UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Triennale

Il *labelling*: i migliori *tool* disponibili e la loro applicazione per il rilevamento di persone e cartelli stradali e per la *pose estimation* di oggetti 3D

Relatore
prof. Marko Bertogna

Laureando
Tobia Poppi

Ottobre 2021

Sommario

Lo sviluppo e l'utilizzo sempre crescente dell'intelligenza artificiale e del machine learning da parte di smartphone, platform e altre applicazioni d'uso quotidiano richiedono un'immensa quantità di dati contenenti le informazioni necessarie agli algoritmi per poter imparare a svolgere operazioni in autonomia.

Questo elaborato descrive un lavoro di ricerca sulle tecniche utilizzate per etichettare dati e immagini al fine di poterli utilizzare per l'addestramento di modelli di machine learning. In seguito vengono illustrati gli sviluppi di due progetti per casi d'uso reali.

Indice

Capitolo 1

Il *labelling*

1.1 Introduzione al Machine Learning

Il Machine Learning è una delle materie più rilevanti nell'ambito dell'Intelligenza Artificiale. In particolare è un ramo dell'informatica che permette ad una macchina di imparare ad eseguire un compito senza che venga appositamente programmata per farlo, grazie all'individuazione di schemi ricorrenti tra i dati.

La sua definizione più classica e formale, utile per capire i principali elementi in gioco, è quella formulata da Tom M. Mitchell nel 1997:

> "Si dice che un programma apprende dall'esperienza E con riferimento ad alcune classi di task T e con misurazione della performance P, se le sue performance nel task T, come misurato da P, migliorano con l'esperienza E."

Analizzando un esempio concreto: un modello di Machine Learning ha l'obiettivo di riconoscere in un determinato soggetto la presenza o meno di tumori al seno, analizzando le immagini di mammografie e biopsie.

In questo caso il task T è il compito di diagnosticare correttamente la presenza di un tumore, la performance P è la percentuale di diagnosi effettuate correttamente, e l'esperienza E è il cosiddetto **dataset** di immagini **etichettate** dall'uomo.

Con immagine "**etichettata**" intendiamo l'immagine (in questo caso della mammografia) insieme a dei metadati ad essa associati che possono descrivere essa, o elementi di essa (in questo caso il metadato necessario è solamente uno: l'informazione sulla presenza o meno del tumore).

Dunque, assumendo che il modello in questione sia veramente un modello di Machine Learning, stando alla definizione di Mitchell più aumenta l'esperienza E, più aumenta la performance P.

L'obiettivo finale è proprio quello di massimizzare la performance P, e nasce da qui l'importanza di avere un grande *dataset* di immagini etichettate con la miglior precisione possibile.

In generale si può parlare di tre diverse categorie di algoritmi di Machine Learning:

- **Supervised Learning:** per apprendere, gli algoritmi supervisionati utilizzano dati già etichettati.
- **Unsupervised Learning:** per apprendere, gli algoritmi non supervisionati, detti anche *algoritmi di clustering*, utilizzano dati non etichettati. Le stesse classi della variabile target (cioè l'output dell'algoritmo) non sono note.
- **Reinforcement Learning:** anche in questo caso i dati non sono etichettati, con la differenza che l'algoritmo interagisce con l'ambiente esterno, ricevendo volta per volta feedback positivi o negativi in base alle scelte che compie.

I casi d'uso del Machine Learning al giorno d'oggi sono numerosissimi, e lo saranno sempre più in futuro. Per citare qualche esempio, in ambito medico può venire utilizzato per prevedere se un paziente ricoverato per colpa di un infarto avrà o meno un secondo infarto, sulla base di dati clinici e demografici.

In ambito industriale la maggior parte dei modelli di Machine Learning è utilizzata per fare *Anomaly Detection*, cioè per identificare anomalie inattese, e per la *Predictive Maintenance*, una tipologia di manutenzione che viene attuata dopo aver identificato una serie di parametri allo scopo di avere una previsione sul tempo residuo prima di incorrere in un guasto.

In ambito *computer vision* le applicazioni d'uso sono veramente innumerevoli: rilevamento/riconoscimento degli oggetti, riconoscimento facciale, scansione di impronte digitali, riconoscimento di testi scritti a mano, rilevamento di gesti, ecc...

1.2 Data Labelling e Data Annotation

Il data labelling è l'operazione di etichettatura dei dati. Spesso si parla anche di *data annotation*.

Nonostante la differenza sia molto sottile, generalmente con *data annotation* indichiamo la tecnica che effettivamente viene utilizzata per etichettare i dati, che permette quindi ad una macchina di capirli e memorizzarli.

Il *data labelling* invece consiste nell'aggiungere un significato a dati di diverso tipo, in modo da addestrare successivamente un algoritmo di machine learning.

Per poter effettuare il *data labelling* è pertanto necessaria una preliminare operazione di *annotation*. Nella prassi i due termini vengono spesso utilizzati in modo interscambiabile, quindi utilizzerò entrambe le espressioni per riferirmi al processo di etichettatura dei dati.

Ribadisco la cruciale importanza di questa operazione: la correttezza e la precisione con cui vengono etichettati i dati ha un grande peso sull'efficienza e sulle performance del modello.

I dati da dover etichettare possono essere di diversi tipi: generalmente testo, immagini, video o audio. In ambito *computer vision* generalmente si lavora su video e immagini. Le tipologie di *annotation* più utilizzate per questi dati sono generalmente di tre tipi:

- **Classification**

  La **classification**, detta anche *image recognition*, è la tipologia più semplice di annotazione. Il suo obiettivo è quello di riconoscere la presenza di un determinato oggetto all'interno dell'immagine al fine di etichettare l'immagine in questione con una scritta che identifica l'oggetto che rappresenta.

  Per fare un esempio, una domanda che la *classification* si pone può essere: *"È presente un semaforo in questa immagine?"* Se la risposta è sì, allora l'immagine verrà etichettata con la scritta *'semaforo'*, un metadato ad essa allegato, che rappresenta la vera e propria *annotation* dell'immagine.

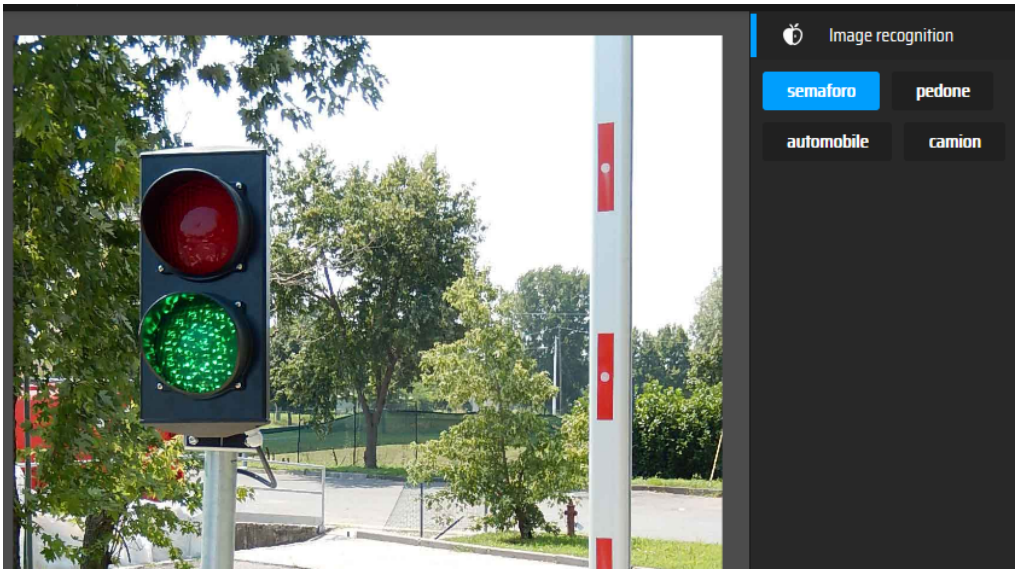

Figura 1.1: Immagine etichettata come *'Semaforo'*.

- **Object Detection**

  L' **Object Detection** è una delle tecniche di *annotation* più utilizzate. A differenza della *classification*, che vuole semplicemente capire se un determinato oggetto è presente o meno all'interno di un'immagine, l' **object detection** ha l'obiettivo di identificare la presenza e la posizione di uno o più oggetti all'interno di un'immagine. Il metodo più utilizzato per indicare la posizione di oggetti all'interno di immagini o video sono le *bounding box*.

In figura 1.2 è possibile notare che l'immagine è stata etichettata tramite *object detection*. Le *bounding box* appartenenti alla classe *car* sono di colore viola, mentre quelle appartenenti alla classe *trafficlight* sono rosse.

In questo caso quindi, le *annotation* dell'immagine contengono informazioni sulle *bounding box*: a quale immagine fanno riferimento, a quale classe fanno riferimento e la loro posizione nell'immagine. Come vedremo questi dati possono essere espressi attraverso diversi formati.

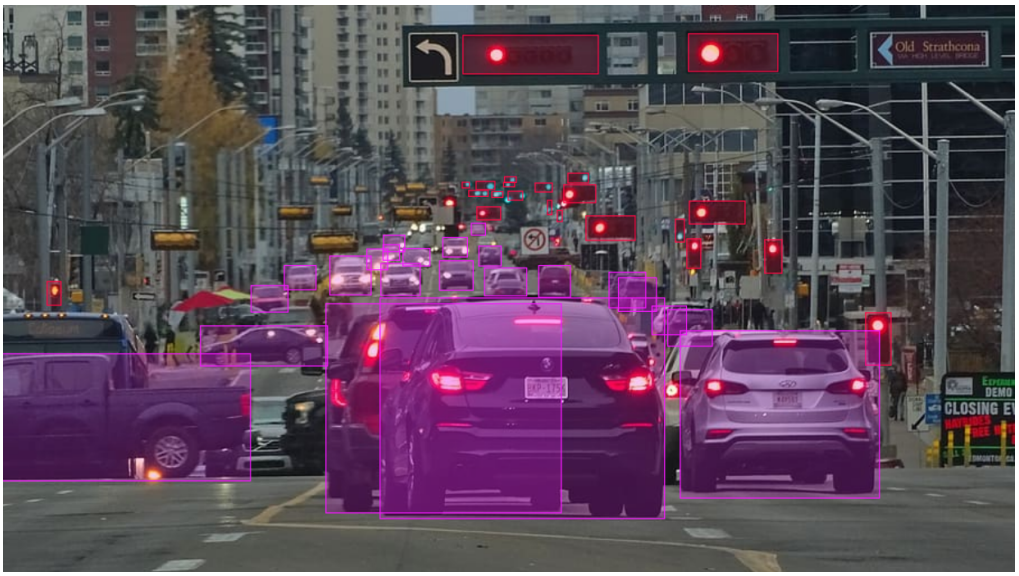

Figura 1.2: *Object Detection* all'interno di uno scenario automotive.

- **Segmentation**

  La **Segmentation** è la tipologia più complessa di *annotation* per le immagini. A differenza delle altre, l'analisi riguardo l'appartenenza di una classe avviene in modalità *pixel-wise*, cioè viene discriminata la classe di ogni pixel dell'immagine. Ogni area dell'immagine (gruppo di pixel) assegnato ad una classe viene chiamato segmento. La *segmentation* si suddivide a sua volta in tre sottocategorie:

  – **Semantic Segmentation**

    All'interno dell'immagine ogni segmento si riferisce ad una diversa classe, ma allo stesso tempo ogni diversa istanza della classe in questione appartiene allo stesso segmento.

    Faccio un esempio concreto: l'immagine in questione rappresenta uno scenario *automotive* e le classi possono essere di tipo *persona*, *veicolo*, *strada* e *cielo*. Nella *semantic segmentation* ogni pixel dell'immagine è assegnato ad una di queste quattro classi. Però due istanze diverse della classe

persona (due persone differenti nell'immagine) appartengono allo stesso segmento. Questo diventerà uno svantaggio quando sarà necessario contare gli oggetti di una determinata classe o tracciare i singoli oggetti nel tempo.

– **Instance Segmentation**
  L'*instance segmentation* considera diverse istanze della stessa classe come segmenti diversi, e non è detto che ogni pixel dell'immagine debba appartenere ad una classe. Infatti, considerando nuovamente l'esempio sopracitato, in questo le classi *strada* e *cielo* verrebbero rimosse, lasciando non classificati i pixel che teoricamente dovrebbero ad esse appartenere.

– **Panoptic Segmentation**
  La *panoptic segmentation* è il tipo di annotation più dettagliato e complesso di tutti, ed è la combinazione di *semantic* ed *instance segmentation*. Ad ogni pixel viene assegnata una delle classi; ogni singolo oggetto di ciascuna classe è identificato da diverse istanze di segmento.

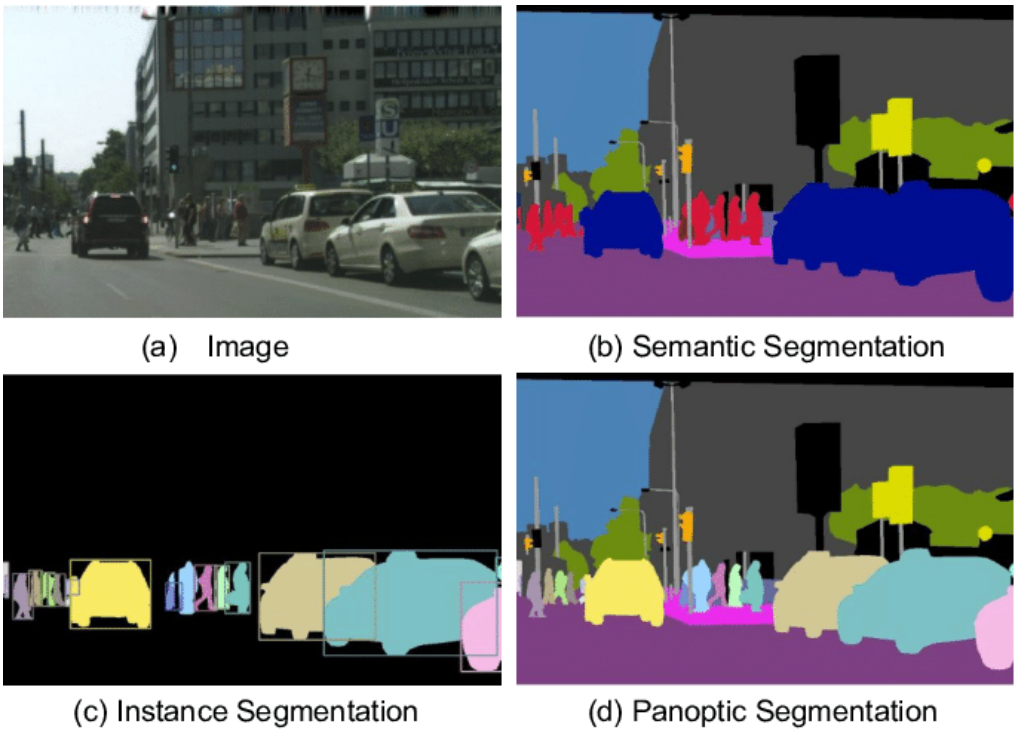

Figura 1.3: Le tre tipologie di *segmentation* su un'immagine di esempio.

Per effettuare il labelling di dati esistono diversi strumenti che possono aiutare nello svolgimento dell'operazione, e spesso anche velocizzarla.

Oltre alle tipologie di annotation già elencate, è possibile utilizzare diversi altri elementi per descrivere le immagini:

- Bounding Box: sono le più utilizzate per i task di **Object Detection**. Consistono in un rettangolo che racchiude un'area dell'immagine, associato ad una classe.
- Landmarks: utilizzati per la **Keypoint Detection**, consistono in singoli punti tracciati sull'immagine ed associati ad una classe.
- Poligoni: poligoni che racchiudono un'area dell'immagine, associati ad una classe.
- Cuboidi 3D: *bounding box* che si estendono anche sull'asse z.
- Linee e Splines: linee e curve tracciate sull'immagine ed associate ad una classe.

1.3 Formati di Annotation in computer vision

Il formato di annotation detta le regole con le quali vengono create le strutture dati dentro cui inseriamo le informazioni sulle etichette delle immagini.

I formati di annotation in ambito *computer vision* sono svariati, ed è importante adottare degli standard per la rappresentazione dei dati in modo da facilitarne il processo di lettura e scrittura, rendendo così universale il linguaggio utilizzato per descrivere ad esempio la posizione di una *bounding box* in un'immagine.

Di seguito vengono descritti i tre formati di annotation più utilizzati in *computer vision*:

1.3.1 COCO

COCO sta per *Common Objects in Context* ed è un formato che supporta cinque tipi di annotation: l'*object detection*, la *keypoint detection*, la *semantic* e la *panoptic segmentation* e l'*image captioning* (che significa assegnare ad un'immagine una sua didascalia testuale).

Le annotation in formato COCO [1] sono salvate utilizzando un file di tipo JSON, che sta per *JavaScript Objection Notation*. JSON è un formato molto utile per lo scambio di dati fra diverse applicazioni. Inoltre il modo con cui i dati vengono salvati al suo interno è facilmente interpretabile e manipolabile da diversi linguaggi di programmazione come JavaScript o Python.

```
0 {
1     "info": info,
```

```
    "images": [image],
    "annotations": [annotation],
    "licenses": [license],
}

info{
    "year": int,
    "version": str,
    "description": str,
    "contributor": str,
    "url": str,
    "date_created": datetime,
}

image{
    "id": int,
    "width": int,
    "height": int,
    "file_name": str,
    "license": int,
    "flickr_url": str,
    "coco_url": str,
    "date_captured": datetime,
}

license{
    "id": int,
    "name": str,
    "url": str,
}
```

Questa è la struttura dati principale, che deve essere integrata dalla definizione del campo *annotations*. Questo può avere una diversa struttura in base a quale delle 5 tipologie di annotation precedentemente elencate ci si sta riferendo. Riporto sotto l'esempio di formato solamente per le annotation di tipo *Object Detection* (bounding box) e per la *segmentation*.

```
annotation{
    "id": int,
    "image_id": int,
    "category_id": int,
    "segmentation": RLE or [polygon],
    "area": float,
    "bbox": [x,y,width,height],
    "iscrowd": 0 or 1,
}

```

```
categories[{
    "id": int,
    "name": str,
    "supercategory": str,
}]
```

Il campo "id" rappresenta un identificativo univoco per l'annotation, "image_id" indica a quale immagine si riferisce, e "category_id" a quale classe appartiene.

Il campo *"iscrowd"* contiene un valore booleano, che è 0 se ogni istanza di segmento rappresenta un singolo oggetto, ed è 1 se invece si vuole includere in una stessa istanza una lista di oggetti.

Il campo *"segmentation"* può avere due diversi tipi di valori: *RLE* viene usato quando *iscrowd* è uguale a 1, e sta per *Run-length Encoding*, che è un algoritmo di compressione di tipo *lossless* utilizzato per le immagini; quando invece *iscrowd* è uguale a 0, *"segmentation"* assume come valore una lista di poligoni. Generalmente se *iscrowd* è 0 vogliamo che un solo oggetto corrisponda ad un solo segmento. Tuttavia più poligoni devono poter comporre un solo segmento, nel caso in cui l'oggetto in questione fosse occluso.

"Bounding box" è una lista di valori di tipo *float* che indicano rispettivamente le coordinate x e y del punto superiore sinistro della *bounding box*, la sua lunghezza e la sua altezza. Queste misure vengono prese a partire dall'angolo superiore sinistro dell'immagine, il quale è indicizzato a zero.

"Area" è il valore della superficie della *bounding box* ($width * height$).

Nel caso di *Object Detection* i campi inerenti alla *segmentation* vengono lasciati vuoti, mentre nel caso di *Segmentation* la *bounding box* viene comunque inserita, considerando l'area che racchiude completamente ogni oggetto.

1.3.2 YOLO

YOLO significa *You Only Look Once* ed è un algoritmo molto diffuso per fare *real-time Object Detection*. Senza addentrarsi sui dettagli di funzionamento dell'algoritmo, è importante sapere che si basa su una particolare rete neurale che esegue un solo ciclo di *forward propagation* e calcola contemporaneamente le probabilità di appartenenza ad una classe e i dati relativi alle *bounding box*.

Data la sua particolare velocità, accuratezza e capacità di apprendimento è molto utilizzato, e il suo formato di output delle predizioni è diventato uno degli standard:

```
<object-class> <center_x> <center_y> <width> <height>
```

Per ogni file di immagine viene creato un file *.txt* corrispondente che contiene una annotation (come quella riportata sopra) per ogni riga.

Object-class indica il numero identificativo della classe di appartenenza. I nomi delle classi sono salvati su un altro file chiamato *'obj.names'* e la riga in cui sono scritti corrisponde all'identificativo della classe (partendo da 0).

Center_x, *center_y*, *width* e *height* sono valori di tipo *float* relativi al centro della *bounding box* e alle sue dimensioni, successivamente scalati tra 0 e 1 in base alle dimensioni dell'immagine [7].

$$center\_x = \frac{center\_x\ [px]}{image\_width\ [px]} \quad (1.1) \qquad center\_y = \frac{center\_y\ [px]}{image\_height\ [px]} \quad (1.2)$$

$$width = \frac{width\ [px]}{image\_width\ [px]} \quad (1.3) \qquad height = \frac{height\ [px]}{image\_height\ [px]} \quad (1.4)$$

Inoltre sono presenti i file *'train.txt'* e *'obj.data'* che indicano il primo la lista dei file relativi al training-set, e il secondo le informazioni relative al numero di classi e ai nomi degli altri file nominati, in modo da poterli rintracciare automaticamente nel caso in cui abbiano nomi differenti.

1.3.3 Pascal VOC

Pascal VOC (*Visual Object Classes*) [11] è una *collection* di dataset utilizzati per *l'object detection*, spesso utilizzati come *benchmark* per confrontare diversi modelli, ed avere una stima delle performance. Da esso nasce quindi il suo formato di annotazione, che viene salvato su file di tipo *XML*.

XML è un metalinguaggio che sfrutta una sintassi che permette di definire il significato degli elementi contenuti in altri documenti.

Il dataset presenta un file *XML* per ogni immagine in esso contenuta.

Le bounding box sono inserite nell'elemento *<bndbox>* che è un sottoelemento di
<object>.

Xmin e *ymin* indicano le coordinate del punto inferiore sinistro della *bounding box*, mentre *xmax* e *ymax* indicano le coordinate del punto superiore destro della *bounding box*.

```
0 <annotation>
1   <folder>Trafficlight</folder>
2   <filename>001.png</filename>
3   <path>/Trafficlight/001.png</path>
4   <source>
5     <database>Unknown</database>
6   </source>
7   <size>
8     <width>360</width>
```

```
    <height>360</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>36</name>
    <pose>Frontal</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <occluded>0</occluded>
    <bndbox>
      <xmin>80</xmin>
      <xmax>130</xmax>
      <ymin>64</ymin>
      <ymax>40</ymax>
    </bndbox>
  </object>
</annotation>
```

Capitolo 2

Labelling tools

In questo capitolo mostro i risultati di una ricerca sugli strumenti di etichettatura ad oggi disponibili sul mercato, che reputo allo stato dell'arte per aiutare nell'operazione di labelling, valutando i loro vantaggi e svantaggi riguardo a costi, tempi per portare a termine determinati *task*, e *User Experience*.

Tutti i software che ho scelto di prendere in considerazione sono *open source*. La scelta è dettata da tre motivi principali: il primo è che, soprattutto in ambito accademico e di ricerca, trovo utile che siano fruibili a chiunque.

Il secondo motivo è legato ad un maggior controllo del software: lavorare su un software *open source* può comportare il sorgere di problemi dovuti a incompatibilità o *bug*, a maggior ragione se è poco conosciuto ed è stato rilasciato recentemente. In compenso il vantaggio consiste nel poter risolvere i problemi semplicemente analizzando, consultando, e talvolta modificando il codice sorgente del software.

Il terzo motivo riguarda la *community* che sta dietro ad ogni progetto *open source*, la quale, secondo la mia esperienza personale, mi è sempre parsa disposta ad aiutare gli altri utenti, generando *thread* utili a chiunque effettui una ricerca, e portando ognuno il proprio contributo al codice.

2.1 Makesense.ai

Makesense.ai [16] è un progetto *open source* utile per fare annotation di immagini. È utilizzabile in due modalità, la prima tramite il sito web, la seconda installando il software in locale. L'installazione può avvenire clonando il *repository* di Github, per poi installare le dipendenze necessarie per il corretto funzionamento tramite NPM (*Node Package Manager*); oppure attraverso Docker, una piattaforma che permette la creazione e distribuzione di applicazioni con rapidità, riducendo al minimo i problemi relativi alle dipendenze. Le sue applicazioni sono contenute all'interno di

containers che forniscono un'astrazione aggiuntiva ad una applicazione standard, grazie alla virtualizzazione a livello di sistema operativo.

2.1.1 Features supportate

Makesense supporta le seguenti *features*:

- Bounding box
- Landmarks
- Poligoni
- Linee

2.1.2 Autolabelling

Il software supporta due modelli di machine learning già addestrati.

- **SSD model**

  È un modello addestrato grazie al dataset COCO in grado di eseguire *object detection* sulle immagini importate sul software. Questo rende il processo di labelling molto più veloce, perché le *bounding box* saranno già posizionate con una media-buona precisione, talvolta anche già assegnate alla corretta classe. Le operazioni da fare manualmente sono quindi quelle di correzione sulle imprecisioni dei bordi delle *bounding box*, la correzione o assegnazione della classe di appartenenza dell'oggetto, e l'eventuale inserimento manuale della *bounding box* su oggetti non rilevati.

- **PoseNet model**

  PoseNet è un modello che viene utilizzato per stimare la posizione di persone all'interno di immagini e video, ricercando i *keypoints* corrispondenti a determinate parti del corpo. Dall'insieme delle *landmarks* è possibile risalire ad una stima della posizione nella quale si trova il soggetto.

Figura 2.1: A sinistra un'immagine in seguito alla *Pose Detection* effettuata dal modello PoseNet. A destra la stessa immagine dopo aver corretto manualmente le posizioni inserite con imprecisione dal modello.

Come è possibile notare nell'immagine 2.2, a sinistra sono rappresentate con dei pallini azzurri le posizioni stimate dal modello PoseNet. Ognuna di queste corrisponde ad una classe (left wrist, right wrist, left ankle, right ankle, ecc...). La correttezza di questi punti deve essere verificata manualmente dall'uomo. Infatti cliccandoci sopra l'annotation viene confermata, e si può così modificare in seguito (sia la sua posizione, sia la sua classe). Viene anche data la possibilità di confermare tutte le *landmarks* con un solo comando nel caso in cui nessuna *landmark* sia stata inserita erroneamente dal modello.

Nell'immagine di destra invece sono presenti, in colore bianco, le *landmarks* già modificate e corrette dall'intervento umano. La stima della loro posizione in alcuni casi era sbagliata, ma grossolanamente il risultato è abbastanza preciso, soprattutto perché le classi vengono assegnate con un'ottima precisione. Questo velocizza estremamente il processo di labelling manuale.

2.1.3 Formati supportati

I formati ad oggi supportati per l'importazione sono:

- YOLO, solamente per le *bounding box*.
- COCO, per *bounding box* e poligoni.

Per l'esportazione invece i formati supportati sono:

- CSV, per *landmarks*, linee, *bounding box* ed etichette testuali.
- YOLO, per le *bounding box*.
- VGG, per i poligoni.
- COCO, per i poligoni.

Sulla documentazione del software sono inoltre indicati diversi formati per i quali gli sviluppatori hanno intenzione di implementare il supporto in futuro.

2.1.4 Valutazione complessiva

Concludendo, i vantaggi di questo software sono la sua semplicità nell'utilizzo, il fatto che offra una versione accessibile anche online, senza limiti di utilizzo e senza che le immagini utilizzate vengano caricate su server per scopi all'utente sconosciuti.

I due modelli di AI presenti sono utili nel velocizzare i task di etichettatura di *object detection* e *pose estimation*.

Tuttavia il software presenta qualche svantaggio nella sua limitazione riguardo i formati supportati e riguardo l'interazione con modelli addestrati dall'utente. Se l'utente volesse esportare le annotations in formato CSV, che è il formato standard suggerito, non avrebbe modo di reimportarle in seguito per effettuare un'eventuale modifica, senza prima convertire le annotation in un altro formato. Inoltre non è supportata, in nessun formato, l'importazione di landmarks, linee o annotation testuali. Le funzioni utente sono ridotte al minimo, non è possibile lavorare su più progetti contemporaneamente e gli unici dati supportati in input sono immagini.

2.2 Label Studio

Label Studio (labelstud.io) [20] è un software di labelling *multi-Domain*. Ciò significa che, a differenza degli altri *tool* che ho analizzato, è progettato per poter etichettare diversi tipi di dati, oltre a quelli standard per la *computer vision*.

Il software può essere utilizzato per annotare dati *raw* oppure per migliorare e aggiustare etichette già predette da un algoritmo di machine learning.

Label Studio può essere installato in locale tramite Docker, pip o Anaconda, oppure si può effettuare il suo deploy su una piattaforma *cloud* a scelta tra Google Cloud, Heroku o Microsoft Azure.

La *User Experience* è molto positiva, il software è utilizzabile dall'utente in maniera molto intuitiva e l'interfaccia grafica è piacevole. È possibile lavorare

su più progetti contemporaneamente, ed è presente la funzione di multi-utente, estremamente utile per poter lavorare in team sullo stesso progetto.

I dati supportati sono: immagini, audio, testo, HTML, serie temporali e video.

2.2.1 Template implementati

Label Studio è facilmente estendibile, e aggiungere *templates* di lavoro destinati a nuovi determinati task è possibile. Tuttavia i *template* implementati di default sul software sono molteplici:

- **Computer Vision:** *semantic segmentation* con poligoni e con maschere, *object detection*, *image classification*, *image captioning*, *visual question answering* e riconoscimento ottico dei caratteri.
- **Natural Language Processing:** rispondere a domande, classificazione di testi e argomenti, traduzione automatica, riconoscimento di identità nominate, riepilogo automatico, estrazione delle relazioni, tassonomia.
- **Audio Processing:** riconoscimento vocale, *speaker segmentation*, classificazione dell'intento, *sound event detection*, riconoscimento della qualità del segnale.
- **Conversational Artificial Intelligence:** Classificazione dell'intento, selezione della risposta, generazione della risposta, decisione coreferenziale.
- **Time Series Analysis:** Riconoscimento delle attività, rilevamento del punto di cambiamento, *detection di outliers* e anomalie, predizione di serie temporali, qualità del segnale.
- **Structured Data Parsing:** Riconoscimento di Hypertext ed entità HTML, PDF, Tabular Data, labelling di pagine web.
- **Ranking e Scoring:** ricerca di documenti, ricerca delle immagini in base al contenuto, classificazione a coppie, regressione a coppie.
- **Video:** video *classification*, video *timeline segmentation*.

2.2.2 Autolabelling

Attraverso Label Studio Machine Learning SDK è possibile interagire facilmente con i propri modelli di machine learning. In particolare l'utente può procedere con tre diversi metodi:

- **Pre-labellare** i dati attraverso modelli predittivi per poi aggiustare le annotations manualmente.

- Fare **online learning**, riaddestrando il modello continuamente dopo aver aggiustato le annotations da lui predette.
- Fare **active learning**, etichettando solamente gli scenari più complessi del proprio dataset.

2.2.3 Formati supportati

Il formato supportato per l'importazione è JSON, e ciò si può fare rispettando la sintassi definita dalla documentazione del software.

Per l'esportazione i formati supportati sono:

- JSON, lista di elementi in formato JSON usato per esportare sia i dati sia le annotations di un dataset.
- JSON_MIN, versione più ristretta della precedente, in cui vengono omessi i campi specifici di Label-Studio.
- CSV, valori separati da virgola.
- TSV, valori separati da *tab*.
- CONLL2003, formato utilizzato per la *CoNLL-2003 named entity recognition challenge*.
- COCO, per task di *object detection* e *segmentation*.
- Pascal VOC, per task di *object detection* e *segmentation*.
- Brush labels to NumPy PNG, esporta le annotations come immagini PNG o array numpy 2D.
- ASR_MANIFEST, per le trascrizioni di audio in seguito a riconoscimento vocale.
- YOLO, per i task di *object detection*.

2.2.4 Valutazione complessiva

Label Studio è un ottimo software *open source* per creare annotations *general purpose*. È possibile lavorare su moltissimi ambiti, dalla *computer vision* all'intelligenza conversazionale. Questo è un grande vantaggio che permette all'utente di etichettare dati per svariati casi d'uso.

Un altro vantaggio è quello di poter integrare il software con altri *tool* personali già esistenti, sia in modalità *frontend* sia in modalità *backend*.

Sulla pagina github del *tool* è presente una sezione chiamata *public roadmap* che mostra le intenzioni future di implementazione, miglioramento, e aggiunta di *features*, utile ad avere una prospettiva di ciò che gli sviluppatori hanno intenzione di sviluppare per migliorare il software. La *user experience* è positiva e il software è facilmente utilizzabile e configurabile dall'utente.

Un aspetto negativo è che la sua versatilità in tanti diversi ambiti dell'intelligenza artificiale lo rende un po' meno performante ed efficiente se utilizzato esclusivamente per una tipologia di task.

Dunque consiglio il suo utilizzo quando è necessario un mezzo per poter lavorare su svariate tipologie di dati, operando in più ambiti dell'*Artificial intelligence*.

2.3 CVAT

CVAT, *Computer Vision Annotation Tool* [10] è un annotation tool *open source* destinato all'ambito della *computer vision*.

L'installazione può essere eseguita in locale utilizzando docker, sia su sistemi Linux sia su sistemi Windows (attraverso *WSL2*) sia su Mac OS Mojave.

Esiste anche una versione demo accessibile online, che permette l'utilizzo del software senza installarlo in locale. Questa modalità implica però un limite di 10 task per ogni utente e un massimo di 500 Megabyte di dati da poter caricare.

Il codice è distribuito sotto la licenza del MIT (licenza di software *open source* creata dal *Massachusetts Institute of Technology*).

2.3.1 Features supportate

CVAT supporta le seguenti features:

- Bounding box
- Poligoni
- Linee Spezzate
- Punti (Landmarks)
- Cuboidi
- Cuboidi per task 3D
- Tag

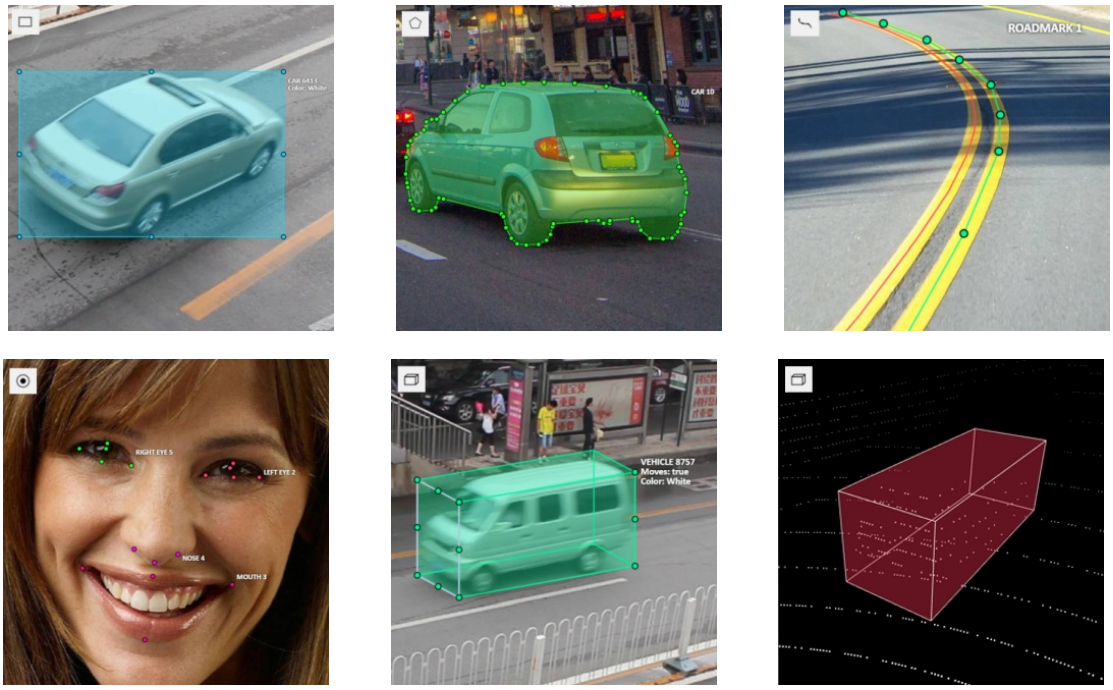

Figura 2.2: Sono riportati esempi di annotation – da sinistra verso destra – rispettivamente di *bounding box*, poligono, linea spezzata, *landmarks*, cuboide e cuboide su un task 3D.

2.3.2 Interpolazione

Il software può etichettare non solo immagini ma anche video. Il modo teorico per poter etichettare un video sarebbe quello di annotare ogni suo singolo frame; ciò risulterebbe particolarmente lungo, ripetitivo e laborioso. Il software offre una modalità di etichettatura chiamata *track* che, sfruttando la tecnica dell'interpolazione, semplifica notevolmente il processo.

L'interpolazione — matematicamente parlando — è un metodo che serve ad individuare nuovi punti sul piano cartesiano partendo da un insieme di punti già definito, cercando una funzione $f(x)$ alla quale appartengono tutti i punti in questione.

Quindi la modalità *track* consente di posizionare un'etichetta (ad esempio una *bounding box*) su un frame del video.

Automaticamente la stessa etichetta viene inizialmente replicata sui frame seguenti, nella stessa posizione. Se l'oggetto in questione è in movimento occorre spostare opportunamente l'etichetta.

La chiave di volta dell'interpolazione sta nel fatto che l'aggiustamento delle annotation non deve per forza essere eseguito per ogni frame dell'immagine, ma può

essere fatto anche dopo che l'oggetto ha compiuto uno spostamento più o meno grande.

Avendo quindi a disposizione i punti delle annotation già collocate, si tratta di eseguire una loro interpolazione al fine di ottenere i punti a loro corrispondenti per tutti i frame che non sono stati manualmente etichettati.

La modalità di etichettatura *track* può essere utilizzata per qualsiasi delle *feature* che il software supporta.

2.3.3 Autolabelling

Su CVAT si possono utilizzare 11 modelli di machine learning che offre di default in modalità *serverless* (disponibili su cloud, ed utilizzati solamente al momento della richiesta):

- **Deep Extreme Cut** Deep Extreme Cut, chiamato anche *DEXTR*, è basato su *framework* OpenVINO ed è di tipo *interactor*. Ha come scopo quello di tracciare automaticamente un poligono sul contorno di un oggetto, fornendo inizialmente i primi punti manualmente. [2]

- **Faster RCNN** È un detector utilizzato per fare *Object Detection* degli oggetti più popolari. Il modello è presente in due versioni: una basata su *framework* OpenVINO, l'altra basata su TensorFlow. [5]

- **Mask RCNN** È un detector utilizzato per fare *instance segmentation*. Anche in questo caso modello è presente sia in versione su *framework* OpenVINO sia su TensorFlow. [8]

- **YOLO v3** Anche questo modello è un detector in grado di fare *object detection* degli oggetti più utilizzati in *computer vision*. Appartiene al *framework* OpenVINO. [21]

- **Object reidentification** È un modello di tipo *reid*, cioè di re-identificazione (in particolare delle persone), anche questo basato su OpenVINO. [9]

- **Semantic Segmentation for ADAS** È un modello di tipo detector che esegue la *semantic segmentation* su scenari di tipo ADAS, che sta per *Advanced driver-assistance systems*. Il *framework* utilizzato è quello di OpenVINO. [14]

- **Text detection v4** Text Detection v4 [19] è un modello che utilizza il *framework* OpenVINO che serve per il riconoscimento di testi all'interno di immagini. È basato sull'architettura *PixelLink* [3].

- **SiamMask** SiamMask è un modello di tipologia *tracker* che ha l'obiettivo di eseguire velocemente la *detection* e il tracciamento di oggetti nello spazio. Il *framework* utilizzato è PyTorch. [15]

- **f-BRS** Anche questo modello è basato su *framework* PyTorch. È di tipo *interactor* e ha un funzionamento simile al modello *DEXTR*: si ottiene una *segmentation* di un oggetto selezionando punti dell'immagine positivi (appartenenti all'oggetto) e negativi (non appartenenti all'oggetto). Così facendo il modello riesce man mano ad avere le informazioni necessarie a tracciare la maschera del poligono che meglio rappresenta l'oggetto nell'immagine. [4]

- **Inside-Outside Guidance** Inside-Outside Guidance si basa anch'esso su PyTorch e il funzionamento è molto similare a quello di *f-BRS*, con l'unica differenza che prima di posizionare i punti l'utente procede a tracciare la *bounding box* che meglio racchiude l'oggetto. In questo modo l'algoritmo è facilitato nella ricerca della giusta *segmentation*. [6]

- **RetinaNet** RetinaNet è basato su PyTorch ed è un detector destinato a task di *object detection*. Supporta anche il suo utilizzo su GPU. [12]

2.3.4 Formati supportati

Il formato supportato per l'importazione sono:

- CVAT for images, è un file in XML in cui per ogni immagine si riportano i suoi oggetti e ogni oggetto può avere più attributi.

- CVAT for videos, è un file in XML che rappresenta l'interpolazione che rappresenta lo spostamento dei vari punti dei poligoni in base ai *frames*.

- PASCAL VOC

- YOLO

- MS COCO Object Detection

- TFrecord, utilizzato per salvare una sequenza di record binari.

- MOT, Multiple Object Tracking Benchmark.

- LabelMe 3.0, è un annotation *tool* online utilizzato in *computer vision*. Questo formato è compatibile con il tool in questione.

- ImageNet, un database di immagini organizzato sulla base della gerarchia WordNet. Il dataset è gratuito per i ricercatori e per usi non commerciali.

- CamVid, Cambridge-driving Labeled Video Database, il primo dataset di video con un'etichettatura di tipo *semantic segmentation*. Le etichette del dataset forniscono la cosiddetta *ground truth*, cioè descrivono correttamente la realtà della scena in questione.

- WIDER Face, un dataset di rilevamento facciale.
- VGGFace2, un dataset per il riconoscimento facciale che utilizza immagini scaricate da *Google Image Search*.
- Market-1501, un dataset per la re-identificazione di persone.
- ICDAR15/15, dataset per il riconoscimento e la lettura di testi all'interno di immagini.

Per l'esportazione sono supportati tutti i formati già elencati per l'importazione, con l'aggiunta di **Datumaro**, che è un *tool* utilizzato per creare, trasformare ed analizzare dataset.

2.3.5 Valutazione complessiva

CVAT è in definitiva uno dei software *open source* più valido ed utilizzati in ambito *Computer Vision*.

Esso è fornito di manuale e di una documentazione completa che spiega dettagliatamente le varie funzioni del programma e il loro utilizzo, con anche alcuni video esplicativi o dimostrativi che mostrano a livello pratico l'utilizzo di alcune *features* più particolari.

I formati di input e output dei dati supportano i dataset più utilizzati. Inoltre supporta moltissime *features* tra cui i cuboidi 3D, un elemento che è difficile da trovare su altri *tool* gratuiti.

Offre molteplici modelli di machine learning integrati, destinati a diversi casi d'uso. Questi possono automatizzare o velocizzare estremamente i task.

Viene più dettagliatamente illustrato nel capitolo 3 lo sviluppo di un progetto in cui viene scelto questo software per l'applicazione su un caso d'uso reale.

2.4 6D-PAT

6D-PAT (6D Pose Annotation Tool) è un software di etichettatura che prende in input immagini 2D e modelli 3D. Il *tool* consente all'utente di annotare dove, nello spazio 2D delle immagini, i modelli 3D sono collocati.

6D-PAT è un *tool open source* sviluppato da Florian Blume, un ricercatore dell'università tecnica di Berlino. Il progetto fu iniziato nel 2017 ed è tuttora in continuo miglioramento grazie al lavoro di Florian e ai feedback di diversi utenti che utilizzano il software.

Ho anche personalmente fornito un piccolo contributo risolvendo un *bug* sull'ultima *release*. Il problema riguardava il fallimento del salvataggio su disco dei dati relativi alle posizioni.

Il codice del software è distribuito attraverso una licenza GPL 3.0 ed è sviluppato in *c++*.

Il software può essere installato lanciando una delle release che si trovano sulla pagine github, oppure tramite docker, oppure compilando direttamente il codice sorgente attraverso l'*IDE* QtCreator.

2.4.1 Il funzionamento di base

Il programma consente all'utente di caricare una serie di immagini e una serie di modelli 3D. Scelta un'immagine e il modello 3D dell'oggetto di cui si vuole trovare la posizione, si procede ad abbinare manualmente una serie di punti dell'immagine ai punti corrispondenti dell'oggetto 3D. (figura 2.3)

Le coppie di punti da dover posizionare sono almeno sei, e per facilitarne la collocazione è possibile ruotare il modello in modo da rendere la sua posizione quanto più possibile analoga alla posizione dell'oggetto sull'immagine.

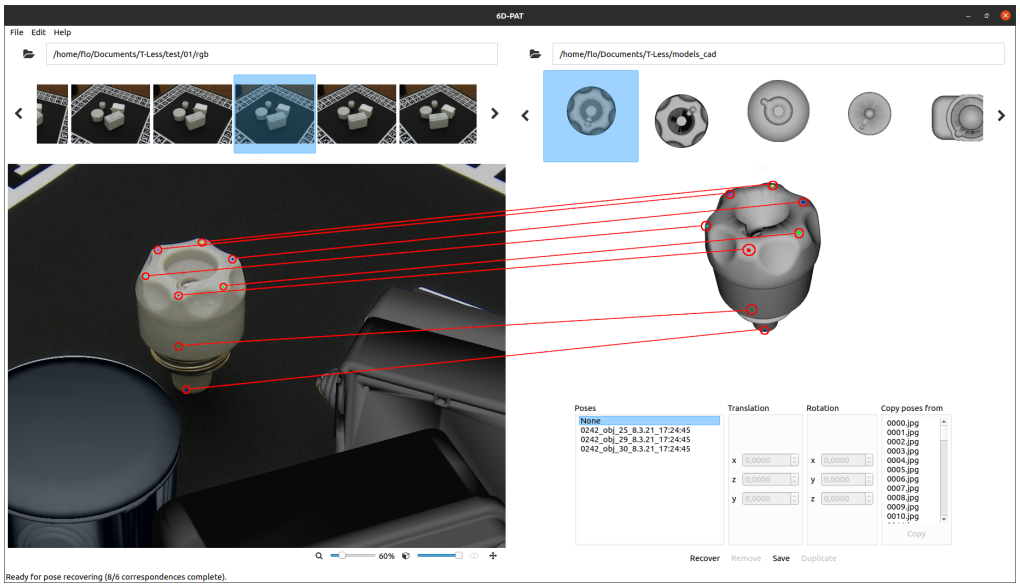

Figura 2.3: Schermata principale del *tool*, evidenziate in rosso le linee che indicano le corrispondenze dei punti sul modello 3D ai punti sull'immagine.

Una volta che le sei coppie di punti sono state correttamente inserite è possibile continuare ad aggiungere punti per migliorare la precisione del risultato oppure premere il tasto *recover* che esegue il calcolo della stima della posizione dell'oggetto, situando poi sull'immagine una copia del modello 3D nella posizione precedentemente calcolata.

Il modello può essere manualmente traslato e ruotato per aggiustare possibili imprecisioni commesse dalla funzione di stima della posizione.

2.4.2 Parametri della camera

Nei progetti di *computer vision* in cui si ha a che fare con task di *pose estimation*, e non solo, è importante tenere conto dei parametri della camera che si sta utilizzando.

I parametri delle camere si suddividono in intrinseci ed estrinseci:

- I parametri intrinseci sono grandezze che dipendono da caratteristiche fisiche della camera, e indicano la mappatura fra le coordinate geometriche e le coordinate in pixel nell'immagine.
  - Lunghezza focale x e y: $\boldsymbol{f_x}$ e $\boldsymbol{f_y}$. Talvolta viene indicata solamente f poiché quelle lungo x e lungo y possono essere ricavate da quest'ultima, dividendola per le rispettive dimensioni del pixel. (Espresse in px se esplicitate entrambe, espressa in mm se esplicitata solamente la lunghezza focale).
  - Il centro ottico: $\boldsymbol{(c_x,\ c_y)}$. Questa coppia di coordinate indica il centro ottico, quindi l'origine del sistema di riferimento del piano dell'immagine. (Espresso in px).
  - Aspect ratio: $\boldsymbol{\alpha}$. L'*aspect ratio* è il rapporto tra la dimensione verticale e orizzontale del pixel del sensore. (Espresse in mm se sono esplicitate le dimensioni del pixel, senza unità di misura se è espresso il rapporto).
  - Coefficienti di distorsione radiale: $\boldsymbol{k_1}$ e $\boldsymbol{k_2}$. Dato che ogni lente reale introduce una distorsione, è bene prenderla in considerazione.
- I parametri estrinseci sono invece parametri che indicano la traslazione e la rotazione della camera rispetto ad un sistema di riferimento fisso.
  - Vettore di traslazione: $\boldsymbol{T}$. È un vettore di tre elementi che rappresenta una traslazione nello spazio 3D.
  - Matrice di rotazione: $\boldsymbol{R}$. È una matrice che rappresenta una rotazione considerando 3 gradi di libertà (lungo l'asse x, y, e z). La rotazione viene rappresentata con una matrice. Moltiplicando questa per un vettore, il risultato sarà un secondo vettore che descrive i punti dopo l'applicazione della rotazione.

I parametri intrinseci possono essere utilizzati per risalire alla cosiddetta **camera matrix**, detta anche **matrice intrinseca**. Questa viene chiamata nella documentazione del software K ed è composta utilizzando le lunghezze focali della camera e le coordinate del centro ottico. Si esplicita la sua forma nell'equazione 2.1. La matrice intrinseca in generale è una matrice che trasforma le coordinate 3D della fotocamera in coordinate 2D dell'immagine da essa scattata, e viene dunque utilizzata in ogni problema che necessita una trasformazione di punti dallo spazio 3D a quello 2D e viceversa.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

2.4.3 Stima della posizione

La tecnica utilizzata per stimare la posizione consiste nell'utilizzo di una funzione definita dalla libreria *OpenCV* chiamata **solvePnP** [17].

Si consideri un *array* di punti 3D (punti che appartengono ad un oggetto 3D) e un corrispondente *array* di punti 2D (punti che appartengono ad un'immagine scattata all'oggetto). Tutte le coordinate di tutti i punti dei due *array* sono relative al sistema di riferimento chiamato arbitrariamente w (cioè il sistema di riferimento del mondo).

SolvePnP ha l'obiettivo di trovare la coppia di vettori (il vettore di traslazione e il vettore di rotazione) tale per cui è possibile trasformare l'*array* di punti 3D relativi al sistema di riferimento w in un *array* di punti 3D relativi al sistema di riferimento c: quello che ha come punto di origine il centro della camera.

In figura 4.1 i punti 3D sono indicati in arancione, le loro coordinate sono relative al sistema di riferimento del mondo (w) e sono espresse come X_w, Y_w e Z_w nell'equazione 2.4. Queste tre variabili si possono considerare anche come singoli punti per facilitare l'interpretazione del calcolo matriciale, ma esse possono anche essere vettori riga contenenti le rispettive coordinate di un insieme di punti, in modo tale da eseguire il calcolo per tutti i punti utilizzando una sola iterazione.

I punti gialli sono posti sul piano dell'immagine e sono le proiezioni dei punti arancioni, espresse come u e v nell'equazione 2.4.

Nell'equazione viene usata la *camera matrix* per effettuare la conversione dallo spazio 3D a quello 2D, e viene moltiplicata per $\mathbf{\Pi}$ (la matrice di proiezione).

$$\Pi = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.2}$$

$$^{c}T_{w} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

$\mathbf{\Pi}$ (equazione 2.2) rappresenta la matrice di proiezione prospettica e in questo caso essa non applica alcuna trasformazione alla *camera matrix*, se non l'aggiunta di una colonna di zeri alla fine, operazione necessaria al fine di poter moltiplicare la matrice risultante per la matrice $^{c}\boldsymbol{T_w}$.

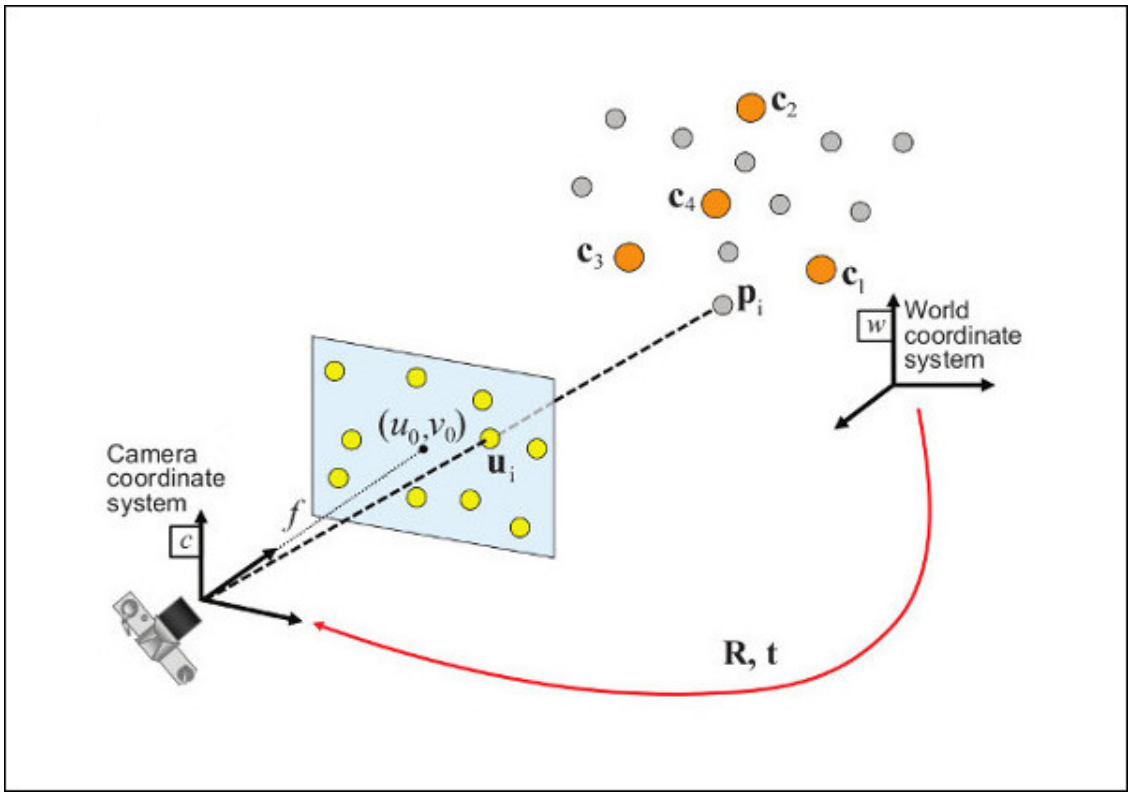

Figura 2.4: Rappresentazione dello scenario in cui si vuole stimare la posizione dei punti nello spazio rispetto alla camera.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2.4}$$

${}^{c}\boldsymbol{T}_{\boldsymbol{w}}$ è la matrice che si ottiene dal vettore di traslazione e dal vettore di rotazione: i due output di SolvePnP. Questa equazione è la base di funzionamento della funzione utilizzata dal software 6D-PAT per risalire alla posizione degli oggetti.

Riassumendo SolvePnP ha bisogno di quattro variabili in input:

- Il vettore di punti appartenenti ai modelli 3D, quindi una lista di vettori in uno spazio 3D rappresentati dalle 3 coordinate x y e z.
- Il vettore di punti posizionati sull'immagine, cioè un elenco di vettori 2D contenenti le due variabili x e y del punto sull'immagine.

- La *Camera Matrix*.
- I coefficienti di distorsione della lente.

La funzione restituisce le due variabili **tvec** e **rvec** che rappresentano rispettivamente il vettore di traslazione e il vettore di rotazione.

In seguito il software risale alla *rotation matrix* (matrice di rotazione) grazie alla funzione di *OpenCV* chiamata **Rodrigues** [13]. L'utilizzo pratico del software verrà spiegato meglio nel capitolo 4.

Capitolo 3

Primo caso d'uso: *Detection* di persone e cartelli stradali

3.1 Obiettivo

L'azienda **Montini** progetta e produce carrelli elevatori elettrici speciali, studiati appositamente per l'ottimizzazione di magazzini con particolari esigenze.

Il laboratorio **HiPeRT Lab**, in collaborazione con Montini, ha l'obiettivo di realizzare un modello di machine learning che riesca a rilevare la presenza e la posizione di persone e di segnali stradali di *"stop"* su immagini prodotte da una telecamera posta sul carrello elevatore.

La parte di etichettatura dunque è estremamente importante per avere un buon dataset attraverso cui addestrare al meglio il modello.

3.2 Apprendimento Incrementale

Tra gli annotation tool analizzati durante la fase di ricerca il più adatto a questo scenario è **CVAT**, data la sua ottima predisposizione all'ambito *computer vision* e alla sua estrema comodità durante il processo di annotation, che permette all'utente di muoversi molto velocemente da un'immagine all'altra utilizzando semplici scorciatoie da tastiera. Inoltre la vastità di formati da esso supportati faciliterà il processo di importazione delle annotation, evitando di eseguire ulteriori conversioni.

Il primo dataset di immagini da etichettare è composto da 271 immagini già precedentemente annotate grazie ad una rete YOLO. Nonostante la rete utilizzata in partenza riesca molto spesso a riconoscere i soggetti d'interesse, la precisione non è elevata e gli errori commessi sono molto frequenti. È normale riscontrare questo problema quando si utilizza una rete per fare inferenza su un dataset molto diverso da quello che è stato inizialmente utilizzato per effettuare il training.

Dunque si è deciso di procedere con un approccio di apprendimento incrementale. È possibile addestrare una rete neurale utilizzando un dataset iniziale abbastanza ristretto e con etichette posizionate in modo non eccessivamente accurato, per poi fare inferenza su nuovi dati non etichettati, correggere manualmente le etichette predette, e dare di nuovo in input alla rete gli ultimi dati corretti per un nuovo training.

Questo approccio comporta un continuo miglioramento delle performance e della precisione del modello, e allo stesso tempo velocizza il processo manuale di etichettatura dei dati, rendendolo semi-automatico.

Partendo dal primo dataset di immagini già etichettate il procedimento si suddivide in quattro fasi principali:

- Preparazione e conversione dei dati e delle annotation nel giusto formato.
- Importazione del dataset su CVAT.
- Correzione delle annotation e classificazione degli scenari di interesse.
- Esportazione e splitting dei dati.

3.3 Preparazione e conversione dei dati

Il dataset fornito inizialmente è composto da 271 immagini in formato *jpg*. Ad ogni immagine corrisponde un numero di file di testo *txt* equivalente al numero di classi del problema. In questo caso le classi sono due, **person** e **stop sign**, quindi i file di testo sono 542 (metà nominati *"persons<image_id>.txt"* e metà nominati *"signals<image_id>.txt"*).

In ognuno dei file di testo sono riportate le annotation che rappresentano le bounding box della classe in questione contenute all'interno dell'immagine con l'*image_id* uguale a quello presente nel nome del file *txt*.

Le annotations a loro volta sono riportate all'interno del file in formato YOLO e ad ogni riga corrisponde una *bounding box*. Tuttavia le *prediction* che escono in output da una rete YOLO adottano un formato temporaneo, e hanno bisogno di uno *scaling* per poter essere utilizzate. Lo *scaling* viene effettuato per mezzo delle formule 1.1, 1.2, 1.3 e 1.4 spiegate al capitolo 1.

Inoltre l'identificativo della classe deve essere posto come primo valore della riga che rappresenta la *bounding box* mentre nei file *txt* forniti non viene riportato.

Ho implementato uno script *python* che permettesse la conversione dei formati delle annotations, effettuando lo *scaling* ed inserendo la giusta classe della annotation in base alla classe riportata sul nome del file, compattando così la coppia di file *txt*, inizialmente presenti per ogni immagine, in un solo file *txt* per ogni immagine,

Class Name	**Class ID**
Person	0
Stop Sign	1

Tabella 3.1: Tabella che riporta le classi e i loro corrispondenti *Class Id*.

nominato con lo stesso nome dell'immagine corrispondente. Inoltre è stata necessaria l'aggiunta dei file *'obj.data'*, *'obj.names'* e *'train.txt'*, necessari ad interpretare correttamente il formato YOLO, come spiegato nella sezione 1.3.2. Infine ho implementato uno script Python che permettesse la conversione delle annotation anche in formato *COCO*, per un'eventuale utilità futura.

3.4 Importazione del dataset su CVAT

I dati sono pronti per essere importati su CVAT in modo da poter iniziare a lavorare sulle etichette.

CVAT è stato installato in locale ed è quindi raggiungibile dal link *"localhost:8080"*, verrà in seguito chiesta un'autenticazione: per effettuare il login si utilizzano le credenziali dell'account *admin* impostate durante la fase di installazione.

La prima operazione da eseguire è quella di creare un nuovo *task* cliccando sul pulsante *'create new task'*. In seguito verrà visualizzata una pagina di creazione task uguale a quella rappresentata in figura 3.1.

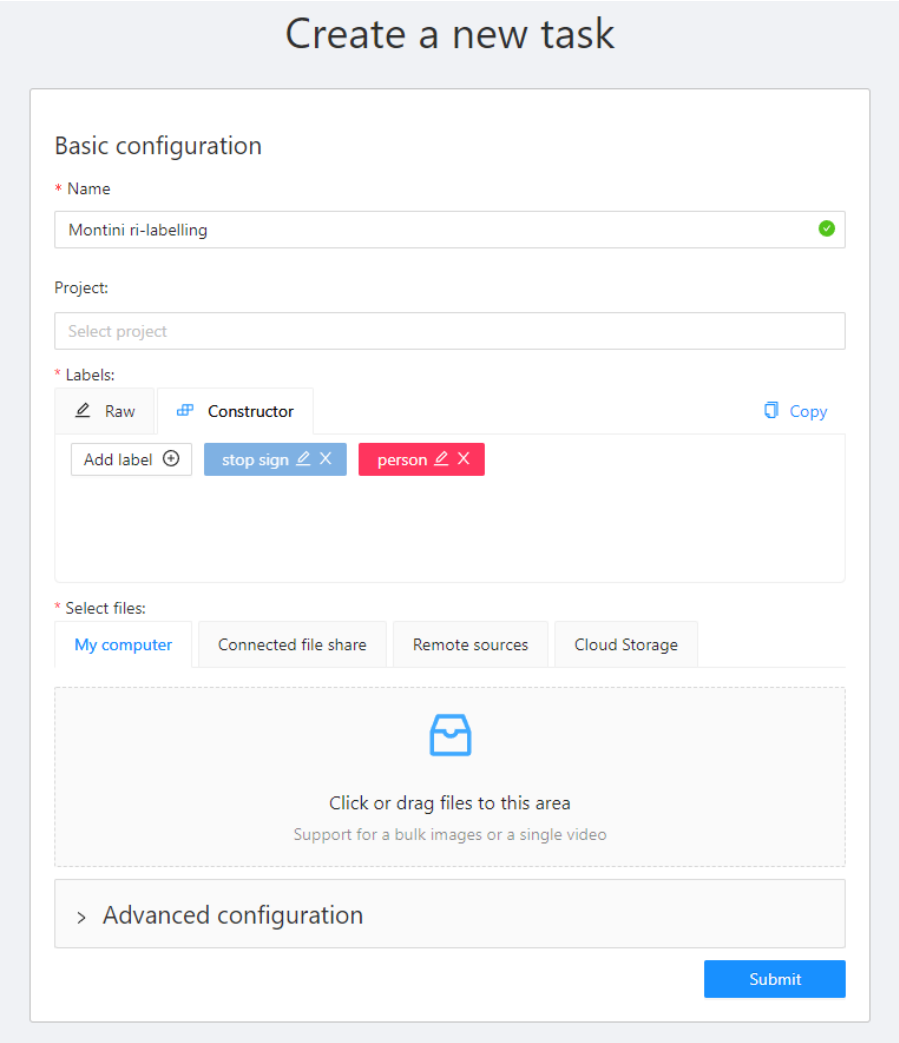

Figura 3.1: Finestra di creazione task.

L'utente deve scegliere un titolo per il task, assegnarlo ad un progetto, e scegliere le **label**. Le label rappresentano le classi che il task deve supportare. In questo caso vengono settate *person* e *stop sign*, e ad ognuna viene assegnato un colore. Esse possono essere create utilizzando un costruttore di base attraverso la casella *constructor*, oppure possono essere create manualmente tramite una lista di tipo json che si inserisce sulla casella *raw*. È bene controllare dopo la creazione attraverso il *constructor* che le label abbiano il giusto *Class ID* corrispondente a quello delle annotation già esistenti, che importeremo successivamente.

In seguito si procede con il caricamento delle immagini del dataset nella casella *select files*.

Infine cliccando sull'ultima sezione della finestra si possono configurare le impostazioni avanzate. In questo caso la maggior parte delle impostazioni avanzate non

sono d'interesse per il task da eseguire, tuttavia sulla casella *image quality* è meglio settare il valore 100% piuttosto che il valore di default per ogni progetto, 70%. Il dataset fornito ha già intrinsecamente una bassa qualità, e inoltre le immagini non compaiono in numero esageratamente elevato, pertanto non c'è motivo di effettuare una compressione.

Cliccando su *submit* il task verrà creato. A questo punto rimangono da importare le annotation. Partendo dalla pagina **Tasks**, che elenca tutti i task creati, cliccando sui tre puntini di fianco alla scritta *actions* compare un menu a tendina che elenca le possibile opzioni che si possono eseguire sul task.

Selezionando la voce *"Upload Annotations"* comparirà una lista di tutti i formati supportati per l'importazione di annotation; in questo caso quello corretto è il formato *YOLO*, quindi si procede con il caricamento dell'archivio contenente i file delle annotation creati con lo script Python.

3.5 Correzione delle annotation e classificazione degli scenari di interesse

Per effettuare il processo di etichettatura si clicca sul pulsante *open* a fianco al task corrispondente, e successivamente in fondo, sulla sezione **Jobs**, si accede alla schermata di lavoro cliccando sul link azzurro "**job #n**". La schermata di lavoro appare come in figura 3.2.

Tramite i pulsanti di navigazione nella parte superiore della finestra si può avanzare nello scorrimento delle immagini.

Le annotation già importate vengono visualizzate sulle immagini: l'obiettivo è quello di correggere quelle esistenti ed aggiungere quelle mancanti. Le annotation già esistenti possono essere interamente modificate cancellandole, spostandole, modificandone la grandezza o cambiandone la classe di appartenenza.

La fase di etichettatura è stata affiancata da due attività importanti: il monitoraggio dei tempi e la classificazione degli scenari di interesse.

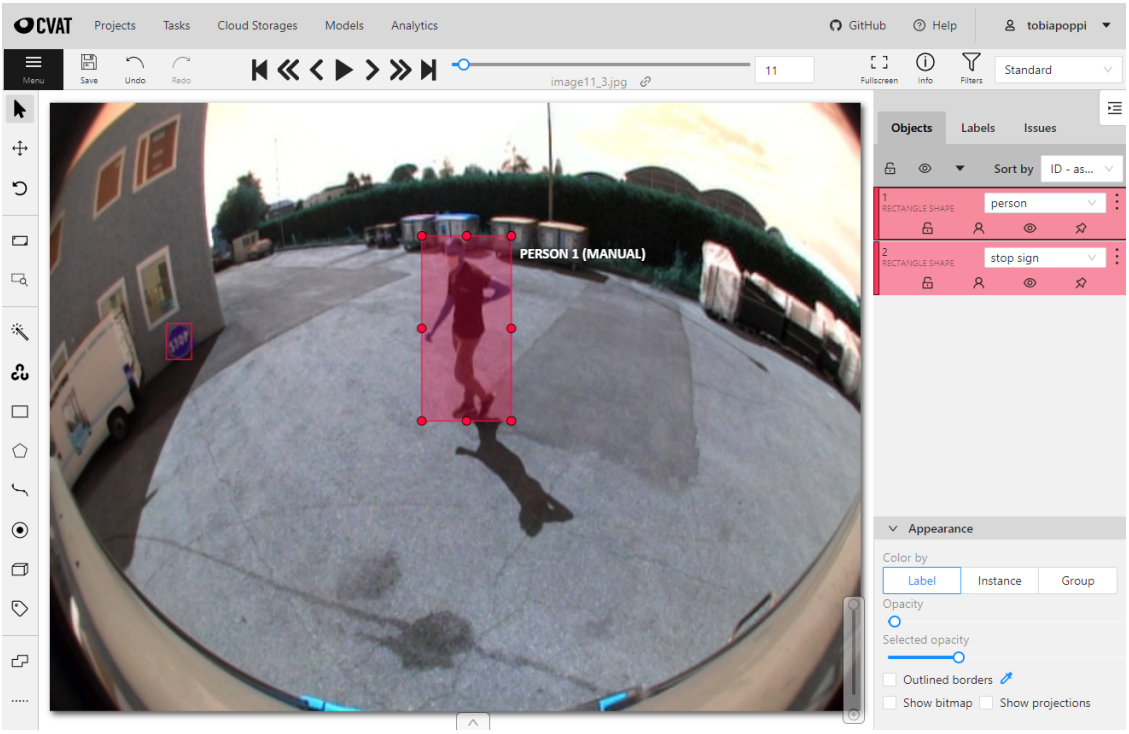

Figura 3.2: Schermata di lavoro principale.

3.5.1 Monitoraggio dei tempi

Il monitoraggio dei tempi consiste in un cronometraggio del tempo impiegato per correggere tutte le annotation di una singola immagine, per tutte le immagini del dataset.

Effettuare una misurazione dei tempi è importante per avere una media del tempo necessario per eseguire un task, oltre che per ottenere una stima di quanto tempo sarà necessario per effettuare l'etichettatura di un numero di immagini più consistente.

In aggiunta a questo dato, il tempo impiegato per eseguire la correzione delle annotation fornisce un feedback su quanto il modello utilizzato per etichettare è migliorato rispetto al training precedente.

	etichettatura correttiva (re-labelling)
n_immagini	271
tempo di etichettatura	3706 s
tempo di etichettatura	1 ora 1 minuto e 46 secondi
tempo di etichettatura medio	13.68 s/img

Tabella 3.2: Risultati del monitoraggio delle tempistiche dell'etichettatura dell'intero dataset (etichettatura correttiva).

Allo scopo di dimostrare una stima di quanto si velocizza il processo di etichettatura partendo da dati già annotati attraverso la rete YOLO rispetto a etichettare da zero il dataset manualmente, è stata eseguita una etichettatura interamente manuale per il 15% del dataset. Per distinguere i due task viene utilizzato il termine *etichettatura correttiva* o *re-labelling* per il primo e *etichettatura interamente manuale* per il secondo.

	etichettatura interamente manuale	**etichettatura correttiva (re-labelling)**
n_immagini	40	40
tempo di etichettatura	862 s	547 s
tempo di etichettatura	14 minuti e 22 secondi	9 minuti e 7 secondi
tempo di etichettatura medio	21.55 s/img	13.68 s/img

Tabella 3.3: La tabella mostra i risultati del cronometraggio dei tempi dell'etichettatura interamente manuale confrontati con i risultati dei tempi dell'etichettatura correttiva precedentemente eseguita.

I risultati dei tempi riportati in tabella 3.3 dimostrano che, etichettando il dataset in modalità correttiva (*re-labelling*) il tempo impiegato – preso in quantità proporzionata al numero di immagini utilizzate per l'etichettatura interamente manuale – corrisponde al 63.5% del tempo impiegato per l'etichettatura completamente manuale. Ciò equivale a velocizzare del 37% circa il task, con un risparmio di circa 8 secondi per ogni immagine etichettata.

3.5.2 Classificazione degli scenari di interesse

La classificazione degli scenari di interesse consiste nell'individuare e nel raggruppare le immagini che vengono etichettate in base ad uno scenario ricorrente che si vuole correggere durante il nuovo training della rete neurale.

Il motivo per cui si vogliono creare questi raggruppamenti di immagini è spiegato dal rischio di *overfitting* del modello di machine learning.

Intuitivamente sarebbe bene tornare a dare in input alla rete neurale tutte le etichettature corrette, anche quelle che hanno subito una minima correzione, in modo da addestrare la rete neurale nel migliore dei modi. Nella realtà però occorre prestare parecchia attenzione durante l'addestramento di un modello di machine learning al fenomeno di *overfitting*.

Un modello si dice che *overfitta* quando il suo *bias* è molto contenuto mentre la sua *varianza* è alta. Ciò significa che il modello ha interpretato in maniera troppo univoca i dati del training set, ottenendo così ottime performance utilizzando gli stessi dati del training, ma ottenendo performance molto scarse in fase di *testing* cioè utilizzando il modello su un altro set di dati.

Dunque se forniamo al modello tanti nuovi dati che contengono piccole correzioni di etichettature che già inizialmente erano sufficientemente corrette, otterremo un modello che avrà poca capacità di generalizzare su dati diversi da quelli forniti.

Le tipologie di scenari ritrovate in fase di etichettatura sono principalmente quattro.

- Tipo 1: Immagini che non hanno subito alcuna correzione di etichettatura.
- Tipo 2: Immagini che hanno subito minime correzioni di etichettatura (minimi spostamenti della *bounding box* e minimi aggiustamenti delle loro dimensioni).
- Tipo 3: Immagini in cui sono state aggiunte etichette su *stop sign* che non erano stati rilevati. (**falsi negativi**)
- Tipo 4: Immagini in cui sono state tolte etichette di tipo *person* che erano state posizionate su oggetti che non sono di tipo *person*. (**falsi positivi**)

Dunque durante l'etichettatura ogni immagine è stata ulteriormente classificata con un codice relativo alla tipologia di scenario incontrato.

3.6 Esportazione e splitting dei dati

Dopo la fase di etichettatura il dataset con le relative annotation è pronto per essere esportato, quindi accedendo sulla schermata *tasks* e cliccando sul pulsante *actions* a fianco al giusto task, comparirà un menu sul quale si possono selezionare due voci. La prima è **dump annotations** la quale esporta solamente le annotation del

dataset, utile nel caso in cui avessimo già scaricato i dati di input. La seconda è **export as a dataset** la quale effettuerà il download dell'intero dataset.

Entrambe le modalità di esportazione consentono di scegliere il formato dati con cui si vogliono salvare i dati, indipendentemente dal formato che si è utilizzato per l'importazione delle annotation.

Quindi il dataset è stato esportato in formato *YOLO*. Successivamente è avvenuto uno splitting del dataset.

Come spiegato nella sezione 3.5.2, il dataset da dare in input al nuovo training della rete dovrà contenere solamente alcuni dei dati, scartando quelli che contengono le correzioni meno significative.

Per questo motivo è stato eseguito uno splitting del dataset in base alla tipologia di scenario a cui le immagini fanno riferimento.

Le immagini con uno scenario di tipo 1 o 2 vengono scartate per il successivo training della rete, mentre quelle con uno scenario di tipo 3 e 4 (cioè i falsi negativi e i falsi positivi) vengono mantenute nel dataset finale.

Lo *splitting* del dataset è avvenuto tramite uno script Python che prende in input il dataset esportato da CVAT e il file contenente la classificazione delle immagini in base allo scenario.

Capitolo 4

Secondo caso d'uso: *Pose annotation* di oggetti per eseguire la *pose estimation* su un robot *pick and place*

4.1 Obiettivo

L'azienda **Cifarelli spa** è specializzata dal 1967 nella produzione di macchine professionali per l'agricoltura.

Nasce la collaborazione con il laboratorio HiPeRT Lab allo scopo di sviluppare una soluzione per un robot che esegue l'impacchettamento di componenti lungo la catena di assemblaggio.

Questa operazione avviene tramite una tipologia di robot chiamata *Pick and Place* data la loro funzione di raccogliere oggetti e posizionarli in un determinato spazio.

Durante la fase di raccoglimento è necessario fornire al robot la posizione in cui deve spostarsi per raccogliere l'oggetto. Questa operazione viene eseguita tramite un modello di *pose estimation* (es. *Efficient Pose* [22] o *Augmented Autoencoder* [18]).

Al fine di addestrare i modelli che eseguono la *pose estimation* servono dei dataset di immagini contenenti gli oggetti che si dovranno raccogliere e la loro *ground truth* (la posizione reale).

Per questa tipologia di task le tecniche che si possono utilizzare per annotare la posizione di oggetti sono diverse e più complesse rispetto al caso analizzato al capitolo 3. Per questo caso d'uso si è scelto di utilizzare uno dei software analizzati

al capitolo 2 per annotare le posizioni degli oggetti 3D sullo spazio delle immagini 2D: 6D-PAT (2.4).

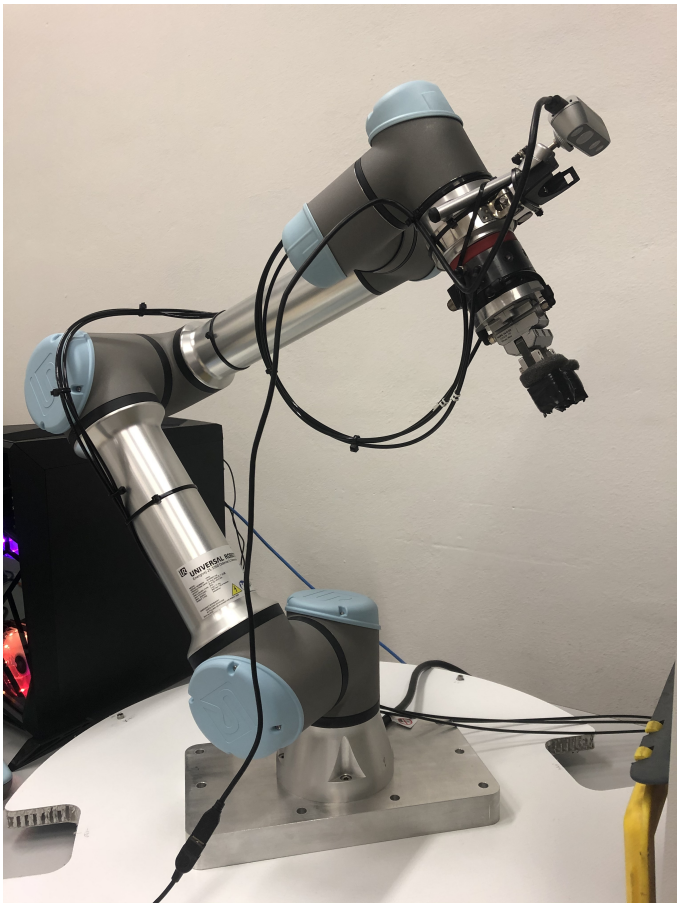

Figura 4.1: Robot *pick and place Universal Robots UR5e* utilizzato per l'attività di impacchettamento.

4.2 Acquisizione dei dati

Per acquisire i dati sono state scattate 100 fotografie con la camera *Realsense D435i*, che è la stessa che verrà utilizzata nella configurazione finale del robot. Le fotografie sono state scattate simulando il più possibile l'ambiente in cui si lavorerà, utilizzando quindi una delle box standard fornite da Cifarelli e ricreando la stessa luminosità attraverso lampade magnetiche posizionate all'interno della box.

All'interno della box sono stati inseriti svariati oggetti dello stesso tipo (in questo caso *chiavi candela*, si può osservare un esempio in figura 4.2). Per ottenere un buon numero di fotografie e garantire che ogni immagine abbia gli oggetti posizionati in modo diverso è stato utilizzato uno script Python che scatta in autonomia un

Figura 4.2: Chiave candela.

numero x di immagini dato un tempo t espresso in secondi che determina il tempo che trascorre tra uno scatto e il seguente. Adottare questa tecnica permette di accelerare i tempi di acquisizione del dataset automatizzando lo scatto delle foto e lasciando all'utente un tempo ragionevole per poter muovere gli oggetti tra uno scatto e l'altro.

Nel dataset è stata successivamente effettuata una selezione delle immagini in base alla loro utilizzabilità. Dopo aver cancellato le immagini che per errore di tempistiche raffiguravano le mani dell'operatore durante gli spostamenti dei pezzi e quelle che erano più sfocate, da 100 immagini scattate il dataset si è ridotto a 60 immagini.

Come risoluzione delle immagini è stata scelta la $640x480$ poiché lavorare con reti neurali su immagini pesanti comporterebbe un costo computazionale elevato, mentre a noi serve una risposta più o meno immediata, dopo lo scatto della foto, riguardo la posizione del pezzo. Il formato delle immagini è *jpg*, quindi è supportato dal software in quanto accetta in input formati *jpg* e *png*.

Figura 4.3: Posizione del robot durante lo scatto delle immagini e durante il calcolo della posizione.

4.3 Configurazione del software e preparazione dei dati necessari

Come prima operazione deve essere opportunamente configurato 6D-PAT e deve essere preparato il dataset in modo da renderlo comprensibile al software.

Una volta aperto il software, accedendo al menu *settings* la prima impostazione da inserire è, sotto alla tendina *Load Save*, la spunta su **Default JSON**. Selezionando questa modalità di elaborazione dati il software elaborerà due file *json* che

devono essere manualmente inseriti: *'camera_info.json'* e *'ground_truth.json'*.

4.3.1 Camera_info.json

Questo file *json* contiene la *camera matrix* della fotocamera utilizzata per scattare le foto, e il formato che deve avere è il seguente:

```
0 {
1     "image_filename.jpg/png": {
2         "K": [fx, 0, cx, 0, fy, cy, 0, 0, 1],
3         "nearPlane": 50,
4         "farPlane": 2000
5     }
6 }
```

Per ogni immagine presente nel dataset deve essere indicata la sua K (*camera matrix*). f_x, f_y, c_x e c_y rappresentano alcuni dei parametri intrinseci della camera elencati nella sezione 2.4.2 (lunghezze focali e centro ottico).

Questi parametri sono stati ottenuti tramite previa *calibrazione* della camera.

I due parametri *nearPlane* e *farPlane* sono opzionali, e se vengono omessi il software gli assegna di default i valori 50 e 2000. Questi due valori rappresentano il piano più vicino e il piano più lontano rappresentabili nello spazio immaginario 3D che si forma sull'immagine 2D.

Per dare quindi origine al file *json* ho implementato uno script Python che lo creasse automaticamente, assegnando la stessa K ad ogni immagine contenuta nel dataset.

Tuttavia non sempre si hanno a disposizione i parametri intrinseci della camera, soprattutto se si stanno effettuando prove e tentativi frettolosi. Per questo motivo ho creato una versione dello script Python che calcola una *camera matrix* approssimata a partire da soli due parametri: la fov_x e la fov_y. Questa soluzione è giusto adottarla solamente in fase di prova, successivamente è consigliabile utilizzare la corretta *camera matrix* della fotocamera utilizzata.

Il file deve essere salvato nella stessa directory in cui sono salvate le immagini del dataset.

4.3.2 Ground_truth.json

Questo è il file di output sul quale verranno salvate tutte le *pose* annotate sulle immagini. Se l'operazione da eseguire è quella di etichettare le posizioni sulle immagini da zero, allora è sufficiente creare un file *json* e salvarlo in una qualsiasi directory. Se invece serve importare sul software le *pose annotation* già esistenti per poi modificarle, allora vanno inserite su questo file con il formato seguente:

```
0 {
1     "image_filename.jpg/png": {
```

```
2           "R": [r11, r12, r13, r21, r22, r23, r31, r32, r33],
3           "t": [t1, t2, t3],
4           "id": pose_id,
5           "obj": obj_model_filename.ply
6       }
7   }
```

R indica la *rotation matrix* e t è il vettore di traslazione. L'ottenimento di questi due valori è spiegato nella sezione 2.4.3.

Id indica un identificatore univoco della *pose* e *obj* indica il nome del file contenente il modello 3D che rappresenta l'oggetto annotato.

4.3.3 Impostazione dei percorsi

A questo punto tutti i file necessari al software sono pronti ed è possibile configurare sul software i percorsi nei quali acquisire i dati.

Andando sul menu di *settings*, sotto la voce *Paths* configureremo in *'Images path'* il il percorso della directory contenente le immagini e il file *'camera_info.json'* (che verrà automaticamente individuato e letto dal software). Su *'Object models path'* settiamo il percorso della directory contenente i file dei modelli 3D. Infine su *'Poses file path'* settiamo il percorso del file *'ground_truth.json'*.

4.4 Etichettatura delle immagini

Il software è pronto per la fase di etichettatura. Sulla schermata principale è possibile selezionare sulla sinistra una delle immagini del dataset, e sulla destra uno dei modelli 3D di cui si vuole annotare la posizione.

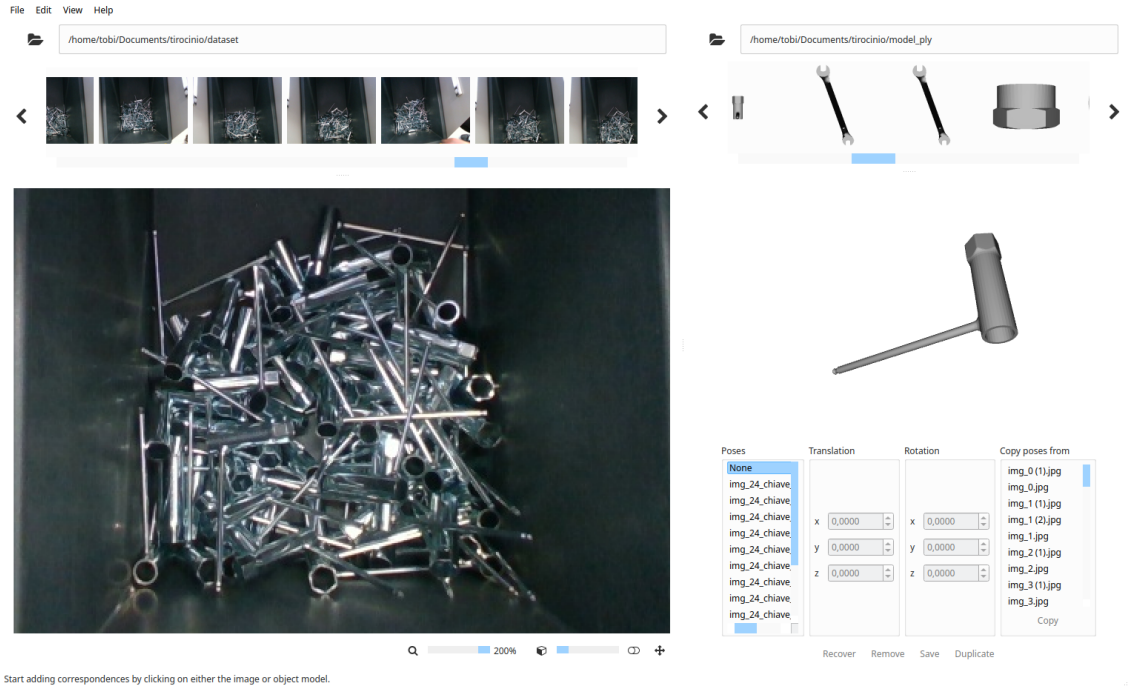

Figura 4.4: Schermata principale di lavoro di 6D-PAT.

A questo punto per iniziare l'annotation di una *pose* si clicca su un punto dell'oggetto 3D (dove comparirà un punto colorato) e successivamente sul punto corrispondente dell'oggetto sull'immagine 2D (dove comparirà un punto di colore uguale a quello del modello). Ripetendo questo procedimento per un totale di almeno sei coppie di punti si abiliterà il pulsante **recover** in basso a destra. Cliccandoci sopra il software eseguirà la funzione **SolvePnP** spiegata nella sezione 2.4.3, generando sull'immagine una copia del modello 3D collocata nella posizione stimata dalla funzione. Nel caso in cui l'oggetto non fosse posizionato in modo preciso è possibile traslarlo con il tasto sinistro del mouse e ruotarlo con il tasto destro del mouse. Un altro modo per cambiare la traslazione è modificare manualmente i valori delle coordinate x y e z della casella *Translation*.

Una volta inserite tutte le annotation necessarie su un'immagine cliccando il pulsante **save** in basso a destra, le informazioni di tutte le *pose* annotate verranno scritte sul file *'ground_truth.json'*.

Per ogni immagine sono state annotate in media nove posizioni di oggetti: quelli più in superficie, che sono più facili da raccogliere per il robot, e quelli con la posizione più semplice da interpretare nel caso in cui fossero intersecati da altri oggetti nell'immagine.

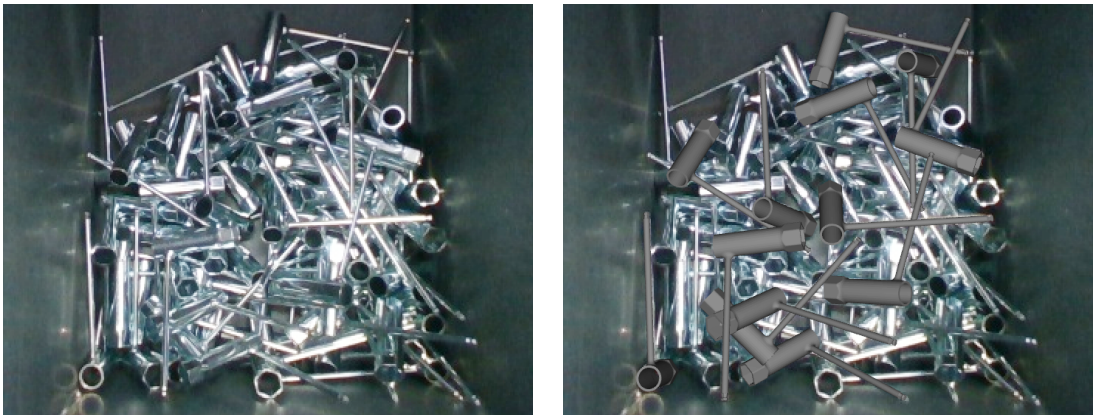

Figura 4.5: Un campione del dataset, sulla sinistra la foto originale e sulla destra la rappresentazione delle *pose* dopo essere state annotate.

4.4.1 Tempi di etichettatura

Anche in questo caso durante l'etichettatura delle posizioni sono stati monitorati i tempi impiegati, e sono riportati in tabella 4.1

TEMPI	
n_immagini	60
Tempo di etichettatura totale	36553 secondi
Tempo di etichettatura totale	10 ore 9 minuti e 13 secondi
Tempo di etichettatura medio per immagine	609 secondi
Tempo di etichettatura medio per immagine	10 minuti e 9 secondi
Tempo di etichettatura medio per annotation	79.39 secondi
Tempo di etichettatura medio per annotation	1 minuto e 19 secondi

Tabella 4.1: Tabella che riporta i tempi impiegati per l'annotation delle *pose* sulle immagini, riportando tempo totale, media per ogni immagine, e media per ogni annotation.

Questo metodo di etichettatura è particolarmente lento, e per posizionare le annotation in modo preciso, al contrario dell'inserimento di *bounding box*, è necessario un notevole sforzo da parte dell'utente.

4.5 Problemi incontrati

Durante il processo di etichettatura delle immagini è stato riscontrato un problema legato al posizionamento della *pose* sull'immagine. La *pose* dopo avere inserito le sei coppie di punti veniva collocata in un punto completamente diverso da quello voluto. Inizialmente si sospettava che il problema fosse legato alla matrice intrinseca utilizzata in modo sbagliato.

Successivamente, facendo svariati tentativi si è giunti alla conclusione che la funzione di calcolo della posizione (*SolvePnP*) ha difficoltà nell'analizzare punti troppo vicini tra di loro.

Eseguire la funzione partendo da punti appartenenti ad un'immagine $640x480$ comporta un calcolo sbagliato dei due vettori finali. La conferma è stata ricevuta dal fatto che il software etichettasse correttamente e con estrema precisione immagini con risoluzione doppia rispetto a quella del dataset.

Il dataset è stato quindi etichettato attraverso un flusso di lavoro differente rispetto a quello proposto dal software, tuttavia il procedimento non è sfociato nel richiedere un tempo maggiore di quello che sarebbe servito inizialmente.

Per ogni immagine sono state tracciate le sei coppie di punti in modo da ottenere la prima annotation. Successivamente questa è stata traslata e ruotata manualmente fino all'ottenimento della corretta posizione. Per tutte le annotation successive è bastato utilizzare il tasto *duplicate* in basso a destra. Questa funzione permette di duplicare una *pose* e successivamente traslarla nella sua corretta posizione, evitando così di dover inserire le coppie di punti.

Capitolo 5

Conclusioni

5.1 Primo caso d'uso

L'etichettatura delle immagini del primo caso d'uso è andata a buon fine, e non sono state incontrate particolari problematiche durante l'evoluzione del progetto.

Come spiegato precedentemente al capitolo 3 il progetto avrà un'evoluzione continua, con l'obiettivo di continuare a riaddestrare la rete di machine learning in modo da ottenere risultati sempre più accurati, aumentando col tempo la capacità di generalizzare del modello.

	Primo Training	**Training di Perfezionamento**
n^immagini training set	244	119
n^immagini validation set	27	14
precision	0.62	0.94
recall	0.59	1.00
F1-score	0.61	0.97
mAP con IoU thr 50%	69.29 %	100 %

Tabella 5.1: Risultati di *validation* della rete prima e dopo la fase di perfezionamento (*finetuning*).

Nella tabella 5.1 sono stati riportati i punteggi di *validation* effettuati dai due modelli di machine learning.

Il primo è quello ottenuto dalla prima fase di training, partendo da una rete YOLO standard, il secondo è quello ottenuto a seguito del training di perfezionamento, eseguito riaddestrando il modello iniziale con le immagini ottenute dal

re-labelling.

Dalla tabella risulta che il miglioramento è notevole e si riscontra su tutte le metriche riportate. In particolare, prendendo come punteggio delle performance riassuntivo la *mean average precision* calcolata con una *threshold* di IoU (*Intersection over Union*) del 50%, si osserva un aumento del 30% delle performance, raggiungendo il 100%.

Questi dati denotano senza dubbio un miglioramento del modello, tuttavia sono da interpretare con cautela a causa delle piccole dimensioni del *training-set* e del *validation-set* utilizzati per il perfezionamento.

5.2 Secondo caso d'uso

L'obiettivo del progetto, cioè arrivare ad avere un modello di *pose estimation* che permetta al robot di riconoscere le posizioni degli oggetti, è estremamente difficile da ottenere tramite etichettatura semi-automatica di immagini reali. Al fine di raggiungere un buon addestramento del modello è necessaria una grande quantità di dati etichettati.

I tempi necessari ad etichettare posizioni di oggetti attraverso questo software sono troppo elevati per poterne ottenere una quantità sufficiente.

Quindi al fine di ottenere i dati per addestrare le reti neurali si è deciso di utilizzare dataset sintetici, realizzati tramite simulatori che ricreassero attraverso modelli 3D foto simili a quelle che verranno scattate dal robot. Questo processo permette di conoscere in partenza il *ground truth* degli oggetti (la loro posizione reale), e di avere quindi le immagini già etichettate.

Addestrando inizialmente la rete tramite i dataset sintetici, sarà poi possibile importare sul *tool* 6D-PAT le *pose* predette dalla rete, ed effettuare il re-labelling, modificando la posizione e la rotazione degli oggetti laddove fossero stati collocati in modo non preciso.

6D-PAT garantisce un'elevata precisione delle annotation, quindi è ottimale utilizzare il software in questa modalità per ottenere un flusso di apprendimento incrementale continuo.

5.3 Sviluppi futuri

Tra gli sviluppi futuri sono compresi anche i continui miglioramenti dei due progetti già illustrati, tramite l'approccio di apprendimento incrementale.

È anche in programma lo sviluppo di un terzo caso d'uso che riguarda l'etichettatura video di veicoli, persone, biciclette, ecc... tramite *bounding box* tridimensionali (cuboidi) in scenari *automotive*.

Anche in questo caso, il proposito è quello di lavorare tramite il software CVAT, poiché ne è derivata un'esperienza estremamente positiva a seguito dell'etichettatura del dataset per il secondo caso d'uso (capitolo 3), e poiché è l'unico software tra quelli elencati che supporta l'etichettatura di cuboidi 3D su immagini 2D.

L'obiettivo è ancora una volta quello di avere uno strumento comodo per effettuare la ri-etichettatura di immagini già etichettate da reti neurali che necessitano di correzioni per riaddestrare la rete, continuando ad adottare un metodo di miglioramento incrementale che, con l'aumentare dei dati già etichettati, aumenti le performance della rete neurale insieme alla velocità di etichettatura dei dati non ancora etichettati.

Bibliografia

[1] *COCO Data Format*. https://cocodataset.org/#format-data.

[2] *Deep Extreme Cut*. https://github.com/openvinotoolkit/cvat/tree/develop/serverless/openvino/dextr/nuclio.

[3] Dan Deng et al. *PixelLink: Detecting Scene Text via Instance Segmentation*. https://arxiv.org/pdf/1801.01315.pdf. 2018.

[4] *f-BRS*. https://github.com/openvinotoolkit/cvat/blob/develop/serverless/pytorch/saic-vul/fbrs/nuclio.

[5] *Faster RCNN*. https://github.com/openvinotoolkit/cvat/tree/develop/serverless/openvino/omz/public/faster_rcnn_inception_v2_coco/nuclio.

[6] *Inside-Outside Guidance*. https://github.com/openvinotoolkit/cvat/blob/develop/serverless/pytorch/shiyinzhang/iog/nuclio.

[7] Jan Markan. *Data annotation formats used in object detection*. https://janmarcan.com/articles/data-annotation-formats-in-object-detection/. 2020.

[8] *Mask RCNN*. https://github.com/openvinotoolkit/cvat/blob/develop/serverless/openvino/omz/public/mask_rcnn_inception_resnet_v2_atrous_coco/nuclio.

[9] *Object reidentification*. https://github.com/openvinotoolkit/cvat/blob/develop/serverless/openvino/omz/intel/person-reidentification-retail-300/nuclio.

[10] OpenVINO. *CVAT: Powerful and efficient Computer Vision Annotation Tool*. Open source software available from https://github.com/openvinotoolkit/cvat. 2020-2021. URL: https://github.com/openvinotoolkit/cvat.

[11] *Pascal VOC*. http://host.robots.ox.ac.uk/pascal/VOC/.

[12] *RetinaNet*. https://github.com/openvinotoolkit/cvat/blob/develop/serverless/pytorch/facebookresearch/detectron2/retinanet/nuclio.

[13] *Rodrigues*. https://docs.opencv.org/3.4.15/d9/d0c/group__calib3d.html#ga61585db663d9da06b68e70cfbf6a1eac.

[14] *Semantic segmentation for ADAS*. https://github.com/openvinotoolkit/cvat/blob/develop/serverless/openvino/omz/intel/semantic-segmentation-adas-0001/nuclio.

[15] *SiamMask*. https://github.com/openvinotoolkit/cvat/blob/develop/serverless/pytorch/foolwood/siammask/nuclio.

[16] Piotr Skalski. *Make Sense*. https://github.com/SkalskiP/make-sense/. 2019.

[17] *SolvePnP*. https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga549c2075fac14829ff4a58bc931c033d.

[18] Martin Sundermeyer. *Augmented Autoencoders: Implicit 3D Orientation Learning for 6D Object Detection*. https://arxiv.org/pdf/1902.01275.pdf.

[19] *Text Detection v4*. https://docs.openvinotoolkit.org/2020.1/_models_intel_text_detection_0004_description_text_detection_0004.html.

[20] Maxim Tkachenko et al. *Label Studio: Data labeling software*. Open source software available from https://github.com/heartexlabs/label-studio. 2020-2021. URL: https://github.com/heartexlabs/label-studio.

[21] *YOLO v3*. https://github.com/openvinotoolkit/cvat/blob/develop/serverless/openvino/omz/public/yolo-v3-tf/nuclio.

[22] Wenqiang Zhang. *EfficientPose: Efficient Human Pose Estimation with Neural Architecture Search*. https://arxiv.org/pdf/2012.07086.pdf.