

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 mayoría 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 campeon/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

```
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         362
weighted avg           0.21      0.20      0.20         362
```

```
Execution output from Oct 17, 2023 1:54 PM
2KB
Stream
12/12 [=====] - 33s 3s/step
Matriz de confusión:
[[60 10  1  0 10]
 [40 10  1  0 14]
 [63 15  1  0  7]
 [39 10  1  0  4]
 [58 12  0  0  6]]
Accuracy por clase:
aatrox: 0.7407407407407407
ahri: 0.15384615384615385
akali: 0.011627906976744186
akshan: 0.0
alistar: 0.07894736842105263

Informe de clasificación:
      precision    recall  f1-score   support

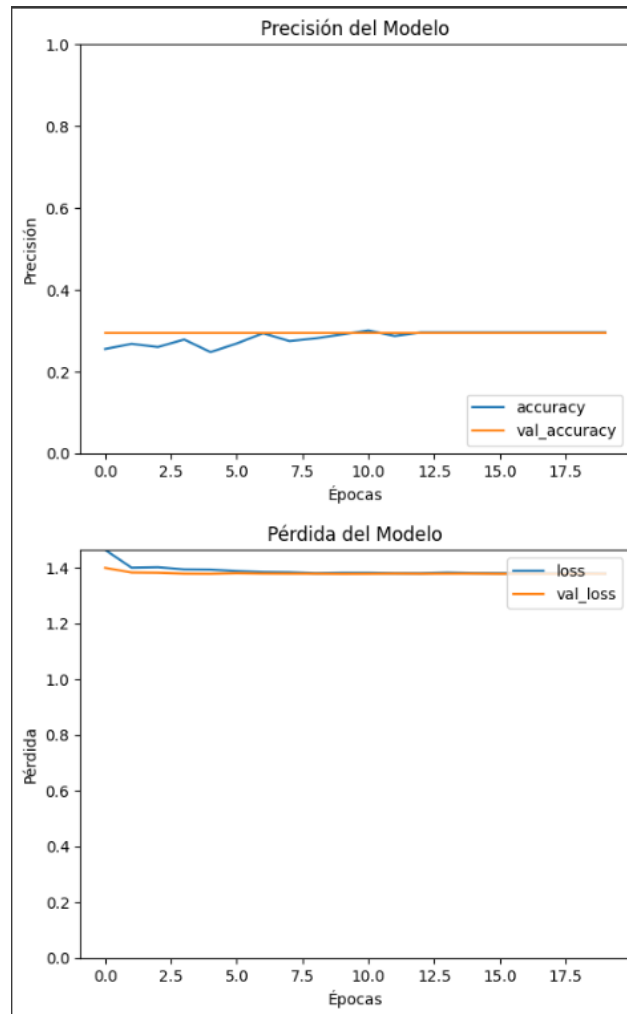
aatrox      0.23      0.74      0.35         81
ahri         0.18      0.15      0.16         65
akali        0.25      0.01      0.02         86
akshan       0.00      0.00      0.00         54
alistar      0.15      0.08      0.10         76

accuracy                0.21         362
macro avg              0.16      0.20      0.13         362
weighted avg           0.17      0.21      0.13         362
```

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
drmunido: 0.0
garen: 0.0
jhin: 0.0

Informe de clasificación:
              precision    recall  f1-score   support

   listar      0.29        1.00      0.46         76
  drmunido      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              0.29         258
  macro avg      0.07        0.25      0.11         258
 weighted avg      0.09        0.29      0.13         258

```