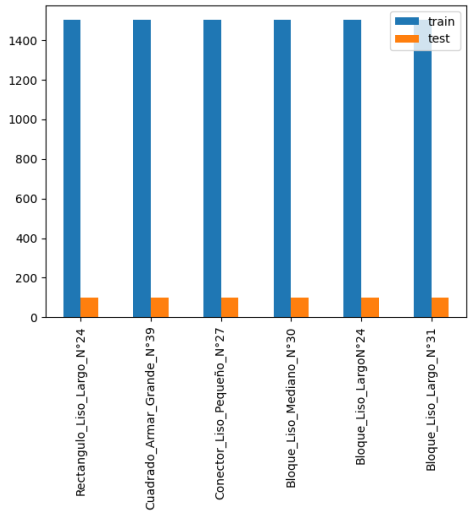


Imagen 7. *Distribución de los datos de entrenamiento y prueba*

Finalizado eso, lo siguiente que se realizó fue otra función donde se imprimieron algunas de las muestras que teníamos para nuestro entrenamiento con su etiqueta para poder observar que de verdad todo estaba en orden antes de entrenar nuestro modelo.

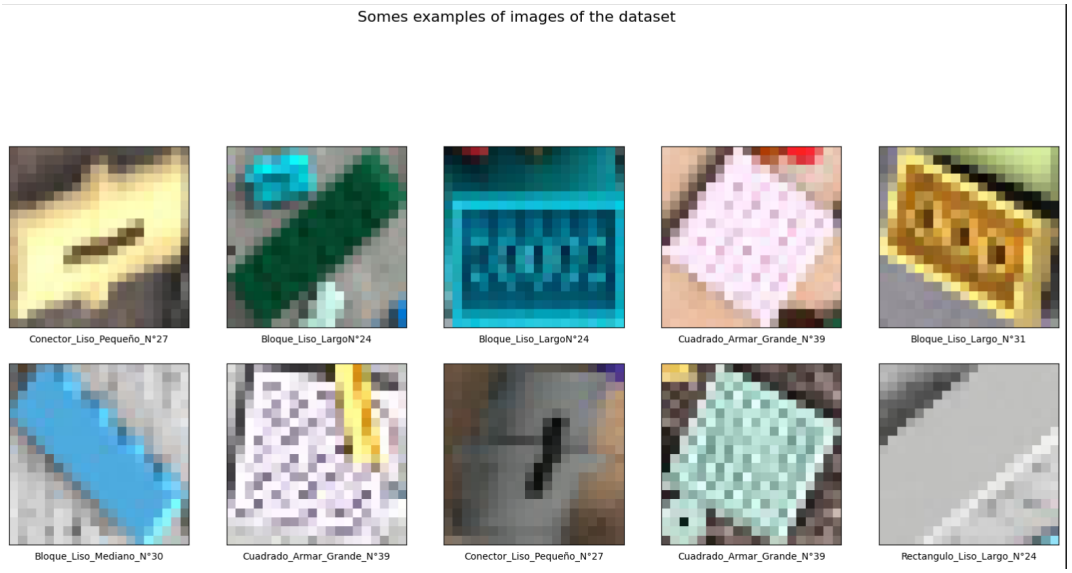


Imagen 6. *Ejemplos de las imágenes del dataset*

Modelo 1

En la sección de generación de modelo, el primer modelo que se propuso fue un modelo donde se tenían unas cinco capas: la primera de ellas era de Conv2D para dividir en pequeñas capas de 2D, luego tenemos una capa de Maxpooling que realiza la discretización de la muestra , pasamos una capa flaaten donde aplanaban las capas a 1D, y finalizamos con dos capas Dense, una de ellas con una función de activación relu y otra soft as para conectar las capas.

Modelo 1

```
model1 = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32,(3,3),activation= 'relu',input_shape = (150,150,3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128,activation=tf.nn.relu),
    tf.keras.layers.Dense(6,activation=tf.nn.softmax)

])
```

Imagen 7. *Configuración del Modelo 1*

Además fue aplicado *sparse_categorical_crossentropy* como parámetro de Loss, ya que tenemos más de dos categorías como posibles resultados. Por otro lado, también recordamos que fue aplicada la función Adam para optimizar el modelo debido a que este realiza un ajuste automático del learning rate. Y para finalizar, como ajustes finales será entrenado 10 épocas y tendrá un batch_size de 256.

```
model1.compile(optimizer= 'adam', loss = 'sparse_categorical_crossentropy',metrics=['accuracy
```

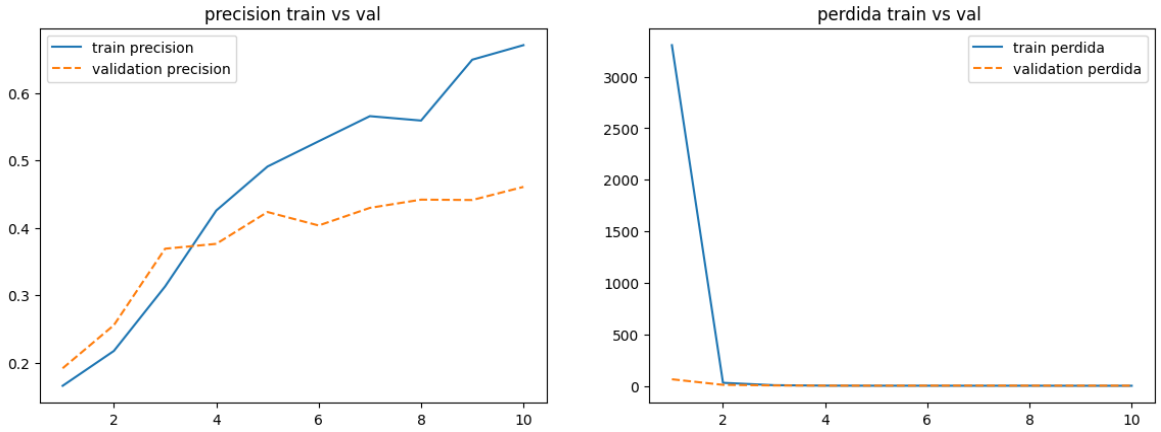
Python

```
history = model1.fit(train_images,train_labels,batch_size=256,epochs=4,validation_split= .2)
```

Python

Imagen 8 . *Compilación y Entrenamiento del modelo*

Finalizado el entrenamiento de nuestro modelo, nos podemos dar cuenta que comparando que el modelo al entrenar nos presenta una precisión muy lejos de estar bien, siendo del **67%** mientras que el caso es todavía peor al momento de validarlo, siendo que ni siquiera llega a la mitad, teniendo una precisión de **46%**. Además la pérdida en ambos casos sigue siendo bastante grande, incluso el modelo al momento de validar , duplica la pérdida que tenía al momento de entrenar. Por lo tanto, todavía tenemos una gran área de oportunidad para mejorar.



Imágenes 9 y 10 . *Gráficas de Comparación de Resultados*

Modelo 2

Para el nuevo modelo los cambios que se realizaron fueron los siguientes:

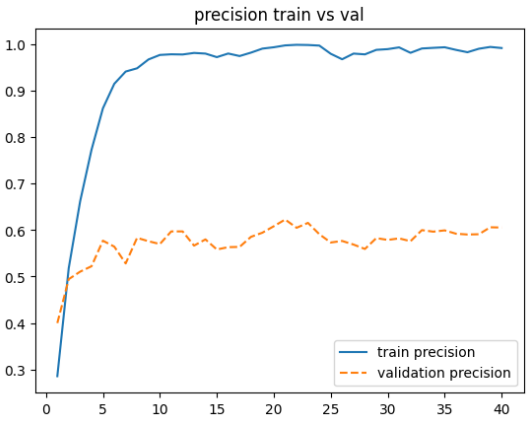
- Se agregaron dos capas, tanto una Conv2D y una capa MaxPooling2D
- Cambiar el batch_size a 128 y conservamos la misma cantidad de épocas

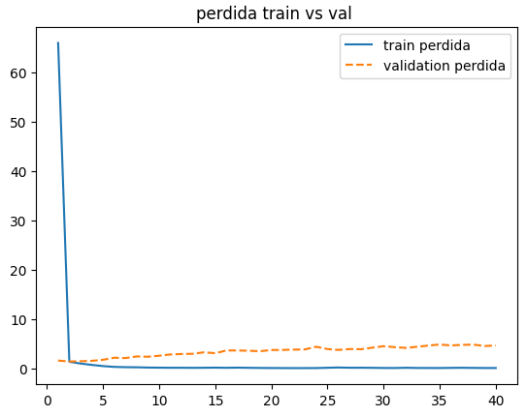
Y los resultados que obtuvimos con este cambio tampoco fueron tan prometedores como se esperaba, ya que aunque mejoró bastante la precisión del modelo al ser entrenado, dando un porcentaje del 97%, al momento de validar la diferencia sigue sin ser mucha dando una precisión del 57%

Modelo 3

Para el nuevo modelo los cambios que se realizaron fueron los siguientes:

- Cambiar el batch_size a 64 y aumentamos a 40 épocas

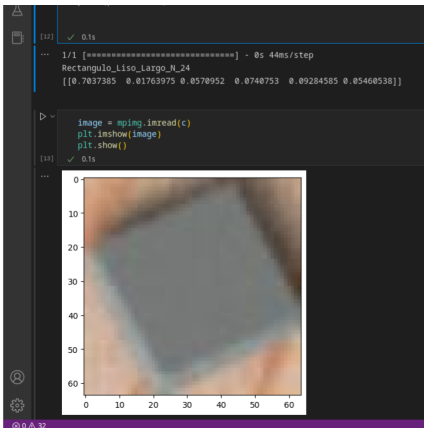




Imágenes 11 y 12 . *Gráficas de Comparación de Resultados Modelo 3*

Lamentablemente como pudimos observar, es que realmente no hubo una gran mejoría en comparación de los demás modelos, ya que llegaba solamente a un máximo 65% de precisión. No obstante esto no significa que el modelo sea malo, ya que tras haber realizado la actividad me di cuenta de que existían varios factores que podrían afectar esta precisión. La primera de ellas es una de las más importantes, siendo la cantidad de salidas que tiene el modelo, ya que si lo comparamos con uno de decisión binaria, este es más sencillo porque solamente tiene que elegir entre dos casos, no obstante para este escenario se tuvo que elegir un total de 6 , por lo que complicaba su evaluación. Por otra parte, las similitud que tienen las piezas de LEGO entre sí es otro problema ya que es fácil entender porqué el modelo puede llegar a confundir al modelo y generar fallos en la precisión.

Resultados



Imágenes 13. *Clasificación de una imagen final*

Finalmente me gustaría agregar que aunque el desarrollo de este entregable no haya resultado el que se esperaba de manera inicial, este mismo hecho no hace más que dejarnos más áreas de oportunidad para poder mejorar el modelo, siendo una de estas generar un modelo de red neuronal más complejo, con mejores hiperparametros, y compilación. No obstante, el limitante lamentablemente en la mayoría de los casos es el hardware con el que se cuenta que es un cuello de botella muy importante y que en varias ocasiones es lo que evita que las mejoras. Para terminar, en verdad se me hizo muy interesante toda la parte de deep learning porque siento que aunque la curva de aprendizaje no es tan amigable al inicio, cuando ya comprendes que es lo que está sucediendo te imaginas la cantidad de situaciones en la que lo podrías aplicar

Referencias

- s/f. (s/a). *B200C LEGO Classify Conv2D.* Kaggle. Recuperado el 25/11/22 de: https://www.kaggle.com/code/stpeteishii/b200c-lego-classify-conv2d