Relazione progetto di Machine Learning, relativa all'esame del 20/07/2023.
Emanuele Cannizzo.

## Problema

Il problema affrontato nell'ambito di questa ricerca consiste nella rilevazione degli equipaggiamenti di sicurezza in contesti lavorativi reali. L'obiettivo principale è individuare e classificare in modo accurato gli equipaggiamenti di sicurezza, quali caschi, giubbotti riflettenti e altri dispositivi di protezione individuale. Il contesto applicativo si riferisce al settore della sicurezza sul lavoro, in cui è fondamentale determinare se gli operatori indossano correttamente l'equipaggiamento di sicurezza al fine di prevenire incidenti sul luogo di lavoro.
La principale sfida affrontata riguarda la carenza di dataset annotati specificamente per questo tipo di problema. Pertanto, si è optato per un dataset di immagini generato virtualmente tramite l'utilizzo del simulatore Gta5 e successivamente etichettato automaticamente.
Si dimostra come l'impiego di un dataset virtuale, combinato con un piccolo dataset reale, possa migliorare le prestazioni del modello di detection addestrato esclusivamente sulle immagini reali.
Questo progetto è basato sull'articolo "Learning Safety Equipment Detection using Virtual Worlds" (https://aimh.isti.cnr.it/vw-ppe/), nel quale viene alleanata una rete YOLO per rispondere a questi problemi. Perciò nel corso di questo progetto dove possibile si cercherà di fare un parallelo tra il modello proposto nell'articolo e quello sviluppato qui.

## Dataset

Sono stati usati due dataset: uno generato virtualmente e l'altro composto da immagini del mondo reale. I dataset originali sono disponibili allo stesso link dell'articolo riferimento: https://aimh.isti.cnr.it/vw-ppe/.
Per comodità di utilizzo sono state create due copie di essi, aventi lo stesso contenuto ma con una struttura interna differente.

### Dataset Virtuale

Il dataset, scaricabile dal link ShuffledVirtualDataset è composto da 20 cartelle zip numerate da 0 a 20, e da due file zip chiamati "train.virtual.zip" e "valid.virtual.zip". Questi ultimi contengono due file di testo che elencano le immagini di training e di validazione rispettivamente, specificando la cartella in cui si trovano e il loro nome. Le liste contenenti I path delle immagini sono costruite in modo tale che immagini di una stessa cartella siano in posizioni consecutive nella lista. Ad ogni immagine è associato un file txt con stesso path e stesso nome, contente I target e le bouding boxes presenti nell'immagine. Nel dettaglio ogni riga del file rappresenta un target e le sue 4 coordinate relative al

bounding box. Il target è un numero compreso tra 0 e 6. Le bounding boxes hanno il seguente formato: x_center, y_center, width e height. Sono valori compresi tra 0 e 1 in proporzione all'immagine.

Il dataset originale era contenuto in una singola cartella zip di circa 131 GB, rendendo il suo utilizzo complicato in situazione con risorse limitate. Perciò, per semplificare l'utilizzo, il dataset è stato suddiviso in 20 cartelle più piccole, da circa 6GB ciascuna, in modo da scaricare solo la cartella necessaria e liberare spazio In memoria.
Questo utilizzo è stato ideato principalmente per l'allenamento su due servizi di notebook in Cloud: Google Colab il quale ogni volta la runtime venga disconnessa elimina tutti I dati salvati, e Amazon Sagemaker Studio Lab il quale offre una memoria di soli 25 GB.
Inoltre le immagine sono state inserite in ogni cartella in modo randomico, perciò non è necessario effettuare altri shuffle.

Per generare il dataset originale (in riferimento a quello dell'articolo) è stato utilizzato il Rockstar Advanced Game Engine (RAGE) per generare degli script C# che generassero gli ambienti di simulazione all'interno di Gta5. In particolar modo il RAGE engine ha permesso di configuare le seguenti opzioni:

- Camera: fissa il punto di vista dal quale guardare la scena;
- Pedoni: è possibile inserire pedoni con o senza equipaggiamenti di sicurezza, e scegliere anche le azioni da effettuare (parlare, camminare, combattere, …);
- Luogo: è possibile scegliere in quale mappa del gioco costruire la scena;
- Ora del giorno;
- Condizioni meteorologiche.

Per il dataset di train sono state ottenute un totale di 126900 immagini, generate in 9 differenti mappe del gioco, ognuna delle quali con 3 differenti condizioni meteorologiche, e con una media di 12 persone per immagine.
Il dataset di validazione è stato estratto da una sola mappa, con 3 condizioni meteorologiche, e consiste di 353 immagini sempre con una media di 12 persone per immagine.


Ear Protection
Gilet
Gilet


Person
Person
No Gilet
Person
Person
Person


Protection
Ear Protection
Gilet


Ear Protection
Gilet
Ear Protection
Gilet

## Dataset Reale:

Il dataset, ottenbile al seguente link: RealDatasetZip, ha un struttura simile a quella del dataset virtuale. Contiene due cartelle zip: train.zip e valid.zip, contenenti le immagini di training e di validazione. Le due liste, contenenti le informazioni riguardo al path delle immagini sono contenute in due zip chiamate “train.real.zip” e “valid.real.zip”. I dettagli sulla codifica dei target e delle bounding boxes sono identici a quelli del dataset virtuale.

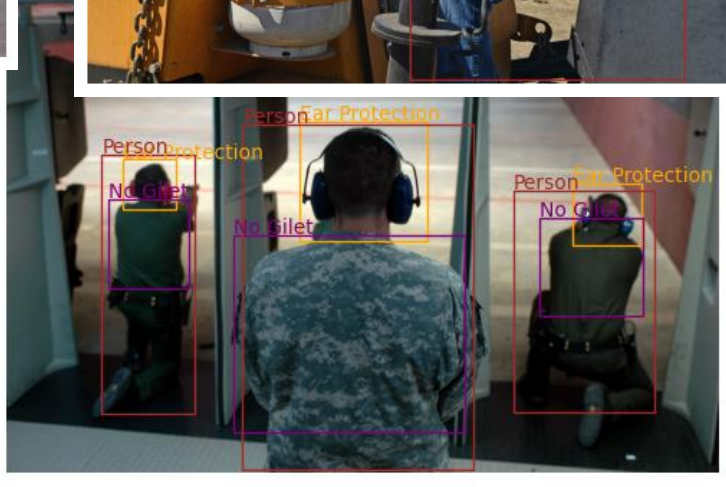
Il dataset reale è composto da un totale di 219 immagini, di cui 110 utilizzate per il training, e 109 utilizzate per la validazione. Le classi sono state etichettate manualmente, insieme alle bounding boxes.

Le categorie utilizzate in entrambi I dataset sono le seguenti:

- Head: la testa di una persona che non indossa il casco protettivo;
- Helmet: una testa che indossa un casco protettivo;
- Welding Mask: una testa che indossa una maschera per saldature;
- Ear Protection: una testa che indossa delle cuffie anti-rumore;
- Chest: il torace di una persona senza l’HVV;
- High Visibility Vest (HVV): un torace con indosso un gilet ad alta visibilità;
- Person: il corpo intero di una persona.

Si noti che nei file contenenti le informazioni sui target, le classi sono numerate da 0 a 6. Invece il modello Faster-RCNN richiede numeri a partire da 1. A causa di questo errore di valutazione la classe 0 (Head) è stata invalidata. Pertanto nelle metriche che seguiranno la classe Head avrà sempre un punteggio 0.

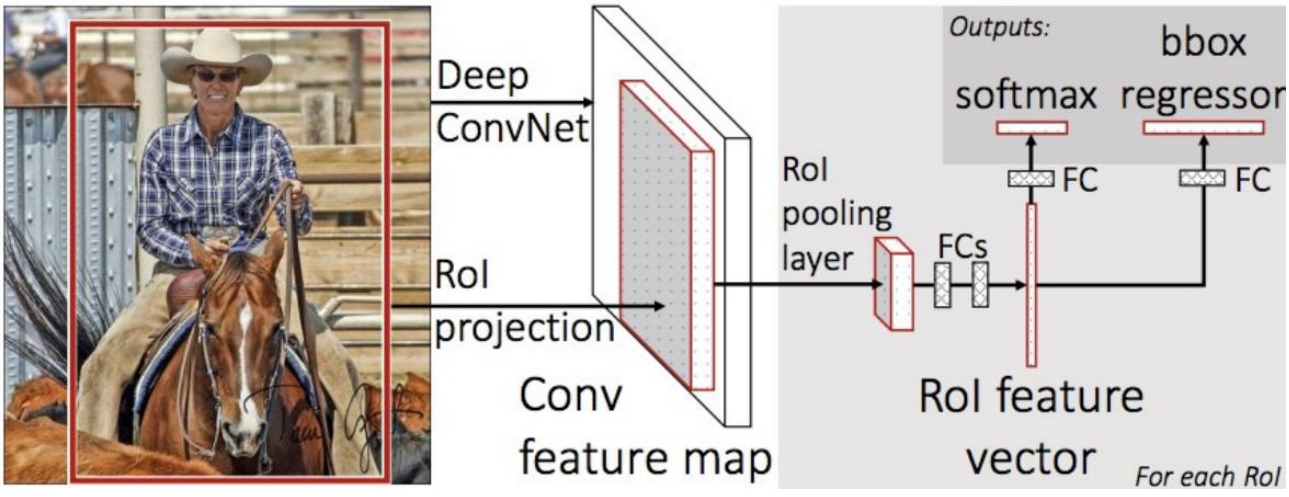




## Metodo:

Il modello utilizzato per la detection si basa sull'architettura Faster R-CNN (Region Convolutional Neural Network), che rappresenta lo stato dell'arte per la precisione delle predizioni. Questo modello è costituito principalmente da tre parti:

- Backbone: una rete convoluzionale usata per estrarre le features delle immagini di input;
- Region Proposal Network: la RPN è una rete neurale convoluzionale che prende in input le features estratte dal backbone e applica un algoritmo di region proposal. Questo algoritmo genera delle proposte di regioni, che sono potenziali regioni dell'immagine che potrebbero contenere oggetti di interesse. Ogni proposta di regione è associata a un punteggio di confidenza che indica la probabilità che la regione contenga un oggetto;
- Classificatore: il classificatore prende in input le regioni proposte dalla RPN e le utilizza per predire la classe dell'oggetto (softmax) e la bounding box associata (regressor).



Per sfruttare il transfer learning, è stato utilizzato un modello Faster R-CNN preaddestrato sul dataset Coco. Questa tecnica consente di ridurre il carico computazionale, poiché i pesi preaddestrati sul vasto e diversificato dataset Coco sono già in grado di ottenere ottime prestazioni di classificazione. Ci si aspetta che i pesi ottimali per il nuovo dataset non si discostino significativamente da quelli originali.
E' stato quindi eseguito un processo di fine-tuning del modello preaddestrato sul dataset Coco utilizzando il dataset virtuale. Questo processo ha consentito al modello di adattarsi alle specificità del nuovo dataset virtuale.
Secondo questa filosofia, è stato effettuato un ulteriore fine-tuning del modello ottenuto dal dataset virtuale utilizzando il dataset reale.
Questo approccio ha permesso di colmare la mancanza di immagini provenienti da contesti lavorativi del mondo reale, partendo da un modello che già possedeva una certa conoscenza delle classi da predire, tenendo in considerazione le differenze di rendering tra un mondo generato virtualmente e uno reale.
Infine, per avere un punto di riferimento, è stato allenato il modello preaddestrato sul dataset Coco utilizzando il solo dataset di addestramento reale.

## Valutazione

Durante la fase di valutazione, la metrica principale utlizzata è stata la mean Average Precision (mAP), che ha permesso di valutare l'efficacia del modello nella rilevazione degli oggetti di interesse. Inoltre, sono state calcolate le AP per le singole classi presenti nel dataset.
La mAP è una metrica comunemente utilizzata nella letteratura sull'object detection. Per calcolarla, sono considerate tre misure chiave: Intersection over Union (IoU), Precision (Pr) e Recall (Rc).

- Intersection over Union (IoU): viene calcolata come il rapporto tra l'area di intersezione e l'area di unione tra il bounding box reale e quello predetto. Una IoU più alta indica una migliore sovrapposizione tra il bounding box previsto e quello reale.
- Precision (Pr): è una misura di quanto sono accurate le previsioni del modello, ossia la percentuale di predizioni corrette. Viene calcolata come il rapporto tra i TP e la somma dei TP e dei falsi positivi (FP). Una Precision più alta indica meno falsi positivi.
- Recall (Rc): è una misura di quanto bene il modello individua tutte le istanze positive, includendo sia i veri positivi (TP) che i falsi negativi (FN). Viene calcolata come il rapporto tra i TP e la somma dei TP e dei falsi negativi (FN). La Recall indica la capacità del modello di trovare tutte le istanze rilevanti. Una Recall più alta indica meno falsi negativi.

Nella definizione classica di Average Precision (AP) si sceglie una soglia di IoU che serve per scartare dalla valutazione tutte le bounding boxes che non raggiungono tale soglia. Quindi si genera la curva precision-recall, e si ottiene l'AP come area sottesa. Nel nostro caso è stata impiegata la definizione di Average Precision definita dal dataset Coco, la quale prevede di calcolare l'AP nel modo usuale per 10 livelli di IoU compresi tra 0.5 e 0.95 con step di 0.05 per poi farne la media. Infine la mAP è stata ottenuta come media della AP tra le varie classi.
Infine, per avere un metro di paragone tra I risultati ottenuti qui e quelli ottenuti dall'articolo di riferimento, verrà anche usata come metrica la classica AP, definita come area sottesa alla curva precision-recall ad un IoU di 0.5.

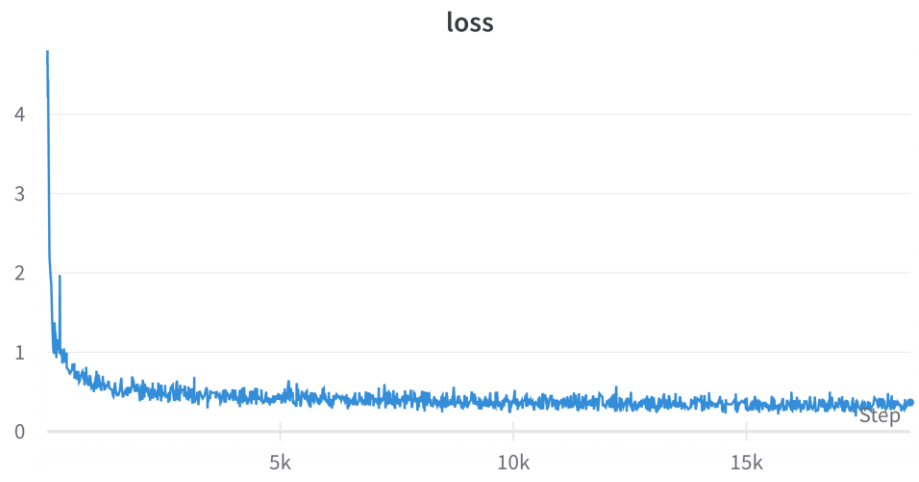
## Esperimenti

Sono stati generati 3 modelli:

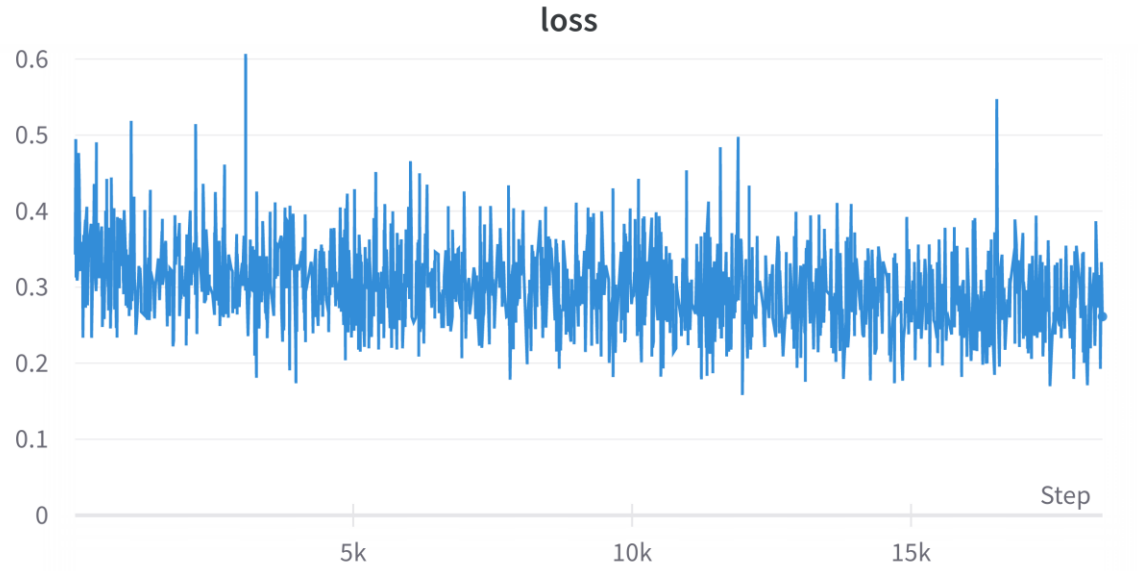
- FCV (Faster Coco Virtual): il modello Faster-RCNN preaddestrato su Coco e finetuned sul dataset Virtuale;
- FCR (Faster Coco Real): il modello Faster-RCNN preaddestrato su Coco e finetuned sul dataset Reale;
- FCVR (Faster Coco Virtual Real): il modello Faster-RCNN preaddestrato su Coco, finetuned prima sul dataset Virtuale e poi su quello Reale.

Tutti I modelli sono stati allenati su Cloud, nello specifico su Google Colab e su Amazon Sagemaker Studio Lab. Entrambi I servizi offrono una Gpu Nvidia Tesla T4.
Il modello FCV, ha richiesto una media di 8 ore per epoch, a causa della vastità del dataset di training.
Considerando tale modello testato sul daset di validazione reale, si nota dalla tabella 1 come la mAP cresca lentamente, raggiungendo il valore massimo di 0.643 all'epoch 6, che forse potrebbe essere stato superato se l'allenamento fosse continuato. Sotto sono raffigurate le loss delle prime due epoch. Si vede che dopo la prima epoca, l'algoritmo converge molto lentamente, e con oscillazioni considerevoli.

Epoch 0.



Epoch 1.




Consideriamo ora sempre il modello FCV valutato invece sul dataset di validazione reale, I cui risultati sono visibili nella tabella 2.
Le prestazioni incominciano a calare dopo la seconda epoca a causa dell'overfitting sul mondo virtuale. Più il modello viene allenato più I suoi parametri vanno ad ottimizzarsi per il mondo virtuale, evidenziando sempre di più le differenze con il mondo reale.
In particolare le classi dove il modello ha maggiori difficoltà sono: Welding Mask, Ear Protection e Chest. Questo potrebbe essere dovuto al fatto che nel mondo reale ci siano molte più varianti di questi oggetti rispetto a quelli che il simulatore include.
Pertanto, i parametri ottenuti dalla seconda epoch sono stati utilizzati come punto di partenza per costruire il modello FCVR. Qui bastano 20 epoch per ottenere prestazioni migliori rispetto al modello FCR, passando da una mAP di 0.35 (tabella 3) ad una di 0.422 (tabella 4).

Tabella 1: FCV sul dataset di validazione Virtuale

| Epochs | Head | Elmet | Welding Mask | Ear Protection | No Gilet | Gilet | Person | mAP |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0.67 | 0.52 | 0.70 | 0.72 | 0.75 | 0.72 | 0.587 |
| 2 | 0 | 0.73 | 0.63 | 0.75 | 0.74 | 0.78 | 0.75 | 0.626 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 3 | 0 | 0.73 | 0.63 | 0.76 | 0.73 | 0.77 | 0.75 | 0.624 |
| 4 | 0 | 0.73 | 0.66 | 0.77 | 0.78 | 0.78 | 0.76 | 0.639 |
| 5 | 0 | 0.71 | 0.62 | 0.77 | 0.76 | 0.78 | 0.76 | 0.629 |
| 6 | 0 | 0.73 | 0.67 | 0.77 | 0.77 | 0.79 | 0.77 | 0.643 |
| 7 | 0 | 0.72 | 0.63 | 0.78 | 0.76 | 0.78 | 0.75 | 0.637 |

Tabella 2: FCV sul dataset di validazione Reale

| Epochs | Head | Elmet | Welding Mask | Ear Protection | No Gilet | Gilet | Person | mAP |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0.21 | 0.07 | 0.09 | 0.11 | 0.19 | 0.24 | 0.131 |
| 2 | 0 | 0.2 | 0.07 | 0.11 | 0.12 | 0.2 | 0.24 | 0.132 |
| 3 | 0 | 0.18 | 0.07 | 0.11 | 0.09 | 0.18 | 0.21 | 0.119 |
| 4 | 0 | 0.23 | 0.04 | 0.12 | 0.1 | 0.18 | 0.2 | 0.125 |

Tabella 3: FCR sul dataset di validazione Reale

| Epochs | Head | Elmet | Welding Mask | Ear Protection | No Gilet | Gilet | Person | mAP |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0.01 | 0.00 | 0.01 | 0.01 | 0 | 0.05 | 0.012 |
| 5 | 0 | 0.37 | 0.19 | 0.16 | 0.23 | 0.3 | 0.42 | 0.238 |
| 15 | 0 | 0.44 | 0.33 | 0.37 | 0.31 | 0.46 | 0.51 | 0.347 |
| 39 | 0 | 0.48 | 0.32 | 0.39 | 0.33 | 0.44 | 0.52 | 0.354 |

Tabella 4: FCVR sul dataset di validazione Virtuale

| Epochs | Head | Elmet | Welding Mask | Ear Protection | No Gilet | Gilet | Person | mAP |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0.49 | 0.25 | 0.28 | 0.28 | 0.43 | 0.48 | 0.317 |
| 3 | 0 | 0.54 | 0.4 | 0.45 | 0.38 | 0.47 | 0.53 | 0.395 |
| 18 | 0 | 0.55 | 0.46 | 0.49 | 0.42 | 0.5 | 0.54 | 0.422 |

Infine ecco un rapido confronto con le prestazioni ottenute dal modello gemello costruito con YOLO. Anche con yolo sono stati generati I tre modelli: YCV, YCR, YCVR.

Nella tabella 5 si vede che il modello Yolo YCV è superiore. Ma questo potrebbe essere dovuto al fatto che esso è stato allenato il doppio delle epoch rispetto al modello FCV.

Invece nella tabella 6 è possibile riscontare la migliore accuratezza dei modelli o allenato solo sul dataset virtuale o finetuned su esso. Questo è concorde con il fatto che solitamente tra I due modelli, Yolo è il più veloce, ma Faster-RCNN è il più accurato.

Tabella 5: Daset di Validazione Virtuale

| Model | Head | Elmet | Welding Mask | Ear Protection | No Gilet | Gilet | Person | mAP |
|---|---|---|---|---|---|---|---|---|
| YCV | 0.897 | 0.867 | 0.755 | 0.890 | 0.897 | 0.900 | 0.897 | 0.872 |
| FCV | 0 | 0.727 | 0.671 | 0.774 | 0.771 | 0.789 | 0.769 | 0.821 |

Tabella 6: Dataset di Validazione Reale

| Model | Head | Elmet | Welding Mask | Ear Protection | No Gilet | Gilet | Person | mAP |
|---|---|---|---|---|---|---|---|---|
| YCVR | 0.788 | 0.733 | 0.663 | 0.740 | 0.747 | 0.786 | 0.871 | 0.761 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| FCVR | 0 | 0.818 | 0.762 | 0.766 | 0.793 | 0.853 | 0.905 | 0.700 |
| YCR | 0.441 | 0.522 | 0.423 | 0.620 | 0.591 | 0.607 | 0.806 | 0.573 |
| FCR | 0 | 0.743 | 0.602 | 0.598 | 0.689 | 0.857 | 0.886 | 0.625 |

## Demo

Per utilizzare la demo è sufficente scaricare il pacchetto python “streamlit” ed eseguire il comando “streamlit run streamlit.py”. Verrà istanziato un server locale sul quale è possibile interagire con il modello traminte interfaccia web. L’interfaccia è molto semplice:

- E’ possibile selezionare il modello da utilizzare;
- E’ possibile selezionare quale dataset utilizzare per il testing;
- E’ possibile selezionare l’immagine sulla quale testare il modello;

Una volta eseguiti questi passaggi compariranno le immagini di ground-truth e di predizione. Inoltre è possibile selezionare quali classi mostrare (utile nel caso di immagini sovra-affollate) e il treshold di confidenza per le predizioni.
Purtroppo dalla demo non è possibile scaricare il modello, questo perché il download tramite wandb non è affidabile su streamlit. Perciò è necessario scaricare il modello tramite il codice di esempio definito nel main.
Invece I dataset vengono scaricati senza problemi. Da notare che quando l’applicazione viene interrotta il download continua nel background. Una volta selezionato il daset conviene non selezionare un altro dataset finquando il primo dataset non è stato completamente scaricato.

## Codice

Dataset.py: qui è definita la struttura del dataset.

- La __init__ vuole due parametri obbligatori: il path in cui è contenuto il dataset, e il nome del file che contiene la lista con le immagini. Questo file deve trovarsi all’interno del path definito nel dataset, e inoltre il path delle singole immagini definite all’interno del file, deve essere relativo al file di cui fanno parte. I due dataset usati rispettano tale vincolo.
- Quando un immagine viene richiesta, essa è letta da file insieme al suo target. Qui viene fatta la conversione del formato delle bounding boxes dal tipo del dataset a quello supportata da Faster-RCNN. Inoltre se un immagine, o il suo relativo target fossero corrotti, viene eliminata dal dataset e viene restituita l’immagine successiva.
- Settando un flag è possibile scaricare automaticamente il dataset. Ogni volta che un immagine viene richiesta, se non è presente nel local storage, viene scaricata la cartella che la contiene.

- E’ presente anche un method utile per il debugging chiamato show_bounding(index), il quale crea un plot dell’immagine con le relative bounding boxes e predizioni.
-

Model.py: qui sono definite tutte le utils utili per lavorare con il modello (Faster-RCNN).

- Nella __init__ viene definito il modello, e vengono create tutte le cartelle utili al logging dei parametri.
- E’ stato fatto ampio uso di Wandb, un sito che permette di fare il logging delle metriche di qualsiasi modello di Machine Learning. E’ stato utilizzato per salvare I parametri del modello, per scaricarli quando necessario e per loggare la loss di training. Tutti I parametri da configurare sono spiegati nel dettaglio all’interno di main.py.
- Il metodo di train prende spunto da una repo ufficiale di pytorch: https://github.com/pytorch/vision/blob/main/references/detection/engine.py . Una volta chiamata la __init__ è possibile usare train() senza inserire altri parametri. E’ stata gestita tutta la conversione del modello e delle immagini al device appropriato nel caso fosse possibile utilizzare cuda.
- Vi è la possibilità di fare il resuming di un training interrotto: I parametri necessari sono definiti nell’__init__ e anch’essi ampiamente documentati nei codici.
- Anche il metodo evaluation() è una versione modificata di quello reperibie al link git poco sopra. Esso vuole come unico argomento il dataloader. E’ stato modificato in modo tale da includere la possibilità di andare a calcolare le metriche relative alle singole classi, oltre alla metricha che considerava la media.
- Il metodo apply_object_detection permette di testare il modello su di un immagine, passata come tensore. Genera quindi un plot con le labels e le bounding boxes delle predizioni. E’ possibile settare un treshold di confidenza e quali classi visualizzare.

Main.py: qui è definito ogni possibile utilizzato del codice. Si mostra come creare un dataset, testarlo, fare il training e l’evaluation.

## Conclusione:

In questa ricerca è stato affrontato il problema della detection degli equipaggiamenti di sicurezza in situazioni lavorative reali utilizzando un modello di machine learning basato sull'architettura Faster R-CNN. Sono stati esplorati diversi metodi per superare la carenza di dataset etichettati, tra cui l'utilizzo di un dataset virtuale generato tramite il simulatore di gioco GTA5 e il transfer learning da un modello preaddestrato sul dataset Coco.

Attraverso il processo di fine-tuning su un dataset virtuale e successivamente su un dataset reale, il modello ha dimostrato buone prestazioni nella rilevazione e classificazione degli equipaggiamenti di sicurezza in situazioni lavorative reali.

Queste potrebbero essere alcune delle azione che si potrebbero intraprendere per migliorare le performance:

- Raccolta di dati reali: Per migliorare ulteriormente il modello, potrebbe essere opportuno ampliare il dataset reale utilizzato per il fine-tuning. Questo consentirebbe di acquisire una maggiore varietà di contesti lavorativi reali e migliorare la capacità del modello di generalizzare su nuove situazioni.
- Data augmentation: Usare tecniche di data augmentation potrebbe migliorare la robustezza del modello e la sua capacità di generalizzazione. Ad esempio, l'applicazione di trasformazioni geometriche più complesse e la generazione di nuovi esempi sintetici potrebbero aumentare la varietà
  d
  e
  i

  d
  a
  t
  i

  d
  i

  t
  r
  a
  i
  n
  i
  n