Segmentacion por Instancias

Santaigo Garcia

I. Introducción

En la actualidad, la inteligencia artificial, las redes neuronales y los modelos de aprendizaje profundo se han integrado cada vez más en la vida cotidiana. Se utilizan para aplicaciones como ChatGPT, vehículos con conducción autónoma, detección de imágenes para semáforos inteligentes, entre otras varias aplicaciones. No es de extrañar que incluso estas tecnologías se empleen dentro de los videojuegos

Un ejemplo sobre cómo la IA ayuda a los jugadores mediante la detección de objetos se vio en el CES 2024 por parte de MSI con el monitor *meg 321urx qd-oled* el cual es capaz de reconocer lo que sucede en la pantalla, procesarlo y mostrar información útil para el jugador dentro de la misma pantalla sin la necesidad de usar recursos de un PC.

Tomando como inspiración este avance en los videojuegos, se plantea un modelo para ser implementado para los jugadores novatos en juegos que pueden llegar a tener curvas de aprendizaje muy elevadas, como son los juegos tipo RPG, Souls Like, simulación, entre otros. En este caso, se parte de la exploración de un juego RPG muy conocido, *Genshin Impact*, juego enfocado principalmente en la exploración y recolección de objetos para obtener y mejorar nuevos personajes.

II. Marco Teórico

Para el proceso de segmentación se usa la arquitectura YOLOv8 [1], esta arquitectura consta de tres partes. La primera es el ”backbone”, dentro de esta parte se encuentra una estructura de capas convolucionales y capas C2f ordenadas de manera intercalada, como se muestra en la figura 1.

Figura 1. Backbone en YOLOv8

Dentro del backbone se observan capas convolucionales y también capas con el nombre C2f [2], Estas capas C2f corresponden a un conjunto de capas convolucionales que dividen una imagen para ser procesadas por dos cuellos de botella, que a su vez son un conjunto de pares de redes convolucionales que ayudan en el proceso de extraer características (figura 2).

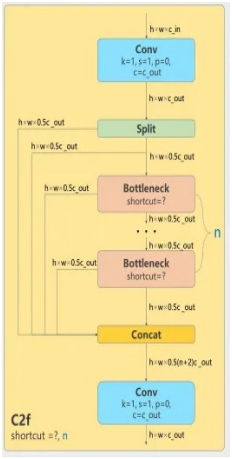

Figura 2. C2f estructura

En el backbone, la red procesa y pasa la imagen por varias capas convolucionales y filtros para extraer características mediante la extracción de patrones, texturas y borde [3].

La segunda parte son las estructuras de cabeza y cuello, estas se incluyen en una sola puesto que el cuello cumple la función de conectar el backbone con la cabeza, haciendo una operación de recolectar los mapas de características del backbone. Por su parte, la cabeza saca los resultados de la red, aquí se generan los bounding boxes asociados a los objetos de cada imagen y se asigna la probabilidad de las etiquetas en cada bounding box. (figura 3).

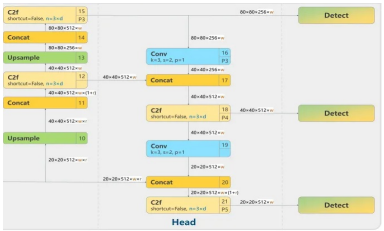

Figura 3. Head en YOLOv8

III. Implementación

Para el desarrollo del laboratorio se hace uso de WLS con una GPU RTX 4070ti con 12GB de V-RAM, y usando las dependencias de *nvidia-cuda*,*tensorflow*, *keras*, *roboflow* y *ultralitics* de la cual se pueden usar el modelo de YOLO.

El primer paso para implementar el modelo fue crear un archivo de python llamado *Modelo*, dentro del cual se hace una crea una funcion que en el que se arma un modelo, se carga un modelo pre-entrenado y se hacer trasnferlaerning del modelo pre-entrenado al modelo original de esta forma [4-5]:

```
model = YOLO('yolov8n-seg.yaml')
model = YOLO('yolov8n-seg.pt')
model = YOLO('yolov8n-seg.yaml').
        load('yolov8n.pt')
```

Luego se procede a entrenar el modelo YOLOv8 con el set de imagenes haciendo uso del archivo data.yaml el cual es un "directorio"sobre donde estan las imagenes de train con sus respectivos labels, teniendo la siguiente estructura:

```
train: ../train/images
val: ../valid/images
test: ../test/images
nc: 7
names: ['Aliado', 'Cofres',
        'Elemetos', 'Enemigo',
        'Eula', 'Fauna',
        'Flora']
```

Al final del entrenamiento YOLO crea una carpeta en la cual se guardan los resultados de los entrenamientos, dentro de estos resultados se encuentran los pesos de aquel modelo que muestra el mejor desempeño para hacer la segmentación, por lo que la función al final regresa dicho modelo, dentro del Notebook se hace un llamado a la función para guardar el modelo y posteriormente hacer predicciones sobre este.

```
from Modelo import Model_segmentation

mejor_model=
Model_segmentation(epochs=50,
                   imgsz=640,
                   batch=64)
```

Una vez con el código implementado se procede a realizar predicciones usando una imagen completamente diferente obtenida de un pequeño gameplay para hacer una predicción de este modo:

```
res=mejor_model.predict('./Eula_01.png',
                        imgsz=640)
imagen = res[0].plot()
plt.imshow(imagen)
plt.show()
```

En la prueba se observa la detección y segmentación del personaje Eula en una imagen, adicionalmente se muestra el nivel de probabilidad con el que el modelo clasifica la imagen (figura 4).

Figura 4. Segmentacion Imagen Prueba

IV. Resultados

El primer resultado obtenido es la curva F1 de confianza, la cual considera tanto la precisión como el recall. En este caso, al usar el último entrenamiento (entreno 14), la curva F1 muestra que los elementos con mejor precisión y mejor recall son Eula y Aliados, mientras que la sección de elementos y enemigos muestra la peor curva F1 (figura 5).

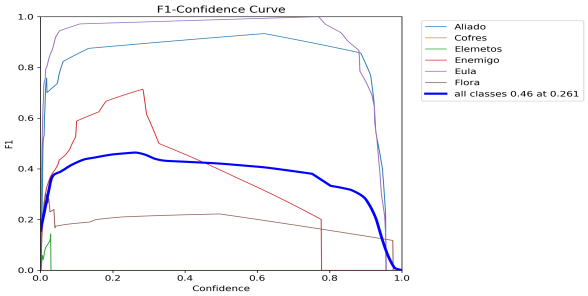

Figura 5. Curva F1

En la curva se refleja la capacidad del modelo para generar una máscara que segmente el objeto. Es natural que el elemento principal, Eula, tenga la mejor curva de F1, seguida por los Aliados. Esto se debe a que el diseño de los personajes aliados es muy similar, todos correspondientes a humanos normalmente encontrados en entornos como ciudades y pueblos. Por su parte, los enemigos tienen un pésimo F1 debido a que existe variedad de enemigos, siendo humanos, plantas, monstruos y gigantes, lo que hace difícil para el modelo distinguir de manera efectiva si un elemento es un enemigo.

Otra métrica para evaluar el desempeño del modelo es la matriz de confusión, en la cual se puede ver qué tan bien el

modelo clasifica y etiqueta los diferentes ejemplos (figura 6).

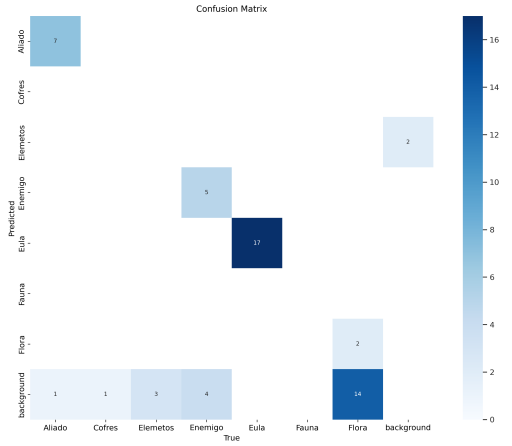

Figura 6. Matriz de confusion

Para el modelo, la etiqueta que mejor clasifica sin cometer errores es 'Eula', seguida por 'aliado' y finalmente 'enemigo'. Los elementos de 'cofres', 'flora' y 'fauna' no son clasificados por el modelo o son clasificados de manera incorrecta, como es el caso de la "flora", la cual es clasificada como background. [6].

Como una aplicación práctica, se usó un pequeño gameplay en el cual se aplica la segmentación para comprobar el desempeño y ver en vivo la velocidad con la que estos modelos son capaces de segmentar en tiempo real un video [7].

V. Conclusiones

El modelo con transfer learning muestra un desempeño bastante bueno si se le dan suficientes datos para el entrenamiento. Esto se corrobora al ver que el elemento 'Eula' es correctamente predicho, incluso mostrando buenos resultados en las métricas de F1 y matriz de confusión

Por su parte, las etiquetas de aliados y enemigos tendrían mejor desempeño si se tomaran más muestras, pues cabe aclarar que si bien los aliados son humanos, estos se diferencian en los colores y patrones de vestimenta. Por otro lado, los enemigos, al mostrar una gran variedad, requieren tomar una cantidad significativa de muestras por cada tipo, lo que llevaría más tiempo y el laboratorio tendría un alcance más amplio, el cual no es el objetivo de esta entrega.

Para mejorar la precisión en la identificación de la flora, es necesario generar etiquetas para aquellas flores que son únicas y no forman parte del fondo en los diferentes escenarios del juego. Además, los elementos como cofres, fauna y otros objetos requieren una mayor cantidad de muestras para que el modelo sea capaz de diferenciarlos, ademas de agregar el background de cada imagen para que estos elementos, que si bien hacen parte de la decoracion de un escenario, puedan ser facilmente diferenciables.

Como reflexión propia, la creación de una aplicación para detección y segmentación de imágenes está muy democratizada debido a que proyectos sencillos y de mediana escala pueden ser desarrollados por personas que tengan recursos limitados. En este laboratorio, se pudo realizar un entrenamiento con más de 100 fotos, teniendo una tarjeta con poca VRAM a comparación de tarjetas gráficas usadas específicamente para tareas más complejas. Esto resulta útil, ya que una tecnología tan útil como la detección y segmentación puede ser usada por el público en general, lo que resulta beneficioso para la sociedad en su conjunto usandose en situaciones como: la seguridad de una comunidad, para simplificar la agricultura, mejorar el tráfico, entre otras aplicaciones que vayan más allá de la conducción autónoma.

REFERENCIAS

1. https://blog.roboflow.com/whats-new-in-yolov8/ #yolov8-architecture-a-deep-dive
2. https://yolov8.org/yolov8-architecture/
3. https://medium.com/@juanpedro.bc22/ detailed-explanation-of-yolov8-architecture-part-1-6da9296b954e
4. https://docs.ultralytics.com/tasks/segment/#train
5. https://www.youtube.com/watch?v=rk7zOBRJWCc
6. https://www.youtube.com/watch?v=q7LwPoM7tSQ&t= 37s
7. https://www.youtube.com/watch?v=5VY8fVfHMok