Momento de Retroalimentación: Módulo 2 Implementación de un modelo de deep learning. (Portafolio Implementación)

José Ángel García López

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Querétaro

Matrícula
A01275108

Diversidad en un mundo globalizado

Benjamín Valdés Aguirre

Santiago de Querétaro, 05 de Noviembre del 2023

A01275108@tec.mx

Introducción

En este portafolio de implementación, abordaremos un emocionante proyecto que se centra en el popular juego "League of Legends". Hemos creado un modelo de deep learning utilizando un conjunto de datos personalizado que comprende 60 campeones del juego. Nuestro objetivo es utilizar este modelo para analizar y predecir el rendimiento de los campeones en el juego.

Data set:

El dataset que se usó para entrenar esta red neuronal fue creado por mi ya que la mayoria de las plataformas de datos de uso libre no tenían el enfoque que yo le quería dar a mi portafolio, por lo que realice una grabación de aproximadamente de 8 a 10 segundos para 60 campeones dentro de la grieta del invocador y mediante el uso de OpenCv dividí cada uno de los videos en frames para tener aproximadamente de 300 a 400 imágenes por camperon/clase, el script para hacer esto se encuentra en este mismo repositorio con el nombre de “nex_file.py” el cual, toma los video de un directorio, los separa en frames y crea una carpeta con el nombre de los archivos.

Modelos obtenidos:

https://drive.google.com/drive/folders/1a3zud4_6eps3YEg2brmeZoJE83kZaerA?usp=sharing

Data Set:

https://drive.google.com/drive/folders/17pW6CfvD8nuoQSSb8pTyxiQMdgxyWJPh?usp=sharing

Código:

El código proporcionado es un cuaderno de Python desarrollado en el entorno de Google Colab (Colaboratory) con el propósito de entrenar un modelo de clasificación de imágenes de campeones del popular juego League of Legends utilizando TensorFlow y Keras. El proceso comienza por montar Google Drive en el entorno de Colab mediante la biblioteca 'google.colab', lo que facilita el acceso a datos y permite guardar el modelo entrenado de manera conveniente en Google Drive.

A continuación, se establece el directorio de trabajo para acceder a las imágenes de los campeones en una ubicación específica dentro de Google Drive, lo cual es esencial para cargar los datos de entrenamiento. Luego, se importan diversas bibliotecas y módulos de TensorFlow, incluyendo componentes para el procesamiento de imágenes y el modelo preentrenado ResNet50, que son cruciales para construir y entrenar el modelo de clasificación.

Se definen parámetros clave para el proceso de entrenamiento, como el tamaño del lote, el tamaño de las imágenes, el número de épocas de entrenamiento y el número de clases, que debe ajustarse al número real de campeones en el juego. Posteriormente, se crea un generador de datos utilizando la función 'ImageDataGenerator' de TensorFlow para preprocesar y cargar las imágenes de entrenamiento y validación, normalizando los valores de píxeles y dividiendo los datos en conjuntos de entrenamiento y validación.

Los conjuntos de datos de entrenamiento y validación se generan utilizando la función 'flow_from_directory', la cual organiza las imágenes en carpetas de acuerdo con sus clases y etiquetas. Luego, se procede a construir un modelo de clasificación de imágenes utilizando ResNet50 preentrenado como modelo base, agregando capas adicionales para adaptar el modelo a la tarea de clasificación de campeones de League of Legends.

Una vez construido el modelo, las capas del modelo base (ResNet) se congelan para evitar que se modifiquen durante el proceso de ajuste. El modelo se compila con un optimizador 'adam' y la función de pérdida 'categorical_crossentropy', además de definir métricas de evaluación, como la precisión.

La fase de entrenamiento del modelo utiliza los generadores de datos de entrenamiento y validación, y el progreso del entrenamiento se registra en la variable 'history'. El modelo entrenado se guarda en un archivo llamado 'lol_champion_classifier4 03/11/23.h5' para su posterior uso.

La biblioteca Matplotlib se emplea para crear gráficos que muestran la precisión y la pérdida del modelo durante el entrenamiento. Asimismo, el código evalúa el modelo en el conjunto de validación, generando una matriz de confusión y un informe de clasificación, lo que proporciona información detallada sobre el rendimiento del modelo.

Finalmente, se realiza una prueba del modelo cargando una imagen de prueba y realizando una predicción para determinar a qué campeón de League of Legends corresponde la imagen. El código culmina mostrando una lista de archivos en el directorio actual, lo cual puede ser útil para verificar la existencia de archivos generados durante el proceso.

Resultados:

El primer modelo lamentablemente no pude recuperar su matriz de confusión sin embargo para el modelo 2 y 3 los resultados fueron los siguientes:

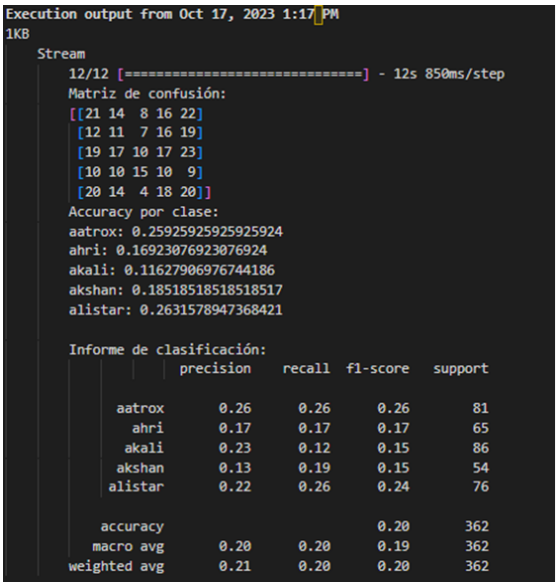
```
Execution output from Oct 17, 2023 1:17 PM
1KB
    Stream
        12/12 [==============================] - 12s 850ms/step
        Matriz de confusión:
        [[21 14  8 16 22]
         [12 11  7 16 19]
         [19 17 10 17 23]
         [10 10 15 10  9]
         [20 14  4 18 20]]
        Accuracy por clase:
        aatrox: 0.25925925925925924
        ahri: 0.16923076923076924
        akali: 0.11627906976744186
        akshan: 0.18518518518518517
        alistar: 0.2631578947368421

        Informe de clasificación:
                      precision    recall  f1-score   support

              aatrox       0.26      0.26      0.26        81
                ahri       0.17      0.17      0.17        65
               akali       0.23      0.12      0.15        86
              akshan       0.13      0.19      0.15        54
             alistar       0.22      0.26      0.24        76

            accuracy                           0.20       362
           macro avg       0.20      0.20      0.19       362
        weighted avg       0.21      0.20      0.20       362
```

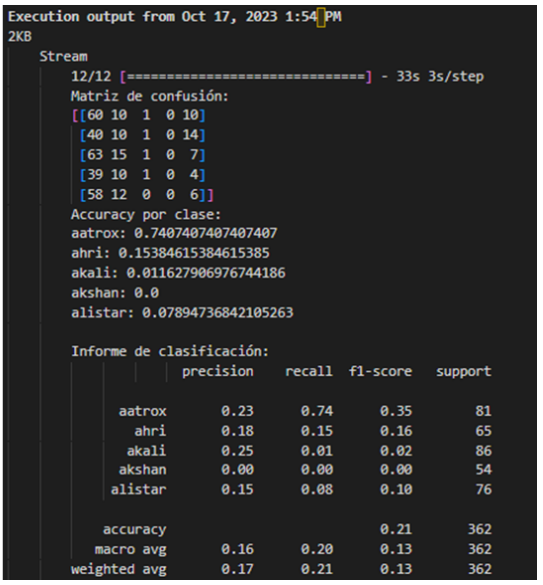
10 epocas

10 epocas + 5 clases + batch size = 32

En este caso podemos ver el desempeño de nuestro modelo bajos diferentes configuraciones, la primera de ellas es una que en resumen, es una CNN básica con tres capas convolucionales, capas de MaxPooling para reducir la dimensionalidad y dos capas completamente conectadas la cual como se puede apreciar, tiene resultados mediocres en cuanto a la identificación de las clases ya que podríamos decir que esta adivinado tras usar solo 10 épocas.

Para el segundo caso, se carga un modelo ResNet50 preentrenado con pesos de "ImageNet" para la extracción de características visuales y se agrega una capa de "Global Average Pooling 2D" para reducir la dimensionalidad de las características, como se puede observar da valores muy buenos en la clase “Aatrox” sin embargo falla completamente en la de “Akshan” lo que puede deberse a que hice un mal uso del framework al configurar las capas a utilizar

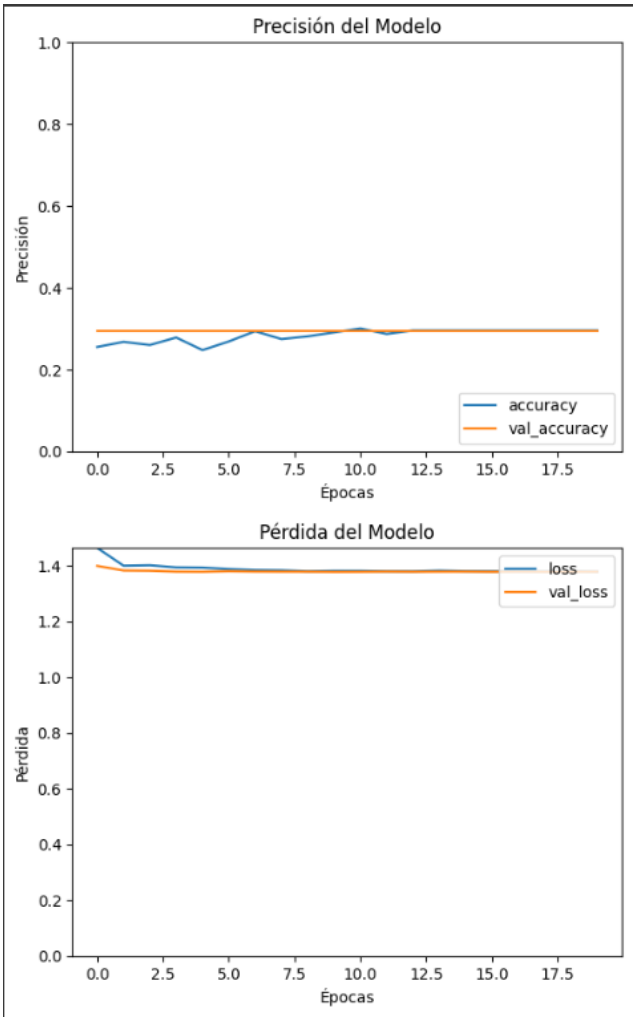




Ultimo modelo:

El modelo es una red neuronal convolucional (CNN) que utiliza la arquitectura ResNet50 como base. Se agrega una capa de agrupación global promedio para reducir la dimensionalidad de las características extraídas por ResNet50. Luego, se siguen dos capas densas con 256 y 128 unidades respectivamente, ambas con funciones de activación ReLU, y se introduce una capa de regularización de abandono con una tasa del 50% para evitar el sobreajuste. La capa de salida consta de un número de unidades igual al número de clases en el problema de clasificación, utilizando una función de activación softmax para la clasificación múltiple. Las capas de ResNet50 se mantienen congeladas para transferir el conocimiento aprendido en la base de datos "imagenet". El modelo se compila con el optimizador 'adam', utiliza la función de pérdida 'categorical_crossentropy' para problemas de clasificación y mide la precisión como métrica de evaluación. En resumen, este modelo es un clasificador de imágenes que utiliza una red neuronal preentrenada (ResNet50) como base y se ajusta para clasificar imágenes en varias clases, con regularización de abandono para evitar el sobreajuste.

Al correrlo, los resultados fueron sumamente mediocres ya que se estancó en valores muy bajo los cuales no resultaron para nada, atribuyó este error a mi poca experiencia usando ResNet.




```
9/9 [==============================] - 27s 3s/step
Matriz de confusión:
[[76  0  0  0]
 [59  0  0  0]
 [55  0  0  0]
 [68  0  0  0]]
Accuracy por clase:
alistar: 1.0
drmundo: 0.0
garen: 0.0
jhin: 0.0

Informe de clasificación:
              precision    recall  f1-score   support

     alistar       0.29      1.00      0.46        76
     drmundo       0.00      0.00      0.00        59
       garen       0.00      0.00      0.00        55
        jhin       0.00      0.00      0.00        68

    accuracy                           0.29       258
   macro avg       0.07      0.25      0.11       258
weighted avg       0.09      0.29      0.13       258
```

Modelos finales + mejoras: 4_LoL_Champ_Cl

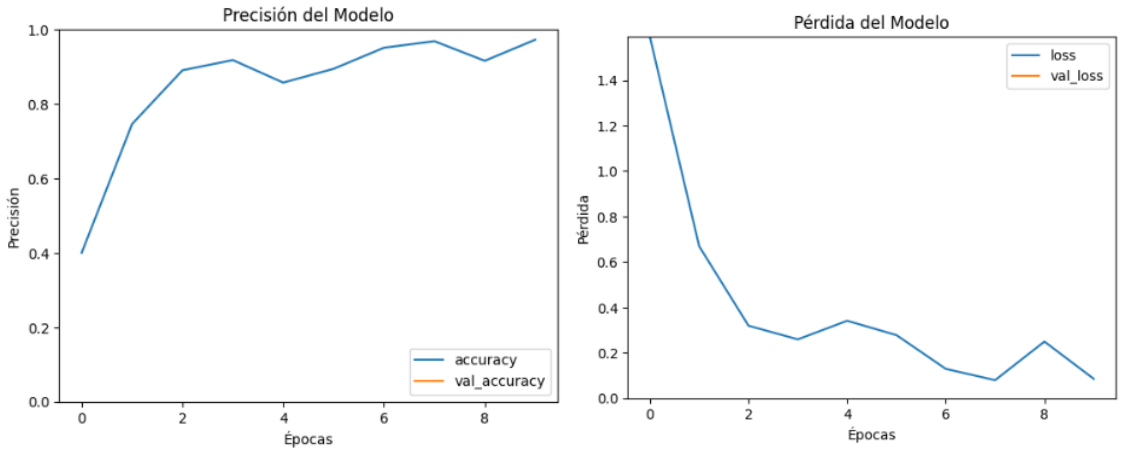
Al analizar las mejoras respecto a modelos anteriores, se evidencia que surgen debido a dificultades en la definición de los modelos iniciales y los datos utilizados para su entrenamiento. Tras revisar los tres modelos anteriores, se observa que la capacidad de reconocimiento del campeón no es tan precisa, posiblemente debido a la escasez de datos en los modelos. Este hecho se reflejó en mi último modelo, el cual solo lograba reconocer la clase "Alistar". Por consiguiente, en esta nueva fase, se reestructuró la estrategia de información y se seleccionaron cuatro campeones con dimensiones mayores y colores más contrastantes respecto al diseño del mapa.

En esta última iteración, se ha optado por modificar la manera en que se distribuye la información tomando la distribución de 70-20-10, así como la selección de los campeones a utilizar, buscando mejorar la capacidad de reconocimiento del modelo. La elección de campeones más grandes y con colores más contrastantes pretende mejorar la precisión del modelo en la identificación de diferentes clases en el contexto del juego.

Para el primer modelo, opte por usar otro modelo el cuál fue VGG, en resumen esta red neuronal convolucional, inicializada con pesos de ImageNet, es adaptada para un nuevo propósito. Tras cargar la VGG16 sin su capa superior y establecer la forma de entrada como imágenes de 220x220 píxeles con tres canales de color, se añaden capas personalizadas. Estas capas incluyen una capa de aplanamiento para transformar los mapas de características tridimensionales en un vector unidimensional. Además, se incorporan dos capas densas, una con 255 neuronas y otra con 512, ambas activadas por la función ReLU, para aprender representaciones más complejas. La capa de salida, compuesta por 4 neuronas activadas mediante softmax, se encarga de la clasificación multiclase. La configuración final se completa con la inmovilización de las capas pre-entrenadas de VGG16 para evitar su modificación durante el entrenamiento del nuevo modelo.

Estos fueron los resultados del modelo los cuales son prometedores ya que finalizó con un 95% de precisión:

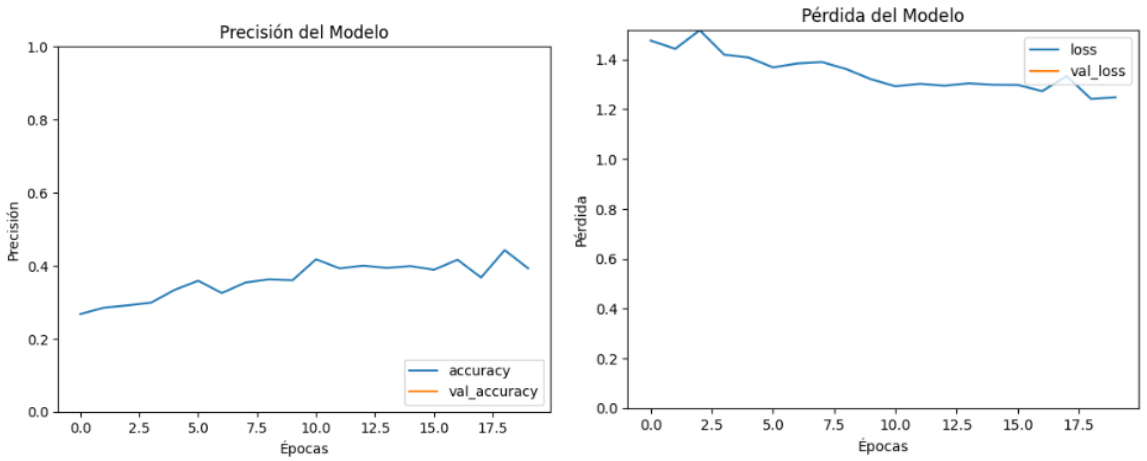
Durante cada época, se procesan 32 lotes de datos (steps_per_epoch), y para la validación del modelo, se utilizan datos de un generador de validación, realizando 25 pasos de validación por época (validation_steps).



Proceso de entrenamiento: optimizers.Adam(learning_rate=0.001) y 10 epocas

Para el segundo modelo retome la arquitectura de ResNet50 y se carga la red ResNet50 pre-entrenada con pesos obtenidos de ImageNet y se omite su capa superior (include_top=False), definiendo el tamaño de entrada como imágenes de 220x220 píxeles con 3 canales de color (RGB). La estructura del modelo resultante se compone de capas convolucionales y de pooling de ResNet50, diseñadas para extraer características complejas de las imágenes. Luego, se agrega un conjunto de capas personalizadas para adaptar la red a un nuevo conjunto de datos: una capa de aplanamiento (Flatten()) para convertir los mapas de características en un vector unidimensional, seguida por una capa densa (Dense()) con 4 neuronas y activación softmax, encargada de clasificar las imágenes en 4 clases distintas. Además, se congela (resnet_model.trainable = False) las capas pre-entrenadas de ResNet50 para evitar su reentrenamiento durante el proceso de adaptación al nuevo conjunto de datos.

Como se puede observar nuevamente tiene problemas para aprender ya que la cantidad de épocas es incluso superior al modelo anterior y aunque presenta una tendencia creciente, parece que tiene a sobrepasar el 50% de precisión:

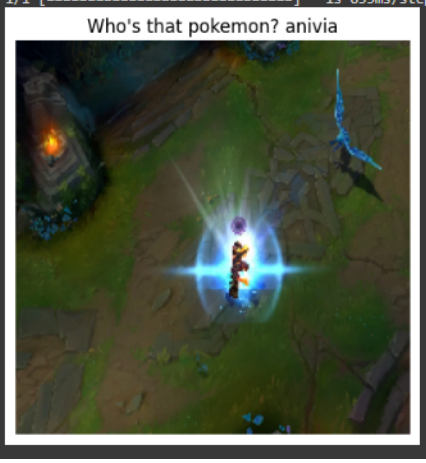


optimizers.Adam(learning_rate=0.0001), epochs=20, steps_per_epoch = 50

Resultados:

```
image = load_img('/content/ability_0034_E1.jpg', target_size=(220, 220))

img = np.array(image)
img = img / 255.0
img = img.reshape(1,220,220,3)

label = best_model.predict(img)

index = np.argmax(label)
plt.imshow(image)
plt.axis("off")

plt.title("El campeon detectado es: %s" %(labels[index]));
```

```
1/1 [==============================] - 1s 833ms/step
```




```
image = load_img('/content/ability_0086_Q1.jpg', target_size=(220, 220))

img = np.array(image)
img = img / 255.0
img = img.reshape(1,220,220,3)

label = best_model.predict(img)

index = np.argmax(label)
plt.imshow(image)
plt.axis("off")

plt.title("El campeon detectado es?  %s" %(labels[index]));
```

```
1/1 [==============================] - 1s 725ms/step
```