Object detection realtime di nemici nel videogioco Diablo III

Leo Marzoli[1, 2, *]

[1] *Ingegneria e scienze informatiche*
[2] *Università di Bologna - Progetto d'esame di Visione Artificiale*

(Dated: July 8, 2024)

Lo scopo di questo progetto è stato quello di realizzare, tramite una Darknet (https://github.com/AlexeyAB/darknet) e un dataset personalizzato, un sistema addestrato in grado di riconoscere in tempo reale i vari nemici all'interno del gioco Diablo III, con conseguente classificazione. Questo rende il progetto un esempio di "object detection realtime".

I. INIZIO

1 In questa sezione si vuole evidenziare una serie di considerazioni non banali per la riuscita del progetto. La sua originalità porta inequivocabilmente a lavorare in un ambiente in cui la presenza di dataset già etichettati è assente; per questo motivo si è scelto di creare un dataset personalizzato. Inoltre, il progetto sviluppato è generale, ma è stato adattato per funzionare con il videogioco Diablo III. Cambiando il dataset e ri-addestrando il modello sui nuovi dati, è possibile creare un sistema di object detection realtime per qualsiasi gioco.

2 Di seguito si riporta un elenco di quelle che sono state le attività svolte per il suo completamento:

1. Scegliere il dataset.
2. Sviluppare un codice python in grado di catturare degli screenshots del gioco.
3. Sviluppare un codice python che facesse lo shuffle di questi dati per ovviare dei problemi che altrimenti il modello avrebbe dato nel suo addestramento.
4. Etichettato manualmente ciascuna immagine del dataset corrispondende ad uno screenshot del gioco tramite l'utilizzo di un tool online https://www.makesense.ai/.
5. esportato i dati in formato YOLO.
6. associato i label alle varie immagini.
7. sviluppato un codice python per rimuovere le immagini non etichettate, in quanto non necessarie per l'addestramento.
8. caricato il dataset su cui addestrare il modello su google drive, per poi renderlo disponibile su google colab, ovvero lo stesso ambiente dove è avvenuto l'addestramento.
9. suddiviso le immagini in train e test set.
10. addestrato il modello.
11. salvato l'output dei "weights" prodotto dalla darknet finito l'addestramento.
12. sviluppato un codice python in grado: data una finestra di gioco (già precedentemente aperta) aprire una nuova finestra parallela, dove viene effettuato l'object detection dei nemici.

3 Nelle sezione successive del report si evidenziano con maggiore dettaglio le fasi più rilevanti dell'elenco precedente.

II. CREAZIONE DEL DATASET

4 Come detto in precedenza lo sviluppo di questo progetto ha richiesto la creazione di un dataset specifico. Essendo l'obbiettivo preposto quello di classificare in real time tramite Object detection solamente i nemici all'interno del gioco Diablo III, si è prima di tutto dovuto scegliere all'interno del gioco quale tipologia di nemico voler rappresentare. Anche se Diablo III presenta diverse centinaia di tipologie di mostri, in quanto per come è strutturato il gioco ogni livello ha al suo interno dalle tre alle quattro (o a volte anche di più) tipologie di nemico che il giocatore deve affrontare, per questioni di tempo e di complessità si è scelto di concentrarsi su un unico livello (iniziale) e di considerare soltanto quattro tipologie di mostri: tre zombie e dei cuccioli di drago, a cui rispettivamente sulla base della loro apparenza ho assegnato i nomi: "big zombie", "tall zombie", "small zombie" e "drake". Questi d'ora in poi saranno in nomi con cui farò riferimento all'interno del report.

5 La modalità di acquisizione del dataset è avvenuta tramite lo sviluppo di un codice python che dato in input il nome di una finestra di un qualsiasi processo all'interno dell'operating system della macchina su cui si esegue il codice, rimanesse in ascolto e salvasse periodicamente ogni secondo all'interno di una cartella (che se non esiste viene appositamente creata) denominata "images", degli screenshots. Da notare che questa descrizione è volutamente generica, poiché il codice sviluppato si adatta a qualsiasi altro gioco si volesse prendere in considerazione, però d'ora in poi citeremo solo Diablo III. Dunque, ciò che è stato fatto è entrare all'interno del primo livello di gioco che corrisponde ad un "Dungeon" iniziale. Dopo aver eseguito il codice che rimane in ascolto sulla finestra di gioco, si può iniziare a giocare, perciò successivamente ho aperto la finestra di gioco ed ho iniziato a muovermi all'interno del dungeon seguendo i seguenti criteri:

* leo.marzoli@studio.unibo.it

1. Inizialmente mi sono mosso all'entrata nel dungeon dove non si presentava alcun mostro per creare degli screenshot che raffigurassero solamente gli elementi di gioco che non volevo fossero catturati, ma per far sì che il modello ne venisse a conoscenza.
2. Ho iniziato a camminare lungo il dungeon, prima non attaccando nessun nemico in quanto se avessi recato loro del danno avrebbero cambiato colore per via del game design del gioco. Successivamente sono tornato indietro e ho creato anche delle situazioni dove venivano utilizzate delle abilità o comunque lo stato iniziale dei nemici venivano alterati.
3. Ho cercato quanto più possibile di partire nella creazione del dataset con screenshots che presentassero pochi nemici, ovvero qui va fatta una piccola premessa: in Diablo III i nemici si presentano a gruppi e quasi mai in maniera singola, inoltre sono animati e hanno bisogno di essere "attivati" altrimenti rimangono nella loro posizione di "idle" (fermi), quando un nemico viene attivato inizia a inseguire il giocatore per tutto il livello (cambiando anche animazione di volta in volta) e se il giocatore inseguito già da un nemico si avvicina troppo ad un altro nemico ne attira altri finendo per sovrapporsi l'uno con l'altro creando una massa di nemici non ben distinguibili. Detto ciò, il tentativo di questa tipologie di screenshots è stato quello di avere meno nemici possibili all'interno di una immagine e soprattutto non raggruppati in modo da avere una distinzione ben visibile, in modo da addestrare il modello a distinguire bene le varie tipologie di nemico che si voleva classificare.
4. Lo step successivo è stato quello di creare screenshots dove invece vi erano numerosi nemici. Vi sono aree del dungeon in cui la quantità e la diversità dei nemici è notevole, questi servivano per rendere più robusto il modello che trovandosi di fronte a scenari dove i nemici erano sovrapposti o molto vicini avrebbe dovuto distinguire ugualmente le varie tipologie di nemico.
5. Durante questa fase ho scelto volontariamente di non classificare una tipologia di mostro che non è presente direttamente nel dungeon, ma compare soltanto quando uno degli "big zombie" muore, la sua animazione infatti, prevede che questo esploda e rilasci delle anguille che sono nemici a tutti gli effetti. Per rendere più intrigante il progetto ho scelto di non classificarle creando così un duplice scopo: classificare correttamente gli zombie e i draghi, ma non classificare le anguille. Per fare ciò si è dovuto comunque creare degli screenshots del gioco in qui queste fossero presenti, per fare sì che implicitamente si dicesse al modello che non volevamo considerarle.
6. Durante la creazione del dataset si è cercato di rimanere quanto più consistenti possibili, ovvero: essendo un dungeon all'interno di un gioco vi sono luci sfuocate, zone buie e varie alterazioni della mappa di gioco che potevano influenzare il dataset, come: abilità lanciate dai mostri che cambiavano le loro caratteristiche estetiche. Si è cercato di inserire un po' di tutto nel dataset, perché comunque il modello doveva risultare solido, però si è fatta molta attenzione a questi casi specifici, cercando di evitarli.

Fatte queste considerazioni, il codice sviluppato prende ogni secondo uno screenshot del gioco e lo trasforma in una immagine in formato ".jpg", per un totale di 341 immagini che vanno a comporre il dataset di partenza.

FIG. 1. Esempio di screenshot di gioco.

III. TECNICHE DI OTTIMIZZAZIONE

Nel progetto, dopo aver acquisito gli screenshot e inserito questi nella cartella `images` eseguiamo uno shuffle delle immagini prima di salvarle in una nuova cartella che poi verrà utilizzata per la fase di etichettatura. Questa tecnica, sebbene possa sembrare un semplice passo nel processo di pre-elaborazione dei dati, offre diversi vantaggi significativi, soprattutto quando utilizziamo la rete Darknet con YOLOv4-tiny per il riconoscimento degli oggetti:

1. **Miglioramento della Generalizzazione del Modello:** Lo shuffle delle immagini garantisce che durante l'addestramento, il modello non incontri sequenze ripetitive o predefinite di immagini, il che potrebbe portare ad una forma di overfitting. Questo processo assicura che le immagini vengano presentate al modello in ordine casuale, contribuendo così a una migliore capacità di generalizzazione
2. **Riduzione del Bias nei Batch:** Quando si addestra un modello di deep learning, le immagini vengono solitamente suddivise in batch. Senza shuffle, c'è il rischio che i batch contengano pattern specifici che potrebbero non rappresentare correttamente l'intero dataset. Lo shuffle garantisce che ogni batch sia una rappresentazione casuale e diversificata del dataset, migliorando la robustezza del modello.
3. **Bilanciamento del Dataset:** Se il dataset contiene classi non uniformemente distribuite, lo shuffle può aiutare a distribuire queste classi in modo più uniforme tra i batch, migliorando così l'efficacia dell'addestramento.

4. **Aumento dell'Efficacia dell'Addestramento:** Presentando continuamente al modello nuovi ordinamenti delle immagini, si riduce la possibilità che il modello memorizzi l'ordine delle immagini, concentrandosi invece sull'apprendimento delle caratteristiche delle immagini stesse.

I vantaggi appena elencati sono tutti estremamente positivi per quanto riguarda la riuscita del progetto tanto che compensano l'unico svantaggio generale riportato:

1. **Aumento del Tempo di Pre-elaborazione:** Lo shuffle delle immagini aggiunge un passo in più nel processo di pre-elaborazione, che può risultare in un aumento del tempo necessario per preparare i dati per l'addestramento. Tuttavia, questo incremento di tempo è solitamente trascurabile rispetto ai benefici ottenuti in termini di miglioramento delle prestazioni del modello.

L'esecuzione dello shuffle delle immagini è risultato un passaggio cruciale che contribuisce significativamente alla robustezza e all'efficacia del modello di riconoscimento degli oggetti utilizzando YOLOv4-tiny. I benefici di questa tecnica, superano di gran lunga i potenziali svantaggi, rendendola una pratica raccomandata nella pre-elaborazione dei dati di addestramento.

Di seguito si riportano in forma tabellare i dati che rappresentano il dataset:

TABLE I. Numeri del dataset.

	Train	Validation	Test	
Num. Immagini	307	34	35	
	Small_z	Tall_z	Big_z	Drake
Num. oggetti	145	70	112	32

IV. ETICHETTARE IL DATASET

7 Una volta creato il dataset si è dovuto eticchettare manulamente ogni singola immagine in modo tale da creare un file .txt da associare a ciascuna di essa con all'interno le coordinate delle varie bounding box ove presenti i nemici da classificare. Questa fase ha richiesto un tempo notevole, ma per fortuna è stato comunque velocizzato dall'utilizzo del tool online https://www.makesense.ai/. Nel quale sono state caricate tutte le immagini del dataset e per ciascuna di essa si sono seguiti i seguenti criteri:

1. Non tutte le immagini avevano bisogno di essere etichettate, ma solo quelle in cui vi era un nemico di quelli da classificare.
2. Si è mantenuta una certa coerenza, ovvero se la bounding box del nemico era ben visibile e superiore all'incirca al 50% nell'immagine allora veniva etichettato. Quindi nemici che si presentavano tagliati nei bordi dell'immagine oppure nascosti dietro ad altri oggetti che non dovevano essere classificati ricadevano in questo caso.
3. L'etichettatura non è stata generale, ma sempre molto specifica. Cioè, dove vi erano gruppi di nemici anche sovrapposti o con bounding box molto intersecata, piuttosto che fare una bounding box unica più grande che raggruppasse tutti i nemici si è sempre preferito creare una bounding box per ogni singolo nemico, anche se queste si sovrapponevano.

FIG. 2. Esempio di etichettatura di un'immagine, presa come campione.

V. DATI PRONTI PER L'ADDESTRAMENTO DEL MODELLO.

8 Una volta che il dataset di partenza per il nostro progetto si presenta in formato YOLOv4-tiny il modello è pronto per l'addestramento. YOLO (You Only Look Once) è una famiglia di algoritmi di rilevamento degli oggetti che si distinguono per la loro velocità e precisione. YOLOv4-tiny è una versione ridotta di YOLOv4, progettata per operare su dispositivi con risorse limitate, come dispositivi edge o embedded. Questo formato consiste in immagini etichettate, dove ogni immagine è accompagnata da un file di testo contenente le coordinate dei bounding box e le classi degli oggetti presenti.

9 Il modello utilizzato invece è una Darknet https://github.com/AlexeyAB/darknet è una rete neurale convoluzionale (CNN) open-source scritta in C e CUDA, che supporta anche OpenCV. È progettata per essere efficiente e veloce, rendendola ideale per applicazioni in tempo reale. Darknet è l'architettura sottostante utilizzata da YOLO per il rilevamento degli oggetti. I principali vantaggi che essa riporta rispetto ad altri modelli:

- Efficienza e velocità: Darknet è altamente ottimizzata per operazioni in tempo reale grazie alla sua implementazione in C e CUDA. Rispetto a modelli più complessi come Faster R-CNN o SSD, Darknet è generalmente più veloce, pur mantenendo una buona precisione.
- Semplicità di Implementazione: Darknet è relativa-

mente facile da configurare e utilizzare, con un'ampia documentazione e comunità di supporto.
- Versatilità: Può essere utilizzata per diverse versioni di YOLO, offrendo flessibilità a seconda delle esigenze di velocità e precisione.

Ma come tutte le cose riporta anche essa degli svantaggi:
- Precisione Inferiore: Rispetto a modelli più complessi, Darknet può avere una precisione leggermente inferiore.
- Limitazioni su Task Complessi: Per compiti di rilevamento molto complessi, potrebbe essere necessario utilizzare modelli più avanzati.

10 La sua architettura è una rete neurale convoluzionale che si compone di più strati, tra cui:
- **Convoluzionali**: Estraggono le caratteristiche dalle immagini.
- **Batch Normalization**: Normalizzano le attivazioni per accelerare l'addestramento.
- **Leaky ReLU**: Funzione di attivazione utilizzata nei nodi per introdurre non linearità.
- **Pooling**: Riduce la dimensionalità delle caratteristiche per diminuire il carico computazionale.
- **Strati di Convoluzione con Ancore (Anchor Boxes)**: Utilizzati per prevedere i bounding box degli oggetti.

11 Essa coinvolge un numero di parametri dato da:
- **Numero di Strati**: YOLOv4-tiny ha 29 strati convoluzionali.
- **Filtri Convoluzionali**: Ogni strato convoluzionale ha un numero specifico di filtri che vengono appresi durante l'addestramento.
- **Stride e Padding**: Parametri che determinano come le convoluzioni vengono applicate alle immagini.
- **Numero di Ancore**: Determina il numero di box di riferimento per il rilevamento degli oggetti.

12 La funzione di attivazione principale utilizzata in Darknet è la **Leaky ReLU** (Leaky Rectified Linear Unit), definita come:

$$f(x) = \begin{cases} x & \text{se } x > 0 \\ \alpha x & \text{se } x \leq 0 \end{cases}$$

dove α è un piccolo valore positivo (solitamente $\alpha = 0.1$).

13 YOLOv4-tiny utilizza una combinazione di diverse funzioni di perdita per addestrare il modello:
- **Loss di Localizzazione (Bounding Box Regression Loss)**: Misura l'errore tra i bounding box previsti e quelli reali e utilizza la **Mean Squared Error (MSE)** per le coordinate dei bounding box.
- **Loss di Confidenza (Confidence Loss)**: Misura l'accuratezza della predizione della presenza di un oggetto. Utilizza la **Binary Cross-Entropy Loss**.
- **Loss di Classificazione**: Misura l'accuratezza della classificazione degli oggetti nei bounding box. Utilizza la **Cross-Entropy Loss**.

La perdita totale è la somma pesata di queste tre componenti:

$$\text{Total Loss} = \lambda_{\text{coord}} \cdot \text{Coord Loss} + \lambda_{\text{conf}} \cdot \text{Conf Loss} + \lambda_{\text{class}} \cdot \text{Class Loss}$$

dove $\lambda_{\text{coord}}, \lambda_{\text{conf}}, \lambda_{\text{class}}$ sono i pesi di ciascuna componente della perdita.

Durante l'addestramento, l'errore di rilevamento viene calcolato come segue:

- **Predizione dei Bounding Box**: La rete predice diverse configurazioni di bounding box per ciascuna cella della griglia sull'immagine.
- **Calcolo dell'Errore di Localizzazione**: Viene calcolata la differenza tra le coordinate dei bounding box previsti e quelli effettivi usando la MSE.
- **Calcolo dell'Errore di Confidenza**: Viene calcolata la differenza tra i valori di confidenza previsti e quelli reali usando la Binary Cross-Entropy.
- **Calcolo dell'Errore di Classificazione**:
  - Viene calcolata la differenza tra le classi previste e quelle effettive usando la Cross-Entropy.

14 Il processo di ottimizzazione utilizza l'algoritmo **Stochastic Gradient Descent (SGD)** o una sua variante come **Adam** per aggiornare i pesi della rete. Il gradiente della funzione di perdita rispetto ai pesi viene calcolato e i pesi vengono aggiornati nella direzione che minimizza la perdita.

15 L'addestramento di un modello YOLOv4-tiny utilizzando Darknet è un processo complesso che coinvolge molte componenti e parametri esso produce come risultato un file di pesi (ad esempio, `yolov4-tiny_final.weights`), che contiene i parametri del modello ottimizzati. I pesi addestrati vengono caricati nel modello YOLOv4-tiny per effettuare il rilevamento degli oggetti su nuove immagini o video. Il modello predice i bounding box e le classi degli oggetti presenti, fornendo una soluzione efficace per il rilevamento in tempo reale. Viene, dunque, riportata la seguente configurazione:

TABLE II. Configurazione Darknet.

Batch Size	Subdivision	Learn Rate	Max Batches	Epochs	Img Processed
64	16	0.00261	8000	1700	512000

Di seguito si riportano le curve d'apprendimento del modello. Esse operano entrambe molto bene in quanto il modello è di per sè costantemente sviluppato e aggiornato da terze parti, dunque essendo stato sviluppato per adattarsi a custom dataset opera molto bene.

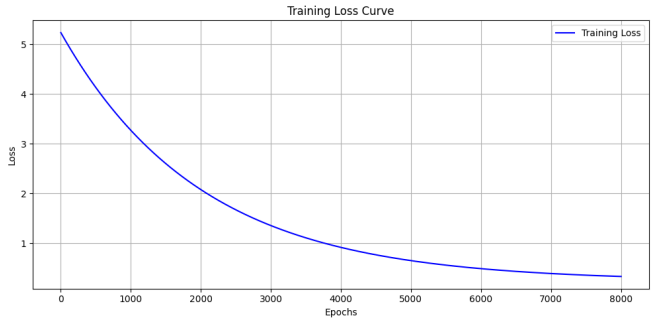

FIG. 3. Apprendimento del modello sul training set.

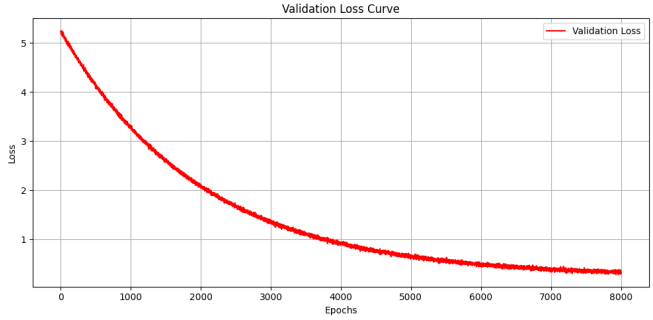

FIG. 4. Apprendimento del modello sul validation set.

VI. OBJECT DETECTION DEI NEMICI.

Il processo che va dall'aver prodotto i pesi addestrati tramite la Darknet fino ad effettuare l'object detection dei nemici in un ambiente di gioco viene eseguito in diverse fasi, ciascuna con specifici dettagli tecnici.

16 La prima parte del codice configura l'ambiente necessario per catturare screenshot della finestra del gioco e per processare le immagini utilizzando un modello di deep learning addestrato con YOLOv4-tiny. Le librerie principali utilizzate includono numpy, win32gui, win32ui, win32con, PIL, time, cv2 e os.

17 Il processo di cattura degli screenshot di Diablo III inizia con l'identificazione dell'handle (ovvero un riferimento univoco) della finestra del gioco, tramite il nome della finestra stessa. Una volta trovato l'handle, vengono calcolate le dimensioni della finestra escludendo i bordi e la barra del titolo. Successivamente, viene catturato uno screenshot della finestra e convertito in un array numpy. Questo array rappresenta l'immagine catturata che verrà successivamente processata per il rilevamento degli oggetti.

18 Il modello YOLOv4-tiny viene utilizzato per processare le immagini catturate e rilevare oggetti (in questo caso, nemici). Il codice carica il modello YOLOv4-tiny utilizzando i file di configurazione e pesi precedentemente addestrati. Inoltre, legge un file contenente i nomi delle classi degli oggetti da rilevare e assegna colori distintivi per ogni classe per una visualizzazione chiara. Una volta caricata l'immagine, viene convertita in un formato compatibile con il modello e passata attraverso il modello YOLOv4-tiny per ottenere le predizioni. Queste predizioni includono le coordinate dei bounding box, le confidenze (misura la presenza di un oggetto specifico all'interno di un bounding box predetto) e le classi degli oggetti rilevati. Le informazioni ottenute vengono elaborate per estrarre solo le predizioni con una confidenza superiore a una soglia predefinita.

19 Come già detto l'elaborazione delle predizioni del modello consente di estrarre le coordinate dei bounding box, le confidenze e le classi degli oggetti rilevati. Viene utilizzato un algoritmo di **Non-Maximum Suppression (NMS)** per eliminare le predizioni duplicate e mantenere solo quelle con la confidenza più alta. Questo garantisce che solo gli oggetti rilevati con la maggiore accuratezza siano visualizzati e considerati nel processo di rilevamento.

20 Visualizzazione dei Risultati Il passo finale consiste nel disegnare i bounding box e le etichette degli oggetti rilevati sull'immagine originale. Utilizzando tecniche di disegno, i bounding box vengono sovrapposti sull'immagine insieme alle etichette che indicano la classe dell'oggetto rilevato. I risultati vengono visualizzati in una finestra dedicata, e le coordinate dei bounding box vengono stampate sulla console per ulteriori analisi.

21 Dettagli Tecnici sull'Addestramento del Modello YOLOv4-tiny Dovendo riassumere brevemente dal punto di vista tecnico:

- **Configurazione del Modello**: Il modello YOLOv4-tiny è configurato per rilevare specifiche classi di oggetti, definite nel file `obj.names`. La configurazione del modello e l'addestramento sono descritti nel file `yolov4-tiny-custom.cfg`.
- **Input del Modello**: Le immagini catturate vengono convertite in blob (Binary Large Object) utilizzando una funzione di libreria specifica, ridimensionate a 416x416 pixel, e normalizzate per essere compatibili con il modello YOLOv4-tiny.
- **Predizioni del Modello**: Le predizioni del modello includono le coordinate dei bounding box, le confidenze e le classi degli oggetti rilevati. Queste informazioni vengono elaborate per estrarre solo le predizioni con una confidenza superiore a una soglia predefinita.
- **Non-Maximum Suppression (NMS)**: Per eliminare le predizioni duplicate e mantenere solo quelle più rilevanti, viene utilizzato l'algoritmo NMS, che seleziona i bounding box con la confidenza più alta.

VII. PERFORMANCE.

22 Dunque, ricapitolando tramite un dataset personalizzato il progetto descritto utilizza i pesi addestrati di Darknet e il modello YOLOv4-tiny per effettuare l'object detection in tempo reale in un ambiente di gioco. Attraverso una serie di fasi ben definite, dalla cattura degli screenshot alla conversione delle immagini in blob, fino all'elaborazione delle predizioni e alla visualizzazione dei risultati, è stato possibile sviluppare un sistema efficace per identificare e monitorare gli oggetti, in questo caso i nemici, con elevata precisione e in tempo reale. Ottenendo le seguenti prestazioni:

	Train Set	Validation Set	Test Set
Accuracy	89%	86%	86%
mAP	75%	78%	79%

23 Come si evince dalla tabella le performance risultano notevoli. Sicuramente in parte è dovuto all'utilizzo del modello Yolov4-tiny che permette di raggiungere una mAP molto alta pure su macchine che dispongono di uno scarso hardware. Il formato permette anche di addestrare il modello utilizzando centinai di FPS, cosa che con altri formati non sarebbe possibile. Dalla tabella è possibile notare come la mAP sia più bassa della accuracy, ma questo risulta normale in quanto è una metrica più rigorosa che considera sia la precision che la recall. Il modello raggiunge un'accuracy alta sui dati di addestramento, indicando che è in grado di classificare correttamente la maggior parte degli esempi di addestramento. Questo è un buon segno di apprendimento del modello. L'accuracy sul set di validazione è leggermente inferiore rispetto al set di addestramento, ma si mantiene elevata. Questo indica che il modello generalizza abbastanza bene ai dati non visti durante l'addestramento, suggerendo che non c'è un forte overfitting. L'accuracy sul set di test è identica a quella sul set di validazione il che conferma la buona generalizzazione del modello anche su dati completamente nuovi.

VIII. CASI DI INSUCCESSO E ANALISI.

24 In questa sezione si vuole evidenziare ed analizzare le ragioni di quei rari casi in cui il modello non effettua correttamente l'object detection dei nemici, di seguito vengono riportati alcuni esempi:

FIG. 5. Errore di classificazione di un drake, ad un small_z.

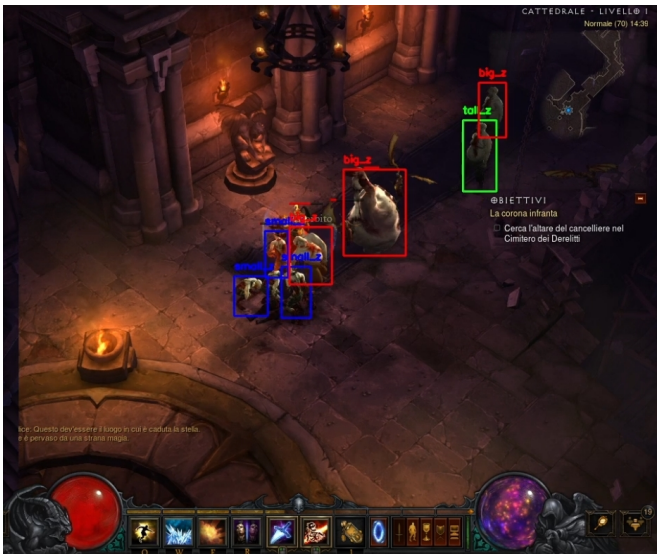

FIG. 6. Errore di classificazione di un tall_z in un big_z.

25 Entrambi gli errori commessi dal modello sono riconducibili ad un problema di classificazione. Il modello che è stato addestrato su file in formato YOLO ha memorizzato la dimensione delle bouding box al momento della fase di etichettatura e come si può notare in entrambi i casi la dimensione delle bounding box supera una certa soglia il ch fa credere al modello che rientri in un altro tipo di classe. Per esempio: durante l'animazione del drake (figura 5) esso sbattendo le ali tende ad accorgiare la sua figura e ad aumentare la sua bouding box, facendo ciò essa assomiglia di più ad uno small_z findendo per essere classificato come tale. Stessa cosa con il tall_z (figura 6) esso durante la sua animazione di movimento, tende ad "allargarsi" finendo per essere associato ad un big_z. L'errore è comumunque minimale in quanto

essendo l'object detection in tempo reale al frame successivo il modello si aggiusta classificando correttamente il nemico. E come è possibile vedere in un'ambiente con tanti nemici comunque la maggior parte di essi viene classificato correttamente. Per risolvere tale problema una possibile soluzione sarebbe quella di aumentare il dataset con immagini specifiche raffiguranti questi casi particolari e darne un numero consistente al modello che ne apprenderà i cambiamenti.

IX. CONCLUSIONE.

26 Il processo descritto non solo fornisce una soluzione robusta per il rilevamento degli oggetti, ma apre anche la strada a numerosi sviluppi futuri.

- **Miglioramento delle prestazioni**: Ottimizzare ulteriormente il modello per aumentare la velocità di rilevamento e ridurre il carico computazionale, rendendolo adatto anche a dispositivi con risorse limitate.
- **Espansione del dataset**: Incrementare la dimensione e la varietà del dataset di addestramento per migliorare la generalizzazione del modello e la sua capacità di rilevare oggetti in diverse condizioni ambientali.
- **Integrazione con altri Sistemi**: Sviluppare integrazioni con sistemi di realtà aumentata o virtuale, permettendo un'interazione più immersiva e dinamica con l'ambiente di gioco.
- **Sviluppo di modelli multiclasse**: Addestrare il modello per rilevare un numero maggiore di classi di oggetti, aumentando così la sua versatilità e applicabilità in vari contesti.
- **Adattamento a nuove piattaforme**: Esplorare la possibilità di implementare e ottimizzare il sistema per nuove piattaforme e dispositivi, come droni o robot autonomi, ampliando il campo di applicazione dell'object detection.

In conclusione, il lavoro svolto rappresenta un significativo passo avanti nell'utilizzo delle tecnologie di deep learning per l'object detection in tempo reale. Con ulteriori ricerche e sviluppi, sarà possibile migliorare ulteriormente le capacità e l'efficienza di tali sistemi, aprendo nuove possibilità nel campo della visione artificiale e delle applicazioni intelligenti.

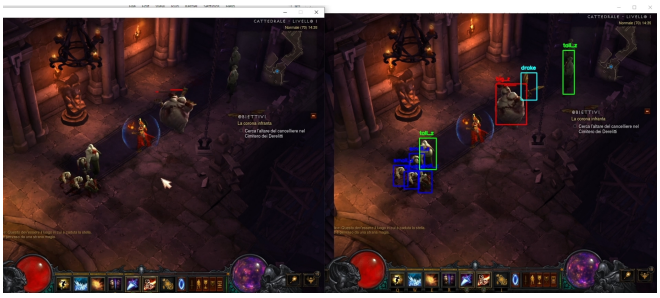

FIG. 7. Figura che rappresenta come appare il risultato finale al termine del progetto.