Politecnico di Torino

Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Informatica

A.a. 2024/2025

Sessione di laurea luglio 2025

Locality-Sensitive Hashing per Open Vocabulary Multi-Object Tracking efficiente

Relatori:

Prof.ssa Lia Morra

Dott. Luca Angioloni

Candidato:

Pietro Siliani

Sommario

I recenti sviluppi nei modelli multimodali, come CLIP, hanno abilitato il paradigma di *open-vocabulary object detection*, in cui gli oggetti possono essere individuati e classificati a partire da una *query* testuale, superando il vincolo di aderenza ad un insieme fisso di categorie. L'applicazione di queste tecnologie in contesti di *tracking* ha portato alla nascita di sistemi *open vocabulary multi-object tracking* (OV-MOT). Tuttavia, la complessità computazionale tipica di questi modelli ne ostacola l'impiego in scenari *real-time*. Il lavoro presentato ha come obiettivo la progettazione, implementazione e valutazione di una soluzione di OV-MOT in tempo reale. Il sistema proposto alleggerisce la fase di ricostruzione delle traiettorie, sostituendo le misure di similarità tradizionali, come *cosine similarity* o meccanismi di *attention*, con la distanza di Hamming tra rappresentazioni binarie apprese appositamente. L'architettura del *tracker* comprende una *embedding net*, una *hashing net* e un modulo di associazione e ricostruzione delle traiettorie. L'*embedding net* è costituita da una *Feature Pyramid Network*, che estrae dei *region embedding* per ciascuna predizione ottenuta da una *backbone* di *open vocabulary object-detection* (OV-OD) preaddestrata. La *hashing net* implementa una funzione di *learnable locality-sensitive hashing* (LLSH), che comprime e quantizza i *region embedding* in *region encoding* binari. L'associazione temporale tra le istanze è infine realizzata confrontando le codifiche binarie mediante distanza di Hamming.
La rete è addestrata sul *training set* di BURST, tramite una combinazione di *triplet loss* e *quantization loss*, con l'obiettivo di garantire la robustezza degli *encoding* prodotti.
Attraverso l'integrazione con una *backbone* di OV-OD efficiente, il *tracker* riesce a raggiungere prestazioni di OV-MOT *real-time*. I risultati sperimentali mostrano che il sistema, denominato BEHEMOTH (***B**inary **E**ncodings by **H**ashing **E**very object for **M**ulti-**O**bject **T**racking through **H**amming-based comparison*), risulta il modello *open vocabulary* più rapido tra le soluzioni attualmente disponibili. Sotto opportune condizioni, BEHEMOTH riesce a raggiungere prestazioni *zero-shot* alla pari dello stato dell'arte. Ciò evidenzia la capacità discriminativa dei *region encoding* ottenuti tramite LSH, e dimostra l'applicabilità di queste tecniche a contesti di *tracking*.

Indice

Introduzione

Obiettivo

L'*object-tracking* rappresenta una tecnica di fondamentale importanza in diversi ambiti, quali videosorveglianza, *automotive*, analisi del traffico e realtà aumentata. La sua funzione è quella di individuare e seguire uno o più oggetti all'interno di una sequenza di immagini, tipicamente rappresentata da un flusso video. Tale processo implica il riconoscimento e l'aggiornamento continuo della posizione di ciascun oggetto man mano che esso si muove all'interno della scena.

Molte delle recenti soluzioni di *object-tracking* basate su reti neurali adottano il paradigma denominato *tracking-by-detection* che prevede la scomposizione del compito di tracciamento in due fasi:

1. Identificazione degli oggetti presenti nel frame corrente.
2. Ricerca delle corrispondenze degli oggetti rispetto a quelli specificati in una *query* o presenti nel frame precedente.

Nel caso del *multi-object tracking*, che richiede di monitorare più oggetti contemporaneamente, la seconda fase può raggiungere una complessità quadratica. Ciò deriva dalla necessità di verificare la similarità, attraverso opportune misure, di ogni possibile coppia *query-target*. L'elevata complessità limita l'utilizzo di tecniche simili in sistemi di *edge computing*, spesso adottati in ambiti quali sicurezza o controllo di processo industriale.

Questa tesi, svolta presso **Aidia s.r.l.**, si propone di snellire il processo di ricerca delle corrispondenze utilizzando una tecnica di *dimensionality reduction* nota come *locality-sensitive hashing* (LSH). Le funzioni di LSH permettono di ricavare, a partire da dati ad elevata dimensionalità, delle rappresentazioni compatte che, pur comportando un'inevitabile perdita d'informazione, cercano di preservare le distanze tra i diversi campioni. Sono talvolta utilizzate per ottenere dei codici binari a partire dai vettori di *feature* ricavati da operazioni di *feature extraction*, tentando di lasciare inalterati gli aspetti semantici che essi modellano.

Il lavoro, inoltre, si concentra sulla progettazione di un sistema *open vocabulary*, ovvero agnostico rispetto all'insieme di categorie su cui è addestrato, ed in grado di generalizzare a classi non presenti nel *training set*. Il vantaggio di questo paradigma è quello di permettere la realizzazione di un sistema rapidamente riconfigurabile ed applicabile in numerosi contesti differenti senza necessità di ripetere l'intera *pipeline* di addestramento da zero.

Il sistema sviluppato, denominato BEHEMOTH (***B**inary **E**ncodings by **H**ashing **E**very*

*object for **M**ulti-**O**bject **T**racking through **H**amming-based comparison*) rappresenta una pipeline di *open-vocabulary multi-object tracking* end-to-end, ottimizzata per l'utilizzo di LSH. Integrando *feature extraction*, *region proposal*, *object detection* e *tracking*, BEHEMOTH massimizza l'efficacia delle tecniche di LSH e riesce a raggiungere prestazioni *real-time*.

I principali contributi del lavoro presentato possono essere riassunti come segue:

i) Progettazione ed implementazione di BEHEMOTH, un sistema di *open-vocabulary multi-object tracking real-time* che integra un modulo di LSH per la quantizzazione dei *feature embedding* delle regioni tracciate.

ii) Sviluppo di un meccanismo di associazione tra istanze basato su distanza di Hamming, che sostituisce le metriche tradizionali, riducendo i costi computazionali nella fase di *tracking*.

iii) Addestramento supervisionato sul *training set* di BURST, mediante una combinazione di *triplet loss* e *quantization loss*, per favorire la separazione dei *feature embedding* nello spazio vettoriale di destinazione.

iv) Validazione sperimentale su *benchmark* standard, che mostra vantaggi e svantaggi della strategia di *tracking* proposta.

Struttura della tesi

Il presente elaborato è strutturato come segue. Nel **Capitolo 1** viene fornita una rassegna della letteratura riguardante le principali tematiche affrontate, tra cui *open* e *closed-vocabulary object tracking*, *open-vocabulary object detection*, ed alcune applicazioni delle funzioni di *locality-sensitive hashing*. Il **Capitolo 2** introduce i concetti teorici alla base del lavoro, include una panoramica sulle principali architetture utilizzate in contesti di *object-detection* e *tracking*, definisce il funzionamento di CLIP, un modello multimodale fondamentale per l'implementazione di soluzioni *open-vocabulary*, e si sofferma sulla formulazione e le caratteristiche delle funzioni di LSH e dell'algoritmo ungherese. Il **Capitolo 3** descrive in dettaglio il metodo proposto: la formulazione del problema di *tracking*, l'architettura di BEHEMOTH, le strategie di addestramento adottate e la *pipeline* di *benchmarking*. Il **Capitolo 4** presenta gli esperimenti condotti durante lo sviluppo del sistema, evidenziando le scelte che hanno portato all'architetture finale di BEHEMOTH, confronta inoltre le prestazioni di diverse soluzioni di *multi-object tracking* e mostra alcuni risultati qualitativi, inquadrando BEHEMOTH all'interno del panorama attuale,. Il **Capitolo 5** contiene analisi ablative supplementari, che contestualizzano e giustificano la scelta di alcuni parametri ed iperparametri della rete. Infine, il **Capitolo 6** conclude il lavoro discutendo i risultati ottenuti, i principali limiti osservati e le possibili direzioni per sviluppi futuri.

Capitolo 1

Stato dell'arte

1.1 Object Tracking

Il riconoscimento e il tracciamento di persone e oggetti in ambienti reali attraverso tecniche di *image processing* e *computer vision* - oggi basate quasi esclusivamente su algoritmi di *deep learning* - rappresentano una componente fondamentale per compiti critici in diversi ambiti, quali il settore *automotive*, l'automazione industriale, il controllo qualità o la sicurezza. Tali applicazioni richiedono spesso la capacità di gestire scene affollate, dinamiche ed eterogenee.
Un'applicazione particolarmente rilevante è l'*object tracking*, in cui, dato uno o più flussi video, è necessario ricostruire la traiettoria degli oggetti che si muovono all'interno del campo visivo, talvolta classificando gli oggetti individuati. Nel caso in cui l'obiettivo sia ricostruire la traiettoria di un singolo oggetto all'interno dell'intera sequenza video si parla di *single object-tracking* (SOT), mentre quando si trattano molteplici traiettorie relative ad oggetti diversi si ha a che fare col *multi object-tracking* (MOT). Data la vastità di ambiti applicativi e le relative sfide, numerosi studi si sono occupati di realizzare sistemi SOT e MOT robusti ed efficienti, tra di essi si annoverano:

- **Single-Object Tracking**
  - GOTURN [1] e Siamese-FC [2] sono tra i primi sistemi che realizzano *single-object tracking* attraverso una CNN addestrata completamente off-line. La principale differenza tra i due consiste nell'area di ricerca dell'oggetto obiettivo: mentre GOTURN addotta una strategia locale, cercando l'oggetto in un intorno dell'ultima posizione nota, Siamese-FC sfrutta una densa griglia di *bounding box* che copre l'intera area dell'immagine, risultando più robusto rispetto ad oggetti dal moto dinamico ed irregolare.
  - Siamese-RPN [3] perfeziona la selezione delle regioni candidate introducendo un modulo di *region proposal* dedicato, addestrato congiuntamente al *tracker*. In particolare, un frame di riferimento contenente l'oggetto di interesse è utilizzato per precalcolare due kernel convoluzionali: ϕ_{cls} e ϕ_{reg}. Successivamente, per ogni frame della sequenza video, è definito un insieme di k *bounding box* di riferimento, chiamate ancore, ed una CNN estrae una *feature map* dal frame di interesse. Quest'ultima partecipa ad una convoluzione con ϕ_{cls} e ϕ_{reg}, i cui risultati sono rispettivamente una vettore di $2k$ canali, che riporta per ciascuna delle ancore un punteggio di *foreground* e uno di *background*, ed un vettore di $4k$ canali, che indica lo spostamento ed il fattore di scala da applicare a ciascuna

ancora. Infine una *pipeline* di *post-processing* seleziona la *bounding box* migliore tra quelle classificate come *foreground*.

- In [4] si individuano ed analizzano le problematiche che impediscono l'addestramento di *tracker* basati su reti profonde più recenti. Una di queste è l'introduzione del *padding* da parte delle reti in questione, che rende le *feature map* ricavate sensibili alla posizione dell'oggetto da identificare. La contromisura adottata consiste in una tecninca di *data augmentation* che applica delle traslazioni casuali alle immagini del *training set*, evitando così che l'oggetto cercato occupi sempre una posizione centrale. Inoltre lo studio dimostra l'efficacia dell'aggregazione di *feature map* prodotte da layer a profondità differente ed introduce la tecnica di *depthwise cross-correlation*, che stablizza ulteriormente l'addestramento. Ciò ha permesso la realizzazione di un sistema con ottime prestazioni che utilizza ResNet-50 [5], ed una sua versione più efficiente tramite MobileNetv2 [6].
- Più recentemente SAMURAI [7] unisce le maschere di segmentazione ottenute da SAM2 [8, 9] con un modello di moto basato su filtro di Kalman [10]. In particolare, SAM2 sfrutta un *memory encoder* per produrre *embedding* vettoriali compatti, chiamati **memorie**, a partire da una maschera di segmentazione, e costruisce una *memory bank* sotto forma di una coda FIFO, formata dalle N memorie più recenti. Le memorie sono poi combinate con le *feature* estratte da un *image encoder* attraverso un meccanismo di *cross-attention*, in modo da produrre una maschera di segmentazione più fedele all'oggetto da tracciare. SAM2, tuttavia, non tiene conto dei fenomeni di occlusione e della velocità di movimento degli oggetti durante la costruzione della propria *memory bank*. SAMURAI affronta il problema calcolando, per ogni maschera, un *object score* s_{obj} ed un *motion score* s_{kf}, oltre che un *mask score* s_{mask} già previsto da SAM2. Un frame è aggiunto alla *memory bank* solo se tutti e tre i punteggi calcolati superano delle soglie di riferimento. Questa procedura irrobustisce la *memory bank* e consente di aumentare le prestazioni di *tracking* rispetto a SAM2 senza necessità di riaddestrare il sistema.

- **Multi-Object Tracking**
  - SORT [11] impiega un filtro di Kalman per prevedere il moto degli oggetti individuati, costruendo le traiettorie sulla base di assegnamenti *frame-to-frame*. Dato un insieme di *target*, definiti da una *bounding box* ed una velocità di movimento, l'algoritmo sfrutta queste informazioni per predirne gli spostamenti. Le regioni predette sono poi messe in corrispondenza con quelle individuate da una Faster-RCNN [12] sul frame successivo. L'assegnamento avviene cercando di massimizzare il punteggio di Intersection-over-Union tra le regioni predette dalla rete e quelle stimate tramite filtro di Kalman. Per farlo si utilizza l'Algoritmo Ungherese [13] in modo da individuare l'ottimo globale ad ogni fase di assegnamento.
  - DeepSort [14] estende SORT impiegando una *Convolutional Neural Network* (CNN) per estrarre *feature* di re-identificazione (ReID), irrobustendo il sistema rispetto alle occlusioni temporanee dei soggetti. Le *feature* di ReID sono organizzate in vettori reali di dimensione pari a 128, per i quali si utilizza la distanza coseno come misura di similarità. Durante la fase di assegnamento, DeepSort sfrutta sia informazioni di moto, derivanti dal filtro di Kalman già impiegato in SORT, sia informazioni visive, rappresentate dalle *feature* di ReID, e calcola un

punteggio di similarità tra traiettorie e nuove predizioni che dipende da entrambi i fattori.

- Il sistema presentato in [15] effettua la segmentazione dei soggetti attraverso Mask-RCNN. La ricostruzione delle traiettorie avviene associando le maschere di frame consecutivi cercando di massimizzarne il punteggio di similarità, che dipende dalla loro intersezione, dalle dimensioni di ciascuna maschera e dalla distanza di Hamming tra le maschere candidate, calcolata a partire da *embedding* binari. Questi ultimi sono ottenuti attraverso una funzione di *Learnable Locality-Sensitive Hashing* (LLSH) (1.4) appresa in modalità *on-line*.

- QDTrack [16] è addestrato insieme alla propria *backbone* di *object detection* e sfrutta un *embedding head*, congiuntamente ad una funzione di loss specificamente progettata, per produrre *embedding* maggiormente rappresentativi degli oggetti da tracciare. La rete è addestrata su coppie di frame temporalmente ravvicinati, da cui sono estratte numerose *bounding box* attraverso una *Region Proposal Network* (RPN). Questo approccio permette di avere un elevato numero di campioni di addestramento, indipendentemente dalla quantità di frame che compone il *dataset*. Infine, l'*embedding head* elabora ciascuna regione individuata per ricavarne una rappresentazione vettoriale, e la ricostruzione delle traiettorie avviene attraverso una fase di *nearest neighbour search*, che utilizza una funzione di *bi-directional softmax* per calcolare la similarità tra i candidati.

- ByteTrack [17] raffina l'algoritmo di tracking separandolo in due fasi: nella prima viene effettuato l'assegnamento delle regioni con confidenza più elevata, combinando le *feature* di ReID con un filtro di Kalman, mentre nella seconda si considerano le regioni rimanenti, utilizzando esclusivamente le predizioni effettuate dal filtro.

- TETer [18] sfrutta un addestramento basato su *class exemplar matching* (CEM) per avvicinare gli *embedding* di regioni rappresentanti un oggetto della stessa categoria. Inoltre, lo studio introduce la metrica Track Everything Accuracy (TETA), con cui le prestazioni di un *tracker* sono valutate calcolando separatamente i punteggi relativi alla classificazione, localizzazione ed associazione degli oggetti, per poi essere condensate in un unico parametro di valutazione. TETA è diventata la metrica standard per la valutazione di sistemi MOT che lavorano con numerose categorie, siano essi *open* o *closed vocabulary*.

Come QDTrack e ByteTrack anche BEHEMOTH utilizza un *embedding head* per la produzione dei descrittori delle regioni individuate, ma sostituice le rappresentazioni vettoriali con codici binari compatti. Inoltre, a differenza di [15], l'addestramento della rete avviene in modalità completamente *off-line*. L'algoritmo di *tracking* di BEHEMOTH mutua la suddivisione tra regioni a bassa ed alta confidenza proposta da ByteTrack, ma elimina l'impiego del filtro di Kalman, limitandosi ad utilizzare le *bounding box* individuate nei frame precedenti. Ciò è motivato dall'eterogeneità dei soggetti considerati all'interno del panorama *open vocabulary*, che rispondono in modo diverso ai modelli di moto stimati dal filtro, limitandone l'efficacia in scene complesse [19].

1.2 Open Vocabulary Object Detection

Sebbene l'*object tracking* sia un compito distinto dall'*object detection*, che prevede l'individuazione e classificazione degli oggetti di interesse all'interno di un'immagine, spesso è desiderabile realizzare sistemi in grado di tracciare e classificare simultaneamente gli oggetti osservati all'interno di un flusso video. Per molto tempo il paradigma di *object detection* ha previsto esclusivamente l'utilizzo di insiemi di categorie statici, attraverso i quali è possibile addestrare le reti a riconoscere un numero limitato di classi. Recentemente, tuttavia, lo sviluppo di sistemi come CLIP [20] e ALIGN [21], addestrati a compiti di *image-to-text* e *text-to image association* ha aperto le porte a modelli *open vocabulary* in cui le categorie di oggetti trattabili non sono definite a priori, ma ottenute a partire da *query* testuali. Tali sistemi si basano su una combinazione di *Convolutional Neural Networks* (ResNet, [5] in particolare), *Transformers* [22] e la loro estensione all'ambito delle immagini (i Visual Transformers o ViT, [23, 24]) per ottenere delle rappresentazioni di testo ed immagini all'interno di uno spazio vettoriale comune, e calcolarne la similarità attraverso misure quali *cosine similarity* (similarità del coseno) o distanza euclidea.
Numerosi studi hanno applicato questa tecnologia per affrontare compiti di *object detection*, si parla in tal caso di *open vocabulary object detection* (OV-OD). Tra di essi:

- RegionCLIP [25] attraversa una fase di *pre-training* come modello *student* di CLIP. Nello specifico, la procedura prevede la definizione di una serie di *concept*, ovvero brevi didascalie che includono i nomi delle classi da identificare, generate a partire da *template* predefiniti. Successivamente, le regioni individuate da una RPN sono associate ad un *concept* utilizzando CLIP. Un *visual encoder* è addestrato tramite *knowledge distillation* a riprodurne i risultati. Gli autori evidenziano che, sebbene le coppie regione-*concept* individuate da CLIP siano rumorose, in quanto il modello non è pensato per *object detection*, la *pipeline* di addestramento permette a RegionCLIP di ottenere un'accuratezza notevole, che può essere migliorata tramite *fine-tuning* su dataset specifici.

- OWL-ViT [26] addestra in modo end-to-end sia *text encoder* che *image encoder* e sfrutta dataset pubblici in tutte le fasi di addestramento. Il *pre-training* avviene su dataset formati da coppie testo-immagine. Attraverso una *contrastive loss* i due *encoder* imparano a generare *embedding* simili per immagini e testo corrispondenti. In seguito un *fine-tuning* su un *ensemble* di dataset pensati per *object detection* permette al modello di raggiungere notevole accuratezza e capacità di generalizzazione. A differenza di altri modelli *open vocabulary*, durante il *fine-tuning* OWL-ViT calcola gli embedding testuali separatamente per ciascuna immagine, invece di precalcolarli per l'intero dataset. In particolare, tali *embedding* vengono generati a partire da un sottoinsieme di categorie che include tutte le classi effettivamente presenti nell'immagine, oltre ad alcune classi assenti, definite "negative" o "pseudo-negative".

- GLIP [27] tratta il compito di *object-detection* come un problema di *grounding* in cui, dato un *prompt* testuale contenente una serie di termini corrispondenti agli oggetti da individuare, si cerca di associare ciascuna regione dell'immagine ad uno di essi. L'addestramento è diviso in due fasi, durante la prima il modello è addestrato ad associare le regioni di un immagine a frasi predeterminate. I pesi ottenuti sono utilizzati come punto di partenza per un addestramento *self-supervised*, che sfrutta milioni di immagini provenienti dal *web*, comprensive di didascalie. Un *parser* apposito estrae i sostantivi dalle didascalie fornite, che sono poi utilizzati da GLIP

per individuare le regioni corrispondenti. Le coppie regione-sostantivo identificate sono utilizzate come riferimento per l'addestramento di un modello *student*. A livello architetturale, gli autori evidenziano l'efficacia del modulo denominato *cross-modality multi-head attention*, utilizzato per effettuare operazioni di *cross-attention* tra gli *embedding* testuali e quelli visivi, in modo da fondere, fin dai primi *layer* degli *encoder* corrispondenti, le due tipologie di informazioni.

- Grounding-DINO [28, 29] riprende la formulazione di *object detection* come *grounding* di GLIP. Rielabora inoltre la procedura di estrazione dei *text embedding* a partire dalle *query* testuali, utilizzando apposite maschere che bloccano le operazioni di *attention* tra parole semanticamente incorrelate. Infine, come GLIP, integra un proprio modulo di *cross-attention* tra *token* testuali e visivi denominato *feature enhancer*.

- OmDet [30] e la sua evoluzione OmDet-Turbo [31] affrontano un *pre-training* su una vasta gamma di dataset per *closed vocabulary object-detection* e *grounding* riuscendo a generalizzare e a raggiungere capacità di riconoscimento *open vocabulary*. OmDet distingue il *prompt* testuale, contenente il tipo di operazione da effettuare (e. g. "detect") dalla *query*, costituita dall'insieme dei nomi delle classi da individuare. Velocizza inoltre la fusione di *embedding* testuali e visivi, attraverso un'architettura *encoder-decoder* appositamente progettata, i cui moduli, indicati dagli autori come *Efficient Language Aware* (ELA), eseguono *self-attention* sulle *feature map* e *cross-attention* tra *token* testuali e visivi. Attraverso l'adozione di ottimizzazioni quali *flash-attention* ([32], e la riduzione del numero di passi di *cross-attention* rispetto a Grounding-DINO, il modello raggiunge un grado di efficienza tale da permettergli di operare in tempo reale.

- YOLO-world [33] è un *tracker real-time* che integra l'architettura di YOLOv8 [34, 35, 36, 37] con il *text-encoder* di CLIP. La *Path Aggregation Network* (PAN) di YOLOv8 è rielaborata per permettere la fusione di *feature* testuali e visive. La RepVL-PAN risultante utilizza dei *text-guided cross-stage partial layers* (*text-guided CSPLayers*) per pesare opportunamente gli elementi delle *feature-map* in base al contenuto della *query* testuale, attraverso l'impiego di *max-sigmoid attention*. Inoltre, i moduli di *image-pooling attention* sono utilizzati per aggiornare gli *embedding* testuali. Essi prevedono il sottocampionamento delle *feature map* attraverso *max-pooling*, e la fusione dei *token* risultanti con le informazioni testuali tramite *cross-attention*.

BEHEMOTH è stato pensato come *tracker* agnostico rispetto al modulo di *object-detection* impiegato, può quindi essere aggiunto come *tracking head* ad una qualsiasi delle reti appena descritte. Tuttavia, l'esigenza di prestazioni *real-time*, ha ristretto le reti candidate a OmdetTurbo-Tiny e YOLO-world. A queste reti è stata aggiunta RegionCLIP, in quanto costituisce una delle *backbone* più utilizzate per i sistemi di *tracking open vocabulary* [19, 38].

1.3 Open Vocabulary Multi-Object Tracking

L'*open vocabulary multi-object tracking* (OV-MOT) nasce dall'applicazione di tecniche OV-OD in contesti MOT, in cui è necessario non solo distinguere e classificare correttamenente le diverse regioni di ciascun frame, ma anche ricostruirne le traiettorie riuscendo a gestire scene dense, ricche di occlusioni e di elementi eterogenei. In ambito OV-MOT le tecnologie

proposte [19, 18, 39, 38] sfruttano un paradigma di *tracking-by-detection* affidandosi ad una *backbone* preposta ad un compito di OV-OD sui singoli frame, a cui si aggiunge un modulo di *tracking* in grado di associare gli oggetti individuati in frame consecutivi e ricostruirne la traiettoria. In particolare:

- OvTrack [19] addestra il proprio *tracker* su un dataset di immagini. Per ciascuna di esse viene generato un duale attraverso tecniche di *data hallucination*, ed il sistema è addestrato ad associare gli oggetti corrispondenti tra le due versioni dell'immagine. La rete integra un *text encoder* ed un *image encoder* addestrati in modo supervisionato a partire dagli *encoder* di CLIP. Durante l'inferenza le traiettorie sono ricostruite assegnando ciascuna predizione al rappresentante di ciascuna traiettoria sulla base della similarità tra i relativi *region embedding*.

- GLEE [39] è un *foundation* model addestrato su milioni di immagini che eccelle in una moltitudine di applicazioni tra cui OV-MOT, tra i sistemi strettamente *open vocabulary* è quello che ottiene il miglior punteggio TETA sul dataset TAO. Gli autori osservano come, addestrando su un numero sufficiente di immagini, la *backbone* di *object detection* riesca a produrre *region embedding* sufficientemente discriminativi da poter essere direttamente utilizzati nel *tracking*, senza bisogno di una *tracking head* apposita. Inoltre, per rendere il *tracker* più robusto, sfruttano una *contrastive loss* addestrando il sistema su coppie di frame provenienti da diversi video. Questo approccio avvicina le rappresentazioni di oggetti simili nello spazio vettoriale di destinazione, ed allontana quelle di oggetti incorrelati.

- AED [38] sfrutta un *sim-decoder* basato su *multi-head deformable cross-attention* [40] per arricchire le *feature* estratte da una *backbone* preaddestrata. Durante l'apprendimento il *sim-decoder* è incentivato a massimizzare la similarità tra gli *embedding* delle istanze di uno stesso oggetto in fotogrammi differenti attraverso una *contrastive loss* apposita. Il sistema è addestrato su *batch* formati da clip di pochi frame. Per ciascuna clip la *contrastive loss* è ottenuta come somma pesata di una *loss spaziale*, che coinvolge solo le predizioni di uno stesso frame, di una *loss temporale*, che è applicata alle istanze individuate in frame consecutivi, e di una *loss cross-clip*, calcolata a partire da tutte le predizioni individuate nella clip. Durante l'inferenza il *sim-decoder* riceve in input delle *object query* e delle *track query*. Le prime sono date dalle predizioni della *backbone* di *object detection* sul frame corrente. Le *track query* corrispondono invece ai rappresentanti di ciascuna traiettoria individuata. Il *sim-decoder* calcola un punteggio di similarità $S_{temporal}$ tra ogni coppia. L'assegnamento ottimo è ottenuto attraverso l'algoritmo ungherese, massimizzando il punteggio di similarità per ogni coppia.

BEHEMOTH sostituisce gli *embedding* vettoriali utilizzati da OvTrack con delle rappresentazioni binarie. Inoltre, è addestrata interamente su un dataset video, e non effettua alcun *fine-tuning* della *backbone* e del *text encoder*. Durante l'addestramento, la suddivisione del dataset in brevi clip è mutuata da AED ma le diverse *contrastive loss* sono sostituite da una singola *triplet loss*. Inoltre la *tracking head* di BEHEMOTH risulta notevolmente più leggera ed efficiente rispetto al meccanismo proposto da AED. Infine, BEHEMOTH implementa un algoritmo di *tracking* più efficiente rispetto alle soluzioni presentate, superando ciascuna di esse in termini di velocità di inferenza (sezione 4.11).

1.4 Locality-Sensitive Hashing

Le funzioni di *Locality-Sensitive Hashing* (LSH) costituiscono una famiglia di funzioni di particolare interesse per operazioni di *dimensionality reduction*. Al contrario delle funzioni di hash classiche, che cercano di produrre output più sparsi possibile indipendentemente dalla distribuzione degli input, le funzioni di LSH hanno lo scopo di produrre output simili per input simili. Il concetto di similarità può essere definito in modi diversi a seconda del contesto applicativo, e differenti tecniche di LSH si adattano meglio a specifiche metriche di similarità. Ad esempio, l'approccio basato su *random hyperplanes* risulta particolarmente efficace quando si utilizza la *cosine similarity* (similarità del coseno) per misurare la similarità tra gli input [41]. I seguenti studi dimostrano l'efficacia delle funzioni di LSH in contesti eterogenei:

- In [42] un layer *fully connected* implementa la funzione di LSH, che viene appresa durante l'addestramento della rete, realizzando di fatto una *learnable* LSH (LLSH).
- HashNet [43] definisce un paradigma per l'addestramento di LLSH, che prevede l'applicazione di un guadagno incrementale alla funzione tangente iperbolica, usata come attivazione. Questa strategia consente una compressione orizzontale progressiva dell'attivazione, permettendo di approssimare la funzione gradino senza compromettere il flusso del gradiente. In questo modo, la rete può apprendere codici binari esatti senza dover ricorrere a loss di quantizzazione ausiliarie.
- Il Reformer [44] è una versione efficiente del transformer che riduce la complessità dell'operazione di *attention* limitandosi a calcolarla tra *query* e *key* assegnate allo stesso indice della tabella hash generata.
- In [45] viene utilizzata una LLSH per la localizzazione di *patch* contenenti oggetti di una classe specifica in immagini ottenute da telerilevamento. L'approccio consiste nel generare codici binari per ogni *patch* e minimizzare la distanza di Hamming rispetto al codice rappresentativo della classe assegnata.
- Luo et al. [46] dimostrano un'applicazione in ambito *image retrieval*, gli *embedding* delle immagini sono estratti attraverso una combinazione di CNN e *attention* e poi proiettati in uno spazio binario a dimensionalità ridotta attraverso una funzione di *random hyperplanes*. I codici binari così generati sono utilizzati per indicizzare tabelle hash multiple ed associare le immagini corrispondenti ad un *bin* per ogni tabella, ciò garantisce un'elevata probabilità che immagini simili siano assegnate allo stesso *bin* in almeno una delle tabelle. I confronti tra immagini sono limitati alle sole coppie assegnate ad uno stesso *bin*.

BEHEMOTH implementa una funzione di LLSH per la produzione di *embedding* binari da utilizzare durante il *tracking*, la cui implementazione estende quella di [42] sostituendo il layer *fully-connected* con un MLP. Al metodo proposto da HashNet è preferito l'impiego di una loss di quantizzazione, in modo da diminuire il numero di iperparametri e semplificare il training congiunto di un *embedding net* e un *hashing net* (sezioni 3.2.2, e 3.2.3). Come [45], anche BEHEMOTH realizza dei confronti mediante distanza di Hamming, ma il contesto *open vocabulary* impedisce di definire a priori i rappresentanti delle classi. Inoltre, questo studio presenta un'applicazione innovativa per le tecniche di LSH, il cui utilizzo è stato tipicamente limitato ad ambiti di *retrieval*.

1.5 Datasets

Il progresso della ricerca negli ambiti di *object detection* ed *object tracking* è stato possibile grazie alla realizzazione di dataset su larga scala, che permettono di addestrare modelli con buone capacità di generalizzazione, vi sono in particolare alcuni dataset che hanno giocato un ruolo fondamentale per l'addestramento ed il benchmarking di sistemi *open vocabulary*:

- MS-COCO [47] è formato da più di 200 000 immagini, e contiene 80 categorie di oggetti per un totale di circa 1.5 milioni di istanze. Le annotazioni non si limitano a localizzazione e classificazione, ma contengono anche maschere di segmentazione e didascalie (*captions*). Il dataset è spesso utilizzato per compiti di *object detection*, *panoptic segmentation*, *image captioning* ma anche *human pose/keypoint estimation*.

- LVIS [48] è un dataset costruito a partire dal test set di COCO 2017, presenta maschere di segmentazione particolarmente dense ed estende notevolmente il numero di categorie, passando dalle 80 di COCO a 1230. La varietà di etichette di classificazione permette di verificare in modo più accurato le capacità di generalizzazione dei modelli addestrati. LVIS è un *federated dataset*, ovvero un dataset formato dall'unione di molteplici dataset più piccoli, ciascuno afferente ad un insieme ristretto di categorie. Ogni dataset costituente garantisce annotazioni esaustive per le classi di riferimento, ma solo per le immagini che lo compongono. Di conseguenza nell'*ensemble* risultante non tutte le immagini presentano annotazioni complete. In particolare, data una categoria c, è possibile che esistano delle immagini in cui le istanze che dovrebbero appartenere a tale categoria non presentano alcuna annotazione. Per indicare questa eventualità si utilizza un flag e_i^c, che, se asserito, indica che l'i-esima immagine è annotata in modo completo per la categoria c. Nel caso in cui e_i^c sia "falso" i falsi positivi della categoria c nell'immagine i non saranno conteggiati durante il benchmarking.

- Il dataset Tracking Any Object (TAO) [49] è stato pensato per applicazioni di *multi-category multi-object tracking*, offre una collezione di 2907 video suddivisi in insiemi di *train, test* e *validation*, per un totale di più di 17 000 traiettorie. Presenta 833 categorie in tutto, 488 estratte dall'insieme di classi di LVIS e 345 nuove categorie denominate *free-form*. Il dataset è annotato ad 1 frame al secondo, i.e. per ogni secondo di video solo un frame presenta delle annotazioni associate. È tuttavia disponibile una versione densamente annotata del *training set* attraverso il dataset BURST [50], che espande TAO aggiungendo maschere di segmentazione per 482 categorie. Come LVIS, anche TAO e BURST sono dei *federated dataset*.

- Alcuni dataset *domain specific* sono spesso utilizzati in ambito *open vocabulary* sia per aumentare le dimensioni del *training set* sia per valutare le capacità di inferenza *zero-shot* dei modelli addestrati. Tra i più comuni si hanno MOTChallenge [51], appositamente studiato per il *tracking* di persone in scene dense e affollate, KITTI [52] e BDD100K [53], orientati invece all'ambito automotive.

  Per quanto riguarda BEHEMOTH, il sistema è addestrato utilizzando il *training set* di BURST, mentre il *validation* e *test set* di TAO sono stati adottati come *benchmark*.

Capitolo 2

Background

2.1 Convolutional Neural Networks

Introdotte da [54], le *Convolutional Neural Networks* (CNN) costituiscono una tecnica di *deep learning* fondamentale per lo sviluppo di algoritmi di *computer vision*. L'architettura di una CNN è basata su tre tipologie di *layer*: ***convolutional***, ***pooling*** e ***fully-connected***, eventualmente combinati con layer di attivazione e normalizzazione.

2.1.1 Convolutional Layer

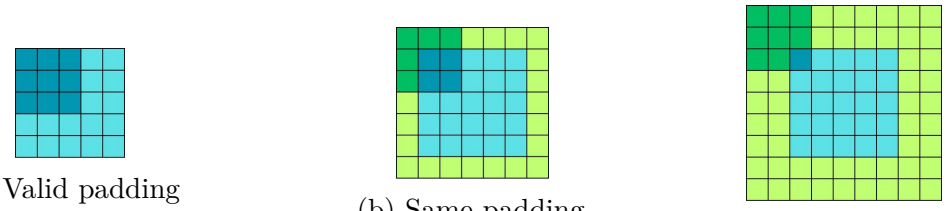
Un layer convoluzionale o *convolutional layer*, è costituito da un insieme di filtri tridimensionali (*kernel*), ciascuno di dimensione $k \times k \times d_i$, dove k è un iperparametro che rappresenta l'estensione spaziale del filtro e d_i la profondità dell'input. Ad esempio nel caso di un'immagine sRGB ciascun *kernel* avrà dimensione $k \times k \times 3$. L'operazione che caratterizza il *convolutional layer* è la convoluzione discreta tra ciascun *kernel* e l'input. Per ogni posizione (i, j) della matrice di input, viene calcolato il prodotto scalare tra il kernel $\boldsymbol{K}$ e la sottomatrice di dimensione $k \times k \times d_i$ centrata in (i, j). L'output corrisponde ad uno scalare, a cui può essere sommato un termine di *bias*.
Dopo ogni passo la finestra di convoluzione subisce uno spostamento, la cui entità dipende da un iperparametro: lo *stride*, ed il processo viene ripetuto fino a coprire l'intera matrice di input. Il risultato dell'operazione di convoluzione è un *activation map* (o *feature map*) di dimensione $h_o \times w_o \times 1$ dove h_o e w_o dipendono rispettivamente dal numero di passi complessivi lungo la prima e la seconda dimensione dell'input. Formalmente, il valore dell'elemento (i, j) dell'output $\boldsymbol{O}$, ottenuto applicando un kernel $\boldsymbol{K}$ di dimensione k a partire da un input $\boldsymbol{I}$ e considerando un bias b è dato dalla seguente equazione:

$$\boldsymbol{O}(i,j) = \left(\sum_{m=0}^{k-1} \sum_{n=0}^{k-1} \sum_{c=0}^{d_i-1} \boldsymbol{I}_{i+m,j+n,c} \cdot \boldsymbol{K}_{m,n,c} \right) + b \tag{2.1}$$

Le *feature map* prodotte da ciascun filtro sono infine concatenate per costruire l'output effettivo del layer, di dimensione $h_o \times w_o \times n$ dove n è il numero di filtri impiegati. Gli iperparametri che definiscono il comportamento di un *convolutional layer* sono:

- **Profondità (d_o)**: La quantità di filtri utilizzati, determina il numero di canali dell'output.
- **Kernel size (k)**: Determina la risoluzione di ciascun filtro $k \times k$.

- **Stride (s)**: La quantita di cui spostare la finestra di convoluzione ad ogni passo.
- **Padding (p)**: Il numero di valori (tipicamento zeri) da aggiungere ai bordi dell'input per modificare la dimensione della *feature map* di output. Esistono tre tipologie di padding:
  - **Valid padding**: Corrisponde all'assenza di padding ($p = 0$), in questo caso il prodotto scalare tra input e kernel viene calcolato solo per le posizioni dell'input che permettono una completa sovrapposizine della finestra di convoluzione sull'input stesso (fig. 2.1a).
  - **Same padding**: Prevede di aggiungere la quantità di padding necessaria a garantire che la *feature map* in output abbia la stessa risoluzione dell'input, sotto l'assunzione di avere stride $s = 1$ (fig. 2.1b).
  - **Full padding**: Corrisponde ad aggiungere valori di padding per permettere di considerare qualsiasi possibile sovrapposizione, anche parziale, della finestra di convoluzione con l'input (fig. 2.1c).



(a) Valid padding (b) Same padding (c) Full padding

Figura 2.1: Esempio di diverse modalità di padding con $h_i = w_i = 5$ e $k = 3$

Detta i la risoluzione dell'input lungo una delle prime due dimensioni (altezza o larghezza) la risoluzione o della dimensione corrispondente dell'output è determinata dalla seguente formula:

$$o = \lfloor \frac{i + 2p - k}{s} + 1 \rfloor \tag{2.2}$$

Il numero complessivo di pesi n_w del *convolutional layer* si ottiene come:

$$n_w = d_i \cdot d_o \cdot k^2 \tag{2.3}$$

mentre il numero di termini di bias n_b è uguale al numero di filtri utilizzati:

$$n_b = d_o \tag{2.4}$$

Di conseguenza, il numero complessivo di parametri del layer n_p è dato da:

$$n_p = \left(k^2 \cdot d_i + 1\right) \cdot d_o \tag{2.5}$$

2.1.2 Pooling layer

I layer di *pooling* sono comunemente utilizzati dopo il layer convoluzionali per ridurre la dimensionalità spaziale delle *feature map* prodotte, in modo tale da ridurre il numero di parametri richiesti dai layer successivi.
Un *pooling layer* è caratterizzato da due iperparametri:

- **Estensione spaziale** (f): definisce la dimensione della finestra su cui viene eseguita l'operazione.
- **Stride** (s): indica l'entità dello spostamento della finestra ad ogni passo.

Un layer di *pooling* corrisponde ad un filtro di dimensione $f \times f$ che applica un'operazione di aggregazione, separatamente per ogni canale, a ciascuna delle sottomatrici ottenute spostandosi lungo le prime due dimensioni dell'input di uno stride s, in modo simile a quanto avviene con l'operazione di convoluzione nel *convolutional layer* (2.1.1). Le modalità di *pooling* tipicamente utilizzate sono due:

- **max pooling**: in cui il risultato dell'operazione di *pooling* è dato dal massimo valore presente in ciascuna regione $f \times f$ considerata.
- **average pooling**: in cui il risultato dell'operazione di *pooling* è ottenuto come media dei valori che compongono la sottomatrice $f \times f$.

La dimensione dell'output $h_o \times w_o \times d$ dato un input di dimensione $h_i \times w_i \times d_i$ è determinata da:

$$w_o = (w_i - f)/s + 1 \tag{2.6}$$

$$h_o = (h_i - f)/s + 1 \tag{2.7}$$

$$d_o = d_i \tag{2.8}$$

2.1.3 Fully-connected layer

I layer *fully-connected* sono tipicamente posizionati alla fine dell'architettura, dove agiscono da classificatori o regressori, determinando l'output finale della rete.
Un singolo layer di questo tipo è costituito da un insieme di neuroni, ognuno caratterizzato da un vettore di pesi $\boldsymbol{w}_k$ ed un termine di bias b_k. Ogni neurone applica una trasformazione lineare all'input $\boldsymbol{x}_i$ definita da:

$$\boldsymbol{x}_{o,k} = \boldsymbol{w}_k \cdot \boldsymbol{x}_i + b_k \tag{2.9}$$

L'output complessivo di un layer è ottenuto concatenando gli scalari prodotti dalle trasformazioni lineari di ciascun neurone. Gli iperparametri che ne definiscono il comportamento sono:

- **Dimensione dell'input** (n_i) Il numero di elementi del vettore di input.

- **Numero di neuroni (n_o)**: Il numero di elementi di ciascun layer, che equivale alla dimensione dell'output.

Al contrario dei layer convoluzionali (2.1.1) e di *pooling* (2.1.2), un *fully-connected layer* richiede un input monodimensionale. Per questo motivo la *feature map* prodotta dai layer precedenti della CNN è sottoposta ad un'operazione di *flattening*, in modo tale da poterla trattare come un vettore.

Il numero di pesi n_w e di bias n_b di un *fully-connected layer* sono dati da:

$$n_w = n_i \cdot n_o \tag{2.10}$$

$$n_b = n_o \tag{2.11}$$

Ed il numero complessivo di parametri n_p può essere ottenuto come:

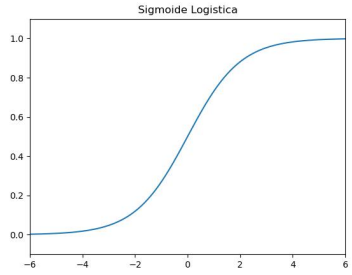
$$n_p = (n_i + 1) \cdot n_o \tag{2.12}$$

2.1.4 Activation layer

Un *activation layer* applica una funzione di attivazione all'output del layer precedente. Lo scopo di tale funzione è quello di introdurre una non linearità all'interno della rete, aumentandone la capacità espressiva e permettendole di catturare *pattern* complessi. Esistono numerose funzioni di attivazione, di seguito sono elencate alcune tra le più utilizzate, corredate dai grafici corrispondenti:

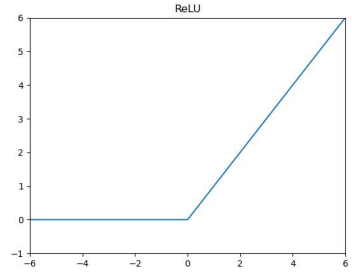
- **Sigmoide logistica:**

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.13}$$



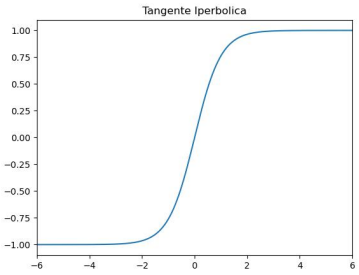
- **ReLU:**

$$f(x) = \max(0, x) \tag{2.14}$$



- **Tangente iperbolica:**

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \qquad (2.15)$$

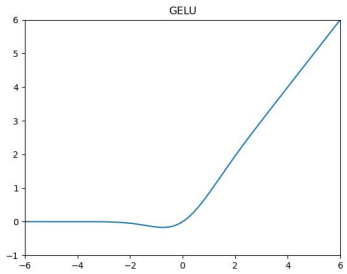


- **GELU:**

$$f(x) = \frac{x}{2}\left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right) \qquad (2.16)$$

$$f(x) \approx x \cdot \sigma(1.702x), \quad \text{con} \quad \sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$



2.1.5 Normalization layer

Un aspetto critico nell'addestramento delle reti neurali è la variazione delle distribuzioni delle attivazioni nei layer intermedi. I layer di normalizzazione sono utilizzati per mitigare questo problema, la loro funzione è quella di ottenere una distribuzione più regolare degli input, applicando fattori di scala e di traslazione appropriati. L'uso dei layer di normalizzazione migliora la stabilità dell'addestramento e accelera la convergenza del modello. Uno dei layer di normalizzazione più comunemente utilizzati nelle CNN e il *batch normalization layer*.
Un layer di *batch normalization* è caratterizzato da due parametri apprendibili, β e γ, e da uno stato $(\boldsymbol{\mu}_{mov}, \boldsymbol{\sigma}_{mov})$. Durante l'addestramento gli input della rete sono organizzati in *mini-batch*, ovvero insiemi di vettori di *feature* provenienti da diversi *data point* del *training set*. Dato $\boldsymbol{X}^t$ l'input ad un generico passo t dell'addestramento, di dimensione $D \times N$ dove D è il numero di features ed N il numero di *data points* che lo compongono, un layer di normalizzazione ne calcola media e deviazione standard in questo modo:

$$\boldsymbol{\mu}^t = \begin{bmatrix} \mu_1^t \\ \mu_2^t \\ \vdots \\ \mu_D^t \end{bmatrix}, \quad \text{dove} \quad \mu_j^t = \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{X}_{ij}^t \quad j = 1, \ldots, D \qquad (2.17)$$

$$\boldsymbol{\sigma}^t = \begin{bmatrix} \sigma_1^t \\ \sigma_2^t \\ \vdots \\ \sigma_D^t \end{bmatrix}, \quad \text{dove} \quad \sigma_j^t = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(\boldsymbol{X}_{ij}^t - \mu_j^t\right)^2} \quad j = 1, \ldots, D \qquad (2.18)$$

E lo stato del layer viene aggiornato attraverso una media mobile di momento α:

$$\boldsymbol{\mu}_{mov}^t = \alpha \boldsymbol{\mu}_{mov}^{t-1} + (1 - \alpha)\, \boldsymbol{\mu}^t \qquad (2.19)$$

$$\boldsymbol{\sigma}_{mov}^{t} = \alpha \boldsymbol{\sigma}_{mov}^{t-1} + (1-\alpha)\,\boldsymbol{\sigma}^{t} \tag{2.20}$$

Infine l'output del layer $\boldsymbol{Y}^t$ è ottenuto come:

$$\boldsymbol{Y}^{t} = \gamma \cdot \hat{\boldsymbol{X}}^{t} + \beta \tag{2.21}$$

dove:

$$\hat{\boldsymbol{X}}^{t} = \frac{\boldsymbol{X}^{t} - \boldsymbol{\mu}^{t}}{\boldsymbol{\sigma}^{t} + \epsilon} \tag{2.22}$$

in cui ϵ è una piccola costante necessaria per garantire stabilità numerica.

In fase di inferenza la media e la deviazione standard non sono calcolate sulla base dell'input, che spesso consiste in un singolo *data point*, ma sono estratte dallo stato del layer, dunque il termine $\hat{\boldsymbol{X}}^t$ nell'equazione 2.21 è ottenuto come:

$$\hat{\boldsymbol{X}}^{t} = \frac{\boldsymbol{X}^{t} - \boldsymbol{\mu}_{mov}^{t}}{\boldsymbol{\sigma}_{mov}^{t} + \epsilon} \tag{2.23}$$

2.2 Transformers

Basata sul meccanismo di *self-attention* introdotto dal celebre *"Attention is all you need"* [22], l'architettura del Transformer ha rappresentato una svolta fondamentale nel trattamento di dati sequenziali, riuscendo a superare le limitazioni delle reti neurali ricorrenti. L'input di un Transformer è costituito da una sequenza di *embedding* (i.e. vettori di *feature*), detti *token*. L'elaborazione di ciascun *token* avviene in parallelo, preservandone le informazioni contestuali attraverso il meccanismo di *self-attention*. L'architettura del Transformer è basata su due moduli fondamentali: l'***encoder*** e il ***decoder***. La figura 2.2 mostra il diagramma di flusso di un Transformer.

2.2.1 Transformer encoder

L'*encoder* costituisce il primo stadio di elaborazione di un Transformer, il suo scopo è quello di trasformare gli *embedding* in una rappresentazione densa, che catturi le relazioni tra essi indipendentemente dalla loro posizione. Tuttavia, poiché l'elaborazione dei *token* all'interno del Transformer avviene in parallelo e non tiene conto, di per sé, della posizione degli elementi nella sequenza, è necessario introdurre esplicitamente l'informazione posizionale. Questa precauzione consente al modello di distinguere *token* identici in posizioni diverse e di preservare l'ordine della sequenza durante l'elaborazione. In assenza di tali informazioni, l'*encoder* tratterebbe l'input come un insieme non ordinato di elementi, perdendo completamente la struttura sequenziale dei dati.
L'aggiunta di informazioni posizionali avviene sommando a ciascun *embedding* un vettore $\boldsymbol{PE}_{pos}$ che dipende dalla posizione *pos* del *token* associato, questi vettori, denominati *positional encoding*, sono ottenuti secondo:

$$\boldsymbol{PE}_{pos} = \begin{bmatrix} PE_{pos,0} \\ PE_{pos,1} \\ \vdots \\ PE_{pos,d-1} \end{bmatrix}, \quad dove \quad \begin{aligned} PE_{pos,2i} &= \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \\ PE_{pos,2i+1} &= \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right) \end{aligned} \tag{2.24}$$

dove d indica la dimensione di ciascun *embedding*.

Successivamente vengono applicate tre trasformazioni lineari a ciascun *token*, per generare tre vettori di *feature*, chiamati ***query***, ***key*** e ***value***, i parametri di ciascuna trasformazione sono appresi durante l'addestramento della rete. In seguito, durante il passo di *self-attention*, per ogni coppia di *query* $\boldsymbol{q}_{pos}$e *key* $\boldsymbol{k}_j$ viene calcolato uno *score* ottenuto dal loro prodotto scalare:

$$s_{pos,j} = \boldsymbol{q}_{pos} \cdot \boldsymbol{k}_j \tag{2.25}$$

Si ottiene così un vettore di *score* per ogni *query*:

$$\boldsymbol{s}_{pos} = \begin{bmatrix} s_{pos,0} \\ s_{pos,1} \\ \vdots \\ s_{pos,n-1} \end{bmatrix}$$

dove n è il numero di *token* in input. Lo *score* così ottenuto è poi normalizzato e moltiplicato per la matrice di *value* $\boldsymbol{V}^T$:

$$\boldsymbol{W}_{pos} = \sigma\left(\frac{\boldsymbol{s}_{pos}}{\sqrt{d}}\right) \cdot \boldsymbol{V}^T, \quad \text{dove} \quad \sigma(\boldsymbol{x}_m) = \frac{e^{\boldsymbol{x}_m}}{\sum_n e^{\boldsymbol{x}_n}} \tag{2.26}$$

e l'output del passo di *self-attention* per ogni token è dato da:

$$\boldsymbol{w}_{pos} = \sum_{i=0}^{n-1} \boldsymbol{W}^*_{pos,i} \tag{2.27}$$

In genere, per aumentare la capacità rappresentativa dell'*encoder*, si utilizzano più "teste" (*head*) in parallelo, ciascun *head* è indipendente dalle altre ed è caratterizzata dai propri parametri apprendibili. L'output è ottenuto concatenando le rappresentazioni $\boldsymbol{z}_i$ di ciascun *token* ottenute dalle diverse *head*, e applicando un'ulteriore trasformazione lineare, per proiettare l'output in un vettore con la dimensionalità attesa. L'architettura dell'*encoder* presenta inoltre una *skip connection*, ovvero un collegamento che somma l'input x all'output del modulo che lo elabora $F(x)$, che in questo caso corrispondono rispettivamente agli *embedding* ed il risultato dell'operazione di *self-attention*. L'utilità delle *skip connection*, introdotte con ResNet [5], è quella di migliorare il flusso del gradiente in reti profonde, scongiurando i fenomeni di *vanishing* ed *exploding gradient*, che rischiano di rendere l'addestramento estremamente instabile. Di fatto una *skip connection* evita che la rete impari una funzione di *mapping* diretta tra l'input x e l'output desiderato $H(x)$, spingendola invece ad apprendere un **residuo** $F(x) = H(x) - x$.
Nel caso dell'*encoder* la *skip connection* realizza la seguente operazione:

$$\boldsymbol{z}_{pos} = \boldsymbol{x}_{pos} + \boldsymbol{w}_{pos}$$

in seguito si applica un'operazione di *layer normalization*:

$$\boldsymbol{z}'_{pos} = LayerNorm(\boldsymbol{z}_{pos})$$

seguita da due layer *fully connected*:

$$\boldsymbol{y}_{pos} = fc_2(fc_1(\boldsymbol{z}'_{pos}))$$

e il risultato finale dell'*encoder* si ottiene applicando un'ulteriore *skip connection* e *layer normalization*:

$$\boldsymbol{y}'_{pos} = LayerNorm\left(\boldsymbol{y}_{pos} + \boldsymbol{z}'_{pos}\right)$$

Il modulo di *encoding* di un Transformer è ottenuto attraverso una sequenza di layer di questo tipo, l'output di ciascun *encoder* costituisce l'input per il successivo, e la rappresentazone finale di ciascun *token* è data dall'output dell'ultimo layer della sequenza.

2.2.2 Transformer decoder

La struttura di un *decoder* è identica a quella di un *encoder*, la differenza principale tra i due moduli riguarda il passo di *attention* e la struttura degli input accettati.
Durante il primo passo di decodifica l'input della pila di *decoder* è costituito dall'*embedding* di un *token* speciale, indicato come $< sos >$ (*start of sequence*). Ad ogni passo di decodifica la pila di *decoding* produce il prossimo token della sequenza di output, e l'insieme dei *token* prodotti fino al passo i-esimo costituisce l'input della pila di *decoding* al passo $i + 1$-esimo. Il procedimento si ripete fino a quando la pila di *decoding* non predice un *token* speciale, indicato come $< eos >$ (*end of sequence*).
L'operazione di *self-attention* che caratterizza l'*encoder* è sostituita dall'*encoder-decoder attention*, in cui *keys* e *values* sono ottenute a partire dagli *embedding* prodotti dall'encoder, mentre le *query* derivano dagli *embedding* dei *token* in input al decoder.
Al termine della procedura di decodifica si ottiene una sequenza di *token*, che costituisce l'output finale del Transformer.

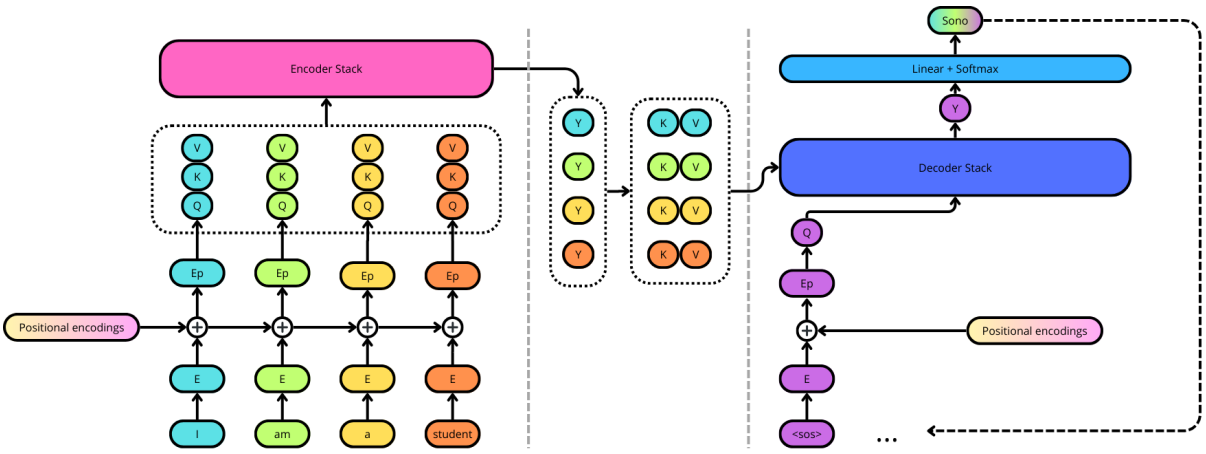


Figura 2.2: Diagramma di flusso di un Transformer

2.2.3 Vision Transformer

Introdotto da [23], il Vision Transformer (o ViT) (fig. 2.3) costituisce un adattamento dei concetti caratterizzanti il Transformer all'ambito delle immagini.
L'input è costituito da un'immagine, che viene suddivisa in *patch* quadrate di risoluzione fissata (e.g. 16x16). Ciascuna *patch* è sottoposta a *flattening* e fatta passare attraverso un layer *fully connected* (2.1.3) per ottenere il relativo *patch embedding*. La sequenza di *patch embedding* è poi elaborata dalla pila di *encoding* come osservato in 2.2.1, a differenza del Transformer classico il ViT non presenta una pila di *decoding*, e gli *output* dell'encoder

possono essere direttamente utilizzati per le operazioni di classificazione e regressione. In particolare, nel caso di classificazione delle immagini, la sequenza di *patch embeddings* viene estesa con un *classification token* prima di essere elaborata dall'*encoder*, l'*embedding* finale del *classification token* è poi usato come input del classificatore, con lo scopo di predire la classe di appartenenza dell'immagine.

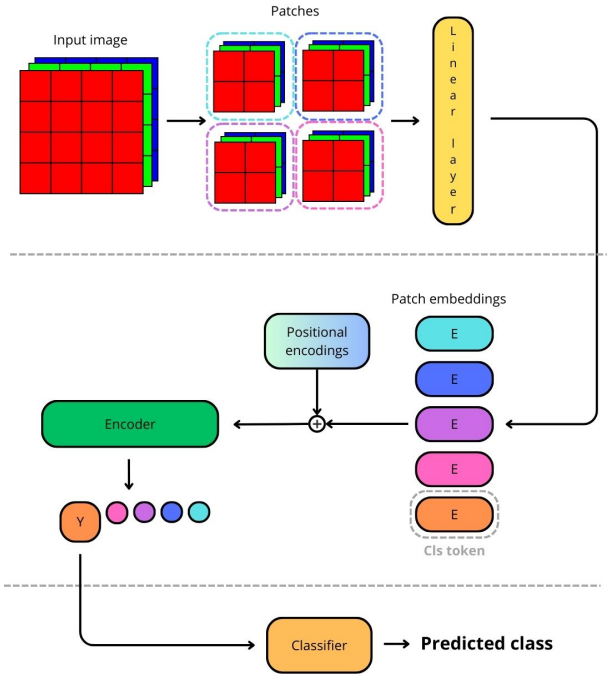


Figura 2.3: Diagramma di flusso di un Vision Transformer

2.3 CLIP

Contrastive Language-Image Pre-training, o CLIP, rappresenta una tecnica di *pre-training* supervisionato che permette di unire *natural language processing* (NLP) e *computer vision* per l'addestramento di un modello multimodale. Prevede di addestrare un *image encoder* ed un *text encoder* in modo congiunto, mappando i rispettivi output all'interno di uno spazio vettoriale condiviso. Il risultato è un modello in grado di catturare e correlare gli aspetti semantici sia di testo che di immagini. Questa tecnica, introdotta da [20], ha permesso lo sviluppo del modello omonimo, che costituisce una *backbone* di fondamentale importanza nella maggior parte dei sistemi *open vocabulary* (sezioni 1.2 e 1.3). Oltre all'ambito *open vocabulary*, CLIP risulta applicabile in molteplici campi, come *optical character recognition* (OCR), *action recognition*, geolocalizzazione ed *image classification*.

2.3.1 Addestramento e architettura

WebImageText

CLIP è addestrato su un dataset costruito appositamente, composto da circa 400 milioni di coppie testo-immagine raccolte dal web. È stato costruito attraverso una procedura automatizzata, cercando coppie il cui testo contenesse almeno una *query* tra quelle presenti in una lista di 500 000 parole. Le parole da cercare sono state ottenute selezionando quelle che compaiono almeno 100 volte nella versione inglese di Wikipedia [55], ed aggiungendo digrammi come articoli e preposizioni. Il dataset risultante, denominato **WebImageText** dagli autori, ottiene un buon bilanciamento, con circa 20 000 coppie per ogni *query*.

Addestramento

La procedura di addestramento prevista da CLIP è pensata per essere scalabile ed efficiente, senza compromettere la capacità rappresentativa del modello. Ogni *batch* di addestramento è composto da N coppie testo-immagine, il modello deve individuare, tra gli N^2 possibili assegnamenti, gli N accoppiamenti "positivi", ovvero corrispondenti al *ground truth*. Per fare questo si utilizzano un *image encoder* ed un *text encoder* per generare *embedding* multimodali all'interno dello stesso spazio di destinazione, e le coppie sono ricostruite sulla base della *cosine similarity* tra gli *embedding* ottenuti. Durante l'addestramento il modello cerca di massimizzare la similarità tra le rappresentazioni corrispondenti agli N campioni positivi, e minimizzare simultaneamente quella tra *embedding* di accoppiamenti negativi. Infine l'addestramento adotta tecniche di *data augmentation* minime, che si limitano al *random cropping* delle immagini in input. Questa scelta è primariamente giustificata dalla dimensione del dataset, che rende trascurabile il rischio di *overfitting*. L'efficacia di questa strategia di *pre-training* è corroborata dall'accuratezza di *zero-shot image classification* dimostrata dal modello risultante. Su ImageNet CLIP riesce a raggiungere le stesse prestazioni di un classificatore basato su ResNet-101, senza alcun *fine-tuning* sul relativo *training set*. Risulta inoltre robusto a variazioni, anche sensibili, di soggetto, prospettiva, stile ed altre caratteristiche delle immagini, al contrario dei classificatori tradizionali, che raggiungono le prestazioni attese solo su dati che corrispondono alla distribuzione dell'insieme su cui sono stati addestrati. Le prestazioni ottenute dal modello in campi come l'OCR o *text* ed *image retrieval*, che talvolta raggiungono lo stato dell'arte, sono un'ulteriore dimostrazione della versatilità del sistema e dell'efficacia di CLIP come *pipeline* di *pre-training*.

Architettura

Image Encoder Esistono più versioni di CLIP, addestrate con *image encoder* differenti, costituiti da varianti di ResNet e ViT. Gli *encoder* basati su ResNet sostituiscono il *layer* di *global average pooling* finale con un *layer* di *attention pooling*. L'*attention pooling* prevede un singolo passo di *self-attention* tra le diverse *patch*[1] $p_{i,j}$ della *feature map* in input. a cui è aggiunto un ulteriore *token* p_{avg}, ottenuto attraverso *global average pooling*. Il *token* in output corrispondente a p_{avg} è utilizzato come *image embedding* finale. Durante l'addestramente l'*image encoder* è addestrato da zero, senza perciò utilizzare i pesi di una versione analoga preaddestrata su ImageNet.

Text Encoder Il *text encoder* corrisponde alla pila di *encoding* di un *Transformer* a 12 *layer*, 8 *head* e *token* di dimensione 512. Gli input hanno una lunghezza massima di 76 *token*, e sono rappresentati dalla codifica BPE (sezione 2.3.2) di un vocabolario di circa 50 000 parole. Sono inoltre delimitati da caratteri speciali di $< sos >$ ed $< eos >$, e l'output corrispondente al *token* $< eos >$ è utilizzato come rappresentazione dell'input nello spazio di destinazione. L'*embedding* di una *query* testuale può essere memorizzata in una *cache*, così da ammortizzare il costo computazionale del *text encoder* nel caso in cui lo stesso *prompt* sia utilizzato più di una volta.

2.3.2 Byte-pair Encoding

Algoritmo originale

L'algoritmo *byte-pair encoding* (BPE), descritto per la prima volta nel 1994 da Philip Gage, nasce come metodo di compressione per sequenze di simboli. Dato un vocabolario V di parole p, ed una stringa $S_0 = \{p_1, ..., p_n \mid p_i \in V \ \forall i\}$, l'algoritmo prevede come prima cosa di assegnare una rappresentazione in byte a ciascuna parola del vocabolario. Successivamente, la coppia di byte consecutivi più frequente all'interno di S_0 viene aggiunta al vocabolario, e riceve una rappresentazione unica, tutte le istanze di quella coppia sono quindi sostituite con la codifica appena assegnata, ottenendo così S_1. L'algoritmo procede iterativamente fino a quando nella string S_n ogni coppia di byte compare al più una sola volta.

Character-level BPE

Il *Character-level* BPE è una versione leggermente modificata dell'algoritmo, adottata come standard di rappresentazione per gli input testuali di numerosi modelli linguistici, come quelli della famiglia GPT [56], nonchè CLIP stesso. In questa formulazione le parole corrispondono a caratteri, ed il vocablario è rappresentato da un alfabeto. La procedura di compressione è la medesima, ma l'algoritmo si ferma dopo un numero di iterazioni predeterminato, senza necessariamente raggiungere il massimo grado di compressione. Lo scopo è quello di ottenere, a partire da una stringa S - spesso formata da un *ensemble* di documenti di testo - un vocabolario di *token* di dimensione fissata. Attraverso i *token* ottenuti è possibile codificare qualsiasi stringa che utilizzi il vocabolario di partenza, indipendentemente dalla sua lunghezza, si tratta quindi di un sistema efficiente per la

[1] i.e. *embedding* corrispondenti a ciascuna posizione (i, j) di una *feature map*

rappresentazione di input testuali, indipendente da altri standard di codifica e adatto all'addestramento di modelli in ambito NLP.

Byte-level BPE

Il prinicpale limite del *character-level* BPE consiste nel vincolo costituito dall'alfabeto di partenza. Il sistema non permette infatti la codifica di stringhe che utilizzino caratteri non previsti dal vocabolario iniziale. Il *byte-level* BPE è una versione alternativa della codifica che risolve il problema evidenziato. Nel *byte-level* BPE le stringhe sono trattate come *stream* di byte, ottenuti a partire dalla rappresentazione UTF-8 del contenuto, ed il vocabolario iniziale è formato dai singoli byte della sequenza, indipendentemente dal dato che essi rappresentano. Questa versione del BPE permette di ottenere un vocabolario di *token* con gli stessi vantaggi del *character-level* BPE, ma senza fare riferimento all'alfabeto di un linguaggio specifico.

2.3.3 Prompt Engineering

Nell'*image classification* classico non esiste ambiguità tra le categorie, in quanto i modelli predicono degli indici numerici in corrispondenza biunivoca con l'insieme di classi. Questa garanzia non è invece valida per CLIP, principalmente a causa di termini polisemantici, in presenza di omonimi è infatti impossibile disambiguare tra i potenziali significati in assenza di contesto. Ad esempio, come riportato dagli autori di CLIP, la parola *"boxer"*, priva di contesto, potrebbe riferirsi sia alla razza canina che ad una professione sportiva. Inoltre, per via della struttura di WebImageText (sezione 2.3.1), CLIP è addestrato su dati testuali corrispondenti a didascalie o intere frasi, non singole parole. Per massimizzare le prestazioni del sistema risulta quindi fondamentale che le *query* testuali contengano anche delle informazioni contestuali, e non si limitino ai soli nomi delle classi da individuare. Gli autori evidenziano come l'utilizzo di *prompt* generici come *"A photo of a {label}"*, al posto di *"{label}"*, possano aumentare l'accuratezza di classificazione di CLIP. Per dataset che trattano classi specifiche è possibile estendere questo *prompt* aggiungendo informazioni contestuali. Ad esempio l'utilizzo del prompt *"A photo of a {label}, a type of pet"* aumenta le prestazioni su immagini di cuccioli, o ancora *"A photo of a {label}, a type of food"* risulta particolarmente adatto per la classificazione di alimenti.
Infine, gli autori evidenziano come il sistema possa trarre vantaggio da *ensemble* di *prompt*. Data una collezione di *prompt* con strutture o significati differenti[2], è possibile distillare una rappresentazione compatta degli *embedding* di ciascuno di essi, con impatto minimo sull'efficienza del sistema. La rappresentazione dell'*ensemble* è ottenuta come media degli output generati dal *text encoder* per ciascun *prompt*, e può essere trattata in modo analogo ad un qualsiasi altro *embedding*.

2.4 Locality-Sensitive Hashing

Le tecniche di *Locality-Senstive Hashing* (LSH) costituiscono un approccio efficace per la ricerca di elementi simili in spazi ad elevata dimensionalità. Le funzioni appartenenti a questa famiglia garantiscono che oggetti simili siano assegnati allo stesso *bucket* di una

[2]Gli autori riportano *"A photo of a big {label}"* e *"A photo of a small {label}"* come esempio

tabella hash con elevata probabilità.

2.4.1 Definizione formale

Formalmente, dato (M, d) uno spazio metrico, dove $d : M \times M \to \mathbb{R}$ è la funzione di distanza tra elementi di M, una famiglia $\mathcal{F}$ di funzioni $h : M \to S$ è detta una famiglia LSH per d se dati:

- Una soglia $r > 0$
- Un fattore di approssimazione $c > 0$
- Due probabilità $p_1 > p_2$

soddisfa le seguenti condizioni per qualsiasi coppia $(a, b) \in M$ ed una qualsiasi funzione h scelta con probabilità uniforme da $\mathcal{F}$ ($h \sim \mathcal{F}$):

- Se $d(a, b) < r$ allora la probabilità che $h(a)$ e $h(b)$ collidano è almeno p_1, ovvero:

$$\mathrm{P}_{h\sim\mathcal{F}}\left[h(a) = h(b)\right] \geq p_1$$

- Se $d(a, b) \geq r$ allora la probabilità che $h(a)$ e $h(b)$ collidano è al massimo p_2, ovvero:

$$\mathrm{P}_{h\sim\mathcal{F}}\left[h(a) = h(b)\right] \leq p_2$$

2.4.2 Random hyperplanes

Una famiglia LSH particolarmente versatile ed efficace è quella dei "*random hyperplanes*" (iperpiani casuali), utilizzata per la prima volta in [57] ed analizzata in [41]. Dato uno spazio sorgente $\mathbb{R}^n$, ed un iperpiano $\boldsymbol{r} \in \mathbb{R}^n$ campionato casualmente da una distribuzione Gaussiana n-dimensionale, la funzione *random hyperplanes* corrispondente è definita come:

$$h_{\boldsymbol{r}}(\boldsymbol{u}) = \begin{cases} 1, & \text{se} \quad \boldsymbol{r} \cdot \boldsymbol{u} \geq 0 \\ 0, & \text{se} \quad \boldsymbol{r} \cdot \boldsymbol{u} < 0 \end{cases} \tag{2.28}$$

con $\boldsymbol{u} \in \mathbb{R}^n$.
Si verifica che per ogni coppia $(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{R}^n \times \mathbb{R}^n$ vale:

$$\mathrm{P}_{h_{\boldsymbol{r}}}\left[h_{\boldsymbol{r}}(\boldsymbol{u}) = h_{\boldsymbol{r}}(\boldsymbol{v})\right] = 1 - \frac{\theta(\boldsymbol{u}, \boldsymbol{v})}{\pi} \tag{2.29}$$

dove $\theta(\boldsymbol{u}, \boldsymbol{v})$ indica l'angolo tra $\boldsymbol{u}$ e $\boldsymbol{v}$.
Dall'equazione 2.29 si osserva che al diminuire dell'angolo tra due vettori, aumenta la probabilità che questi siano associati ad uno stesso *bucket* della tabella hash. Ciò rende le funzioni di *random hyperplanes* particolarmente adatte in situazioni in cui la similarità tra due elementi dello spazio sorgente dipende dall'angolo tra essi, un esempio di tale misura di similarità è la *cosine similarity* (similarità del coseno). Combinando m funzioni della famiglia *random hyperplanes* è possibile costruire una funzione di hash $\mathcal{H} : \mathbb{R}^n \to \{0,1\}^m$, ottenuta trattando il risultato di ciascuna funzione della famiglia come il valore di un singolo bit di un vettore all'interno dello spazio di destinazione. In questo modo è possibile costruire tabelle hash arbitrariamente grandi, utilizzabili per fare *clustering* dei campioni provenienti dallo spazio sorgente considerato.

2.4.3 Considerazioni

Una delle applicazioni più comuni di queste tecniche è quella di *k-nearest neighbours search*, in cui, dato un insieme di campioni S, si vogliono individuare i k elementi più vicini ad un campione di riferimento. In particolare è possibile applicare una funzione LSH per dividere gli elementi in diversi *bucket*, limitando la ricerca a tutti e soli gli elementi che condividono lo stesso *bucket*, riducendo il numero di confronti da effettuare e la complessità dell'operazione. Trattandosi di una tecnica probabilistica, la ricerca dei *k-nearest neighbours* attraverso LSH non garantisce di individuare una soluzione ottima, si tratta quindi di una tecnica di approssimazione, che risulta comunque utile in diversi ambiti (sezione 1.4). Tipicamente, per ottenere un'approssimazione migliore, si definisco più funzioni hash, e quindi più tabelle, e la ricerca dei *k-nearest neighbours* di un campione x è limitata agli elementi a cui è stato assegnato lo stesso *bucket* di x in almeno una delle tabelle hash.
Nel caso particolare della funzione di *random hyperplanes* (2.4.2) vi è un'ulteriore possibilità per sfruttare il risultato dell'operazione di hashing. Poichè gli iperpiani utilizzati per costruire le funzioni di hash sono scelti in modo indipendente, e la probabilità di collisione di ciascun bit aumenta al diminuire dell'angolo θ tra i vettori in input (eq. 2.29), si osserva che input con un'elevata *cosine similarity* tendono a produrre un numero di collisioni sui singoli bit maggiore. Di conseguenza è possibile adottare la distanza di Hamming nello spazio binario risultante come surrogato della *cosine similarity* tra i rispettivi elementi dello spazio sorgente. La funzione *random hyperplanes*, perciò, è utilizzabile anche per operazioni di *dimensionality reduction*, permettendo di proiettare spazi vettoriali reali ad elevata dimensionalità in spazi vettoriali binari, che garantiscono costi ridotti sia in termini di memoria che di computazione.

2.5 Algoritmo Ungherese

Introdotto in [13], l'algoritmo ungherese (o metodo ungherese) è uno dei più celebri algoritmi in grado di risolvere il problema dell'assegnamento in tempo polinomiale[3]. L'algoritmo è spesso utilizzato in soluzioni di *multi-object tracking* (sezioni 1.1 e 1.3), come SORT [11], ByteTrack [17] e AED [38], sia durante la fase di inferenza, per determinare l'assegnamento ottimo tra le *tracklets* (i.e. traiettorie parziali) e le predizioni della rete relative all'ultimo frame, sia durante l'addestramento, per assegnare ciascuna predizione alla *bounding box* della regione di *ground truth* corrispondente.

2.5.1 Problema dell'assegnamento

Nella teoria dei grafi il problema dell'assegnamento consiste nel trovare, dato un grafo pesato bipartito G, un accoppiamento[4] massimale[5] M tale che la somma dei pesi degli archi in M sia minima.
Talvolta il problema è formulato sulla base di attività, risorse e costi, in tal caso i due sottoinsiemi di nodi del grafo bipartito G sono indicati come attività T e risorse R. Nella

[3] $O(n^3)$

[4] Un **accoppiamento** è definito come un insieme di archi bipartito senza vertici in comune.

[5] Un accoppiamento M si dice **massimale** se aggiungendo un qualsiasi arco non in M ad M esso non è più un accoppiamento.

formulazione più semplice ogni nodo in T è collegato a tutti e soli i nodi in R, e viceversa. Il costo di ogni assegnamento equivale al peso degli archi corrispondenti. Se T ed R hanno la stessa cardinalità il problema è detto **bilanciato**, altrimenti il problema è **sbilanciato**. La figura 2.4 mostra un'istanza di un problema di assegnamento bilanciato.

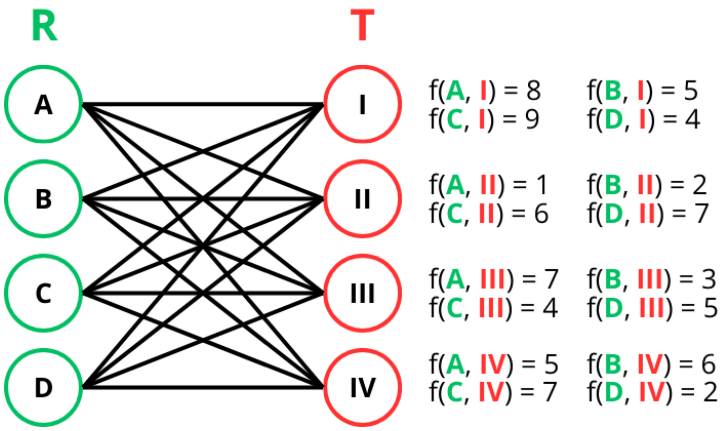


Figura 2.4: Esempio di un problema di assegnamento

2.5.2 Algoritmo

Di seguito è indicata la procedura per la risoluzione del problema dell'assegnamento prevista dal Metodo Ungherese:

1. Calcolare la matrice dei costi C di dimensione $n \times n$.
2. Sottrarre da ciascuna riga il valore minimo della riga stessa.
3. Sottrarre da ciascuna colonna il valore minimo della colonna stessa.
4. Coprire tutti gli zeri con il minimo numero di linee orizzontali e verticali. Se il numero di linee utilizzate n_r è minore di n andare al passo 5; altrimenti andare al passo 6.
5. Individuare il minimo valore k non coperto da alcuna linea, sottrarlo a tutti gli elementi non coperti da alcuna linea e sommarlo a tutti gli elementi che si trovano all'intersezione di due linee, poi ripetere il passo 4.
6. L'assegnamento ottimale è dato da una qualsiasi combinazione degli indici degli zeri della matrice tale che nessun indice di riga o di colonna venga utilizzato più di una volta.

Applicando l'algoritmo al problema mostrato in figura 2.4 si ottiene:

passo 1.

	I	**II**	**III**	**IV**
A	8	1	7	5
B	5	2	3	6
C	9	6	4	7
D	4	7	5	2

passo 2.

	I	II	III	IV
A	8	**1**	7	5
B	5	**2**	3	6
C	9	6	**4**	7
D	4	7	5	**2**

→

	I	II	III	IV
A	7	0	6	4
B	3	0	1	4
C	5	2	0	3
D	2	5	3	0

passo 3.

	I	II	III	IV
A	7	**0**	6	4
B	3	0	1	4
C	5	2	**0**	3
D	**2**	5	3	**0**

→

	I	II	III	IV
A	5	0	6	4
B	1	0	1	4
C	3	2	0	3
D	0	5	3	0

passo 4-a.

	I	II	III	IV
A	5	0	6	4
B	1	0	1	4
C	3	2	0	3
D	~~0~~	~~5~~	~~3~~	~~0~~

$n_r < n \rightarrow$ vai a 5.

passo 5.

	I	II	III	IV
A	5	0	6	4
B	**1**	0	1	4
C	3	2	0	3
D	~~0~~	~~5~~	~~3~~	~~0~~

→

	I	II	III	IV
A	4	0	6	3
B	0	0	1	3
C	2	2	0	2
D	~~0~~	**6**	**4**	~~0~~

passo 4-b.

	I	II	III	IV
A	4	0	6	3
B	~~0~~	~~0~~	~~1~~	~~3~~
C	2	2	0	2
D	~~0~~	~~6~~	~~4~~	~~0~~

$n_r = n \rightarrow$ vai a 6.

passo 6.

	I	**II**	**III**	**IV**
A	4	$\underline{0}$	6	3
B	$\underline{0}$	0	1	3
C	2	2	$\underline{0}$	2
D	0	6	4	$\underline{0}$

Assegnamento ottimo:

A $\leftrightarrow$ **II** $\diamond$ **B** $\leftrightarrow$ **I** $\diamond$ **C** $\leftrightarrow$ **III** $\diamond$ **D** $\leftrightarrow$ **IV**

Costo: $5 + 1 + 4 + 2 = 12$

Capitolo 3

Metodo

3.1 Tracking

L'*object tracking* è un'operazione complessa, che espande l'ambito di *object detection* ed introduce un requisito ulteriore, ovvero la ricostruzione delle traiettorie degli oggetti di interesse. La soluzione sviluppata si basa sul paradigma di *tracking-by-detection*, che scompone il *tracking* in due fasi: predizione e associazione. Durante la fase di predizione ciascun frame è elaborato individualmente da una rete preposta alla localizzazione e classificazione delle regioni di interesse, mentre con la fase di associazione si assegna ciascuna regione ad una traiettoria esistente, o se ne creano di nuove se necessario. L'associazione può essere effettuata in modalità *frame-by-frame*, ricostruendo le traiettorie iterativamente man mano che i frame vengono processati, oppure a valle delle operazioni di localizzazione su tutti i frame. In entrambe le modalità l'implementazione tipica prevede la ricerca dell'assegnamento ottimo, ma solo la seconda garantisce l'individuazione dell'ottimo globale.
In questo studio è stata adottata la prima metodologia, in quanto è l'unica applicabile ad un contesto *real-time*.
Con particolare riferimento a BEHEMOTH, esso integra un modulo di *locality-sensitive hashing* nella fase di predizione (sezione 3.1.2), per comprimere e quantizzare delle rappresentazioni vettoriali che descrivono le regioni individuate. I codici binari ottenuti sono utilizzati durante la fase di associazione (sezione 3.1.3) per determinare un punteggio di similarità tra le regioni candidate (eq. 3.3). Ciò riduce il costo computazionale e la memoria richiesta durante questo processo rispetto ad approcci classici, basati su misure di similarità tra rappresentazioni vettoriali reali. Il diagramma in figura 3.1 illustra il flusso che caratterizza il processo di *tracking-by-detection*.

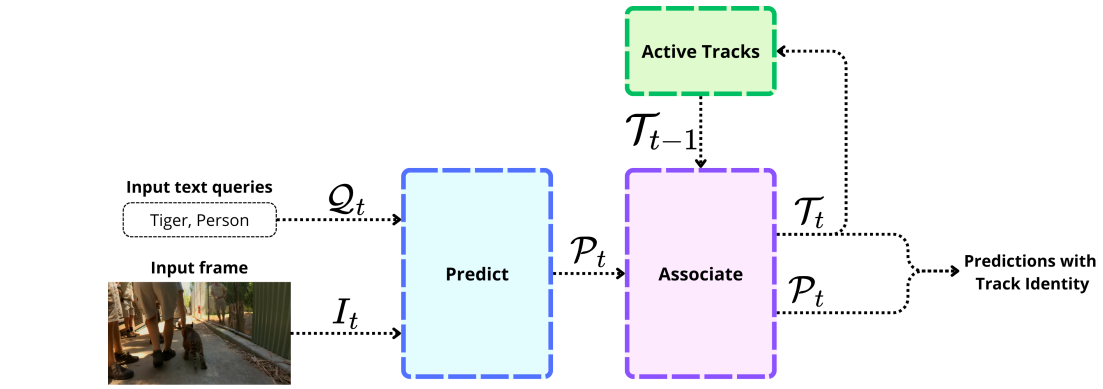
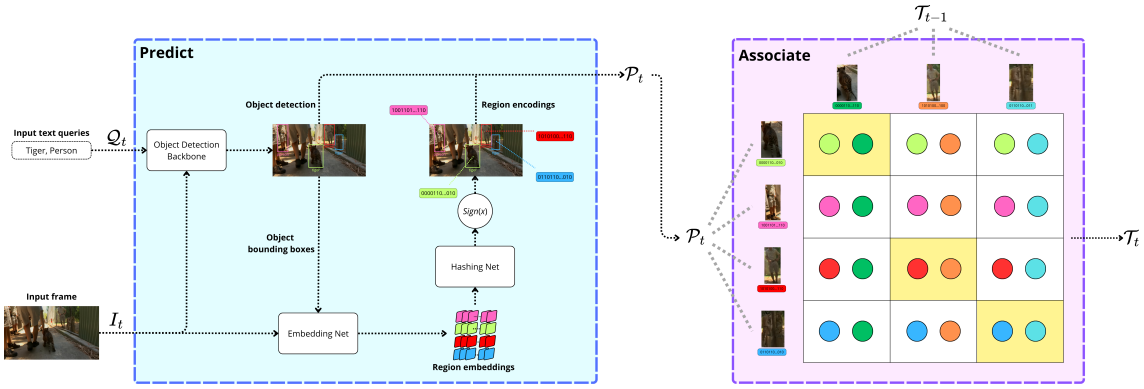


Figura 3.1: Diagramma a blocchi del processo di *open-vocabulary tracking-by-detection*. I diagrammi 3.2a e 3.2b mostrano in maggiore dettaglio le fasi di associazione e predizione.



(a) Modulo di predizione. Ad un istante t la rete elabora l'insieme di *query* $\mathcal{Q}_t$ ed il frame I_t e genera un insieme di predizioni $\mathcal{P}_t$

(b) Modulo di associazione. Il problema dell'assegnamento è definito tra gli elementi di $\mathcal{P}_t$ ed i rappresentanti delle tracce in $\mathcal{T}_{t-1}$

Figura 3.2: Schema riassuntivo dell'architettura utilizzata.

3.1.1 Formulazione del problema

L'input del problema di OV-MOT considerato è caratterizzato da una sequenza di T fotogrammi in ingresso $\mathcal{I}_T = \{I_t\}_{t=0}^{T-1}$, ed un insieme di C classi $\mathcal{Q} = \{Q_c\}_{c=0}^{C-1}$, ciascuna corrispondente ad una *query* testuale. Attraverso la fase di predizione, una rete di *open-vocabulary object-detection* fornisce un insieme di N_t predizioni $\mathcal{P}_t = \{p_t^i\}_{i=0}^{N_t-1}$ per ogni fotogramma I_t. Una predizione p_t^i è una tupla $(b_t^i\ ,\ c_t^i,\ conf_t^i)$ dove:

- $b_t^i \in \mathbb{R}$ è la **bounding box** che definisce la regione corrispondente.
- $c_t^i \in [0, C-1]$ è l'**etichetta di classe** assegnata, corrispondente all'indice di una classe in $\mathcal{Q}$.
- $conf_t^i \in [0,1]$ è la **confidenza** della predizione, i.e. la *likelihood* dell'assegnamento regione-classe proposto.

L'obiettivo del *tracking* è stimare, per ogni frame I_t, un insieme di K traiettorie $\mathcal{T}_t = \{\tau_t^j\}_{j=0}^{K-1}$. Ogni traiettoria τ_t^j è una sequenza temporale di rilevamenti associati ad una stessa identità j ed effettuati sugli elementi della sequenza parziale $\mathcal{I}_t = \{I_{t'} \in \mathcal{I}_T\ |t' \leq t\}$.

Ogni traiettoria è inoltre caratterizzata da un rappresentante $p_{\tau_t^j}$, tipicamente identificato con l'ultima predizione corrispondente all'identita j. La fase di associazione consiste nella ricostruzione delle traiettorie, che avviene risolvendo un problema di assegnamento. Date le predizioni del frame t-esimo $\mathcal{P}_t$ e l'insieme di traiettorie attive $\mathcal{T}_{t-1}$, è necessario determinare una corrispondenza ottimale tra gli elementi dei due insiemi. È inoltre possibile individuare delle nuove traiettorie $\mathcal{T}_t^{new}$ o eliminare delle traiettorie ritenute obsolete $\mathcal{T}_t^{discard} \subseteq \mathcal{T}_{t-1}$. Formalmente l'operazione di associazione si riconduce alla ricerca di una funzione:

$$\phi_t : \mathcal{P}_t \to \mathcal{T}_{t-1} \tag{3.1}$$

e l'insieme $\mathcal{T}_t$ è definito come:

$$\mathcal{T}_t = \left\{ \mathcal{T}_{t-1} \setminus \mathcal{T}_t^{discard} \right\} \cup \mathcal{T}_t^{new} \tag{3.2}$$

La soluzione ottimale per ϕ_t è spesso individuata attraverso l'algoritmo ungherese (sezione 2.5), minimizzando una funzione di costo definita sulle coppie di predizioni e rappresentanti di traiettorie. Per ogni frame I_t l'output del sistema è costituito sia dalle predizioni $\mathcal{P}_t$ sia dalle traiettorie $\mathcal{T}_t$.

3.1.2 Predizione

Durante la fase di predizione (figura 3.2a), per determinare l'insieme di rilevazioni $\mathcal{P}_t$, le regioni in output alla *backbone* di OV-OD sono filtrate attraverso *non-maximum suppression* (NMS). Inoltre, tutte le predizioni con un livello di confidenza $conf_t^i < 0.3$ sono eliminate. Questi passaggi di *post-processing*, oltre ad essere conformi a quanto avviene in numerosi sistemi di *object-detection*, eliminano preventivamente predizioni duplicate, errate o eccessivamente rumorose, facilitando la ricostruzione delle traiettorie durante la fase di associazione e riducendo la propagazione degli errori.
Infine, il sistema implementato estende ciascuna predizione $p_i^n \in P_t$ aggiungendo alla tupla $(b_t^i\ ,\ c_t^i,\ conf_t^i)$ un quarto elemento $r_t^i \in \{0,1\}^m$, chiamato *region encoding*. Il *region encoding* di una predizione è una sequenza di m bit che cattura e codifica le caratteristiche visive della regione dell'immagine corrispondente alla *bounding box* b_i^t. I *region encoding* sono estratti da una rete dedicata, costituita da due moduli denominati *embedding net* ed *hashing net*. Il funzionamento e l'architettura di *backbone*, *embedding net* ed *hashing net* è descritto in dettaglio nella sezione 3.2.

3.1.3 Associazione

La fase di associazione (figura 3.2b) ricostruisce in modo iterativo le traiettorie finali a partire dalle predizioni sui singoli frame, il rappresentante $p_{\tau_t^j}$ di ciascuna traiettoria τ_t^j corrisponde all'ultima predizione che vi è stata assegnata. Le nuove predizioni sono messe in corrispondenza con la traiettoria che possiede il rappresentante più simile, mentre generano nuove traiettorie se una tale corrispondenza non può essere trovata. Questo meccanismo è basato sull'assunzione secondo cui regioni rappresentati lo stesso soggetto in frame ravvicinati siano molto simili, sia in termini di caratteristiche visive, come colore, *background* e *foreground*, sia per proprietà spaziali, definite ad esempio da posizione, altezza e larghezza.
Ad un istante t, ciascun elemento $\tau_t^j \in \mathcal{T}_t$, è rappresentato da una tupla $(j,\ age_t^j,\ p_{\tau_t^j})$ dove:

- j è l'identificativo univoco dell'identità catturata dalla traiettoria.
- age_t^j è un contatore intero che misura il numero di frame passati dall'ultimo aggiornamento della traiettoria.
- $p_{\tau_t^j}$ è il rappresentante della traiettoria.

L'implementazione della procedura segue l'algoritmo **ByteTrack** [17], che è composto da due sottofasi principali. Durante la prima sono considerate solo le predizioni ad elevata confidenza, mentre la seconda si occupa dell'assegnamento delle predizioni rimanenti. In generale, al passo t-esimo, dato l'insieme delle traiettorie $\mathcal{T}_{t-1}$ e quello delle predizioni $\mathcal{P}_t$, l'insieme di predizioni ad elevata confidenza è definito come $\mathcal{P}_t^{high} = \{p_t^i \in \mathcal{P}_t | conf_t^i > t_{conf}\}$, dove $t_{conf} \in [0,1]$ è una soglia fissata ($t_{conf} = 0.4$ negli esperimenti condotti). Le predizioni a bassa confidenza formano invece l'insieme $\mathcal{P}_t^{low}$.
Inoltre, data una qualsiasi coppia $(\tau_{t-1}^j, p_t^i) \in \mathcal{T}_t - 1 \times \mathcal{P}_t$, e considerate le coppie di *region encoding* $(r_{\tau_{t-1}^j}, r_{p_t^i})$ e di *bounding box* $(b_{\tau_{t-1}^j}, b_{p_t^i})$ associate, è possibile definire le seguenti funzioni di costo:

$$c_{visual}(\tau_{t-1}^j, p_t^i) = \frac{HammingDistance\left(r_{\tau_{t-1}^j}, r_{p_t^i}\right)}{len(r)} \tag{3.3}$$

$$c_{spatial}(\tau_{t-1}^j, p_t^i) = \left(1 - IoU(b_{\tau_{t-1}^j}, b_{p_t^i})\right) \tag{3.4}$$

dove $len(r)$ esprime il numero di bit che compongono un *region encoding*.

L'equazione 3.3 modella la distanza visiva tra gli elementi della coppia, utilizzando come *proxy* la distanza di Hamming tra i relativi *region encoding*, l'equazione 3.4, invece, calcola la distanza spaziale tra gli elementi, che è tanto maggiore quanto più basso è il punteggio di *intersection over union* tra le loro *bounding box*. Per procedere all'assegnamento si definisce una matrice di costi, il costo per una coppia $(\tau_{t-1}^j, p_t^i) \in \mathcal{T}_t - 1 \times \mathcal{P}_t^{high}$ è dato da:

$$c(\tau_{t-1}^j, p_t^i) = \alpha c_{visual}(\tau_{t-1}^j, p_t^i) + (1-\alpha) c_{spatial}(\tau_{t-1}^j, p_t^i), \quad \text{con } \alpha \in [0,1] \tag{3.5}$$

Il risultato condensa distanza visiva e spaziale in un unico valore, attribuendo pesi diversi alle due misure sulla base del fattore α, negli esperimenti condotti è stato fissato $\alpha = 0.6$ (questo valore è giustificato nella sezione 5.2.1). Successivamente, si utilizza l'Algoritmo Ungherese (sezione 2.5) per trovare la soluzione ottimale. Dagli assegnamenti ottenuti sono scartati tutti quelli il cui costo supera una soglia $t_{assignment} = 0.5$ (giustificata in sezione 5.2.2). L'insieme di traiettorie che hanno ricevuto un assegnamento è indicato con $\mathcal{T}_t^a$, mentre le restanti traiettorie formano $\mathcal{T}_t^u$, allo stesso modo le predizioni di $\mathcal{P}_t^{high}$ assegnate ad una traiettoria formano l'insieme $\mathcal{P}_t^{high,a}$, le rimanenti formano $\mathcal{P}_t^{high,u}$.
Si procede quindi con l'assegnamento delle predizioni in $\mathcal{P}_t^{low}$, ovvero quelle a bassa confidenza, con le traiettorie ancora disponibili, rappresentate da $\mathcal{T}_t^u$. La modalità di assegnamento è analoga a quella utilizzata nella prima sottofase, ma la funzione di costo considera solo la distanza spaziale tra i candidati:

$$c(\tau_{t-1}^j, p_t^i) = c_{spatial}(\tau_{t-1}^j, p_t^i) \tag{3.6}$$

e la soglia $t_{assignment}$ è impostata a 0.1.
La distinzione tra predizioni ad alta e a bassa confidenza si basa sull'assunzione che un livello di confidenza troppo basso corrisponda ad una regione dell'immagine particolarmente difficile da classificare, a causa di occlusioni, rumore, risoluzione insufficiente o altri problemi simili. Di conseguenza anche la capacità rappresentativa del relativo *region encoding* diminuisce e l'assegnamento può dunque affidarsi solo alle proprietà della *bounding box* della regione stessa.
Alla fine della seconda sottofase l'insieme $\mathcal{T}_t^a$ viene aggiornato inserendo le traiettorie assegnate, e si definiscono gli insiemi $\mathcal{P}_t^{low,a}$ e $\mathcal{P}_t^{low,u}$, come avvenuto per $\mathcal{P}_t^{high,a}$ e $\mathcal{P}_t^{high,u}$ al termine della prima sottofase.
Infine si effettuano i seguenti aggiornamenti:

- Tutte le traiettorie in $\mathcal{T}_t^a$ sono aggiornate inserendo i dati della predizione corrispondente di $\mathcal{P}_t^{high,a} \cup \mathcal{P}_t^{low,a}$ e resettando il loro valore di *age* a 0.
- Si incrementa di 1 il valore di *age* per ciascuna traiettoria in $\mathcal{T}_t^u$ e, data una soglia t_{age}, si definisce l'insieme $\mathcal{T}_t^{u,kept} = \{\tau \in \mathcal{T}_t^u | age_\tau < t_{age}\}$. Negli esperimenti condotti è stata fissata $t_{age} = 60$.
- Si crea una nuova traiettoria per ciascuna predizione in $\mathcal{P}_t^{high,u}$, definendo quindi un insieme T_t^{new}.
- Tutte le predizioni in $\mathcal{P}_t^{low,u}$ sono scartate.
- Si definisce l'insieme $\mathcal{T}_t = T_t^a \cup T_t^{u,kept} \cup T_t^{new}$

L'utilizzo di t_{age} per filtrare le traiettorie non assegnate è necessario per permettere al sistema di gestire l'eventuale scomparsa di alcuni soggetti dal flusso video, che possono uscire dal campo visivo o scomparire in seguito a un cambio di inquadratura. Inoltre le predizioni a bassa confidenza in $\mathcal{P}_t^{low,u}$ vengono rimosse e non generano alcuna nuova traiettoria.
Al termine della fase di associazione ciascuna predizione in $\mathcal{P}_t \setminus \mathcal{P}_t^{low,u}$ è arricchita con l'*id* j della traiettoria associata, e $\mathcal{T}_t$ rappresenta il nuovo stato del sistema.

3.1.4 Classificazione

La fase di associazione determina gli accoppiamenti tra traiettorie e nuove predizioni, minimizzando il costo complessivo dell'assegnamento. Tuttavia, non prevede alcun controllo sulla coerenza semantica tra le etichette di classe delle regioni associate. In altri termini, data una traiettoria τ, il cui rappresentante p_τ ha un'etichetta di classe c_{p_τ}, non vi è alcuna garanzia che la prossima regione assegnata alla traiettoria corrisponda anch'essa alla categoria c_{p_τ}. L'assenza di un vincolo esplicito sulla congruenza tra le categorie permette al sistema di mantenere una certa robustezza rispetto ad errori di classificazione, che, altrimenti, causerebbero un'eccessiva frammentazione delle traiettorie. Può però risultare vantaggioso sfruttare l'informazione temporale e semantica delle regioni appartenenti ad una stessa traiettoria per correggere gli errori di classificazione introdotti dalla *backbone* di *object-detection*. A tal proposito, durante la fase di associazione - una volta stabilito l'assegnamento ottimo - il sistema di *tracking* può aggiornare l'etichetta di classe del rappresentante di ciascuna traiettoria, basandosi sulle etichette delle regioni precedentemente associate. Per farlo sono state esplorate due strategie, mutuamente esclusive, descritte di seguito:

- ***Highest score***: la categoria assegnata al nuovo rappresentante corrisponde a quella della predizione con il punteggio di confidenza più alto tra il rappresentante corrente e la nuova regione associata alla traiettoria.
- ***Majority voting***: la categoria assegnata al nuovo rappresentante è determinata dalla classe più frequentemente predetta tra le ultime n regioni assegnate alla traiettoria, con $n = 10$.

Le differenze in termini di accuratezza di classificazione tra le due strategie sono mostrate con l'esperimento descritto nella sezione 4.3.

3.2 Architettura

L'architettura utilizzata per effettuare le predizioni sui frame video è formata da tre moduli principali, una ***backbone*** di *open-vocabulary object detection*, una ***embedding net*** ed un modulo di LLSH, denominato ***hashing net***. La coppia *embedding net* e *hashing net* rappresenta il fulcro del sistema di *tracking* ideato, a cui è stato dato il nome di BEHEMOTH. Il funzionamento e le modalità di interazione tra questi componenti sono descritti di seguito.

3.2.1 Backbone

La rete di *object detection* ha lo scopo di individuare e classificare le regioni corrispondenti ai nomi degli oggetti specificati nella *query* testuale. Considerata la mole di risorse necessaria ad addestrare un sistema di *open-vocabulary object detection* è stato deciso di sfruttare dei modelli pre-addestrati, i cui pesi sono disponibili pubblicamente. In particolare sono stati condotti degli esperimenti sfruttando tre diversi modelli: **RegionCLIP** [25], **Omdet-Turbo** [30] e **YOLO-World** [33], il primo ha prestazioni migliori in termini di localizzazione e classificazione, ma richiede maggiori risorse computazionali durante l'inferenza, gli altri due invece sono – al momento della stesura di questo elaborato – gli unici modelli di *open-vocabulary object detection* a raggiungere prestazioni *real-time*, sebbene a scapito di una minore accuratezza. Le reti sopra menzionate differiscono per architettura ed implementazione, ma, ad alto livello, si basano sullo stesso meccanismo. Sono formate da un *text encoder*, un modulo di *image encoding* e *object detection* ed un classificatore finale.
Il *text encoder* riceve una *query* testuale contenente i nomi degli oggetti da individuare, e li mappa in uno spazio vettoriale ad elevata dimensionalità, ottenendo un *embedding* per ciascuna delle categorie specificate. In questa fase tutte e tre le reti sfruttano il *text encoder* di CLIP [20] (sezione 2.3), addestrato a proiettare informazioni testuali in uno spazio vettoriale latente condiviso col proprio *image encoder*. Gli *embedding* sono memorizzati in una *cache*, in modo da ridurre sensibilmente il costo computazionale delle operazioni di inferenza nel caso in cui l'insieme di classi rimanga costante nel corso di elaborazioni successive.
Nonostante CLIP offra un modulo di *image encoding* preaddestrato, tutte e tre le reti adottate ne sviluppano uno proprio, basandosi su architetture diverse. RegionCLIP sfrutta una *Feature Pyramid Network* (FPN) [58] costruita su **ResNet-50** o su **ResNet-50x4** [5] a seconda della versione, Omdet-Turbo sfrutta una **Swin-T** [24], mentre YOLO-World è basato sull'architettura di **YOLOv8** [34]. Lo scopo dell'*image encoder* è duplice: da

un lato integra funzionalità di *region proposal*, che permettono alla rete di individuare le regioni dell'immagine con maggiore probabilità di contenere gli oggetti cercati; dall'altro si occupa di proiettare il contenuto visivo di queste regioni all'interno dello stesso spazio vettoriale utilizzato per gli *embedding* testuali. Alle regioni identificate dal modulo di *region proposal* è inoltre applicata *non-maximum suppression* (NMS), per eliminare le *bounding box* ridondanti.
Una volta ottenuti gli *embedding* di ciascuna regione, il classificatore costruisce una matrice di costi, ottenuti calcolando il prodotto scalare tra gli *embedding* testuali e quelli relativi all'immagine. Le righe della matrice corrispondono alle regioni dell'immagine, le colonne alle etichette di classe. Infine la funzione di *softmax* è applicata separatamente ad ogni riga della matrice, ottenendo una distribuzione di probabilità che indica, per ciascuna regione, la *likelihood* di appartenenza a ognuna delle classi. L'indice di classe assegnato ad una regione è quello che ne massimizza la *likelihood* in questione. L'output della *backbone* di *object detection* è costituito dall'insieme delle regioni individuate, caratterizzate da una *bounding box*, un'etichetta di classe, la confidenza della predizione e, in alcuni esperimenti, anche dal relativo *embedding* visivo calcolato dal modulo di *image encoding*.

3.2.2 Embedding Net

Motivazione

Gli *embedding* vettoriali assegnati a ciascuna regione dal modulo di *image encoding* risultano adeguati per l'operazione di classificazione, ma non sono sufficientemente robusti per essere utilizzati durante il *tracking*.
Una possibile soluzione consiste nel riaddestrare la *backbone* sia su dataset di immagini che su dataset video, cercando di ottimizzare allo stesso tempo le prestazioni di localizzazione, classificazione e rappresentazione delle regioni attraverso un unico addestramento. Ciò risulta estremamente difficile da realizzare in pratica, per una serie di motivi. Innanzitutto, per mantenere le prestazioni delle reti preaddestrate, sarebbe necessario avere accesso agli stessi dati originariamente utilizzati per addestrarle, spesso composti sia da dataset pubblici, facilmente reperibili, che da dataset personalizzati e non pubblicamente accessibili. Inoltre, le risorse computazionali ed i tempi stimati per l'addestramento da zero di questi sistemi non sono risultati compatibili con le risorse a nostra disposizione. Sebbene sia possibile alleviare queste problematiche limitandosi ad un *fine-tuning* delle reti, partendo dalle loro versioni preaddestrate, ciò renderebbe necessario l'utilizzo di un dataset video pubblico, dotato di annotazioni *open-vocabulary* ed estremamente variegato, per mantenere elevata la capacità di generalizzazione della rete e riuscire allo stesso tempo a raffinare i *region embedding* prodotti da essa, un dataset simile al momento non è pubblicamente disponibile. Il candidato maggiormente qualificato è TAO [49], il cui *training set* è formato da 500 sequenze video, talvolta con soggetti ripetuti, e che, pur essendo risultato funzionale per l'apprendimento del solo compito di *tracking* (sezione 3.3.1), non è sufficientemente variegato per garantire un apprendimento robusto della funzione di classificazione.
Infine, per quanto siano necessari ulteriori dati sperimentali per poterlo affermare con certezza, è ragionevole ipotizzare che i moduli di *image encoding* delle *backbone* utilizzate non abbiano una capacità espressiva sufficiente ad ottenere dei *region embedding* che risultino adatti sia alla classificazione *open-vocabulary* che al *tracking*. Per tutti i motivi sopra elencati è stato deciso di addestrare un modulo di *region embedding* dedicato.

Funzionamento

L'*embedding net* ha lo scopo di ricavare rappresentazioni vettoriali significative per le diverse regioni individuate dalla *backbone*. Questo modulo è costituito dall'accoppiamento di una *backbone* convoluzionale ed una FPN, la prima produce *feature map* profonde a diversa risoluzione a partire dall'immagine in input, la seconda sfrutta le *bounding box* ottenute dalla fase di *object detection* per selezionare le regioni corrispondenti delle *feature map* a disposizione, attraverso un'operazione di *ROI align*. La rete produce delle *feature map* di risoluzione variabile per ciascuna regione, i *region embedding* corrispondenti sono ottenuti applicando un'operazione di *global average pooling* seguita da un *layer* lineare. Sono stati condotti esperimenti utilizzando diverse *backbone*, tra cui **EfficientNet-B0**, **EfficientNet-B3** [59] e **MobileNetV3** [60]. In alcuni esperimenti è stato adottato un modulo di *self-attention*, denominato **AttentionNet**, per sostituire l'*embedding net* convoluzionale sopra descritta, nel tentativo di riutilizzare i *region embedding* prodotti dalla *backbone* di *object detection*. Il modulo di **AttentionNet** riceve i *region embedding* direttamente dalla *backbone*, che sono utilizzati come *token* per eseguire tre passi di *self-attention*. I *positional encoding* di ciascun *token* sono appresi dalla rete a partire dalle caratteristiche spaziali delle regioni corrispondenti agli *embedding* in input. In particolare da ogni regione si estrae una tupla $(x_1, x_2, y_1, y_2, w, h, A, ar)$ dove:

- $\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{y_1}$ e $\boldsymbol{y_2}$: sono le coordinate della *bounding box* della regione.
- $\boldsymbol{w}$ e $\boldsymbol{h}$ sono rispettivamente la larghezza e l'altezza della regione.
- $\boldsymbol{A}$ è l'area occupata dalla regione.
- $\boldsymbol{ar}$: è il rapporto d'aspetto della regione.

Tutti i valori che formano la tupla sono inoltre normalizzati nell'intervallo [0,1], in modo da essere indipendenti rispetto alla risoluzione dell'immagine sorgente. La tupla così definita è elaborata da un *layer* lineare, che produce i *positional encoding* da sommare ai *token*.

3.2.3 Hashing Net

L'*hashing net* della rete riceve i *region embedding* prodotti dall'*embedding net* e ne produce le corrispondenti versioni quantizzate denominate *region encoding*, che hanno la forma di vettori $\overline{\boldsymbol{r}} \in \mathbb{R}^{len(r)}$.
L'*hashing net* è costituita da un *multi layer perceptron* (MLP) ed approssima una funzione di LLSH (sezione 1.4), gli output infatti, pur non essendo vettori in $\{0,1\}^{len(r)}$, sono generati in modo tale che i singoli elementi abbiano valori il più possibile vicini a -1 o 1[1], questo comportamento è ottenuto mediante l'uso di una loss di quantizzazione (sezione 3.3.2) durante l'addestramento. Il primo *layer* dell'MLP non utilizza il termine di bias, ed i pesi sono inizializzati secondo una distribuzione gaussiana $\mathcal{N}(0,1)$, in questo modo il layer implementa di fatto una funzione *random hyperplanes*, e gli iperpiani ottimi sono appresi durante l'addestramento della rete. I *layer* successivi manipolano il risultato della

[1]Si noti che un qualsiasi vettore in $\{0,1\}^n$ può essere messo in corrispondenza biunivoca con un vettore in $\{-1,1\}^n$ semplicemente mappando gli 0 in -1.

funzione *random hyperplanes* e ne migliorano la quantizzazione. Durante l'inferenza la quantizzazione effettiva si ottiene applicando la funzione segno all'output, ciò garantisce di ottenere un vettore binario $\boldsymbol{r}$, formalmente:

$$\boldsymbol{r} = \begin{bmatrix} r_0 \\ r_1 \\ \vdots \\ r_{len(r)-1} \end{bmatrix}, \quad \text{dove} \quad r_i = sign(\overline{r_i}) = \begin{cases} 1, & \text{se} \quad \overline{r_i} \geq 0 \\ -1 & \text{se} \quad \overline{r_i} < 0 \end{cases} \tag{3.7}$$

Durante l'addestramento, invece, la loss di quantizzazione è calcolata a partire dagli output grezzi dell'MLP. Lo scopo dell'*hashing net* è quello di comprimere l'informazione contenuta nei *region embedding*, in questo modo la similarità tra una predizione ed una traiettoria può essere espressa in funzione della distanza di Hamming dei codici corrispondenti (si veda l'eq. 3.3), meno complessa da calcolare rispetto ad altre misure come distanza euclidea o distanza coseno. Inoltre il passaggio da *region embedding* in $\mathbb{R}^n$, a codici binari, riduce sensibilmente l'utilizzo di memoria del sistema, e quindi il costo della memorizzazione dell'insieme di traiettorie $\mathcal{T}$ durante il *tracking* (sezione 3.1).

3.3 Addestramento

Tra i moduli descritti nella sezione 3.2, l'*embedding net* e l'*hashing net* sono addestrati congiuntamente, mentre per la *backbone* di *object detection* sono utilizzati i pesi pubblici resi disponibili dai rispettivi sviluppatori. Lo scopo dell'addestramento è quello di ottenere un modello in grado di produrre *region encoding* simili per regioni appartenenti alla stessa traiettoria. La similitudine tra due *region encoding* r_i, r_j di lunghezza $len(r)$ è modellata dalla seguente formula:

$$sim(r_i, r_j) = len(r) - HammingDistance(r_i, r_j) \tag{3.8}$$

La rete implementa una funzione $h : [0{,}255]^{H \times W \times 3} \rightarrow \{0{,}1\}^{len(r)}$, che mappa ciascuna regione di un'immagine in un codice binario. Come descritto all'inizio della sezione 3.1.3, dato un flusso video generico, si assume che lo stesso soggetto mantenga coerenza visiva in frame temporalmente vicini. Pertanto, l'aspettativa è che gli output della funzione h, ovvero i *region encoding*, siano discriminativi rispetto all'identità visiva della regione di input, consentendo una rappresentazione robusta rispetto a lievi variazioni temporali e spaziali.

3.3.1 Dataset e Data Processing

Scelta del dataset

Per poter addestrare la rete al compito prefissato in modo efficace è necessario utilizzare un dataset video variegato, con numerosi soggetti di aspetto e dimensioni diverse, in modo da garantire che la rete finale abbia una buona capacità di generalizzazione. Inoltre è necessario lavorare su un dataset pensato per *multi-object tracking*, per mantenere elevata la capacità discriminativa della rete in scene complesse.
La scelta è quindi ricaduta su TAO [49], che attualmente costituisce lo standard per il *benchmarking* di sistemi di *open-vocabulary multi-object tracking*.

Il *training set* è formato da 500 sequenze video della durata di un minuto e mezzo circa, provenienti da contesti eterogenei e annotate alla frequenza di 1 FPS, i.e. un frame annotato per ogni secondo di video. Esiste anche una versione densamente annotata di TAO, chiamata BURST [50], che fornisce annotazioni a 5 FPS, aumentando notevolmente la quantità di frame annotati. La versione del dataset utilizzata per addestrare il sistema è stata BURST, mentre il *benchmarking* è stato effettuato sulla versione originale di TAO. Durante l'addestramento ogni sequenza è suddivisa in brevi clip disgiunte, composte ciascuna da 5 frame annotati consecutivi, ogni clip corrisponde ad un *data point* su cui sono calcolate le funzioni di loss (3.3.2) a partire dai *region encoding* corrispondenti.

Augmentations e preprocessing

Ciascun frame del dataset attraversa una *pipeline* di *preprocessing* piuttosto semplice, per renderlo compatibile con il formato atteso dalla rete. In particolare il valore di ciascun canale c viene prima riscalato nell'intervallo [0,1], poi normalizzato secondo la rispettiva media e deviazione standard μ_c, σ_c calcolate a partire da ImageNet [61], dove:

$$\begin{aligned} \mu_r &= 0.485, \quad \sigma_r = 0.229 \\ \mu_g &= 0.456, \quad \sigma_g = 0.224 \\ \mu_b &= 0.406, \quad \sigma_b = 0.225 \end{aligned}$$

Infine il frame viene ridimensionato in modo da corrispondere alla risoluzione attesa dal modello, in particolare l'*embedding net* (sezione 3.2.2) accetta immagini con risoluzione di 512×512 pixel, mentre la risoluzione dell'immagine in input alla *backbone* di *object detection* varia a seconda del modulo utilizzato: RegionCLIP accetta immagini di risoluzione variabile, ed esegue un ridimensionamento dinamico, portando ciascuna dimensione al multiplo di 32 più vicino al valore originale, mentre per OmDet e YOLO-World la risoluzione è stata fissata a 640×640 pixel.
Inoltre, per aumentare la variabilità del dataset e la capacità di generalizzazione della rete, ad ogni frame possono essere applicate delle *augmentation*. Ciascuna *augmentation* è utilizzata con probabilità p_a, più *augmentation* possono essere applicate cumulativamente ad uno stesso frame. Le *augmentation* utilizzate sono:

- **Ribaltamento Orizzontale**, con probabilità $p_r = 0.5$
- **Color Jitter**, con probabilità $p_j = 0.4$ e perturbazioni scelte con probabilità uniforme tra i seguenti intervalli:
  - [0.6,1.4] per il valore di luminosità.
  - $[0.8, 1.2]$ per i valori di contrasto e saturazione.
  - $[-0.05, 0.05]$ per il valore di *hue*.
- **Rumore Gaussiano** $\mathcal{N}(0,0.1^2)$, con probabilità $p_g = 0.4$
- **Random Occlusion**, con probabilità $p_o = 0.3$. In questo caso la probabilità di applicazione non riguarda il singolo frame ma ciascuna regione di *ground truth* contenuta in esso. Il funzionamento di questa *augmentation* è descritto con maggiore dettaglio nel paragrafo **Random Occlusion** (3.3.1).

Random Occlusion Uno degli aspetti critici del *tracking* è la capacità di mantenere stabili le traiettorie in corrispondenza di occlusioni parziali dei soggetti e di reidentificarli in seguito ad occlusioni totali. Per rendere la rete più robusta in questo compito è stata introdotta una tecnica di *augmentation* ad-hoc, denominata *random occlusion*.
La strategia prevede di applicare, per ogni regione di *ground truth* del frame originale, una *patch* di occlusione con probabilità p_o. L'area coperta dalla *patch* di occlusione è scelta con probabilità uniforme in un intervallo compreso tra il 10% ed il 33% dell'area della regione, ed il rapporto d'aspetto è scelto, sempre con probabilità uniforme, nell'intervallo $[0.5625, 1.7778]$. In alcuni casi può accadere che una *patch* di occlusione non riguardi solo la regione di riferimento, ma che vada a coprire, totalmente o parzialmente, anche un'altra regione, ad esempio nel caso in cui la seconda sia contenuta all'interno della prima. Per questo motivo tutte le regioni che, a causa di una o più *patch* di occlusione, sono coperte per più del 50% della propria area sono eliminate dalle annotazioni di *ground truth*.
Un'occlusione può essere generata in due diversi modi:

- Con probabilità $p_u = 0.33$ la *patch* corrisponde ad un regione di colore uniforme, con $R = 104$, $G = 116$, $B = 124$ (figura 3.3).
- Con probabilità $p_{cp} = 0.67$ si utilizza una strategia più raffinata, basata sulla tecninca **CutPaste**, introdotta nello studio di [62]. Con questa modalità una *patch* di occlusione è ottenuta copiando una regione casuale dell'immagine e replicandola nella zona che si desidera occludere (figura 3.4).

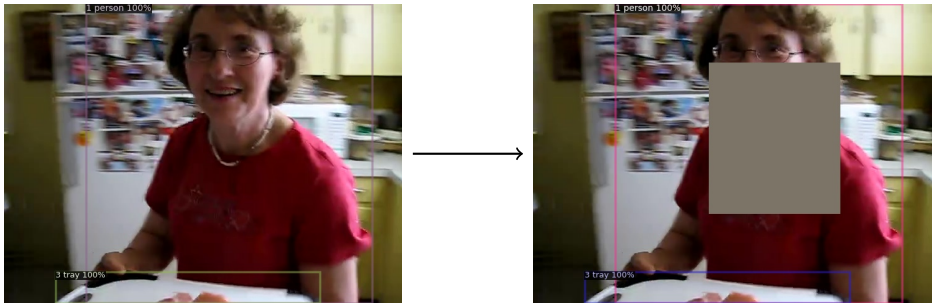


Figura 3.3: Esempio di occlusione uniforme

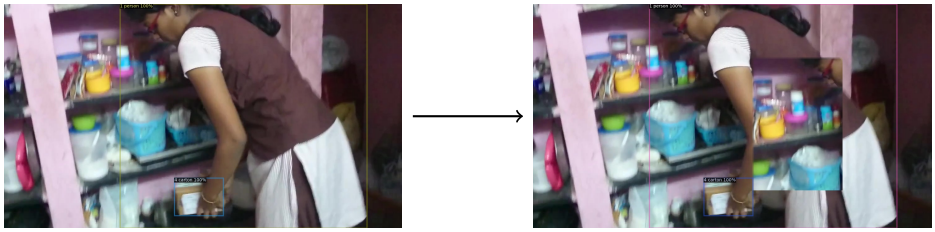


Figura 3.4: Esempio di occlusione attraverso CutPaste

3.3.2 Loss

Per permettere alla coppia formata da *embedding net* e *hashing net* (sezioni 3.2.2 e 3.2.3) di raggiungere l'obiettivo prefissato, ovvero la generazione di *region encoding* binari

utilizzabili per discriminare in modo accurato tra le regioni appartenenti a traiettorie diverse, è stata utilizzata una coppia di due loss: la *triplet loss* e la *quantization loss*. Gli input di entrambe le loss sono ottenuti a partire dai *region encoding* prodotti dall'*hashing net*, senza che vi sia applicata la funzione segno, in modo da non compromettere il calcolo dei gradienti. Gli input sono quindi dei vettori $\overline{\boldsymbol{r}} \in \mathbb{R}^{len(r)}$, dove $len(r)$ è la dimensione di un *region encoding*.

Triplet Loss

La *triplet loss* ha lo scopo di spingere il modello a massimizzare la distanza tra *region encoding* di regioni appartenenti a traiettorie differenti, aumentando in questo modo la capacità rappresentativa della rete. L'input della *triplet loss* è una tripletta di *region encoding* $(\overline{\boldsymbol{r_a}}, \overline{\boldsymbol{r_p}}, \overline{\boldsymbol{r_n}})$ dove $\overline{\boldsymbol{r_a}}$ ed $\overline{\boldsymbol{r_p}}$ appartengono a regioni della stessa traiettoria e sono chiamati rispettivamente **ancora** e **positivo** (o **esempio positivo**), mentre $\overline{\boldsymbol{r_n}}$ è chiamato **negativo** (o **esempio negativo**) e codifica la regione di una traiettoria diversa.
La *triplet loss* per una singola tripletta è formalmente definita come:

$$l_t(\overline{\boldsymbol{r_a}}, \overline{\boldsymbol{r_p}}, \overline{\boldsymbol{r_n}}) = \max(d(\overline{\boldsymbol{r_a}}, \overline{\boldsymbol{r_p}}) - d(\overline{\boldsymbol{r_a}}, \overline{\boldsymbol{r_n}}) + m, 0) \tag{3.9}$$

dove d è una misura di distanza nello spazio vettoriale di riferimento e m è detto margine.

Il margine rappresenta il valore minimo della differenza tra $d(\overline{\boldsymbol{r_a}}, \overline{\boldsymbol{r_p}})$ e $d(\overline{\boldsymbol{r_a}}, \overline{\boldsymbol{r_n}})$ necessario affinché la loss sia 0. Maggiore è il valore del margine, maggiori sono sia la capacità di separazione che si richiede alla rete, sia la difficoltà del processo di apprendimento. Inoltre, valori eccessivamente elevati del margine spingono la rete a produrre *region encoding* appartenenti a *cluster* notevolmente separati. Controintuitivamente, questo causa una diminuzione della capacità rappresentativa della rete, in quanto due regioni con differenze moderate, ma non eccessive, dovute ad esempio a un'occlusione temporanea del soggetto, rischiano di essere mappate in *region encoding* notevolmente distanti nello spazio di destinazione.
La misura di distanza d utilizzata per implementare la *triplet loss* è definita a partire dalla *cosine similarity* degli input come:

$$d(\overline{\boldsymbol{r_i}}, \overline{\boldsymbol{r_j}}) = 1 - CosineSim(\overline{\boldsymbol{r_i}}, \overline{\boldsymbol{r_j}}) \tag{3.10}$$

ed è stata scelta come *proxy* della distanza di Hamming, che non può essere direttamente calcolata a partire da vettori reali.
Un aspetto critico della *triplet loss* riguarda la generazione delle triplette: se queste sono troppo semplici, ovvero se $d(\overline{\boldsymbol{r_a}}, \overline{\boldsymbol{r_p}}) < d(\overline{\boldsymbol{r_a}}, \overline{\boldsymbol{r_n}})$ il rispettivo contributo alla loss sarà pressoché nullo, se invece si selezionano fin da subito solo le triplette più difficili, ovvero quelle per cui $d(\overline{\boldsymbol{r_a}}, \overline{\boldsymbol{r_p}}) > d(\overline{\boldsymbol{r_a}}, \overline{\boldsymbol{r_n}})$ si rischia un *overfitting* sugli esempi più difficili ed un *clustering* eccessivo degli output, compromettendo la capacità di generalizzazione della rete. Per questo sono state implementate una serie di tecniche di *triplet mining*, in modo da selezionare triplette diverse in fasi differenti dell'addestramento. In generale ogni modalità di *triplet mining* seleziona una singola tripletta per ciascuna traiettoria, tale che l'ancora e l'esempio positivo siano *region encoding* di regioni appartenenti a quella traiettoria. Più nello specifico, le strategie implementate sono le seguenti:

- **random**: Per ogni taiettoria si seleziona una singola tripletta con probabilità uniforme tra tutte le candidate per la traiettoria in questione.

- **semi-hard**: Per ogni traiettoria si applica un filtro selezionando tutte e sole le triplette per cui: $d\left(\overline{\boldsymbol{r_a}}, \overline{\boldsymbol{r_p}}\right) < d\left(\overline{\boldsymbol{r_a}}, \overline{\boldsymbol{r_n}}\right) < d\left(\overline{\boldsymbol{r_a}}, \overline{\boldsymbol{r_p}}\right) + m$, ovvero quelle in cui l'esempio positivo è più vicino all'ancora rispetto al negativo, a meno del margine m. Tra le rimanenti triplette viene selezionata quella che minimizza $d\left(\overline{\boldsymbol{r_a}}, \overline{\boldsymbol{r_n}}\right)$, tale cioè, da avere l'esempio negativo più vicino all'ancora. Lo scopo è quello di ignorare le triplette troppo semplici, selezionando solo quelle che contribuiscono attivamente alla loss, ma senza ricorrere a triplette troppo difficili in cui l'ancora è più vicina all'esempio negativo che a quello positivo.

- **hardest-positive**: Per ogni traiettoria si seleziona una tripletta a caso tra quelle la cui ancora e positivo massimizzano: $d\left(\overline{\boldsymbol{r_a}}, \overline{\boldsymbol{r_p}}\right)$. In questo modo si assegna maggiore importanza alle regioni che appartangono alla stessa traiettoria ma che sono mappate in *region encoding* notevolmente diversi. L'obiettivo è quello di rendere la rappresentazione più robusta rispetto a occlusioni, rumore ed altri elementi di disturbo.

È stata quindi definita una procedura incrementale che, modificando la modalità di *triplet mining* ed aumentando il valore di margine, complica gradualmente i requisiti di separazione tra *region encoding*, permettendo alla rete di convergere in modo efficace. La tabella 3.1 mostra i valori di margine e le strategie utilizzati nelle diverse epoche di addestramento. Per ogni *batch* la loss finale L_t è ottenuta come media della *triplet loss* calcolata per le singole triplette estratte.
L'aumento progressivo del margine ed il *triplet mining* rendono la *training loss* rumorosa e causano incrementi netti nei valori registrati all'inizio di ciascuna fase di addestramento. Per verificare l'andamento effettivo dell'addestramento la *validation loss* è invece calcolata su tutte le triplette estratte, senza *triplet mining*, ed utilizzando un margine $m = 2$, corrispondente al massimo valore ammesso da $d(\overline{\boldsymbol{r_i}}, \overline{\boldsymbol{r_j}})$ (eq. 3.10). La figura 3.5 mostra *training loss* e *validation loss* corrispondenti ad uno stesso addestramento. Nonostante le fluttuazioni della *training loss*, la curva della *validation loss* converge efficacemente. Si osserva inoltre che l'ingresso in una nuova fase di addestramento permette di uscire da eventuali minimi locali, dimostrando l'efficacia della strategia proposta.

Tabella 3.1: Strategie e valori di margine utilizzati durante l'addestramento.

Epoche (estremi inclusi)	**Strategie**	**Margine**
0-2	random	0.5
3-5	semi-hard	0.5
6-9	hardest-positive	0.5
10-13	semi-hard	0.7
14-18	hardest-positive	0.7
19-23	semi-hard	0.9
24-28	hardest-positive	0.9
29-33	semi-hard	1.0
34+	hardest-positive	1.0

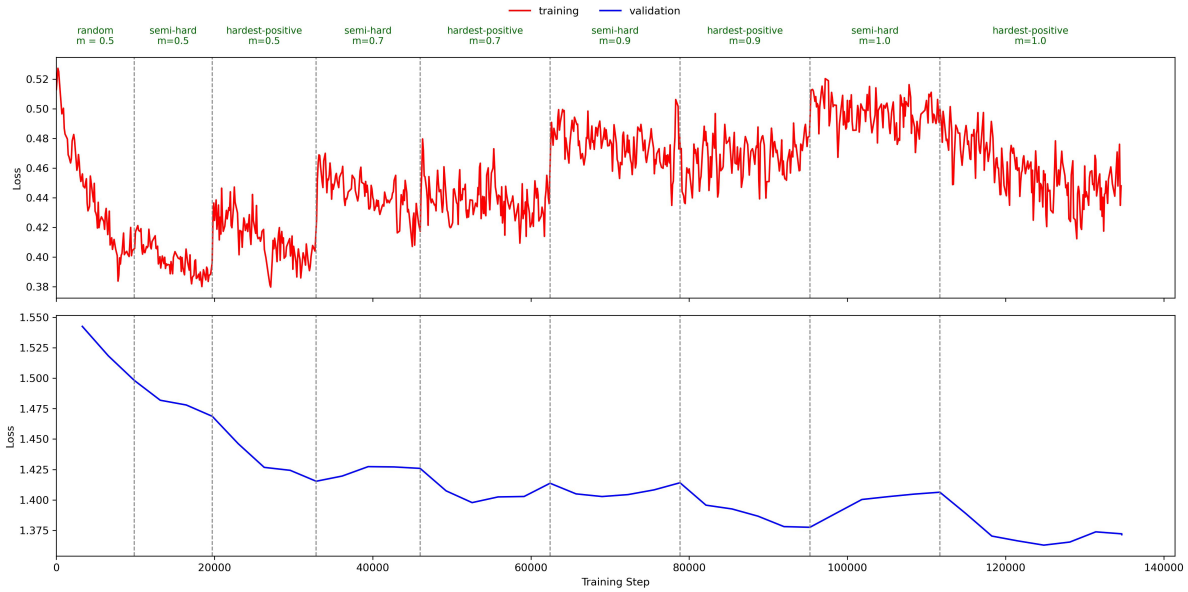


Figura 3.5: *Training loss* e *validation loss* a confronto. La differenza sostanziale tra i valori assoluti registrati è dovuta ai diversi parametri di margine e strategia di selezione adottati.

Quantization Loss

Gli output dell'*hashing net* sono vettori reali $\overline{\boldsymbol{r}} \in \mathbb{R}^{len(r)}$, che vengono successivamente quantizzati attraverso la funzione segno, come indicato nella sezione 3.2.3. Questa operazione introduce inevitabilmente un errore di quantizzazione, tanto maggiore quanto più i singoli elementi di un *region encoding* si allontanano da -1 e 1. Per ovviare a questo problema è stata introdotta una loss di quantizzazione, in modo da portare naturalmente la rete a produrre output che minimizzino l'errore di quantizzazione.
Per un singolo *region encoding* $\overline{\boldsymbol{r}}$ la loss è definita come:

$$l_q(\overline{\boldsymbol{r}}) = \frac{\sum_{i=0}^{len(r)} log\left(cosh\left(|\overline{r_i}| - 1\right)\right)}{len(r)} \tag{3.11}$$

Per ogni *batch* la loss finale L_q è calcolata come media delle loss calcolate per i singoli *region encoding*.

Riduzione

La loss complessiva è data da:

$$L = \gamma_t L_t + \gamma_q L_q \tag{3.12}$$

con $\gamma_t = \gamma_q = 1$

3.3.3 Dettagli e training loop

Come indicato nella sezione 3.3.1 BEHEMOTH è addestrato su clip di 5 frame, raggruppate in *batch* di dimensione b per un totale di $5b$ frame per ogni iterazione. L'algoritmo di ottimizzazione utilizzato è stato *Stochastic Gradient Descent* (SGD), con *momentum*= 0.9, *weight decay*=0.001 (inizialmente 0.0001, poi corretto in seguito ad una fase di *tuning*,

sezione 4.5) ed un *learning rate* iniziale pari a 0.001, diminuito a 0.0001 dopo 36 epoche. Le *triplet loss* e *quantization loss*, calcolate rispettivamente per ogni tripletta ed ogni *region encoding*, sono ridotte prendendone la media aritmetica. Per quanto riguarda la *triplet loss*, vengono escluse le triplette la cui loss è uguale a 0, per evitare di diluire eccessivamente il contributo delle triplette più difficili, e quindi più significative. Ad esempio, sia T l'insieme delle triplette formate a partire dal *batch* in input all'i-esima iterazione, la *triplet loss* complessiva sarà calcolata come:

$$L_t^i = \begin{cases} \frac{1}{|\overline{T}|} \sum_{t \in \overline{T}} l_t^i \text{ , se } |\overline{T}| > 0 \\ 0, \text{ se } |\overline{T}| = 0 \end{cases} \quad , \quad \text{con} \quad \overline{T} = \{t \in T | l_t^i > 0\}$$

All'inizio di ogni epoca il *training set* subisce una permutazione casuale, inoltre vengono ricalcolati i parametri della *triplet loss*, secondo la tabella 3.1. Al termine di un'epoca il modello viene valutato sul *validation set*, anch'esso composto da clip di 5 frame ciascuna, estratte dal *validation set* di TAO. Le metriche calcolate durante questa fase sono:

- **Triplet loss** media.
- **Loss di quantizzazione** media.
- **Loss complessiva** media.
- **Distanza ancora-positivo** e **ancora-negativo** media. Le distanze sono ottenute rispettivamente come distanze coseno tra ciascuna delle ancore selezionate per il calcolo della *triplet loss* ed i rispettivi esempi positivi e negativi.
- **Distanza spaziale** (o **distanza spaziale negativa**) media, minima e massima. La metrica corrisponde alla distanza di Hamming tra il *region encoding* (quantizzato) di una regione e quello di tutte le altre regioni individuate nello stesso frame.
- **Distanza temporale positiva** media, minima e massima. La metrica corrisponde alla distanza di Hamming tra il *region encoding* (quantizzato) di una regione e quello della regione del frame precedente corrispondente alla stessa traiettoria.
- **Distanza temporale negativa** media, minima e massima. La metrica corrisponde alla distanza di Hamming tra il *region encoding* (quantizzato) di una regione e quelli di tutte le regioni del frame precedente corrispondenti a traiettorie diverse.
- **Distanza cross-clip positiva** media, minima e massima. La metrica corrisponde alla distanza di Hamming tra il *region encoding* (quantizzato) di una regione e quelli di tutte le regioni della stessa clip corrispondenti alla stessa traiettoria.
- **Distanza cross-clip negativa** media, minima e massima. La metrica corrisponde alla distanza di Hamming tra il *region encoding* (quantizzato) di una regione e quelli di tutte le regioni della stessa clip corrispondenti a traiettorie diverse.

La loss complessiva media è inoltre utilizzata come criterio per fare *early stopping*: se la loss non diminuisce di almeno un fattore Δ per un numero di epoche pari a p l'addestramento si arresta, ed il modello che ha ottenuto loss di validazione migliore è salvato come modello finale. Negli addestramenti condotti sono stati utilizzati $\Delta = 0.002$ e $p = 5$.
Infine, poichè il modulo di *object detection* non viene addestrato insieme al resto della rete,

la sua funzione è disattivata sia durante l'addestramento che durante la validazione, e le regioni in input all'*embedding net* sono estratte dalle annotazioni di *ground truth* di ciascun frame. Altri sistemi sfruttano le predizioni della *backbone* anche durante l'addestramento, selezionando quelle corrispondenti al *ground truth* sulla base del valore di IoU tra le rispettive *bounding box* [38, 19]. In questo studio, la scelta di addestrare il *tracker* a partire dalle regioni di *ground truth* è stata motivata dalle seguenti osservazioni:

1. L'introduzione di *random occlusion* sintetiche (paragrafo 3.3.1) può alterare, anche significativamente, la qualità delle *detection* della *backbone*, talvolta impedendo l'individuazione delle regioni che subiscono *augmentation*, o alterando significativamente le coordinate delle *bounding box* predette. Le regioni che a causa di ciò non trovano una corrispondenza con un'annotazione di *ground truth* non possono essere utilizzate come esempi positivi per la *triplet loss*. Inoltre, poichè l'addestramento congiunto della *backbone* risulta infattibile per i motivi indicati in 3.2.2, non è possibile migliorarne la robustezza alle occlusioni durante l'addestramento. Di conseguenza si registra sia una diminuzione degli esempi validi per l'addestramento sia una riduzione dell'efficacia della *random occlusion*.

2. Addestrare il *tracker* con le predizioni ottenute da una *backbone* lo incentiva ad adattarsi alla loro struttura e distribuzione. È ipotizzabile che questo migliori l'accuratezza del tracciamento in presenza dello stesso *object detector* utilizzato durante l'addestramento. Allo stesso modo ciò potrebbe ridurre le prestazioni del sistema con altre *backbone*. Di conseguenza, il sistema non risulterebbe più agnostico rispetto alla sorgente delle predizioni. In questo studio è stata preferita la realizzazione di un sistema indipendente dall'*object detector*, per cui è risultato più adatto addestrare direttamente sulle annotazioni del *training set*.

3. Il *training set* di BURST non offre annotazioni esaustive per ogni frame, limitandosi spesso ad un numero esiguo di soggetti rispetto a quelli identificabili dalla *backbone* di *object detection*. Le predizioni così individuate, quando non corrispondenti ad una regione di *ground truth*, possono essere usate come esempi negativi nella *triplet loss*, ma non come esempi positivi. D'altra parte, non si ha nemmeno garanzia che la *backbone* individui tutte le regioni presenti nelle annotazioni. Questo porta ad un'ulteriore riduzione degli esempi positivi validi per l'addestramento.

3.4 Benchmarking

Per la valutazione di BEHEMOTH è stata utilizzata la metrica TETA [18], calcolata a partire dai video del *test set* e del *validation set* di TAO (sezione 3.3.1). TETA costituisce uno standard per la validazione di sistemi *open vocabulary multi-object tracking* e prevede il calcolo di metriche separate per localizzazione, associazione (i.e. *tracking*) e classificazione. Le corrispondenti metriche riassuntive sono *localization accuracy* ($LocA$), *association accuracy* ($AssocA$) e *classification accuracy* ($ClsA$). Poichè il più delle volte i dataset utilizzati in questo ambito non forniscono annotazioni esaustive, TETA evita di punire eccessivamente il *tracker* escludendo dalla valutazione le predizioni ambigue. In particolare ogni regione di *ground truth* è trattata come ancora di un *cluster*, le regioni predette sono assegnate ad un *cluster* se totalizzano un punteggio di IoU superiore a $t = 0.7$ rispetto all'ancora corrispondente. Le regioni che non corrispondono a nessun *cluster* sono escluse dalla valutazione.

Localization Accuracy

La *localization accuracy* è calcolata a partire dagli insiemi TPL, FNL, FPL, che corrispondono rispettivamente ai veri positivi, falsi negativi e falsi positivi di localizzazione. Una regione corrispondente ad una predizione p ($pBox$) è un elemento candidato di TPL se ha un valore di IoU superiore ad una soglia α con almeno una delle regioni di *ground truth* ($gBox$), gli elementi dell'insieme TPL sono selezionati utilizzando l'algoritmo ungherese (sezione 2.5) in modo da massimizzare il punteggio di $LocA$ e $AssocA$. FNL è l'insieme di $gBox$ che non corrispondono a nessuna predizione, infine FPL è formato da tutte le $pBox$ appartenenti ad un *cluster* ma non associate ad alcuna $gBox$. Il punteggio di $LocA$ si ottiene come:

$$LocA = \frac{|TPL|}{|TPL| + |FPL| + |FNL|} \tag{3.13}$$

Association Accuracy

L'*association accuracy* è definita sulla base delle singole regioni $b \in TPL$ in questo modo:

$$AssocA(b) = \frac{|TPA(b)|}{|TPA(b)| + |FPA(b)| + |FNA(b)|} \tag{3.14}$$

dove:

- $TPA(\hat{b}) = \{\bar{b} \in TPL \mid p_t(\bar{b}) = g_t(\bar{b}) = g_t(\hat{b}) \ \wedge \ \bar{b} \neq \hat{b}\}$
- $FPA(\hat{b}) = \{\bar{b} \in TPL \mid p_t(\bar{b}) = g_t(\hat{b}) \ \wedge \ g_t(\bar{b}) \neq g_t(\hat{b})\}$
- $FNA(\hat{b}) = \{\bar{b} \in TPL \mid p_t(\bar{b}) \neq g_t(\hat{b}) \ \wedge \ g_t(\bar{b}) = g_t(\hat{b})\}$

In cui $p_t(b)$ indica l'indice di traiettoria predetto per la regione b e $g_t(b)$ l'indice di traiettoria assegnato alla regione di *ground truth* a cui corrisponde b. La figura 3.6 mostra un esempio schematico degli elementi di TPA, FPA e FNA.
Il punteggio di $AssocA$ complessivo è calcolato come:

$$AssocA = \frac{1}{|TPL|} \sum_{b \in TPL} AssocA(b) \tag{3.15}$$

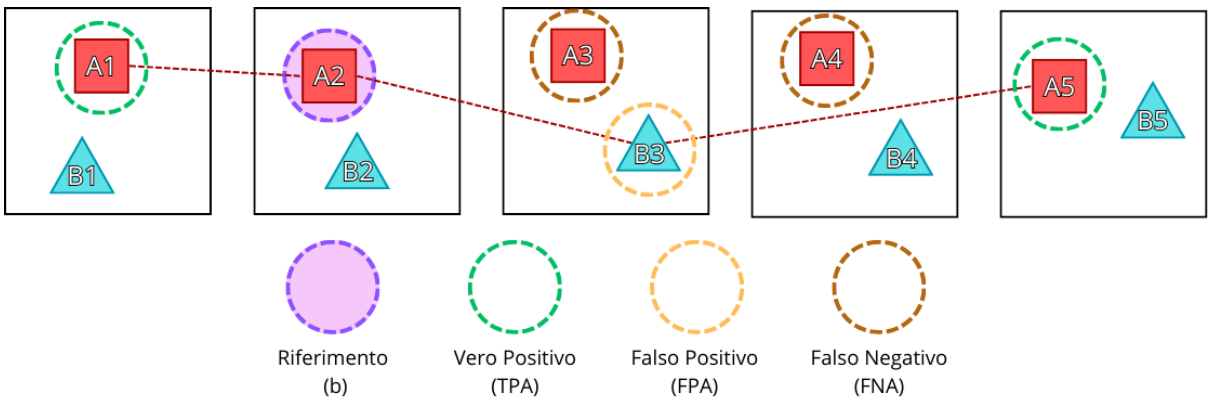


Figura 3.6: Esempio visivo dei concetti di TPA, FPA e FNA per un riferimento b

Classification Accuracy

La *classification accuracy* è ottenuta a partire dagli insiemi TPC, FPC e FNC definiti come segue:

- $TPC = \cup_{\forall c} TPC(c), \quad \text{dove} \quad TPC(c) = \{b \in TPL \mid p_c(b) = g_c(b) = c\}$
- $FPC = \cup_{\forall c} FPC(c), \quad \text{dove} \quad FPC(c) = \{b \in TPL \mid p_c(b) = c \ \wedge \ g_c(b) \neq c\}$
- $FNC = \cup_{\forall c} FNC(c), \quad \text{dove} \quad FNC(c) = \{b \in TPL \mid p_c(b) \neq c \ \wedge \ g_c(b) = c\}$

In cui $p_c(b)$ corrisponde all'etichetta di classe predetta per la regione b e $g_c(b)$ l'indice di classe della regione di *ground truth* a cui corrisponde b. Il punteggio di $ClsA$ è dato da:

$$ClsA = \frac{|TPC|}{|TPC| + |FPC| + |FNL|} \tag{3.16}$$

TETA

La metrica riassuntiva TETA è ottenuta come media aritmetica dei tre valori di *accuracy*:

$$TETA = \frac{LocA + AssocA + ClsA}{3} \tag{3.17}$$

Commenti

Come affermato nella sezione 3.2.1, il modulo che si occupa di classificazione e localizzazione è pre-addestrato e non è sottoposto ad un ulteriore addestramento, pertanto non ci si aspettano variazioni significative nei valori di $ClsA$ e $LocA$ dei diversi modelli prodotti. Inoltre il presente studio si è concentrato sullo sviluppo, implementazione e valutazione del meccanismo di *tracking*, dunque la metrica di riferimento per valutare la qualità del lavoro svolto è stata l'$AssocA$, mentre le altre misure, ed in particolare il punteggio $TETA$, sono servite a contestualizzare la bontà del sistema rispetto ad altri algoritmi già esistenti.

Capitolo 4

Esperimenti e risultati

Di seguito sono riportati gli esperimenti condotti, con le relative motivazioni ed osservazioni. Si noti che, se non esplicitamente indicato, il dataset su cui sono stati calcolati i risultati è stato il *validation set* di TAO, descritto nella sezione 3.3.1.

4.1 Introduzione e obiettivi

Il sistema sviluppato prevede l'implementazione di una tecnica alternativa per fare *open vocabulary multi-object tracking*, che sfrutta un sistema di quantizzazione basato su LLSH. L'aspettativa dunque è quella di riuscire a ridurre i tempi di inferenza rispetto ad altre soluzioni esistenti, sacrificando minimamente l'accuratezza del *tracking*. Le soluzioni prese in considerazione per il confronto e la verifica dei nostri risultati sono state **AOA** [63], **TETer** [18], OVTrack [19] e **RegionCLIP + AED** [38]. Il primo è un modello *closed vocabulary*, da considerare come *baseline*, equivale perciò alle prestazioni minime necessarie affinchè il modello addestrato sia ritenuto valido. TETer è un modello *closed vocabluary* che raggiunge buoni livelli di accuratezza, superando notevolmente la *baseline*. OVTrack ed AED sono entrambi modelli *open vocabulary* non *foundation* che, stando alla piattaforma *PapersWithCode* [64] e alle rispettive pubblicazioni, ottengono i punteggi TETA più alti, con AED che si afferma come miglior modello di OV-MOT. La tabella 4.1 riassume le prestazioni dei modelli appena descritti sul *validation set* di TAO.

Nonostante il sistema di associazione sia basato sull'algoritmo proposto da **ByteTrack** [17], come indicato nella sezione 3.1.3, AOA è stata preferita come *baseline*. ByteTrack, infatti, è un *tracker* ideato per lavorare con un numero ristretto di classi, ed i pesi pubblicamente disponibili sono stati ottenuti addestrandolo sul dataset MOT [51], i cui soggetti sono limitati a persone. Non risulta quindi adatto alla complessità proposta dal panorama *open-vocabulary*. AOA, al contrario, è stato progettato appositamente per il dataset TAO, e riesce a gestire un maggior numero di classi, rappresentando una sfida maggiore.

Si noti che poichè è stato sviluppato un sistema di *tracking* agnostico rispetto alla *backbone* di *object detection*, le metriche di maggiore rilevanza sono TETA, che riassume le prestazioni globali del sistema, e AssocA, che invece rappresenta l'accuratezza del tracciamento effettuato. D'altra parte i valori di ClsA e LocA dipendono in misura maggiore dal sistema di *object detection*, e marginalmente dall'implementazione della fase di associazione, che può talvolta eliminare qualche predizione (si veda la sezione 3.1.3). Pertanto le conclusioni e le decisioni raggiunte a partire dai risultati ottenuti, nonchè i

Tabella 4.1: Risultati dei sistemi scelti come riferimento sul *validation set* di TAO.

Nome	TETA	AssocA	ClsA	LocA
AOA	25.27	30.56	**21.86**	27.461
TETer	33.25	35.02	13.16	51.58
TETer-SwinT	34.61	36.71	15.03	52.10
TETer-HTC	36.85	37.53	15.70	**57.53**
OVTrack	34.7	36.7	18.1	49.3
RegionCLIP + AED	**37.00**	**38.10**	16.2	56.7

relativi commenti, sono dipesi in misura maggiore dai valori delle prime due metriche. La tabella 4.2 riporta inoltre le prestazioni dei modelli di *object detection* utilizzati negli esperimenti, in modo da contestualizzare i valori di ClsA e LocA registrati rispetto alle prestazioni di ciascuna *backbone*.

Tabella 4.2: Risultati delle *backbone* di *object detection*, misurati sul *validation set* del dataset corrispondente. Celle vuote indicano che il dato non è stato fornito dagli sviluppatori.

Nome	COCO mAP	LVIS mAP	LVIS-mini mAP
OmdetTurbo-Tiny	42.5	-	30.3
YOLOv8x-Worldv2	46.7	**28.6**	**35.8**
RegionCLIP-50	**50.4**	28.2	-

Tutti gli esperimenti sono stati condotti in un ambiente **Python 3.9.20**, la versione di **PyTorch** utilizzata è la **2.4.1**, con **torchvision 0.19.1** e **CUDA Toolkit 12.2**.

4.2 Embedding Net

Un elemento determinante per la capacità espressiva del modello è l'*embedding net*, il cui scopo è quello di ricavare i *region embedding* di ciascuna regione individuata dalla *backbone* di *object detection*. È stato quindi deciso di dedicare i primi esperimenti alla ricerca dell'architettura che garantisse i risultati migliori. Oltre all'*embedding net*, è stata variata anche la risoluzione degli input, per determinare se fosse possibile registrare incrementi prestazionali all'aumentare della dimensione delle immagini.

La tabella 4.3 elenca le architetture testate, ulteriori dettagli implementativi sono forniti nella sezione 3.2.2.

La *backbone* di *object-detection* utilizzata è stata **OmdetTurbo-Tiny** [30], per via della sua efficienza, che ha garantito tempi di inferenza più rapidi rispetto alle alternative. La tabella 4.4 mostra invece l'architettura dell'*hashing net* adottata. La decisione di utilizzare *region encoding* a 1024 bit è motivata dallo studio riportato nella sezione 5.1. Infine, tutti gli esperimenti sono stati effettuati adottando la strategia di classificazione denominata *highest score* (sezione 3.1.4).

I risultati ottenuti, riassunti dalla tabella 4.5, mostrano differenze minime tra le architetture

Nome	Architettura	# Self-Attention layers	Ris. Input	Dim. Embedding
BEHEMOTH-B0-224	EfficientNet-B0	-	224×224	512
BEHEMOTH-B3-224	EfficientNet-B3	-	224×224	512
BEHEMOTH-MOB-224	MobileNetV3	-	224×224	512
BEHEMOTH-B0-512	EfficientNet-B0	-	512×512	512
BEHEMOTH-B3-512	EfficientNet-B3	-	512×512	512
BEHEMOTH-MOB-512	MobileNetV3	-	512×512	512
BEHEMOTH-ATT	AttentionNet	3	-	512

Tabella 4.3: Architetture dell'*EmbeddingNet* utilizzate nei primi esperiementi.

Tabella 4.4: Architetture dell'*Hashing Net* utilizzata. La dimensione di output corrisponde al numero di bit che compongono i *region encoding*.

Layer #	Tipologia	Dim. Input	Dim. Output
0	*Linear*	512	1024
1	ReLU	1024	1024
2	*LayerNorm*	1024	1024
3	*Linear*	1024	1024

Tabella 4.5: Risultati al variare di risoluzione ed *embedding net*, confrontati con la *baseline* (AOA) e lo stato dell'arte (AED). I nomi fanno riferimento alle tabelle 4.3 e 4.1.

Nome	TETA	AssocA	ClsA	LocA
Esperimenti				
BEHEMOTH-B0-224	30.089	32.375	<u>11.364</u>	46.529
BEHEMOTH-B3-224	30.293	<u>32.740</u>	**11.637**	46.501
BEHEMOTH-MOB-224	30.012	32.585	10.895	46.556
BEHEMOTH-B0-512	30.170	**32.743**	11.118	<u>46.648</u>
BEHEMOTH-B3-512	<u>32.417</u>	29.992	10.792	**46,767**
BEHEMOTH-MOB-512	**32.743**	29.967	10.839	46.493
BEHEMOTH-ATT	29.988	29.575	10.317	46.472
Baseline				
AOA	25.27	30.56	21.86	27.461
Stato dell'arte				
RegionCLIP + AED	37.00	38.10	16.2	56.7

adottate, attribuibili alla variabilità intrinseca di ciascun addestramento.
Inoltre, tutte le soluzioni proposte si avvicinano al punteggio **AssocA** del modello di *baseline*, talvolta superandolo, mentre non riescono a raggiungere le prestazioni di AED, che mantiene un distacco compreso tra gli 8 ed i 4 punti percentuali. Per quanto riguarda il punteggio **TETA**, la netta superiorità dei modelli addestrati rispetto alla *baseline* è attribuibile principalmente alla migliore capacità di localizzazione da parte della *backbone* di *object-detection* adottata, e parzialmente all'incremento registrato nel punteggio di AssocA. Anche qui AED mantiene un distacco variabile tra gli 8 ed i 5 punti percentuali, che ne evidenzia la robustezza rispetto al *tracker* sviluppato. Basandosi sui risultati ottenuti non è stato possibile determinare in modo univoco l'architettura migliore per realizzare l'*embedding net*, nonostante questo è stato deciso di procedere utilizzando EfficientNet-B0, in quanto è l'unica che ha superato in modo consistente il punteggio

di **AssocA** della baseline in entrambe le sue versioni. La risoluzione dell'input è stata fissata a 512×512, sia per l'incremento, seppur minimo, registrato nell'accuratezza di associazione di **BEHEMOTH-B0-512** rispetto a quella di **BEHEMOTH-B0-224**, sia per una maggiore aderenza all'utilizzo delle FPN descritto in letteratura [58].

4.3 Strategia di classificazione

Come descritto nella sezione 3.1.4 il *tracker* utilizza delle strategie correttive per tentare di mitigare possibili errori di classificazione commessi dalla *backbone*. Per valutare l'efficacia delle due strategie proposte, denominate *highest score* e *majority voting*, è stato condotto un esperimento dedicato, confrontando i punteggi di ClsA ottenuti dal modello **BEHEMOTH-B0-512**.

Tabella 4.6: Risultati di BEHEMOTH-B0-512 al variare della strategia di classificazione.

Strategia	TETA	AssocA	ClsA	LocA
Highest score	30.170	**32.743**	11.118	**46.648**
Majority voting	**31.752**	32.392	**16.243**	46.62

La tabella 4.6 evidenzia i risultati dell'esperimento. Appare evidente il netto miglioramento sul punteggio di ClsA ottenuto con la strategia *majority voting* rispetto a *highest score*, con conseguente incremento del punteggio TETA complessivo. Le variazioni nei valori di AssocA e LocA tra le due soluzioni risultano invece marginali, e possono essere attribuite all'interdipendenza tra le tre componenti di *accuracy* che componongo la metrica TETA (sezione 3.4), più che ad un'alterazione sostanziale del comportamento del *tracker*. Alla luce di questi risultati, è stato deciso di adottare la strategia *majority voting* per tutti gli esperimenti successivi.

4.4 Object Detection Backbone

Le prestazioni del *tracker* dipendono in modo significativo dalla qualità delle predizioni ottenute dalla *backbone* di *object detection*. Per verificare le differenze prestazionali del sistema al variare della *backbone*, sono stati effettuati esperimenti con tre diversi modelli di *open vocabulary object-detection*: **OmdetTurbo-Tiny**, **YoloWorld** e **RegionCLIP**. Le prestazioni di OV-OD delle *backbone* sono visualizzate nella tabella 4.2.
Durante questi esperimenti è stato utilizzato il *tracker* che ha ottenuto il risultato di AssocA migliore nei precedenti esperimenti, ovvero **BEHEMOTH-B0-512** con strategia di *majority voting* per la classificazione. La procedura di inferenza è stata ripetuta con tutte e tre le *backbone* descritte.

I risultati registrati con YOLO-World e RegionCLIP, mostrati nella tabella 4.7, sono commentati e confrontati con quelli di OmdetTurbo-Tiny nei paragrafi successivi.

YOLO-World YOLO-World risulta essere la *backbone* con i tempi di inferenza più rapidi, raggiungendo prestazioni *real-time*, con una velocità di elaborazione superiore ai 32 frame al secondo. Tuttavia, questa rapidità si traduce in una minore accuratezza di localizzazione, evidenziata dalla diminuzione del punteggio di LocA rispetto a quello ottenuto dalle altre due soluzioni. Inoltre, la scarsa precisione nella localizzazione comporta

Tabella 4.7: *Accuracy* di BEHEMOTH-B0-512 con diverse *backbone* di *object-detection*.

Nome	TETA	AssocA	ClsA	LocA	Framerate
Esperimenti					
OmdetTurbo-Tiny	31.752	**32.392**	**16.243**	46.62	16.89 fps
YOLOv8x-Worldv2	25.627	25.803	14.39	36.688	**32.46** fps
RegionCLIP-50	**32.34**	27.405	14.35	**55.264**	1.56 fps
Baseline					
AOA	25.27	30.56	21.86	27.461	-

una significativa riduzione del punteggio di AssocA rispetto alla soluzione che sfrutta OmdetTurbo-Tiny. Ciò impedisce al sistema basato su YOLO-World di raggiungere prestazioni di *tracking* soddisfacenti, ottenendo un punteggio inferiore alla *baseline*.

RegionCLIP Con una velocità di circa 1 frame al secondo, RegionCLIP si dimostra essere la *backbone* meno efficiente dal punto di vista computazionale, ma fornisce prestazioni di localizzazione nettamente superiori rispetto agli altri modelli esaminati. Nonostante ciò, l'accuratezza del tracciamento risulta inferiore rispetto alla *baseline*, evidenziando problematiche di robustezza nel sistema di *tracking*.
La causa del calo di prestazioni non è immediatamente evidente dai risultati quantitavi: intuitivamente, una maggiore precisione nella localizzazione dovrebbe tradursi in un miglioramento dell'accuratezza del *tracking*. Una possibile spiegazione di questo comportamento può emergere dall'analisi qualitativa dei risultati di RegionCLIP ed OmdetTurbo-Tiny. A tale scopo, si considerino gli esempi mostrati in figura 4.1, che riportano annotazioni e predizioni relative a due frame estratti dal *validation set* di TAO, e le analisi riassunte dalla tabella 4.8, che confronta densità e dimensioni delle regioni individuate su tutto il *validation set* di TAO.
È immediato osservare che le regioni di *ground truth* (figura 4.1a) non siano esaustive, ma si limitino ad un sottoinsieme ristretto di soggetti. La sparsità delle annotazioni è tenuta in considerazione dalla metrica TETA, che adotta un sistema di *clustering* per limitare la valutazione alle sole regioni prossime ad una *bounding box* di *ground truth*, escludendo le altre (sezione 3.4). Questo meccanismo è necessario per evitare che predizioni qualitativamente corrette, ma prive di corrispettivi nelle annotazioni di *ground truth*, vengano erroneamente penalizzate. Allo stesso tempo, tuttavia, favorisce reti come RegionCLIP, che generano predizioni estremamente dense e talvolta frammentate (figura 4.1b). Al contrario, predittori più conservativi come OmdeTurbo-Tiny (figura 4.1c) ne traggono meno beneficio. La differenza nella la densità delle regioni rilevate spiega il divario nel punteggio di LocA, nettamente più alto per RegionCLIP.
Inoltre, poichè predizioni più dense e meno localizzate ampliano lo spazio di possibili assegnamenti nella la fase di associazione del *tracker*, è possibile ricondurre la differenza registrata in termini di AssocA ad una debolezza strutturale del sistema, che non risulta sufficientemente robusto per gestire un elevato numero di predizioni. È ragionevole ipotizzare che ciò sia dovuto all'implementazione della funzione di LLSH attraverso l'*hashing net* della rete: un numero elevato di predizioni per frame incrementa le collisioni nello spazio binario di destinazione, compromettendo la capacità discriminativa del *tracker* e generando una frammentazione eccessiva delle traiettorie ricostruite. Tuttavia, i risultati presentati nella sezione 5.1 dimostrano che sostituire *region embedding* con *region encoding* determina un aumento dell'accuratezza di *tracking*. È quindi plausibile che la riduzione

delle prestazioni di associazione sia dovuta anche ad altri fattori, come la dimensione del *training set* o problemi architetturali dell'*embedding net*.
Un ulteriore aspetto qualitativo da valutare riguarda la dimensione delle *bounding box* prodotte dalle *backbone* di *object-detection*: la maggiore frammentazione delle predizioni di RegionCLIP si concretizza spesso in regioni dall'area ridotta, che potrebbero compromettere la qualità dei *region embedding* estratti dalla FPN integrata nell'*embedding net* del sistema.

(a) *Ground Truth* (b) RegionCLIP (c) OmDetTurbo-Tiny

Figura 4.1: Confronto tra le regioni di *ground truth* e quelle individuate da RegionCLIP ed OmDetTurbo-Tiny.

Metodo	**Densità**	**Area**
GT	3.31 annos/frame	105506.99 px
RegionCLIP	40 preds/frame	24217.96 px
OmdetTurbo-Tiny	9.97 preds/frame	65673.62 px

Tabella 4.8: Area e densità medie delle regioni di *ground truth* e delle predizioni di RegionCLIP ed OmdetTurbo-Tiny.

4.5 Hyperparameter Tuning

Per determinare l'impatto di alcuni iperparametri sulle prestazioni del sistema, e contestualmente cercare di migliorarne le prestazioni, è stata condotta una sessione di *hyperparameter tuning* attraverso ricerca Bayesiana. Il modello preso come riferimento è stato **BEHEMOTH-B0-512** con **OmdetTiny-Turbo** come *backbone* e strategia di classificazione *majority voting* (si vedano gli esperimenti 4.2, 4.4, 4.3). Il *framework* adottato per l'implementazione è stato **Optuna** [65].
La metrica ottimizzata è stata il valore della loss ottenuta sul *validation set*, ritenendola un buon *proxy* per valutare le prestazioni del *tracker*. L'insieme di iperparametri coinvolti,

è stato limitato a quelli ritenuti più significativi, in modo da ridurre la dimensione dello spazio di ricerca, essi sono indicati di seguito:

- **Hashing Net**:
  - # *hidden layer*
  - dim. *hidden layer*
- **Embedding Net**:
  - dim. output *embedding*
- **Addestramento**:
  - *learning rate*
  - *weight decay*

Nome	**Valore**	
	BEHEMOTH*	**BEHEMOTH-B0-512**
# *hidden layers*	3	1
dim. *hidden layers*	{1024, 256, 1536}	{1024}
dim. output *embeddings*	256	512
learning rate	0.001	0.001
weight decay	0.001	0.0001

Tabella 4.9: Confronto tra iperparametri originali (BEHEMOTH-B0-512) e quelli individuati attraverso *hyperparameter search* (BEHEMOTH*).

Nome	**TETA**	**AssocA**	**ClsA**	**LocA**	**Loss**
BEHEMOTH-B0-512	31.752	32.392	16.243	**46.620**	1.364
BEHEMOTH*	**32.015**	**33.481**	**16.293**	46.271	**1.358**

Tabella 4.10: Prestazioni del modello ottenuto in seguito ad *hyperparameter tuning* (BEHEMOTH*) e modello di partenza. La voce **Loss** indica la loss minima ottenuta dal modello corrispondente durante le fasi di *validation* effettuate durante l'addestramento.

Gli iperparametri individuati durante questa fase sono elencati nella tabella 4.9, la tabella 4.10 mostra invece le prestazioni del modello corrispondente, indicato con **BEHEMOTH***. BEHEMOTH* ottiene un leggero miglioramento nella *loss* sul *validation* set, che si traduce in un aumento delle capacità di tracciamento, segnalato dal punteggio di AssocA ottenuto. La correlazione tra la diminuzione della *loss* e l'aumento della qualità del *tracking* costituisce un buon indicatore della bontà della *triplet loss* adottata.

4.6 Benchmarking e confronti

Da ultimo il modello ottenuto in seguito ad *hyperparameter tuning* (sezione 4.5), ovvero BEHEMOTH*, è stato confrontato con altre soluzioni di *multi object tracking*, sia dal

punto di vista dell'accuratezza, sia per quanto riguarda i tempi di inferenza. Per i modelli *open-vocabulary* l'accuratezza è stata misurata separatamente per tre diversi insiemi di classi: **Base**, **Novel** ed **All**. Il gruppo **Base** è formato dalle sole classi presenti nel *training set* di TAO, **Novel** al contrario, include le categorie che compaiono esclusivamente nel *validation* e nel *test set*, infine **All** contiene tutte le classi del dataset. Questa distinzione, ormai standard in ambito di OV-MOT [38, 19], risulta fondamentale per evidenziare le capacità *open-vocabulary* dei modelli, contestualizzandone i risultati ed illustrando la loro capacità di generalizzare a classi mai osservate durante l'addestramento.
Infine, i tempi di inferenza sono stati misurati sul *validation set* di TAO, utilizzando un sistema **Ubuntu 20.04.6**, con un processore **AMD Ryzen 7 5800X** ad 8 core, una scheda video **NVIDIA GeForce RTX 3080 Ti** con 12GB di VRAM e 64GB di RAM.

Modalità			All				Base				Novel			
Backbone	**Tracker**	**Split**	**TETA**	**AssocA**	**ClsA**	**LocA**	**TETA**	**AssocA**	**ClsA**	**LocA**	**TETA**	**AssocA**	**ClsA**	**LocA**
RegionCLIP	AED	Val	**37.00**	**38.10**	16.20	**56.70**	**37.30**	**38.40**	17.00	**56.40**	**34.40**	**38.50**	8.70	**55.90**
Integrato	OVTrack	Val	34.70	36.70	18.10	49.30	35.50	36.90	**20.20**	49.30	27.80	33.60	1.50	48.8
RegionCLIP	OVTrack	Val	-	-	-	-	36.30	36.30	18.70	53.90	32.00	33.20	**11.40**	51.40
SwinT	TETer	Val	34.60	36.70	15.00	52.10	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
RegionCLIP	BEHEMOTH*	Val	32.46	27.91	14.90	54.56	32.76	27.87	15.80	54.62	30.18	28.21	8.192	54.14
OmdetTiny-Turbo	BEHEMOTH*	Val	32.01	33.48	16.29	46.27	32.18	33.24	18.37	44.94	31.21	34.54	6.72	52.38
Integrato	AOA	Val	25.27	30.56	**21.86**	27.46	N/A	N/A	17.00	N/A	N/A	N/A	N/A	N/A
RegionCLIP	AED	Test	-	-	-	-	**37.20**	**40.40**	15.60	**55.70**	**27.80**	**29.10**	5.90	48.50
Integrato	OVTrack	Test	-	-	-	-	32.60	35.40	16.90	45.60	24.10	28.70	1.80	41.80
RegionCLIP	OVTrack	Test	-	-	-	-	34.80	36.10	**17.30**	51.10	25.70	26.20	6.10	44.80
SwinT	TETer	Test	-	-	-	-	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
RegionCLIP	BEHEMOTH*	Test	**32.16**	28.23	**14.07**	**54.19**	32.95	29.34	14.90	54.63	26.91	22.73	5.98	**52.03**
OmdetTiny-Turbo	BEHEMOTH*	Test	29.69	**30.62**	13.90	44.54	30.23	31.21	15.35	44.14	26.99	27.67	**6.74**	46.54
Integrato	AOA	Test	-	-	-	-	N/A	N/A	17.00	N/A	N/A	N/A	N/A	N/A

Tabella 4.11: *Accuracy* delle diverse soluzioni di *multi object tracking* a confronto. Le celle indicate con "N/A" indicano che la metrica non è applicabile al modello corrispondente, in quanto *closed voabulary*. Le celle indicate con "-" indicano che il valore non è stato fornito dagli autori delle rispettive pubblicazioni.

Backbone	**Tracker**	**Object Detection**	**Tracking**	**Framerate**	**Speedup**	**Speedup tracking**
RegionCLIP	AED	627.90 ms/frame	55.99 ms/frame	1.46 fps	1×	1×
Integrato	OVTrack	54.50 ms/frame	12.39 ms/frame	14.940 fps	∼ 10.23×	∼ 4.51×
RegionCLIP	OVTrack	627.90 ms/frame	12.39 ms/frame	1.56 fps	∼ 1.06×	∼ 4.51×
SwinT	TETer	51.00 ms/frame	13.28 ms/frame	15.625 fps	∼ 10.70×	∼ 4.21×
OmdetTiny-Turbo	BEHEMOTH*	**48.60** ms/frame	**10.60** ms/frame	**16.89** fps	**∼ 11.55×**	**∼ 5.28×**

Tabella 4.12: Tempi di inferenza medi delle diverse soluzioni di *multi object tracking* a confronto.

4.6.1 Accuratezza

La tabella 4.11 riporta i punteggi TETA dei modelli considerati. Sull'insieme **Base** del *validation set* BEHEMOTH risulta essere il meno accurato. OmDetTiny-Turbo + BEHEMOTH riporta una riduzione del punteggio di AssocA compresa tra l'8% ed il 14%, e una diminuzione di TETA tra il 9% ed il 14%, mentre le percentuali aumentano significativamente in presenza di RegionCLIP come *backbone*.
Osservazioni analoghe valgono per l'insieme **All** del *validation set*, e sull'insieme **Base** del *test set*, dove BEHEMOTH continua a mostrare accuratezza inferiore rispetto allo stato dell'arte, con un netto aumento dello scarto rispetto ad AED, che si dimostra essere il sistema più accurato.
A parità di *backbone*, BEHEMOTH non riesce a raggiungere le prestazioni degli altri

tracker in nessuno degli *split* considerati. Come osservato nella sezione 4.4, BEHEMOTH risulta estremamente sensibile alla *backbone* utilizzata, e non sufficientemente robusto per la gestione di predizioni ad elevatà densità. Le prestazioni di OmDetTiny-Turbo + BEHEMOTH risultano più vicine allo stato dell'arte. Sull'insieme **Novel** del *validation set* supera il punteggio AssocA di entrambe le versioni di OVTrack. Inoltre ottiene un punteggio di TETA maggiore rispetto alla versione di OVTrack con *backbone* integrata, grazie a migliori capacità di localizzazione e classificazione. Sul gruppo **Novel** del *test set* le prestazioni si allineano allo stato dell'arte, talvolta superandolo. OmDetTiny-Turbo + BEHEMOTH ottiene un punteggio TETA migliore rispetto ad entrambe le versioni di OvTrack, e supera RegionCLIP + OvTrack in termini di AssocA. Allo stato attuale tuttavia, l'utilizzo OmdDetTiny-Turbo come *backbone* costituisce un requisito per il raggiungimento di capacità di *tracking* adeguate, ciò rappresenta uno dei limiti più evidenti del sistema. La differenza prestazionale rispetto agli altri *tracker* non può essere attribuita esclusivamente all'utilizo di *region encoding* binari, in quanto, come mostrato dai risultati riportati nella sezione 5.1, essi garantiscono una robustezza maggiore rispetto a *region embedding* reali. Uno dei principali fattori da prendere in considerazione è il *training set*, troppo piccolo per garantire l'apprendimento di *feature* robuste, e con annotazioni troppo poco dense per poter rappresentare correttamente gli output degli *object detector* testati. Inoltre, i risultati dell'esperimento in sezione 4.2 mostrano una scarsa risposta all'*embedding net* (sezione 3.2.2) rispetto a variazioni del *bottom-up* della FPN. Ciò suggerisce sia la presenza di problemi architetturali del modulo, sia una ridotta efficacia della *pipeline* di addestramento ideata. Ulteriori osservazioni sono riportate come futuri sviluppi nella sezione 6.3.

4.6.2 Velocità di inferenza

La tabella 4.12 riporta i tempi medi di inferenza delle soluzioni analizzate. La colonna **Object Detection** include il tempo necessario per le operazioni di *region proposal* e classificazione, mentre la voce **Tracking** si riferisce a tutte le operazioni aggiuntive richieste per il tracciamento, quali la rielaborazione o estrazione dei *region embedding* e la ricostruzione delle traiettorie.
BEHEMOTH si dimostra essere il sistema più rapido in assoluto, raggiungendo prestazioni quasi *real-time*, seguito da TETer e dalla versione più veloce di OvTrack, che registrano prestazioni simili. Considerando esclusivamente la componente di *tracking* BEHEMOTH è oltre 5 volte più veloce di AED, circa il 25% più veloce di TETer ed il 17% più rapido rispetto a OvTrack.
La differenza prestazionale complessiva tra i sistemi più veloci rimane comunque contenuta, con uno scarto massimo inferiore ai 2 frame al secondo. Questo è attribuibile sia al fatto che i tempi assoluti di inferenza sono già molto ridotti, sia al *bottleneck* rappresentato dalla *backbone* di *object detection*, che rimane il principale fattore limitante nella velocità complessiva della *pipeline*.

4.6.3 Considerazioni

BEHEMOTH si conferma un *tracker* molto rapido, e, nei contesti *open vocabulary*, riesce a raggiungere prestazioni simili allo stato dell'arte. Tuttavia, presenta un'accuratezza ridotta rispetto ai sistemi esistenti, ed una dipendenza troppo stretta dalla *backbone di object detection* adottata.
È difficile stabilire se la perdita in accuratezza sia giustificabile rispetto ai guadagni in

efficienza. Attualmente, infatti, nessun *tracker open vocabulary* esistente offre prestazioni sufficientemente affidabili da consentirne l'impiego in contesti applicativi critici, come il settore *automotive*, la sicurezza o il controllo di qualità in ambito industriale.
Infine, è opportuno considerare le differenze nei contesti di sviluppo dei modelli confrontati. BEHEMOTH è stato addestrato su un dataset composto da soli 500 video, con circa 3 annotazioni per frame, e *batch* di dimensioni contenute. Tali scelte sono state dettate dai limiti *hardware* dell'ambiente di sviluppo. Al contrario, gli altri sistemi sono stati addestrati su *ensemble* di più dataset, talvolta proprietari o progettati appositamente. Hanno inoltre beneficiato di maggiori risorse computazionali, che hanno consentito l'utilizzo di *batch* più grandi, con una conseguente riduzione dei tempi di addestramento e una migliore convergenza dei modelli addestrati.
Alla luce di queste considerazioni, è ragionevole ipotizzare che BEHEMOTH non abbia raggiunto il pieno delle proprie potenzialità. Una revisione della *pipeline* di addestramento, accompagnata da un'estensione del *training set*, potrebbe portare a un miglioramento sostanziale delle sue prestazioni. Nonostante le problematiche elencate, il fatto che BEHEMOTH riesca a superare la *baseline* prefissata dimostra l'applicabilità di tecniche di quantizzazione, come il *locality-sensitive hashing*, a scenari di *multi-object tracking*. Questo costituisce uno dei principali risultati ottenuti attraverso il lavoro qui presentato ed evidenzia uno scenario promettente per sviluppi futuri.

4.7 Risultati qualitativi

BEHEMOTH è stato testato su alcune sequenze video in modo da effettuare valutazioni qualitative. Le figure 4.2, 4.3, 4.4, 4.5 e 4.6 mostrano alcuni dei frame più significativi tra quelli ottenuti, campionati con una frequenza di 1 FPS. Le clip scelte afferiscono a contesti eterogenei, in modo da verificare le prestazioni del *tracker* in diversi ambiti. Per ogni clip sono indicate le classi tracciate, che sono state specificate attraverso una *query* testuale. I nomi sottolineati si riferiscono a classi assenti dai *training set* dei dataset utilizzati per l'addestramento della *backbone* e di BEHEMOTH, ma presenti nel rispettivo *split* di *validation* o in quello di *test*. I nomi in **grassetto** indicano classi assenti da qualsisi *split* dei dataset utilizzati. La *backbone* utilizzata per i seguenti risultati è stata OmDetTurbo-Tiny, ad eccezione della clip in figura 4.6, per cui è stata utilizzata RegionCLIP.

Classi: “person”, “motorbike”

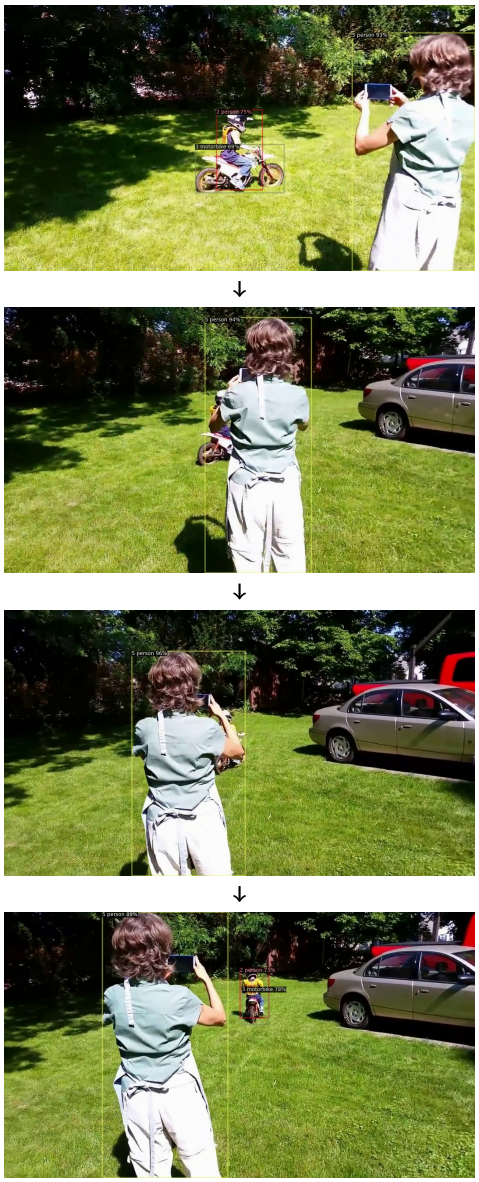


Figura 4.2: Quando ha a che fare con pochi soggetti, BEHEMOTH riesce a resistere ad occlusioni temporanee e a mantenere stabili le identità.

Classi: “person”, “car”

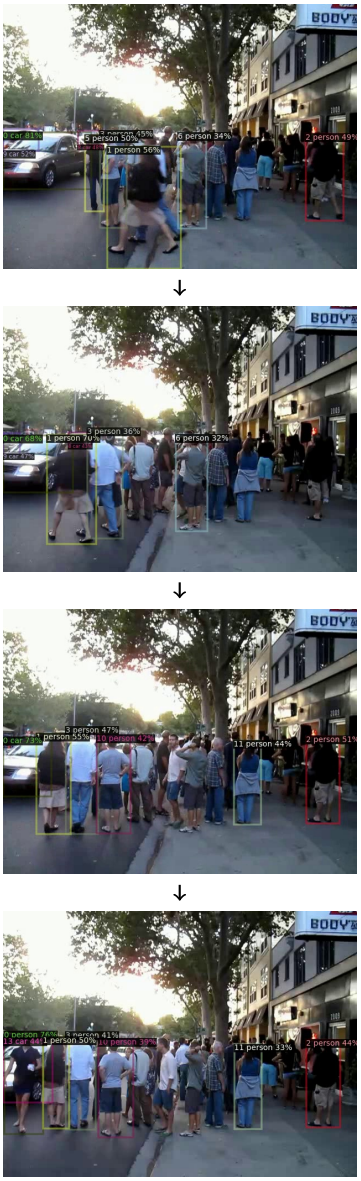


Figura 4.3: In scenari complessi, con un elevato numero di soggetti, BEHEMOTH fatica a mantenere stabili le identità. L’identità **0**, ad esempio, viene prima assegnata ad un automobile e poi ad una persona.

Classi: “**bobcat**”, “heron”

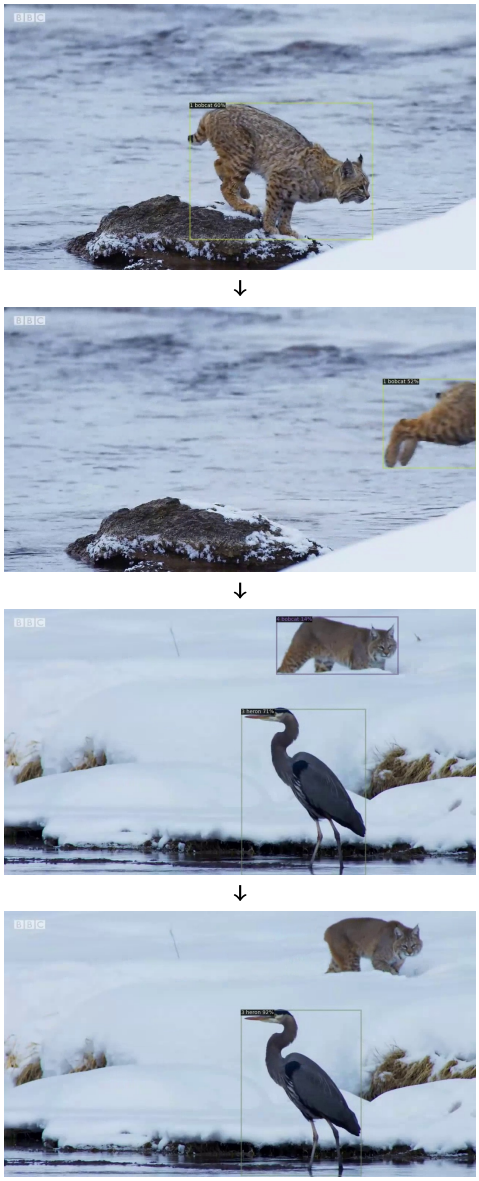


Figura 4.4: In questa clip BEHEMOTH traccia correttamente i soggetti, ma il cambio di inquadratura rende difficile la reidentificazione dello stesso soggetto.

Classi: “polar bear”, “walrus”

Figura 4.5: BEHEMOTH riesce a gestire correttamente anche classi assenti dal *training set*, come “walrus” (“tricheco”).

Classi: “**pikachu**”, “person”

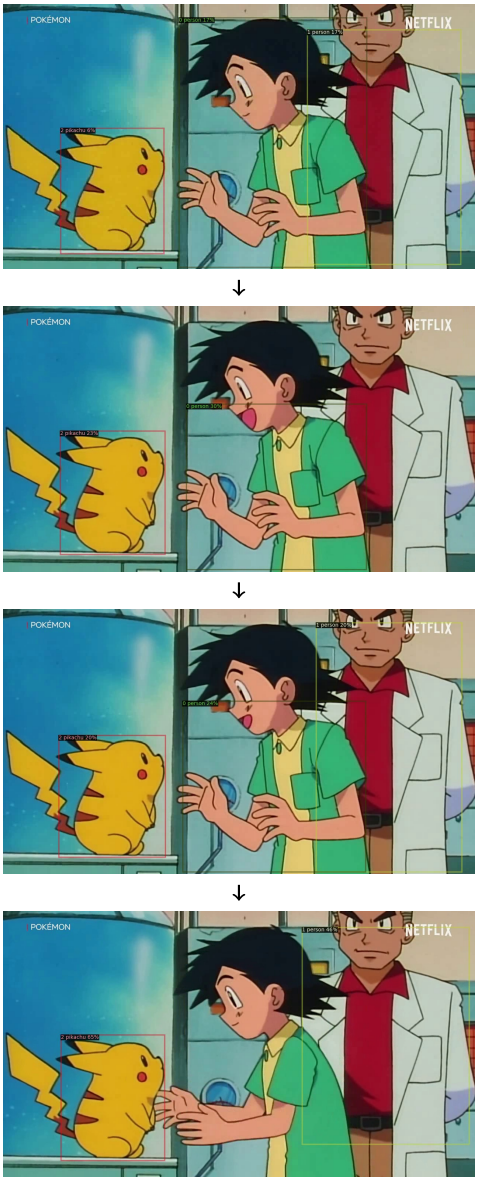


Figura 4.6: Anche se semplice dal punto di vista del *tracking*, questa clip mostra le potenzialità dei sistemi *open-vocabulary*, con cui si riescono a trattare anche soggetti notevolmente differenti rispetto a quelli osservati durante l’addestramento.

Capitolo 5

Studi di ablazione

Il presente capitolo riporta gli studi di ablazione compiuti per verificare l'impatto dei parametri di *tracking* e l'efficacia di alcune scelte implementative. Al contrario dei risultati trattati nel capitolo 4, quelli qui presentati sono stati ottenuti a partire da una versione ridotta del *validation set* composta da 100 video, ovvero circa il 10% dei video presenti. Inoltre i risultati riportati fanno riferimento all'architettura ottenuta in seguito ad *hyperparameter tuning* (sezione 4.5).

5.1 Metrica di distanza

Dim. Embedding / Encoding	Distanza	TETA	AssocA	ClsA	LocA	Tempo medio CPU	Speedup Hamming
256	Coseno	31.28	25.927	21.388	46.525	249 μs	$\sim 3.4\times$
	Hamming	32.11	28.123	20.984	47.224	73 μs	
512	Coseno	31.673	26.663	20.304	**48.053**	220 μs	$\sim 2.8\times$
	Hamming	32.309	28.445	21.319	47.163	78 μs	
1024	Coseno	31.472	26.079	20.698	47.639	327 μs	$\sim 5.6\times$
	Hamming	**32.617**	**29.633**	21.155	47.063	**58** μs	
2048	Coseno	31.339	26.15	**21.485**	46.381	1500 μs	$\sim$ **15.6**$\times$
	Hamming	32.102	28.401	20.583	47.323	96 μs	

Tabella 5.1: Confronto tra distanza di Hamming e Distanza coseno come misura di similarità tra *encoding/embedding*. La voce **Tempo Medio CPU** si riferisce al tempo medio necessario al calcolo della matrice di costo durante la fase di associazione.

È stato condotto uno studio specifico per verificare l'impatto derivante dall'utilizzo di *encoding* binari al posto di *embedding* reali. Il *tracker* progettato, basato su *locality-sensitive hashing*, è stato confrontato con una versione duale, ottenuta rimuovendo l'*hashing net* (sezione 3.2.3) e seguendo la stessa procedura di addestramento descritta nella sezione 3.3. Questa implementazione alternativa impiega i *region embedding*, ottenuti dall'*embedding net* (sezione 3.2.2), come rappresentazioni vettoriali di ciascuna regione. Durante la fase di associazione, la funzione $c_{visual}(\tau_{t-1}^j, p_t^i)$ nell'equazione 3.5, è sostituita da:

$$c_{visual}(\tau_{t-1}^j, p_t^i) = \frac{1 - CosineSimilarity\left(r_{\tau_{t-1}^j}, r_{p_t^i}\right)}{2} \tag{5.1}$$

anche indicata come distanza coseno o *cosine distance*.

I risultati sono mostrati nella tabella 5.1. È evidente il guadagno in efficienza derivante dall'utilizzo della distanza di Hamming rispetto alla distanza coseno, con *speedup* tanto più significativi all'aumentare della dimensione degli *embedding*. Da un confronto tra il tempo medio impiegato dall'algoritmo di *tracking* di BEHEMOTH (tabella 4.12), e quello necessaro al calcolo della metrica di costo, emerge che il *bottleneck* del sistema sia da ricercare altrove. Tuttavia è possibile che su hardware meno performanti lo *speedup* ottenuto dall'impiego della distanza di Hamming possa rappresentare un fattore cruciale al fine di permettere il raggiungimento di prestazioni *real-time*.
Analizzando le metriche TETA emerge un risultato particolarmente significativo. Nonostante i *tracker* basati su distanza coseno possano avvalersi di un maggiore potere rappresentativo, in quanto utilizzano *embedding vettoriali* definiti all'interno di un spazio reale, questi ottengono consistentemente punteggi di AssocA e TETA inferiori alla propria controparte binaria. La causa del fenomeno è attribuibile proprio alla minore capacità rappresentativa degli *encoding binari*. L'impiego di *region encoding* limita la quantità di informazioni codificabili, riducendo il rumore introdotto da dettagli visivi meno rilevanti, ed aumentando la stabilità delle rappresentazioni associate ad uno stesso soggetto all'interno di un flusso video. Rappresentazioni meno dettagliate riducono inoltre l'*overfitting* sui campioni osservati durante l'addestramento, migliorando le capacità di generalizzazione del *tracker*. Le stesse motivazioni giustificano la riduzione di prestazioni ottenuta con *encoding* a 2048 bit rispetto a quelle registrate con *encoding* a 1024 bit.
Infine, le prestazioni ottenute con *encoding* a 1024 bit motivano la scelta di questa dimensione per l'output dell'*hashing net* di BEHEMOTH.

5.2 Parametri di associazione

La fase di associazione descritta in 3.1.3 è regolata da alcuni parametri, il cui valore ha un impatto notevole sulla qualità delle traittorie individuate. Sono stati condotti alcuni studi per regolare in modo ottimale questi parametri, descritti di seguito.

5.2.1 Costo visivo e spaziale

α	TETA	AssocA	ClsA	LocA
0.2	31.217	25.745	**21.458**	46.448
0.4	<u>32.223</u>	<u>28.493</u>	<u>21.200</u>	46.975
0.6	**32.617**	**29.633**	21.155	<u>47.063</u>
0.8	30.244	23.89	19.671	**47.17**

Tabella 5.2: Confronto tra le prestazioni corrispondenti a diversi valori di α.

Nella somma pesata tra costo visivo c_{visual} e spaziale $c_{spatial}$ (eq. 3.5), il parametro α regola l'importanza delle due metriche, e risulta fondamentale per massimizzare le prestazioni del *tracker*. La tabella 5.2 mostra l'impatto di α sull'accuratezza del sistema, a parità degli altri parametri. Si osserva che valori di α sbilanciati in favore dell'una o dell'altra metrica diminuiscono significativamente i punteggi di TETA ed AssocA, ciò dimostra che entrambe le misure di costo giocano un ruolo fondamentale al fine di garantire la massima accuratezza durante il *tracking*.

5.2.2 Soglia di assegnamento

$t_{assignment}$	**TETA**	**AssocA**	**ClsA**	**LocA**
0.2	29.853	22.066	21.343	46.150
0.3	31.065	25.342	21.479	46.376
0.4	32.227	28.116	21.531	47.034
0.5	**32.617**	**29.633**	21.155	47.063
0.6	31.229	25.27	20.584	47.832
0.7	30.122	22.171	**21.618**	**47.935**

Tabella 5.3: Confronto tra le prestazioni corrispondenti a diversi valori di $t_{assignment}$.

La soglia $t_{assignment}$ determina il costo massimo di un assegnamento affinchè esso sia ritenuto accettabile. Valori di $t_{assignment}$ più bassi rendono il *tracker* più rigido e meno robusto rispetto agli errori della *tracking head*. Una soglia troppo elevata, invece, obbliga la rete ad accettare assegnamenti dal costo elevato, provocando frequenti cambi di identità ed impedendo la creazione di nuove traiettorie. Per determinare il valore ottimo di $t_{assignment}$ è stato condotto uno studio specifico, i cui risultati sono mostrati in tabella 5.3. Il valore $t_{assignment} = 0.5$ massimizza le prestazioni di *tracking* ed il punteggio TETA, ed è quindi stato scelto come valore di soglia.

Capitolo 6

Conclusioni

6.1 Osservazioni

Il sistema presentato in questo elaborato integra funzionalità di *detection* e *tracking open vocabulary*, affermandosi come una soluzione efficiente grazie all'integrazione di *locality-sensitive hashing* e all'utilizzo di *backbone* particolarmente leggere. L'esperimento 4.6 dimostra il guadagno in efficienza di BEHEMOTH rispetto ad altri sistemi esistenti, sebbene sull'*hardware* utilizzato la differenza in termini di tempi assoluti rispetto alle soluzioni più rapide risulti trascurabile. I risultati ottenuti confermano le potenzialità delle funzioni LSH come tecniche di *dimensionality reduction* e quantizzazione. In scenari *open vocabulary*, infatti, tali tecniche permettono di raggiungere prestazioni comparabili allo stato dell'arte, dimostrando come approcci alternativi alla classica *cosine similarity*, quali quelli basati su distanza di Hamming, possano rappresentare una direzione promettente per sistemi di *tracking* più leggeri e scalabili. Le potenzialità di questi scenari sono evidenziate anche dalle conclusioni presentate in 5.1. Se da un lato ciò valorizza l'efficacia delle tecniche LSH, che potrebbero risultare adatte anche in altri paradigmi di *tracking real-time*, dall'altro evidenzia i limiti intrinseci delle tecnologie *open vocabulary* attualmente disponibili. Il fatto che BEHEMOTH raggiunga, sotto opportune condizioni, le prestazioni di altri *tracker open-vocabulary*, dotati di algoritmi di *tracking* più complessi, e le prestazioni mediamente insoddisfacenti di ogni *tracker* analizzato, suggeriscono che vi sia ancora ampio margine di miglioramento in questo ambito. Pertanto, risulta difficile giustificare l'utilizzo di una qualsiasi di queste tecnologie in contesti applicativi reali.
Inoltre l'assenza di un dataset video densamente annotato e pubblicamente disponibile, che offra un *training set* sufficientemente variegato e corposo, ostacola l'avanzamento della ricerca in questo ambito. Si tratta, infine, di una tecnologia ancora computazionalmente troppo onerosa per potersi adattare a sistemi *embedded*, nei quali potrebbero emergere in maniera più evidente i benefici derivanti dall'utilizzo di tecniche di quantizzazione binaria. L'insieme di questi fattori evidenzia la necessità di uno sforzo congiunto da parte del mondo scientifico, sia dal punto di vista tecnologico che infrastrutturale, per promuovere un'evoluzione significativa nel campo del *tracking open-vocabulary*.

6.2 Limiti

Uno dei limiti più evidenti del sistema, emerso nell'esperimento descritto in 4.4, consiste nella forte dipendenza dalla *backbone* di *object detection* utilizzata. In particolare, il *tracker* sviluppato riesce a superare la *baseline* solo con una delle tre *backbone* adottate,

risultando sensibile sia al numero che alla dimensione delle regioni individuate. Questo comportamento è attribuibile principalmente al *training set* di TAO, caratterizzato da varianza e densità di annotazione ridotte, che limitano la capacità discriminativa del sistema in ambienti affollati e dinamici.

Un altro aspetto critico è costituito dall'accuratezza subottimale di BEHEMOTH, come mostrato nell'esperimento 4.6. L'utilizzo di tecniche di quantizzazione e di reti più leggere, ma con minore potere espressivo - sia nella *backbone* che nell'*embedding net* del sistema - può giustificare una riduzione delle capacità di *tracking*. Tuttavia il grado di tolleranza sulla diminuzione dell'accuratezza risulta difficile da definire in modo oggettivo, data la mediocrità generale delle soluzioni esistenti. Rimane comunque un obiettivo rilevante cercare di ridurre lo scarto in alcune metriche critiche, come AssocA, considerando anche le potenzialità mostrate da OmDetTiny-Turbo + BEHEMOTH sull'insieme **Novel** di *validation* e *test set*.

Un'ultima criticità, evidenziata dall'esperimento 4.2, riguarda la scarsa influenza che *embedding net* diverse hanno sull'output del sistema. Le cause di questo fenomeno non sono facilmente identificabili, ma tra le ipotesi più plausibili vi sono un'inefficacia parziale della *pipeline* di addestramento, la bassa capacità espressiva delle reti utilizzate ed il già citato problema di ridotta varianza del *dataset*.

6.3 Futuri sviluppi

Si illustrano qui alcuni aspetti di rilievo su cui potranno concentrarsi ricerche future, sia per affrontare le problematiche esposte nella sezione 6.2, sia per estendere le potenzialità delle tecnologie sviluppate.
Come primo passo, riaddestrare il modello su un *dataset* appositamente progettato, con annotazioni ad elevata densità e maggiore varianza nei soggetti, potrebbe aumentare significativamente la capacità discriminativa del sistema, nonché accentuare le differenze prestazionali tra diverse *embedding net*. Tuttavia, la costruzione di un simile *dataset* rappresenta un compito oneroso, che richiede una mole di risorse considerevole. Una soluzione più accessibile potrebbe consistere nell'utilizzo di *ensemble* di *dataset* video specifici per ambiti diversi, o nell'impiego di tecniche di *data augmentation* e *hallucination*, la cui efficacia in ambito *open vocabulary* è stata dimostrata in [19], per produrre un'elevata quantità di campioni sintetici. Il sistema potrebbe inoltre beneficiare di una revisione della *pipeline* di addestramento e della sperimentazione con diverse funzioni di loss, come la *contrastive loss* proposta in [38].

Un ulteriore sviluppo potrebbe derivare dal *fine-tuning* congiunto della *backbone* di *object-detection* e del *tracker*. Ciò permetterebbe non solo un miglioramento delle prestazioni di localizzazione e classificazione, ma anche la possibilità di riutilizzare direttamente i *region embedding* prodotti dalla *backbone*, con conseguente eliminazione del modulo di *embedding net*, aumentando l'efficienza del sistema.

Sebbene le tecnologie *open vocabulary* risultino ancora poco efficienti e accurate per applicazioni in ambienti *embedded*, studi futuri potrebbero esplorare i vantaggi computazionali dell'utilizzo di tecniche LSH in tali sistemi. In questi contesti, la differenza nei tempi di inferenza tra BEHEMOTH e soluzioni alternative, che su GPU risulta minima (sezione

4.12), potrebbe rivelarsi molto più significativa. Infine un *tracker* basato su *region encoding* binari, come quello sviluppato, potrebbe avere particolare rilevanza anche in contesti *closed vocabulary*. Parte della ricerca futura potrebbe quindi concentrarsi sull'eliminazione della componente *open vocabulary*, per approfondire l'applicabilità delle tecniche LSH nel *multi-object tracking* classico, offrendo un'alternativa computazionalmente vantaggiosa ai metodi tradizionali.

Considerazioni finali

Il lavoro svolto in questa tesi ha portato alla progettazione, implementazione e valutazione di un sistema completo per *open vocabulary multi-object tracking*, ottimizzato per l'impiego di tecniche di *locality-sensitive hashing*. Il sistema, denominato BEHEMOTH, si distingue per l'introduzione di una *pipeline* end-to-end che combina efficienza computazionale e flessibilità semantica, consentendogli di operare in modo efficiente in contesti *open vocabulary*. L'approccio proposto mostra come l'integrazione di tecniche di quantizzazione binaria, tradizionalmente associate a compiti di *retrieval*, possa contribuire alla realizzazione di sistemi di *tracking* leggeri. Il lavoro evidenzia inoltre i limiti attuali dell'approccio *open vocabulary*, fornendo spunti per futuri miglioramenti. Si auspica che i risultati ottenuti, insieme alle criticità emerse, possano rappresentare un punto di partenza solido per lo sviluppo di sistemi di OV-MOT più efficienti, robusti e adattabili alle esigenze dei contesti applicativi reali.

Bibliografia

[1] David Held, Sebastian Thrun e Silvio Savarese. «Learning to Track at 100 FPS with Deep Regression Networks». In: *European Conference Computer Vision (ECCV)*. 2016 (cit. a p. 3).

[2] Luca Bertinetto, Jack Valmadre, João F. Henriques, Andrea Vedaldi e Philip H. S. Torr. «Fully-Convolutional Siamese Networks for Object Tracking». In: *Computer Vision – ECCV 2016 Workshops*. A cura di Gang Hua e Hervé Jégou. Cham: Springer International Publishing, 2016, pp. 850–865. ISBN: 978-3-319-48881-3 (cit. a p. 3).

[3] Bo Li, Junjie Yan, Wei Wu, Zheng Zhu e Xiaolin Hu. «High Performance Visual Tracking With Siamese Region Proposal Network». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Giu. 2018 (cit. a p. 3).

[4] Bo Li, Wei Wu, Qiang Wang, Fangyi Zhang, Junliang Xing e Junjie Yan. «SiamRPN++: Evolution of Siamese Visual Tracking with Very Deep Networks». In: *arXiv preprint arXiv:1812.11703* (2018) (cit. a p. 4).

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren e Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385 (cit. alle pp. 4, 6, 17, 33).

[6] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov e Liang-Chieh Chen. «MobileNetV2: Inverted Residuals and Linear Bottlenecks». In: giu. 2018, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474 (cit. a p. 4).

[7] Cheng-Yen Yang, Hsiang-Wei Huang, Wenhao Chai, Zhongyu Jiang e Jenq-Neng Hwang. *SAMURAI: Adapting Segment Anything Model for Zero-Shot Visual Tracking with Motion-Aware Memory*. 2024. arXiv: 2411.11922 [cs.CV]. URL: https://arxiv.org/abs/2411.11922 (cit. a p. 4).

[8] Alexander Kirillov et al. *Segment Anything*. 2023. arXiv: 2304.02643 [cs.CV]. URL: https://arxiv.org/abs/2304.02643 (cit. a p. 4).

[9] Nikhila Ravi et al. *SAM 2: Segment Anything in Images and Videos*. 2024. arXiv: 2408.00714 [cs.CV]. URL: https://arxiv.org/abs/2408.00714 (cit. a p. 4).

[10] R. E. Kalman. «A New Approach to Linear Filtering and Prediction Problems». In: *Journal of Basic Engineering* 82.1 (mar. 1960), pp. 35–45. ISSN: 0021-9223. DOI: 10.1115/1.3662552. eprint: https://asmedigitalcollection.asme.org/fluidsengineering/article-pdf/82/1/35/5518977/35_1.pdf. URL: https://doi.org/10.1115/1.3662552 (cit. a p. 4).

[11] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos e Ben Upcroft. «Simple online and realtime tracking». In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 3464–3468. DOI: 10.1109/ICIP.2016.7533003 (cit. alle pp. 4, 24).

[12] Shaoqing Ren, Kaiming He, Ross B. Girshick e Jian Sun. «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.» In: *NIPS*. A cura di Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama e Roman Garnett. 2015, pp. 91–99. URL: http://dblp.uni-trier.de/db/conf/nips/nips2015.html#RenHGS15 (cit. a p. 4).

[13] H. W. Kuhn. «The Hungarian method for the assignment problem». In: *Naval Research Logistics Quarterly* 2.1-2 (1955), pp. 83–97. DOI: https://doi.org/10.1002/nav.3800020109. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800020109. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800020109 (cit. alle pp. 4, 24).

[14] Nicolai Wojke, Alex Bewley e Dietrich Paulus. «Simple online and realtime tracking with a deep association metric». In: *2017 IEEE International Conference on Image Processing (ICIP)*. 2017, pp. 3645–3649. DOI: 10.1109/ICIP.2017.8296962 (cit. a p. 4).

[15] Peng Dai, Xue Wang, Weihang Zhang e Junfeng Chen. «Instance Segmentation Enabled Hybrid Data Association and Discriminative Hashing for Online Multi-Object Tracking». In: *IEEE Transactions on Multimedia* 21.7 (2019), pp. 1709–1723. DOI: 10.1109/TMM.2018.2885922 (cit. a p. 5).

[16] Jiangmiao Pang, Linlu Qiu, Xia Li, Haofeng Chen, Qi Li, Trevor Darrell e Fisher Yu. «Quasi-Dense Similarity Learning for Multiple Object Tracking». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Giu. 2021, pp. 164–173 (cit. a p. 5).

[17] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Fucheng Weng, Zehuan Yuan, Ping Luo, Wenyu Liu e Xinggang Wang. *ByteTrack: Multi-Object Tracking by Associating Every Detection Box*. 2022. arXiv: 2110.06864 [cs.CV]. URL: https://arxiv.org/abs/2110.06864 (cit. alle pp. 5, 24, 31, 46).

[18] Siyuan Li, Martin Danelljan, Henghui Ding, Thomas E. Huang e Fisher Yu. «Tracking Every Thing in the Wild». In: *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXII*. Tel Aviv, Israel: Springer-Verlag, 2022, pp. 498–515. ISBN: 978-3-031-20046-5 (cit. alle pp. 5, 8, 43, 46).

[19] Siyuan Li, Tobias Fischer, Lei Ke, Henghui Ding, Martin Danelljan e Fisher Yu. «OVTrack: Open-Vocabulary Multiple Object Tracking». In: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2023, pp. 5567–5577 (cit. alle pp. 5, 7, 8, 43, 46, 53, 63).

[20] Alec Radford et al. «Learning Transferable Visual Models From Natural Language Supervision». In: *Proceedings of the 38th International Conference on Machine Learning*. A cura di Marina Meila e Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, lug. 2021, pp. 8748–8763. URL: https://proceedings.mlr.press/v139/radford21a.html (cit. alle pp. 6, 20, 33).

[21] Chao Jia et al. *Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision*. 2021. arXiv: 2102.05918 [cs.CV]. URL: https://arxiv.org/abs/2102.05918 (cit. a p. 6).

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser e Illia Polosukhin. «Attention is all you need». In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964 (cit. alle pp. 6, 16).

[23] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV]. URL: https://arxiv.org/abs/2010.11929 (cit. alle pp. 6, 18).

[24] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin e Baining Guo. «Swin Transformer: Hierarchical Vision Transformer using Shifted Windows». In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 9992–10002. DOI: 10.1109/ICCV48922.2021.00986 (cit. alle pp. 6, 33).

[25] Yiwu Zhong et al. «RegionCLIP: Region-Based Language-Image Pretraining». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Giu. 2022, pp. 16793–16803 (cit. alle pp. 6, 33).

[26] Matthias Minderer et al. «Simple Open-Vocabulary Object Detection with Vision Transformers». In: *ECCV* (2022) (cit. a p. 6).

[27] Liunian Harold Li et al. *Grounded Language-Image Pre-training*. 2022. arXiv: 2112.03857 [cs.CV]. URL: https://arxiv.org/abs/2112.03857 (cit. a p. 6).

[28] Shilong Liu et al. «Grounding dino: Marrying dino with grounded pre-training for open-set object detection». In: *arXiv preprint arXiv:2303.05499* (2023) (cit. a p. 7).

[29] Tianhe Ren et al. *Grounding DINO 1.5: Advance the "Edge" of Open-Set Object Detection*. 2024. arXiv: 2405.10300 [cs.CV] (cit. a p. 7).

[30] Tiancheng Zhao, Peng Liu e Kyusong Lee. «OmDet: Large-scale vision-language multi-dataset pre-training with multimodal detection network». In: *IET Computer Vision* 18.5 (gen. 2024), pp. 626–639. ISSN: 1751-9640. DOI: 10.1049/cvi2.12268. URL: http://dx.doi.org/10.1049/cvi2.12268 (cit. alle pp. 7, 33, 47).

[31] Tiancheng Zhao, Peng Liu, Xuan He, Lu Zhang e Kyusong Lee. *Real-time Transformer-based Open-Vocabulary Detection with Efficient Fusion Head*. 2024. arXiv: 2403.06892 [cs.CV]. URL: https://arxiv.org/abs/2403.06892 (cit. a p. 7).

[32] Tri Dao. *FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning*. 2023. arXiv: 2307.08691 [cs.LG]. URL: https://arxiv.org/abs/2307.08691 (cit. a p. 7).

[33] Tianheng Cheng, Lin Song, Yixiao Ge, Wenyu Liu, Xinggang Wang e Ying Shan. «YOLO-World: Real-Time Open-Vocabulary Object Detection». In: *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2024), pp. 16901–16911. URL: https://api.semanticscholar.org/CorpusID:267320681 (cit. alle pp. 7, 33).

[34] Joseph Redmon, Santosh Divvala, Ross Girshick e Ali Farhadi. «You Only Look Once: Unified, Real-Time Object Detection». In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91 (cit. alle pp. 7, 33).

[35] Joseph Redmon e Ali Farhadi. «YOLO9000: Better, Faster, Stronger». In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 6517–6525. URL: https://api.semanticscholar.org/CorpusID:786357 (cit. a p. 7).

[36] Joseph Redmon e Ali Farhadi. «YOLOv3: An Incremental Improvement». In: (apr. 2018). DOI: 10.48550/arXiv.1804.02767 (cit. a p. 7).

[37] Glenn Jocher, Jing Qiu e Ayush Chaurasia. *Ultralytics YOLO*. Ver. 8.0.0. Gen. 2023. URL: https://github.com/ultralytics/ultralytics (cit. a p. 7).

[38] Zimeng Fang, Chao Liang, Xue Zhou, Shuyuan Zhu e Xi Li. *Associate Everything Detected: Facilitating Tracking-by-Detection to the Unknown*. 2024. arXiv: 2409.09293 [cs.CV]. URL: https://arxiv.org/abs/2409.09293 (cit. alle pp. 7, 8, 24, 43, 46, 53, 63).

[39] Junfeng Wu, Yi Jiang, Qihao Liu, Zehuan Yuan, Xiang Bai e Song Bai. *General Object Foundation Model for Images and Videos at Scale*. 2023. arXiv: 2312.09158 (cit. a p. 8).

[40] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang e Jifeng Dai. *Deformable DETR: Deformable Transformers for End-to-End Object Detection*. 2021. arXiv: 2010.04159 [cs.CV]. URL: https://arxiv.org/abs/2010.04159 (cit. a p. 8).

[41] Moses Charikar. «Similarity estimation techniques from rounding algorithms». In: *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*. A cura di John H. Reif. ACM, 2002, pp. 380–388. DOI: 10.1145/509907.509965. URL: https://doi.org/10.1145/509907.509965 (cit. alle pp. 9, 23).

[42] Kevin Lin, Huei-Fang Yang, Jen-Hao Hsiao e Chu-Song Chen. *Deep learning of binary hash codes for fast image retrieval*. Giu. 2015. DOI: 10.1109/CVPRW.2015.7301269 (cit. a p. 9).

[43] Zhangjie Cao, Mingsheng Long, Jianmin Wang e Philip S. Yu. *HashNet: Deep Learning to Hash by Continuation*. 2017. arXiv: 1702.00758 [cs.LG]. URL: https://arxiv.org/abs/1702.00758 (cit. a p. 9).

[44] Nikita Kitaev, Lukasz Kaiser e Anselm Levskaya. «Reformer: The Efficient Transformer». In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=rkgNKkHtvB (cit. a p. 9).

[45] Lirong Han, Peng Li, Antonio Plaza e Peng Ren. «Hashing for localization (HfL): A baseline for fast localizing objects in a large-scale scene». In: *IEEE Transactions on Geoscience and Remote Sensing* 60 (2021), pp. 1–16 (cit. a p. 9).

[46] Youmeng Luo, Wei Li, Xiaoyu Ma e Kaiqiang Zhang. «Image Retrieval Algorithm Based on Locality-Sensitive Hash Using Convolutional Neural Network and Attention Mechanism». In: *Information* 13.10 (2022). ISSN: 2078-2489. DOI: 10.3390/info13100446. URL: https://www.mdpi.com/2078-2489/13/10/446 (cit. a p. 9).

[47] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV]. URL: https://arxiv.org/abs/1405.0312 (cit. a p. 10).

[48] Agrim Gupta, Piotr Dollár e Ross Girshick. *LVIS: A Dataset for Large Vocabulary Instance Segmentation*. 2019. arXiv: 1908.03195 [cs.CV]. URL: https://arxiv.org/abs/1908.03195 (cit. a p. 10).

[49] Achal Dave, Tarasha Khurana, Pavel Tokmakov, Cordelia Schmid e Deva Ramanan. *TAO: A Large-Scale Benchmark for Tracking Any Object*. 2020. arXiv: 2005.10356 [cs.CV]. URL: https://arxiv.org/abs/2005.10356 (cit. alle pp. 10, 34, 36).

[50] Ali Athar, Jonathon Luiten, Paul Voigtlaender, Tarasha Khurana, Achal Dave, Bastian Leibe e Deva Ramanan. «BURST: A Benchmark for Unifying Object Recognition, Segmentation and Tracking in Video». In: *WACV*. 2023 (cit. alle pp. 10, 37).

[51] Patrick Dendorfer, Aljoša Ošep, Anton Milan, Konrad Schindler, Daniel Cremers, Ian Reid, Stefan Roth e Laura Leal-Taixé. *MOTChallenge: A Benchmark for Single-Camera Multiple Target Tracking*. 2020. arXiv: 2010.07548 [cs.CV]. URL: https://arxiv.org/abs/2010.07548 (cit. alle pp. 10, 46).

[52] Andreas Geiger, Philip Lenz e Raquel Urtasun. «Are we ready for autonomous driving? The KITTI vision benchmark suite». In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3354–3361. DOI: 10.1109/CVPR.2012.6248074 (cit. a p. 10).

[53] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan e Trevor Darrell. «BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning». In: giu. 2020, pp. 2633–2642. DOI: 10.1109/CVPR42600.2020.00271 (cit. a p. 10).

[54] Alex Krizhevsky, Ilya Sutskever e Geoffrey E Hinton. «ImageNet Classification with Deep Convolutional Neural Networks». In: *Advances in Neural Information Processing Systems*. A cura di F. Pereira, C.J. Burges, L. Bottou e K.Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (cit. a p. 11).

[55] *Wikipedia, the free encyclopedia — en.wikipedia.org*. https://en.wikipedia.org/wiki/Main_Page. [Accessed 13-07-2025] (cit. a p. 20).

[56] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei e Ilya Sutskever. «Language Models are Unsupervised Multitask Learners». In: *OpenAI* (2019). Accessed: 2024-11-15. URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf (cit. a p. 21).

[57] Michel X. Goemans e David P. Williamson. «Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming». In: *J. ACM* 42.6 (nov. 1995), pp. 1115–1145. ISSN: 0004-5411. DOI: 10.1145/227683.227684. URL: https://doi.org/10.1145/227683.227684 (cit. a p. 23).

[58] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan e Serge Belongie. *Feature Pyramid Networks for Object Detection*. 2017. arXiv: 1612.03144 [cs.CV]. URL: https://arxiv.org/abs/1612.03144 (cit. alle pp. 33, 49).

[59] Mingxing Tan e Quoc Le. «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks». In: *Proceedings of the 36th International Conference on Machine Learning*. A cura di Kamalika Chaudhuri e Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. PMLR, giu. 2019, pp. 6105–6114. URL: https://proceedings.mlr.press/v97/tan19a.html (cit. a p. 35).

[60] Andrew Howard et al. «Searching for MobileNetV3.» In: *ICCV*. IEEE, 2019, pp. 1314–1324. ISBN: 978-1-7281-4803-8. URL: http://dblp.uni-trier.de/db/conf/iccv/iccv2019.html#HowardPALSCWCTC19 (cit. a p. 35).

[61] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li e Li Fei-Fei. «Imagenet: A large-scale hierarchical image database». In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255 (cit. a p. 37).

[62] Chun-Liang Li, Kihyuk Sohn, Jinsung Yoon e Tomas Pfister. «CutPaste: Self-Supervised Learning for Anomaly Detection and Localization». In: *CoRR* abs/2104.04015 (2021). arXiv: 2104.04015. URL: https://arxiv.org/abs/2104.04015 (cit. a p. 38).

[63] Fei Du, Bo Xu, Jiasheng Tang, Yuqi Zhang, Fan Wang e Hao Li. «1st Place Solution to ECCV-TAO-2020: Detect and Represent Any Object for Tracking». In: *arXiv preprint arXiv: 2101.08040* (2021). URL: https://arxiv.org/abs/2101.08040 (cit. a p. 46).

[64] Papers With Code. *Multi-Object Tracking on TAO*. https://paperswithcode.com/sota/multi-object-tracking-on-tao. Accessed: 2025-05-12. 2025 (cit. a p. 46).

[65] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta e Masanori Koyama. *Optuna: A Next-generation Hyperparameter Optimization Framework*. 2019. arXiv: 1907.10902 [cs.LG]. URL: https://arxiv.org/abs/1907.10902 (cit. a p. 51).