Riconoscimento oggetti

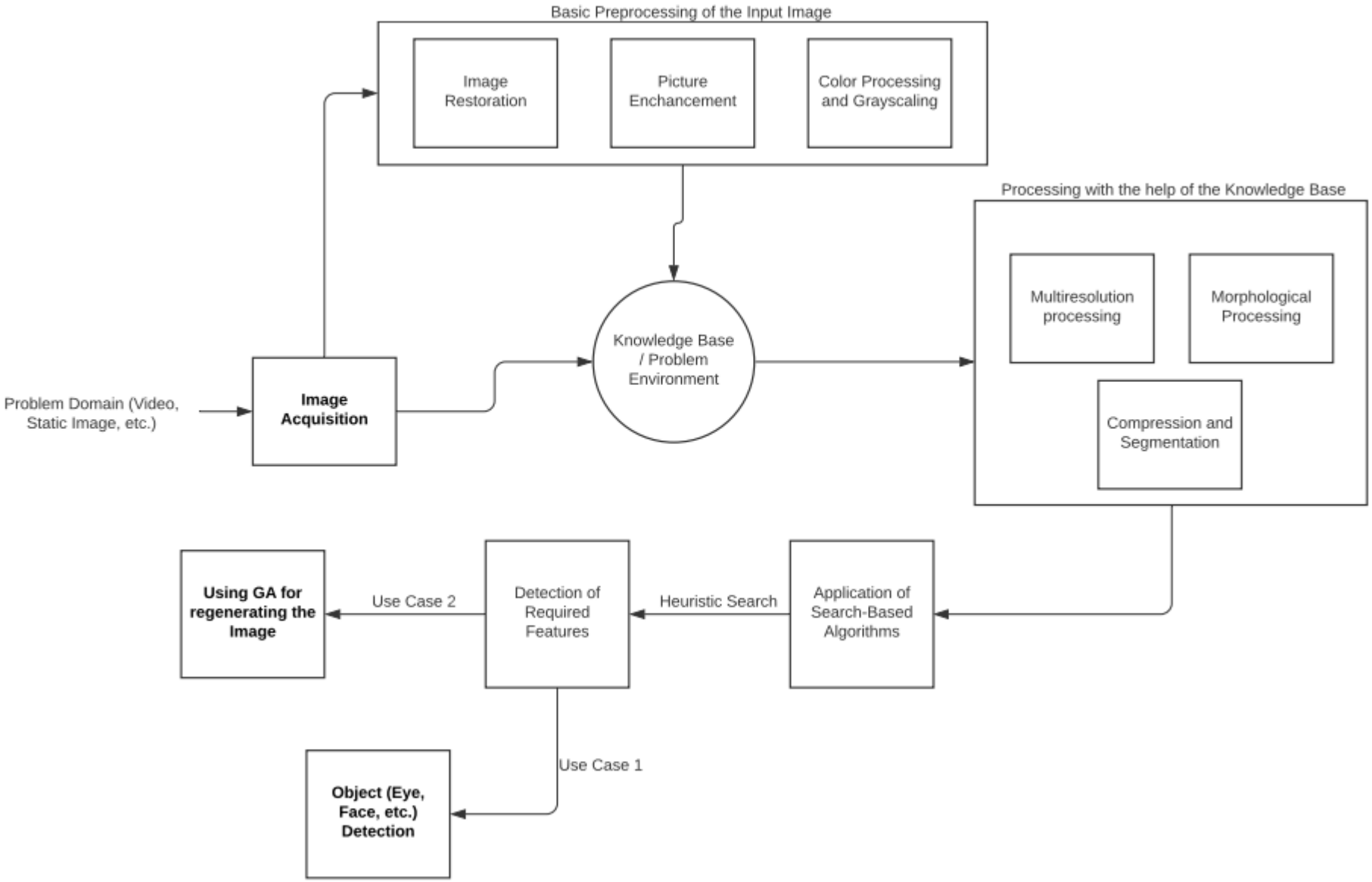
A CURA DI LINO POLO

Image Processing in Python

L’Image Processing non è altro che la conversione di un’immagine in forma digitale al fine di applicare operazioni su di essa, generalmente per applicare filtri o per estrarre feature dall’immagine. Può essere visto come una forma di distribuzione del segnale in cui l’input è generalmente sotto forma di immagini, come immagini statiche, video, film in movimento, ecc.

Le immagini vengono trattate come segnali bidimensionali e convertite in matrici. Lo scopo dell’Image Processing è visualizzare l’immagine in input, ripulirla da qualsiasi rumore, e cercare pattern e riconoscere le caratteristiche al suo interno. Nel dominio della Computer Vision, l’Image Processing viene anche chiamata Digital Image Processing.

L'elaborazione delle immagini è la fase preliminare di feature engineering di qualsiasi problema di Computer Vision. Sotto è mostrata una rappresentazione del funzionamento dell’image processing. Nell’immagine possiamo vedere alcuni termini chiave usati nelle pipeline di elaborazione delle immagini.


Basic Preprocessing of the Input Image
Image Restoration
Picture Enchancement
Color Processing and Grayscaling
Processing with the help of the Knowledge Base
Multiresolution processing
Morphological Processing
Compression and Segmentation
Knowledge Base / Problem Environment
Problem Domain (Video, Static Image, etc.)
Image Acquisition
Using GA for regenerating the Image
Use Case 2
Detection of Required Features
Heuristic Search
Application of Search-Based Algorithms
Use Case 1
Object (Eye, Face, etc.) Detection

Knowledge Base: tutti i problemi richiedono un proprio dominio di lavoro. Ad esempio, un video che mostra una partita di cricket richiederebbe un ambiente generale costituito dalle attrezzature, dal campo di gioco e dai giocatori. Allo stesso modo, anche i processi di elaborazione delle immagini richiedono una propria base di conoscenza, un database in cui sono codificate tutte le informazioni riguardo il dominio del problema. L'algoritmo di Image Processing effettua operazioni continue sulla knowledge base per verificare che il processo stia operando in maniera accurata.

Elaborazione morfologica: è un processo basato sulla forma e dimensione dell'immagine. L'elaborazione morfologica si accorda con operazioni e funzioni che hanno lo scopo di estrarre componenti dell'immagine che rappresenterebbero bene l'immagine nelle sue dimensioni (forma, dimensione e inclinazione).

Image Segmentation: nei sistemi di Computer Vision, la segmentazione è il processo di partizionamento dell'immagine in gruppi di pixel più piccoli o in oggetto. L'obiettivo finale dell'image segmentation è rappresentare l'immagine in maniera più significativa possibile per semplificare il processo di analisi. Viene inoltre assegnata un'etichetta a ogni pixel per facilitare il lavoro agli algoritmi di machine learning e di computer vision.

Step nell’elaborazione di un’immagine: l'Image Processing è un processo che si compone di più fasi che non può essere definito come una successione di passaggi distinti, perché gli step sono dinamici e potrebbero essere ripetuti più di una volta. Vediamo una pipeline generale di elaborazione di immagini:

il primo passo è l’acquisizione dell’immagine, che si occupa del modo in cui si reperisce l’immagine da dare in input all’algoritmo;

dopo aver catturato l’immagine, si effettuano diverse operazioni di pre–processing, come il ridimensionamento e la pulitura dal rumore o la rimozione delle increspature;

le increspature sono la base per classificare le immagini in più gradi di risoluzione. Suddividere le immagini in segmenti o regioni più piccole è un passaggio importante per comprimere l’immagine;

una volta compressa l’immagine la si converte in una matrice, su cui vengono applicati gli algoritmi di ricerca discussi in precedenza;

lo scopo finale di un algoritmo di image processing potrebbe essere quello di rilevare una parte dell’immagine o di generare la stessa immagine con diverse configurazioni.

Opencv

Lettura delle immagini, conversione del colore e Rilevazione dei Bordi

FUNZIONI DI OPENCV PER L'IMAGE PROCESSING

OpenCV è la libreria più usata per l'elaborazione delle immagini e include funzionalità per lavorare coi video, coi volti, e molto altro. L'architettura sottostante OpenCV usa algoritmi di Machine Learning per svolgere i task di image processing. Dal momento che lavorare con le immagini è molto più complicato che lavorare coi numeri, OpenCV dispone di un'ampia collezione di algoritmi che si coordinano tra loro suddividendo il task di riconoscimento in porzioni molto piccole. In queste implementazioni ci sono vari algoritmi di ricerca che abbiamo visto in precedenza. Per iniziare con OpenCV, esamineremo le funzioni di base per la lettura, la scrittura e la visualizzazione di immagini in Python:

Applicazioni Pratiche di OpenCV

Visione Artificiale in Robotica

Utilizzo di OpenCV per la navigazione e il riconoscimento di oggetti.

Implementazioni di robotica visione-based.

Analisi di Immagini Mediche

Applicazioni di OpenCV nell'analisi di immagini mediche, come la segmentazione di organi.

Ruolo di OpenCV nella diagnostica medica.

Automazione Industriale

Controllo di qualità visivo utilizzando OpenCV.

Utilizzo di OpenCV in applicazioni di automazione industriale.

imread(): questa funzione permette all'utente di fornire immagini in input. Legge le immagini in diversi formati, tra cui PNG, JPEG, JPG e TIFF. Prende come argomenti il nume del file, il path e l'estensione.

imshow(): questa funzione si usa per visualizzare un'immagine in Python su una finestra esterna. La finestra che genera il risultato si adatta automaticamente alla dimensione dell'immagine. Anche questa funzione supporta numerosi formati di immagine e prende in input il nome del file, il path e l'estensione.

imwrite(): questa funzione si usa per scrivere dati binary su un file immagine. Il caso d'uso generale è convertire il tipo del file. La funzione write può salvare le immagini in formati come PNG, JPEG, JPG, TIFF, ecc. L'estensione desiderata viene passata come argomento, insieme al nome del file.

Concetti Fondamentali

Immagini in OpenCV: rappresentazione e manipolazione

OpenCV rappresenta le immagini come array NumPy, facilitando la manipolazione.

Ogni pixel è rappresentato da valori di intensità dei canali (per esempio, RGB).

Le dimensioni e la profondità delle immagini sono importanti per le operazioni.

Canali e Spazi Colore

RGB: Canali Rosso, Verde e Blu.

HSV (Hue, Saturation, Value): Rappresentazione alternativa che separa l'intensità del colore.

Altri spazi colore: Grayscale, LAB, ecc.

Filtri e Trasformazioni di Base

Utilizzo di filtri per migliorare o modificare immagini.

Esempi di filtri: blur, sharpen, edge detection.

Trasformazioni geometriche come rotazione e traslazione.

Caricamento di Immagini da File o Telecamera

Utilizzo di cv2.imread() per caricare immagini da file.

cv2.VideoCapture() per acquisire video dalla telecamera.

Visualizzazione delle Immagini con OpenCV

cv2.imshow() per mostrare immagini in una finestra.

cv2.waitKey() per gestire l'interazione con la finestra.

Rilevamento di Contorni e Oggetti

Rilevamento di Contorni

Utilizzo di cv2.findContours() per individuare i contorni in un'immagine.

cv2.drawContours() per disegnare i contorni rilevati.

Identificazione e Analisi di Oggetti

Utilizzo di funzioni come cv2.minAreaRect() per ottenere il rettangolo minimo che circonda un oggetto.

Calcolo di proprietà come area e perimetro.

Slide 8: Riconoscimento di Volto con OpenCV

Utilizzo di Classificatori Pre-addestrati

Addestramento di un classificatore per il riconoscimento facciale.

Utilizzo di cascata di Haar per il rilevamento dei volti.

Esempi Pratici di Riconoscimento di Volto

Implementazione di un'applicazione di riconoscimento facciale utilizzando OpenCV.

Possibili sfide e ottimizzazioni.

Funzione Canny()

La funzione cv2.Canny() è utilizzata per eseguire il rilevamento dei bordi in un'immagine utilizzando l'operatore di Canny. L'operatore di Canny è un algoritmo di rilevamento dei bordi che cerca di identificare le discontinuità nei livelli di intensità di un'immagine. Qui di seguito una breve spiegazione di come funziona la funzione Canny():

cv2.Canny(image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]])

Parametry Canny()

image: L'immagine di input su cui applicare il rilevamento dei bordi. Deve essere un'immagine in scala di grigi.

threshold1 e threshold2: Sono i due valori della soglia usati nell'algoritmo. I pixel con gradienti di intensità superiori a threshold2 vengono sicuramente considerati come bordi, mentre i pixel con gradienti inferiori a threshold1 vengono sicuramente scartati. I pixel con gradienti compresi tra threshold1 e threshold2 vengono inclusi solo se sono connessi a pixel con gradienti superiori.

edges (opzionale): Un'immagine di output che contiene i risultati del rilevamento dei bordi. Se non viene fornito, la funzione restituisce l'immagine dei bordi.

apertureSize (opzionale): Dimensione del kernel per il calcolo del gradiente. Se non viene specificato, viene utilizzato un valore predefinito di 3.

L2gradient (opzionale): Un flag booleano che indica se utilizzare una formula più accurata per il calcolo del gradiente (True) o una formula più veloce (False). Di solito, è sufficiente impostare questo valore su False.

edges (immagine di output):

Questo parametro è opzionale e rappresenta un'immagine di output in cui vengono memorizzati i risultati del rilevamento dei bordi. Se viene fornito, l'output della funzione Canny verrà memorizzato in questa immagine. Se non viene fornito, la funzione restituirà l'immagine dei bordi come risultato. L'immagine di output conterrà i bordi rilevati nell'immagine originale, dove i pixel bianchi rappresentano i bordi.

L2gradient (usare la formula L2 per il calcolo del gradiente):

Questo è un parametro booleano che determina quale formula utilizzare per il calcolo del gradiente. Se L2gradient è impostato su True, viene utilizzata la formula del gradiente L2 (Euclidea) più accurata. Se è impostato su False, viene utilizzata la formula del gradiente L1 (Manhattan) più veloce. In molti casi, è sufficiente impostare L2gradient su False, a meno che non si richieda una maggiore precisione nel calcolo del gradiente.

apertureSize (dimensione del kernel):

Il parametro apertureSize rappresenta la dimensione del kernel utilizzato per il calcolo del gradiente dell'immagine. Il kernel è una finestra scorrevole che attraversa l'immagine per calcolare il gradiente dei pixel. La scelta della dimensione del kernel può influenzare la precisione del rilevamento dei bordi. Un valore comune per questo parametro è 3, che utilizza un kernel 3x3 per il calcolo del gradiente. Un kernel più grande può catturare variazioni più ampie nelle intensità dei pixel.

Face e Eye Detection

Per lavorare con cose come volti e occhi, l'algoritmo suddivide il processo di rilevamento in molte attività (6000 o più) più piccole che rilevano segmenti molto piccoli (raccolta di pixel) dell'immagine e verificano se tali segmenti corrispondono a parti di viso. OpenCV utilizza le cascade per risolvere questa parte del problema. Una cascade è definita come una raccolta di cascade. Le cascade sono una batteria predefinita di test a cui l'algoritmo deve sottoporsi per passare all'iterazione successiva. Le cascade in OpenCV sono file XML che contengono i dati utilizzati dagli algoritmi per l'object detection. Le cascade esistono sia per oggetti umani che per quelli inanimati.

FACE DETECTION

Il riconoscimento facciale è il ramo dell'object detection che tratta l'identificazione di volti in un'immagine. Useremo il file ["haarcascades/haarcascade_frontalcatface.xml"] come cascade per la Face Detection. Il riconoscimento facciale si compone dei seguenti passaggi:

l'immagine viene fornita come input al programma;

si importano le cascade e si inizializza la classe face cascade con l'immagine di input;

si effettua la conversione in scala di grigi, dopodiché si usa la funzione detectMultiScale() per identificare le porzioni di immagine che contengono un volto. Questa funzione prende in input un'immagine in scala di grigi, un fattore di scala (alcune immagini potrebbero essere più vicine alla fotocamera e sembrare più grandi, quindi l'immagine dovrebbe essere ridimensionata per rappresentare tutti gli oggetti possibili in un intervallo di scala specificato), il valore minimo dei vicini che definiscono l'oggetto rilevato vicino al viso e l'argomento della dimensione minima che fornisce la dimensione di ciascuna finestra di attività individuale;

infine, l'algoritmo traccia un rettangolo sui volti individuati.

Rilevazione oggetti con YOLO

Cos'è YOLO?

You Only Look Once (YOLO) è un algoritmo per il rilevamento di oggetti in tempo reale introdotto nel 2015 da Joseph Redmon , Santosh Divvala , Ross Girshick e Ali Farhadi nel loro famoso documento di ricerca " You Only Look Once: Rilevamento di oggetti unificato e in tempo reale ”.

Gli autori inquadrano il problema del rilevamento degli oggetti come un problema di regressione invece che come un compito di classificazione separando spazialmente riquadri di delimitazione e associando probabilità a ciascuna delle immagini rilevate utilizzando una singola rete neurale convoluzionale (CNN).

Cosa rende YOLO popolare per il rilevamento di oggetti?

Alcuni dei motivi per cui YOLO è in testa alla competizione includono:

Velocità

Precisione del rilevamento

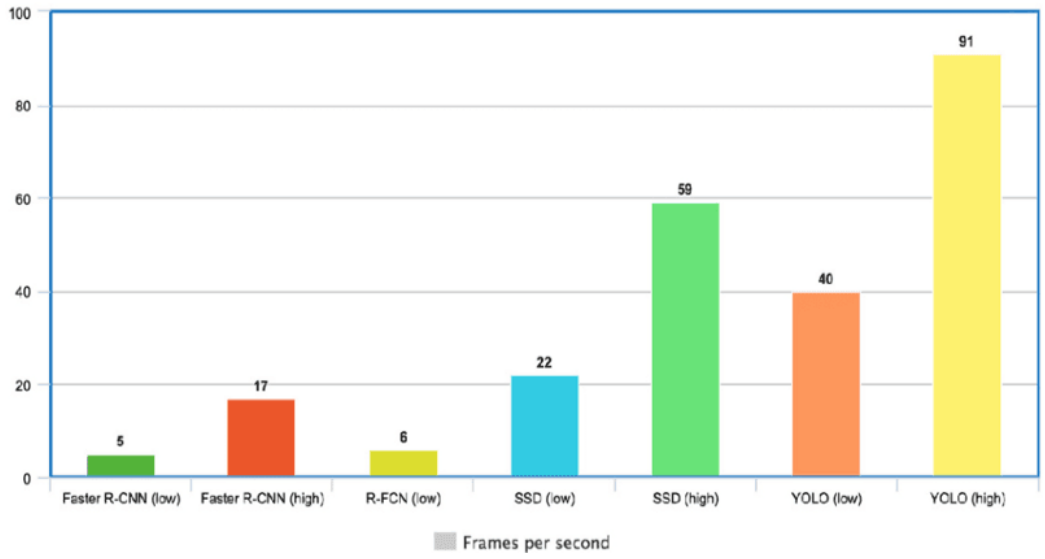
Buona generalizzazione

Open source

1- Velocità

YOLO è estremamente veloce perché non si occupa di pipeline complesse. Può elaborare immagini a 45 fotogrammi al secondo (FPS). Inoltre, YOLO raggiunge più del doppio della precisione media media (mAP) rispetto ad altri sistemi in tempo reale, il che lo rende un ottimo candidato per l'elaborazione in tempo reale.

Dal grafico a lato osserviamo che YOLO è ben oltre gli altri rilevatori di oggetti con 91 FPS.



2- Elevata precisione di rilevamento

YOLO è ben oltre altri modelli all'avanguardia in termini di precisione con pochissimi errori di fondo.

3- Migliore generalizzazione

Ciò è particolarmente vero per le nuove versioni di YOLO, di cui parleremo più avanti nell'articolo. Con questi progressi, YOLO si è spinto un po' oltre fornendo una migliore generalizzazione per i nuovi domini, il che lo rende ottimo per le applicazioni che si affidano al rilevamento di oggetti veloce e robusto.

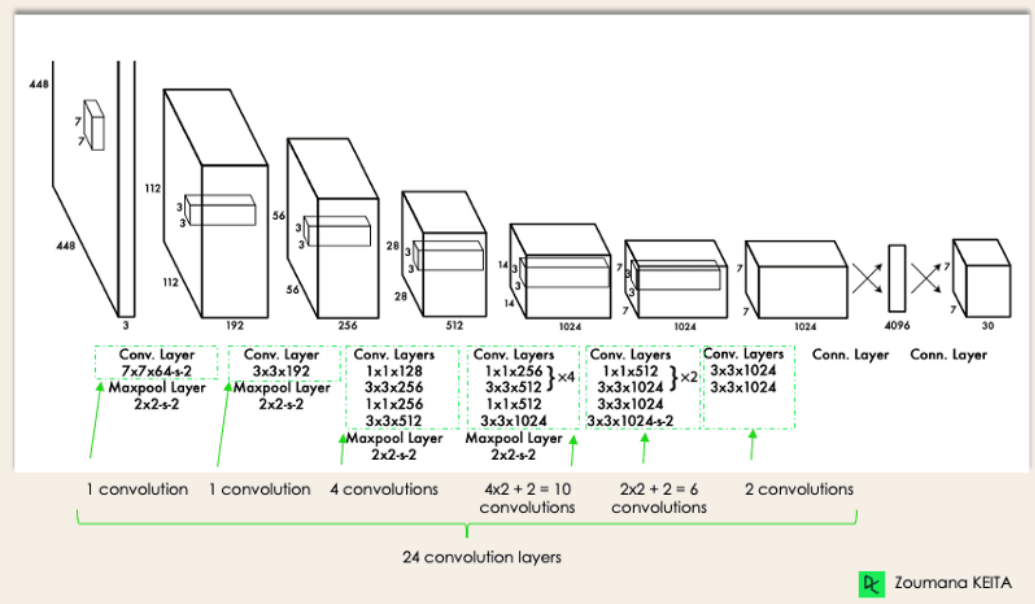
Ad esempio, il documento Automatic Detection of Melanoma with Yolo Deep Convolutional Neural Networks (https://www.researchgate.net/publication/337498273_Automatic_Detection_of_Melanoma_with_Yolo_Deep_Convolutional_Neural_Networks) mostra che la prima versione YOLOv1 ha la precisione media media più bassa per il rilevamento automatico della malattia del melanoma, rispetto a YOLOv2 e YOLOv3.

4- Fonte aperta

Rendere YOLO open source ha portato la comunità a migliorare costantemente il modello. Questo è uno dei motivi per cui YOLO ha apportato così tanti miglioramenti in un tempo così limitato.

Architettura YOLO

L'architettura YOLO è simile a GoogleNet . Come illustrato di seguito, ha complessivamente 24 strati convoluzionali, quattro strati di max pooling e due strati completamente connessi.



L'architettura funziona come segue:

Ridimensiona l'immagine di input in 448x448 prima di passare attraverso la rete convoluzionale.

Viene prima applicata una convoluzione 1x1 per ridurre il numero di canali, seguita poi da una convoluzione 3x3 per generare un output cuboidale.

La funzione di attivazione sotto il cofano è ReLU, ad eccezione dello strato finale, che utilizza una funzione di attivazione lineare.

Alcune tecniche aggiuntive, come la normalizzazione batch e il dropout, rispettivamente regolarizzano il modello e ne impediscono l'overfitting.

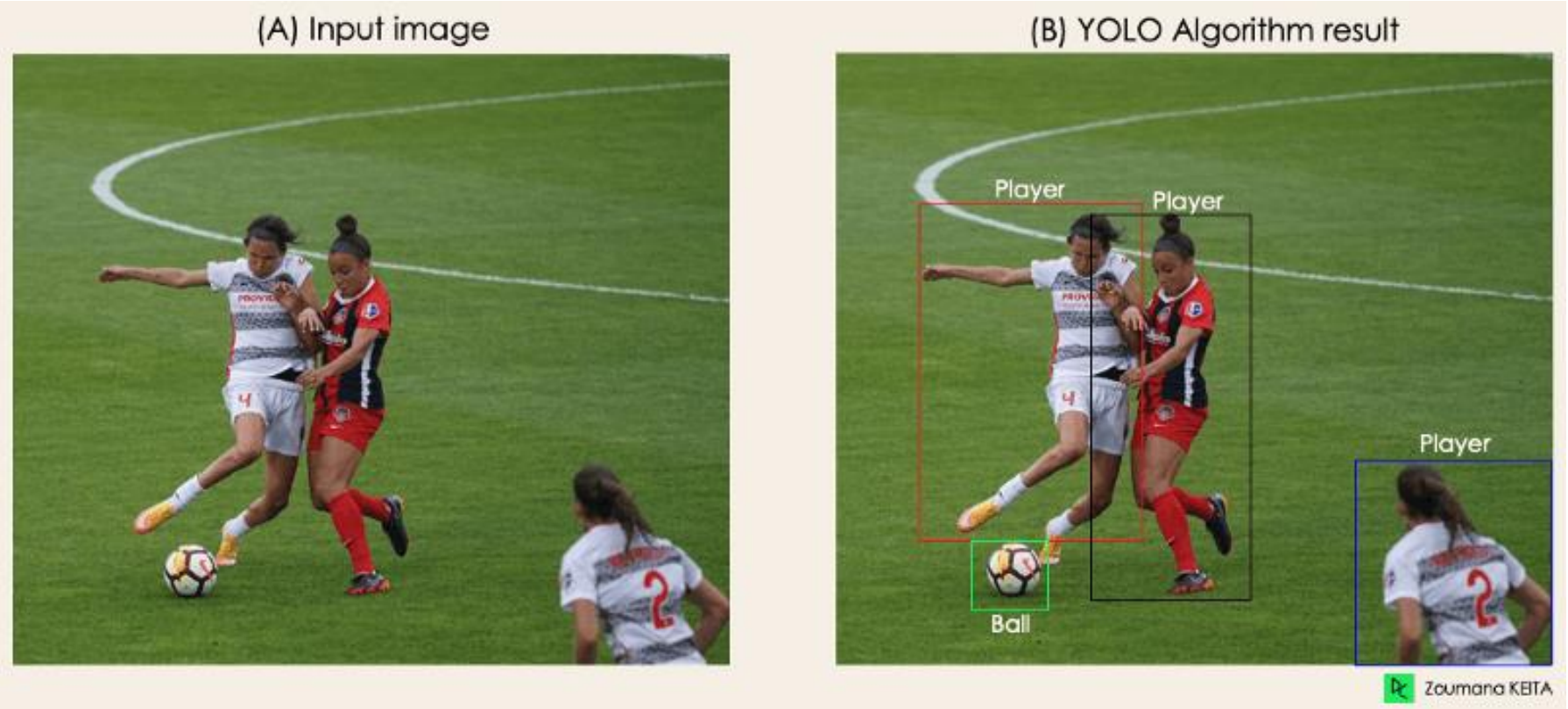
Come funziona il rilevamento degli oggetti YOLO?

Ora che hai compreso l'architettura, diamo una panoramica di alto livello di come l'algoritmo YOLO esegue il rilevamento degli oggetti utilizzando un semplice caso d'uso.

"Immagina di aver creato un'applicazione YOLO che rileva giocatori e palloni da calcio da una determinata immagine.

Ma come spiegare questo processo a qualcuno, soprattutto alle persone non iniziate?

→ Questo è il punto centrale di questa sezione. Comprenderai l'intero processo di come YOLO esegue il rilevamento degli oggetti; come ottenere l'immagine (B) dall'immagine (A)”


(A) Input image
(B) YOLO Algorithm result
Player
Player
Player
Ball
Zoumana KEITA

L’algoritmo funziona sulla base dei seguenti quattro approcci:

Blocchi residui

Regressione del riquadro di delimitazione

Intersection Over Unions o IOU in breve

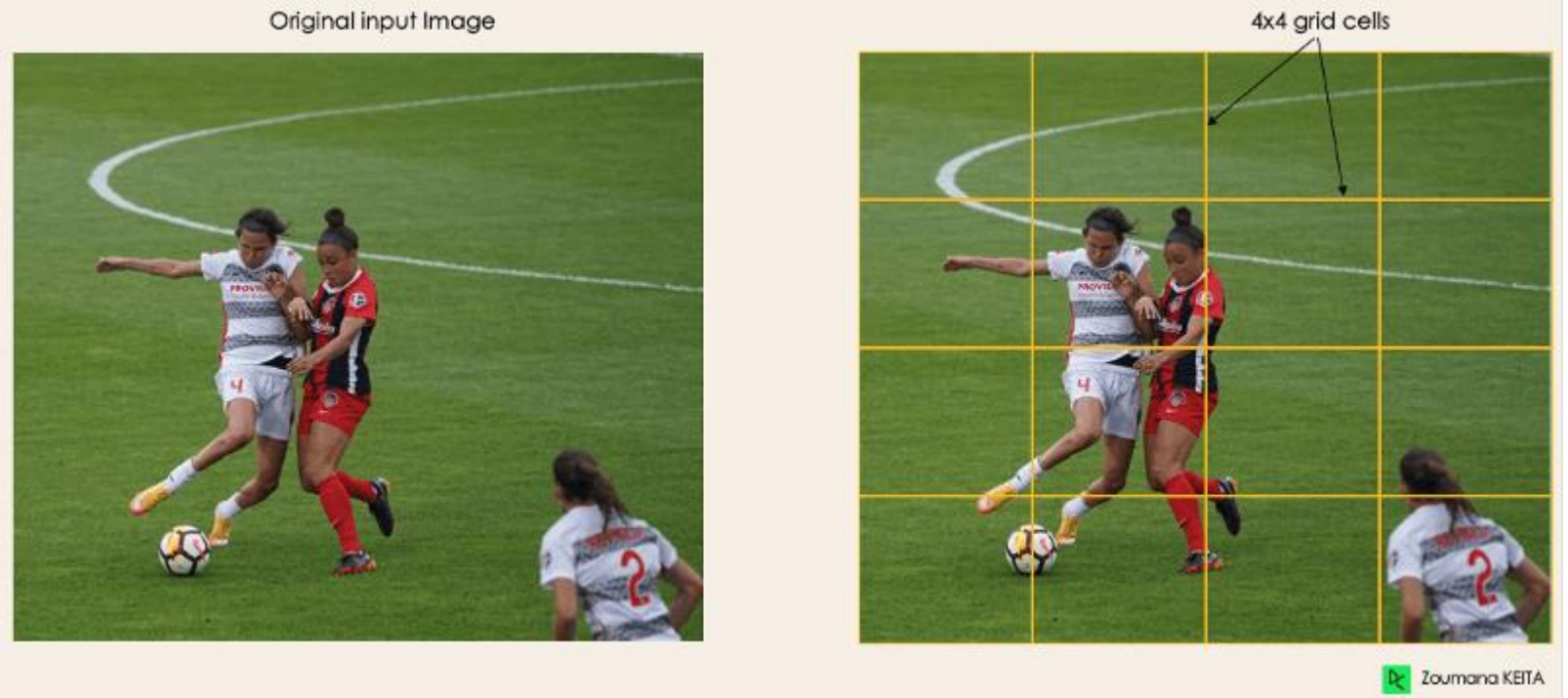
Soppressione non massima.

1- Blocchi residui

Questo primo passaggio inizia dividendo l'immagine originale (A) in NxN celle della griglia di uguale forma, dove N nel nostro caso è 4 mostrato nell'immagine a destra.

Ogni cella della griglia è responsabile della localizzazione e della previsione della classe dell'oggetto che copre, insieme al valore di probabilità/confidenza.

1- Blocchi residui



2- Regressione del riquadro di delimitazione

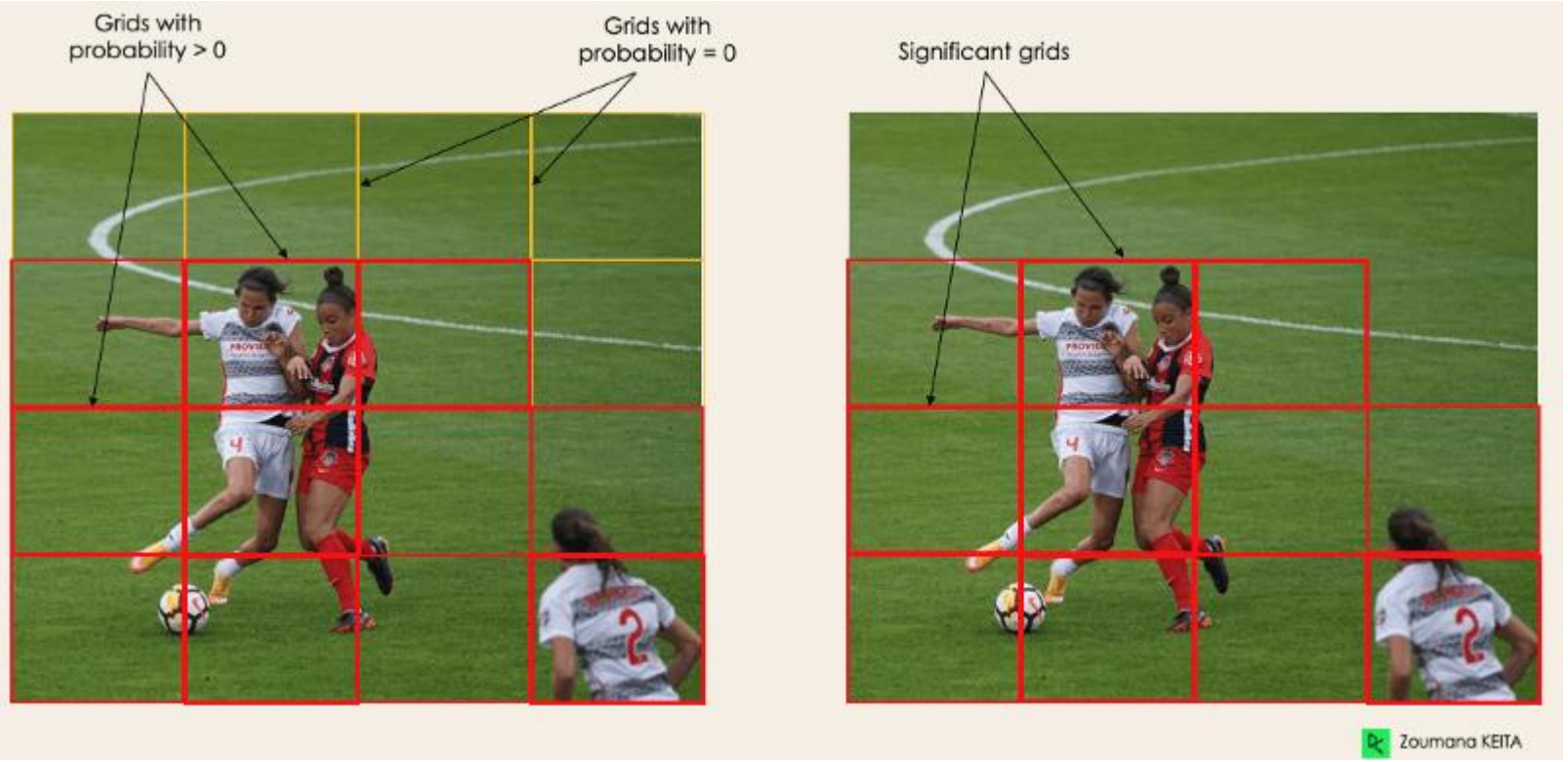
Il passo successivo è determinare i riquadri di delimitazione che corrispondono ai rettangoli che evidenziano tutti gli oggetti nell'immagine. Possiamo avere tanti riquadri di delimitazione quanti sono gli oggetti all'interno di una data immagine.

YOLO determina gli attributi di questi riquadri di delimitazione utilizzando un singolo modulo di regressione nel seguente formato, dove Y è la rappresentazione vettoriale finale per ciascun riquadro di delimitazione.

Y = [pc, bx, di, bh, bw, c1, c2]

Ciò è particolarmente importante durante la fase di training del modello.

- pc corrisponde al punteggio di probabilità della griglia contenente un oggetto. Ad esempio, tutte le griglie in rosso avranno un punteggio di probabilità superiore a zero. L'immagine a destra è la versione semplificata poiché la probabilità di ciascuna cella gialla è zero (insignificante).

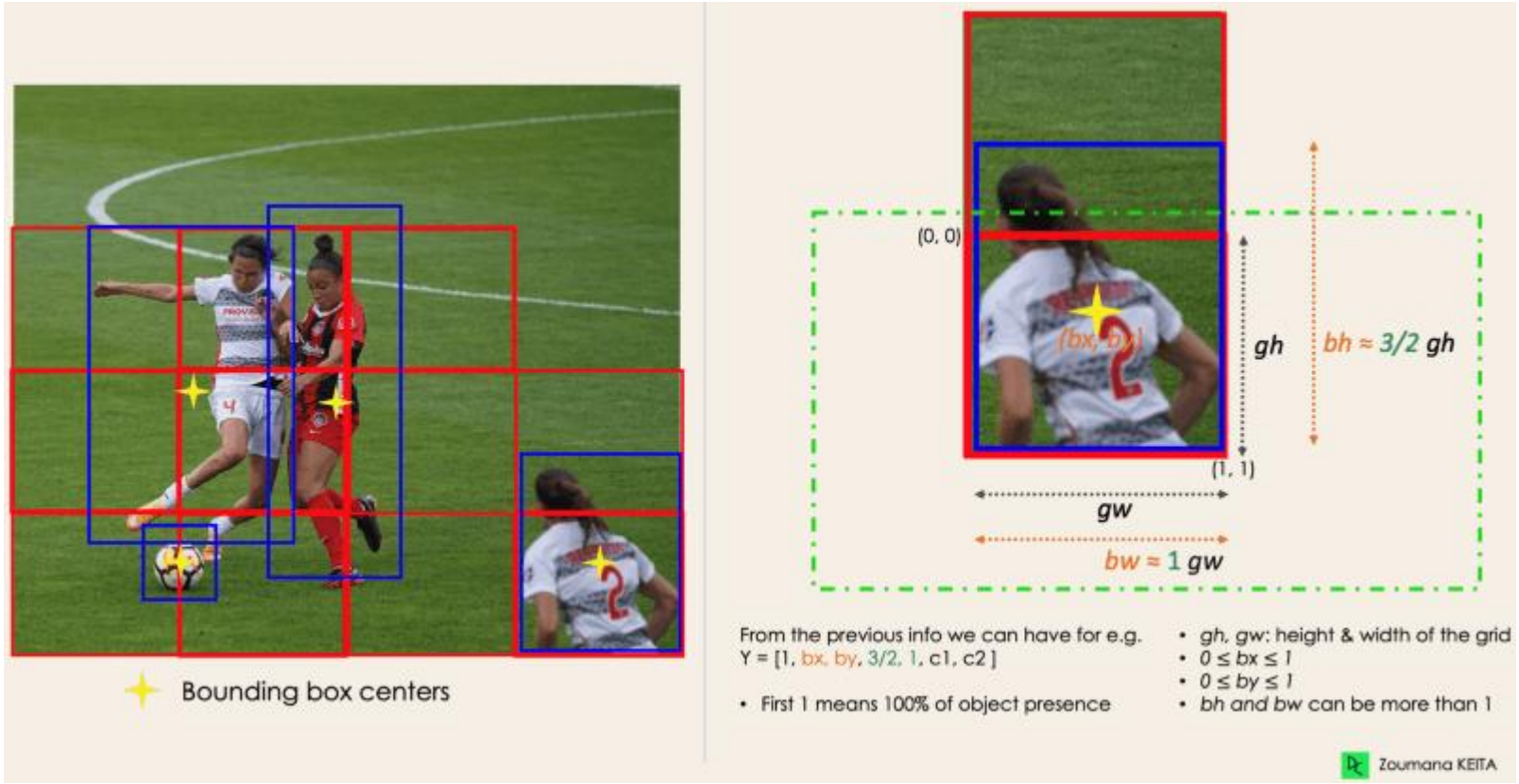

Grids with probability > 0
Grids with probability = 0
Significant grids
Zoumana KEITA

bx, by sono le coordinate xey del centro del riquadro di delimitazione rispetto alla cella della griglia avvolgente.

bh, bw corrispondono all'altezza e alla larghezza del riquadro di delimitazione rispetto alla cella della griglia avvolgente.

c1 e c2 corrispondono alle due classi Player e Ball. Possiamo avere tutte le classi richieste dal tuo caso d'uso.

Per capirci prestiamo più attenzione al giocatore in basso a destra.



3- Intersezione sulle unioni o pagherò

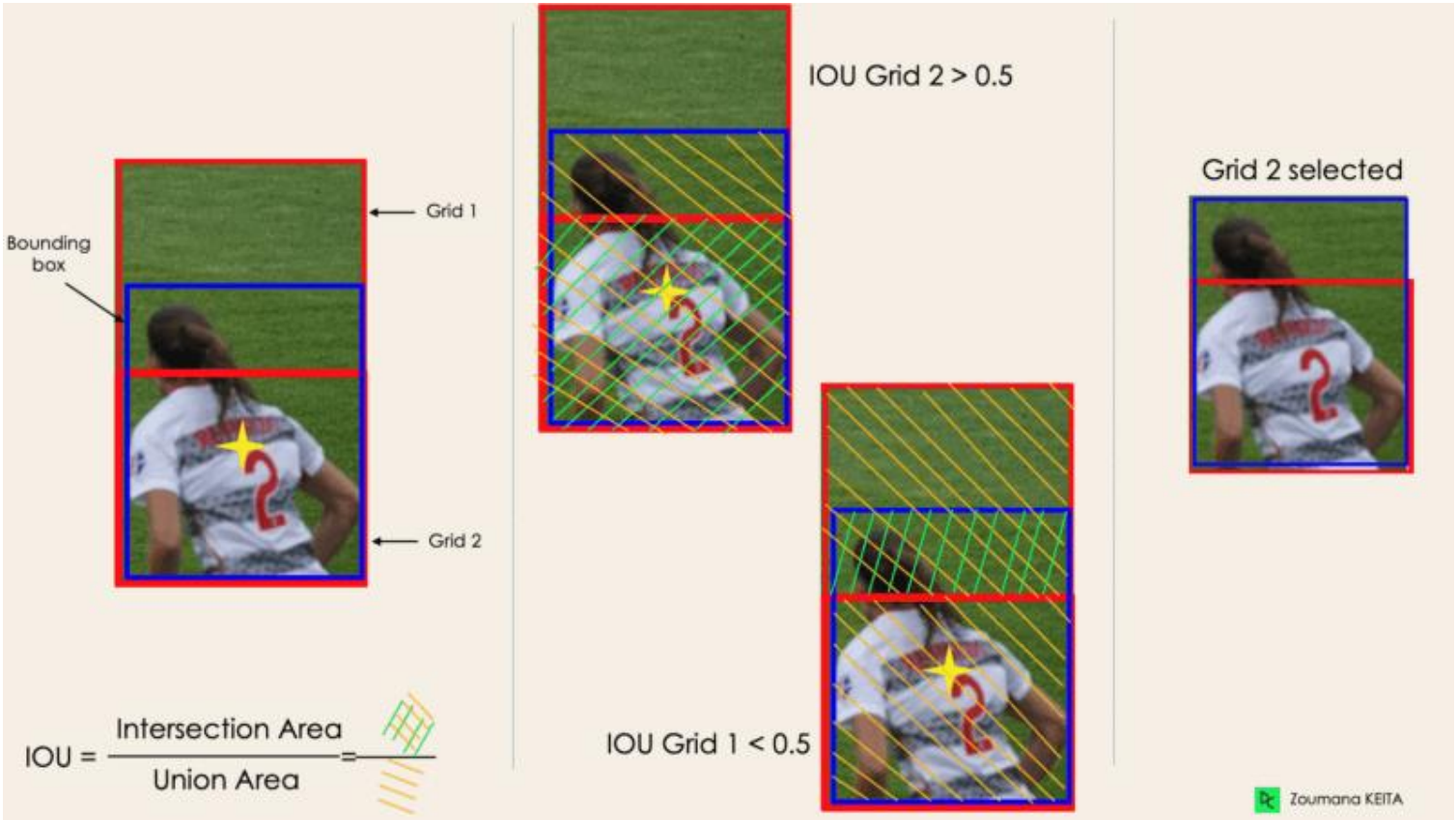
Nella maggior parte dei casi, un singolo oggetto in un'immagine può avere più caselle della griglia candidate per la previsione, anche se non tutte sono rilevanti. L'obiettivo dell'IOU (un valore compreso tra 0 e 1) è quello di scartare tali caselle della griglia per mantenere solo quelle rilevanti. Ecco la logica dietro:

L'utente definisce la soglia di selezione dell'IOU, che può essere, ad esempio, 0,5.

Quindi YOLO calcola l'IOU di ciascuna cella della griglia che è l'area di intersezione divisa per l'area di unione.

Infine, ignora la previsione delle celle della griglia aventi soglia IOU ≤ e considera quelle con soglia IOU >.

Di seguito è riportata un'illustrazione dell'applicazione del processo di selezione della griglia all'oggetto in basso a sinistra. Possiamo osservare che l'oggetto originariamente aveva due candidati alla griglia, quindi alla fine è stata selezionata solo la “Griglia 2”.


Bounding box
Grid 1
Grid 2
IOU = Intersection Area / Union Area
IOU Grid 2 > 0.5
IOU Grid 1 < 0.5
Grid 2 selected
Zoumana KEITA

4- Soppressione non massima o NMS

L'impostazione di una soglia per l'IOU non è sempre sufficiente perché un oggetto può avere più caselle con IOU oltre la soglia e lasciare tutte quelle caselle potrebbe includere rumore.

È qui che possiamo utilizzare NMS per conservare solo le scatole con il punteggio di probabilità di rilevamento più alto.

Applicazioni YOLO

Applicazione nelle industrie

Assistenza sanitaria

agricoltura

Sorveglianza di sicurezza

Auto a guida autonoma

Il rilevamento degli oggetti in tempo reale fa parte del DNA dei sistemi di veicoli autonomi. Questa integrazione è vitale per i veicoli autonomi perché devono identificare correttamente le corsie corrette e tutti gli oggetti e i pedoni circostanti per aumentare la sicurezza stradale. L'aspetto in tempo reale di YOLO lo rende un candidato migliore rispetto ai semplici approcci di segmentazione delle immagini.

Addestramento Yolo

Per addestrare il nostro rilevatore di oggetti, dobbiamo supervisionare il suo apprendimento con annotazioni del riquadro di delimitazione. Disegniamo un riquadro attorno a ciascun oggetto che vogliamo che il rilevatore veda ed etichettiamo ogni riquadro con la classe di oggetto che vorremmo che il rilevatore prevedesse.

Strumenti di etichettatura:

CVAT: https://blog.roboflow.ai/getting-started-with-cvat/

LAbelImg: https://blog.roboflow.ai/getting-started-with-labelimg-for-labeling-object-detection-data/

CoTT: https://blog.roboflow.com/vott/

Mentre disegni i riquadri rilegati, assicurati di seguire le migliori pratiche:

Etichetta tutto intorno all'oggetto in questione

Etichettare interamente gli oggetti occlusi

Evitare troppo spazio attorno all'oggetto in questione