Università degli Studi di Catania
Corso di Laurea Magistrale in Informatica

Mario Benissimo
Rosario Cannavo
Riccardo Raciti

Coin Detection whit FASTERRCNN_RESNET50_FPN

Machine Learning

Prof. Giovanni Maria Farinella
Prof. Francesco Ragusa

Anno Accademico 2022–2023

Indice

1 Introduzione

Il problema affrontato nel progetto proposto è quello dell'object detection applicata alle monete. Le monete sono oggetti comuni nella vita quotidiana e possono presentare una varietà di caratteristiche, come dimensioni, forme, colori e disegni. Una certa categoria di persone potrebbe riscontrare difficoltà nell'utilizzo di tali oggetti a causa di disabilità visive. Lo scopo finale è quello di riuscire a classificare ogni moneta correttamente in modo da fornire un supporto a questa categoria di persone: dopo una corretta classificazione la persona non vedente o con difficoltà potrebbe ad esempio ricevere come una comunicazione sonora la classe di appartenenza delle monete che sono state rilevate. Altri utilizzi si potrebbero riscontrare nella velocizzazione dei pagamenti attraverso le casse automatiche e in tutti quei task che richiedono il conteggio manuale di tali oggetti.

Uno scenario comune è quello in cui più monete di tagli diversi sono disposte su una superficie come un tavolo o il rullo di un registratore di cassa e si vuole conoscere il valore totale o quello specifico di ogni moneta in modo da poter compiere correttamente un pagamento. Una situazione tipo di tale contesto è illustrata in figura 1.

Figura 1: Esempio di monete disposte su una superficie piana

In uno scenario del genere, al crescere del numero di monete e di tagli diventa complicato, senza il supporto visivo riconoscere correttamente tutti gli oggetti, per tale motivo si rende necessario l'utilizzo di uno strumento con le funzionalità proposte. Per la realizzazione di tale progetto è stato dunque effettuato fine tuning su un modello di **Object Detection** che in questa fase iniziale riesce a classificare correttamente una o più monete di taglio "10 centesimi". Ai fini della creazione di tale modello è stato utilizzata la rete **FASTER R-CNN**[6][1] messa a disposizione dalla libreria **PyTorch**[7]. Il modello utilizzato impone delle specifiche riguardo il formato dei dati trattati, si è resa dunque necessaria una fase preliminare per l'adattamento del dataset costruito in modo tale da renderlo conforme alle specifiche richieste. Nello specifico è stato necessario estrapolare dei bounding box a partire da delle maschere, ottenute attraverso l'algoritmo Segment Anything Model

(SAM)[2] sviluppato da META per ogni immagine in modo da etichettare correttamente i dati.

2 Dataset

Il dataset utilizzato è costituito da due componenti principali: una composta da immagini già predisposte per un task di classificazione reperito in rete (dal cui ai fini progettuali sono state prelevate solamente le immagini relative ai 10 centesimi) e una seconda, realizzata appositamente per l'implementazione del progetto in questione. Per la realizzazione del dataset generato appositamente per il progetto sono state effettuate una serie di fotografie attraverso la fotocamera di uno smartphone ad alta risoluzione mantenuto ad una distanza costante rispetto al piano di appoggio delle monete attraverso l'uso di un treppiedi fotografico. Per ogni acquisizione le monete sono state disposte su uno sfondo bianco in posizioni e quantità differenti. Dopo aver effettuato le acquisizioni, ogni immagin è stata ridimensionata manualmente attraverso un software di elaborazione per le immagini digitali alla dimensione $400 \times 400\ px$. Una volte ottenute le immagini ridimensionate è stato possibile effettuare un processo di data augmentation, applicando alle immagini esistenti delle operazioni di rotazione in modo da aumentare la cardinalità del dataset.
Le immagini acquisite sono rappresentate dal campione presente in figura 2.

Figura 2: Campione del dataset acquisito

Mentre le immagini prelevate dal dataset reperito in rete sono rappresentate dal campione in figura 3.

Figura 3: Campione del dataset reperito online

Sia il dataset reperito online che quello costruito appositamente hanno una cardinalità pari a circa 600 immagini, il dataset finale è dunque composto da circa 1200 campioni.

2.1 Nomenclatura utilizzata

Per uniformare la nomeclatura adoperata si è scelto di rinominare i file acquisiti come segue:

1. Taglio della moneta (nel nostro caso "10cent") in modo da essere utilizzato anche come etichetta;
2. Numero progressivo dell'immagine all'interno del dataset;
3. Estensione del file.

Ogni campo è separato dal carattere *underscore* (_). La formattazione utilizzata si presta in questo modo ad eventuali integrazioni future in termini di aggiunta di nuovi tagli di monete o di immagini appartenenti a classi già presenti. E' possibile reperire il codice per questa procedure nella repository, all'interno della directory *Codice*. Il file in questione è disponibile al seguente link: *1-Definizione_Nomi.ipynb*.

2.2 Creazione CSV

Ai fini della creazione di un dataset compatibile con il modello utilizzato è stato necessario seguire le specifiche messe a disposizione dalla libreria PyTorch. Tra le vari opzioni compatibili per l'utilizzo di un dataset personalizzato, si è scelto di utilizzare il metodo che prevede la creazione di un file csv contenente le seguenti informazioni:

1. Il nome dell'immagine;
2. Le classi degli oggetti contornati da una box;
3. I valori numerici delle box.

Un esempio di CSV è presente in figura 4.

	A	B	C	D
1	10cent_22.jpg	[1]	[24.0, 16.0, 130.0, 124.0]	
2	10cent_23.jpg	[1]	[17.0, 17.0, 132.0, 135.0]	
3	10cent_24.jpg	[1]	[26.0, 22.0, 143.0, 128.0]	
4	10cent_25.jpg	[1]	[14.0, 15.0, 131.0, 135.0]	
5	10cent_26.jpg	[1]	[4.0, 20.0, 138.0, 145.0]	
6	10cent_27.jpg	[1]	[15.0, 13.0, 132.0, 134.0]	
7	10cent_28.jpg	[1]	[20.0, 14.0, 129.0, 126.0]	
8	10cent_29.jpg	[1]	[22.0, 32.0, 125.0, 141.0]	
9	10cent_30.jpg	[1]	[22.0, 16.0, 138.0, 131.0]	
10	10cent_31.jpg	[1]	[17.0, 15.0, 135.0, 132.0]	

Figura 4: Esempio di file csv

Le informazioni relative all'estrazione delle box sono trattate con più accuratezza nei capitoli successivi. Per l'inizializzazione del dataset sono richiesti dalla funzione di liberiea utilizzata 5 parametri tra cui ritroviamo:

1. *csv_file*: il path al file csv descritto in precedenza;
2. *root_dir*: il path alla directory contenente le immagini;
3. *width*: la larghezza delle immagini;
4. *heigth*: l'altezza delle immagini;
5. *trasforms*: di default impostata a None, indica eventuali trasformazioni da applicare alle immagini.

E' importante notare che tra le classi conosciute dal modello ne è presente una aggiuntiva che indica il **background** dell'immagine, come mostrato in figura 5.

```
self.coin_frame = pd.read_csv(csv_file, header = None)
print("Dataset caricato correttamente numero immagini:",len(self.coin_frame))
self.root_dir = root_dir
self.transforms = transforms
self.height = height
self.width = width
self.classes = [_, '10_cent']
```

Figura 5: Classi del dataset

Per creare dei dataset che seguono il formato utilizzato da *torch.utils* è necessario implementare due metodi per il corretto funzionamento delle successive operazioni che faranno uso del dataset stesso. In particolare si tratta delle funzioni:

1. *getitem()*: questo metodo viene invocato quando si accede agli elementi del dataset tramite la notazione indicizzata (ad esempio dataset[i]). Esso si occupa semplicemente di restituire l'elemento corrispondente all'indice specificato. Può essere utilizzato per recuperare un'immagine, una coppia di input-output o qualsiasi altro tipo di dato presente nel dataset.;

2. *len()*: questo metodo viene invocato per ottenere la cardinalità del dataset.

In linea con le pratiche standard il dataset composto da circa 1000 immagini è stato partizionato secondo la percentuale $80-20$: l'80% del dataset è stato utilizzato come training set mentre il restante 20% per il validation set. Date le elevate dimensini del dataset, le immagini sono state importate attraverso l'utilizzo di un dataloader, nello specifico il dataloader messo a disposizione da PyTorch *torch.utils.data*. Un ulteriore set di dati di circa 200 immagini viene utilizzato come test set.

2.3 Segmentazione

Per l'estrapolazione delle maschere necessarie all'estrazione delle bounding box è stato effettuato un processo di segmentazione. La segmentazione è un processo chiave nell'ambito dell'elaborazione delle immagini e della visione artificiale che consiste nel dividere un'immagine in regioni o segmenti significativi. L'obiettivo della segmentazione è separare gli oggetti di interesse dall'immagine di sfondo o dal contesto circostante. Questa tecnica è ampiamente utilizzata in una vasta gamma di applicazioni, come la rilevazione degli oggetti, il tracciamento del movimento, la segmentazione degli organi nelle scansioni mediche e la guida autonoma, solo per citarne alcune. Ci sono diversi approcci alla segmentazione delle immagini, che possono essere classificati in due categorie principali: la segmentazione basata su pixel e la segmentazione basata su regione.

2.3.1 Segmentazione basata su pixel

Questo approccio considera ogni pixel dell'immagine come un'unità indipendente e decide se far parte dell'oggetto di interesse o dello sfondo. Alcuni metodi comuni per la segmentazione basata su pixel includono:

1. Segmentazione basata su soglia: si utilizza una soglia per separare i pixel in base ai loro valori di intensità o ad altre caratteristiche. I pixel che superano la soglia vengono etichettati come oggetto, mentre quelli al di sotto vengono etichettati come sfondo;

2. segmentazione basata su edge: Si rilevano i bordi o i gradienti di intensità nell'immagine e si utilizzano per separare gli oggetti dallo sfondo;

3. segmentazione basata su clustering: Si raggruppano i pixel in cluster omogenei in base alle loro caratteristiche, come l'intensità dei colori o le caratteristiche spettrali.

2.3.2 Segmentazione basata su regione

Questo approccio raggruppa i pixel in regioni omogenee in base alle loro proprietà spaziali, di colore o di texture. Alcuni metodi comuni per la segmentazione basata su regione includono:

1. Algoritmi di growing region: si parte da un seme (un pixel o un insieme di pixel) e si espandono le regioni circostanti in base a determinate regole di omogeneità;

2. Algoritmi di divisione e fusione: si inizia con l'intera immagine come una singola regione e si procede a dividere le regioni in segmenti più piccoli in base alle differenze tra i pixel. Successivamente, le regioni possono essere fuse se soddisfano determinate condizioni di omogeneità;

3. Approcci basati su grafo: si costruisce un grafo che rappresenta l'immagine, in cui i pixel corrispondono ai nodi e i collegamenti rappresentano la similarità tra i pixel. Successivamente, si utilizzano algoritmi di taglio del grafo per dividere l'immagine in segmenti.

2.3.3 Semantic Alignment and Matching

Oltre agli approcci di base, sono disponibili molte varianti e metodi più avanzati per la segmentazione, che spesso combinano più tecniche o utilizzano deep convolutional neural networks (CNN) per affrontare sfide più complesse. L'algoritmo utilizzato per la realizzazione del presente progetto prende

il nome di **SAM** , *"Semantic Alignment and Matching"*[2] (Allineamento Semantico e Corrispondenza). SAM mette a disposizione tre metodi per la generazione delle maschere:

1. Generazione automatica delle maschere;
2. Generazione data una bounding box;
3. Generazione dato un punto, invece di delimitare con un rettangolo la zona di cui si vuole ottenere la maschera, si seleziona con un punto colorato l'oggetto e SAM produrrà la maschera dell'oggetto selezionato.

In questa sede la scelta è ricaduta sulla generazione automatica delle maschere in modo da automatizzare totalmente il processo.
La generazione automatica presenta in output una lista contenente le seguenti informazioni:

1. *segmentation*: la maschera con (W,H) larghezza e altezza;
2. *area*: l'area della maschera espressa in pixel;
3. *bbox*: la box dell'oggetto in formato *xywh*, ovvero, il pixel in alto a sinistra descritto in due coordinate, la larghezza e l'altezza;
4. *predicated_iou*: la bontà della predizione del modello (0.9 valore massimo e 0 valore minimo);
5. *point_coords*: il punto di input che ha generato la maschera corrente;
6. *stability_score*: una misura aggiuntiva che indica qualità della maschera;
7. *crop_box*: il ritaglio dell'immagine generato per calcolare la maschera in formato *xywh*.

In figura 6 il codice utilizzato.

```
for i in range(num_image):
  IMAGE_NAME = files[i]
  IMAGE_PATH = dir_path + IMAGE_NAME


  image_bgr = cv2.imread(IMAGE_PATH)
  image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
  sam_result = mask_generator.generate(image_rgb)

  new_masks = sorted(sam_result, key=lambda x: x['area'], reverse=True)

  if new_masks[0]["bbox"][:2]== [0,0]:
    new_masks.pop(0)

  img_np = (new_masks[0]['segmentation'])
  img_tt = torch.from_numpy(img_np)
  img_tt = img_tt[None, : ]

  img = img_tt.long().type(torch.uint8)

  MASK_NAME = IMAGE_NAME[:-4] + "_mask" + IMAGE_NAME[-4] + "png"
  MASK_PATH = dir_path.replace("Images", "Masks", 1) + MASK_NAME

  tensor = img.permute(1, 2, 0)
  array = tensor.numpy()
  cv2.imwrite(MASK_PATH, array)
```

Figura 6: Codice SAM

L'output di SAM figura tutte le maschere che riesce ad estrarre per ordine di riscontro. E' possibile osservare il risultato della segmentazione in figura 7.

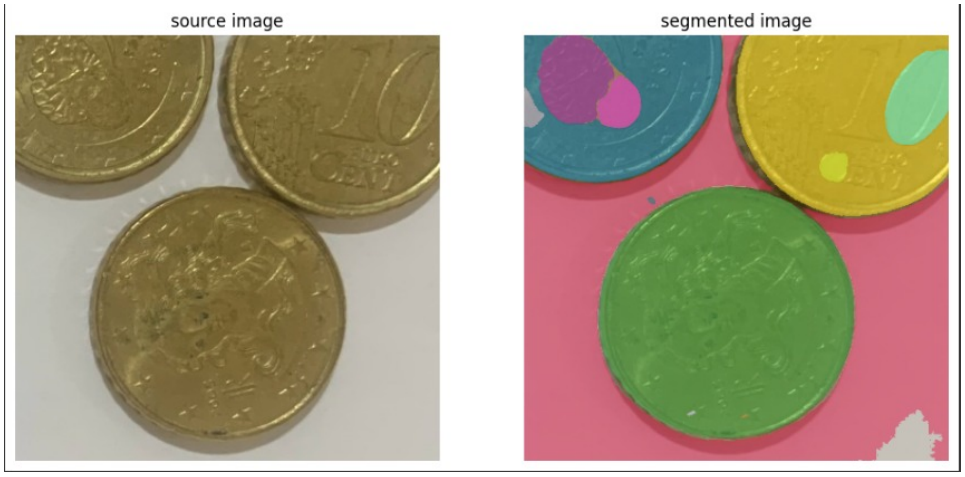

Figura 7: Segmentazione tramite SAM

La prima strategia adottata per rilevare e memorizzare solo le maschhere inerenti alle monete è stata quella di ordinarle per dimensione per poi selezionare il primo elemento della lista, questo perché l'algoritmo è in grado di estrarre maschere molto dettagliate riguardanti le immagini presenti in rilievo sulle monete, tali informazioni non sono però utili in questa fase iniziale. Un secondo accorgimento prevede lo scarto del primo elemento per ordine di dimensione in quanto esso rappresenta la maschera del background (che ha come origine il punto $[0, 0]$). Data quest'osservazione è possibile scartare il primo elemento dell'ordinamento se esso presenta come coordinate $[0, 0]$ in quanto sarà sicuramente il background. Un esempio di maschera estratta è presente in figura 8.

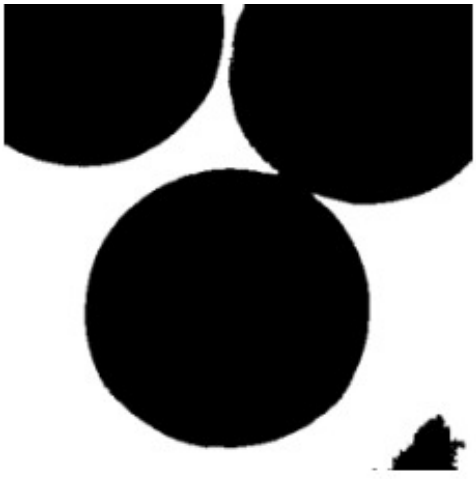

Figura 8: maschera estratta da SAM

Dopodiché vengono eseguiti dei passaggi per salvare la maschera in un formato utile per lo step successivo, la rilevazione delle bounding box. Le maschere vengono salvate utilizzando la nomenclatura "nome moneta_mask". Il codice per l'estrapolazione delle maschere è reperibile al seguente link: *4-Segmetnazione.ipynb*.

2.4 Bounding Box

Le bounding box, anche note come "scatole delimitatrici" o "box di confine", sono un concetto fondamentale nell'ambito del riconoscimento e dell'elaborazione delle immagini. Esse rappresentano un modo per definire e circoscrivere l'area di interesse di un oggetto all'interno di un'immagine. L'obiettivo delle bounding box è quello di delimitare in maniera accurata gli oggetti presenti nelle immagini, consentendo di estrarre informazioni specifiche su di essi e di svolgere una serie di compiti di analisi e classificazione. Le bounding box sono spesso utilizzate come input per algoritmi di machine learning, che possono apprendere a riconoscere gli oggetti sulla base delle informazioni contenute all'interno delle scatole delimitatrici. La rap-

presentazione delle bounding box può variare a seconda del contesto e delle esigenze dell'applicazione. In genere, una bounding box viene rappresentata come un rettangolo all'interno dell'immagine, definito dalle coordinate dei punti di inizio e fine della diagonale o dai quattro angoli del rettangolo. Le coordinate di una bounding box sono spesso espresse come valori relativi rispetto alle dimensioni dell'immagine. Ad esempio, le coordinate potrebbero essere normalizzate nell'intervallo $[0, 1]$, in cui (0, 0) rappresenta l'angolo in alto a sinistra dell'immagine e (1, 1) rappresenta l'angolo in basso a destra. Questo permette una rappresentazione indipendente dalle dimensioni assolute dell'immagine, rendendo più facile la generalizzazione dell'algoritmo di riconoscimento degli oggetti a diverse immagini di dimensioni diverse. Le bounding box sono ampiamente utilizzate in molte applicazioni di visione artificiale e intelligenza artificiale. Alcuni esempi includono:

1. Riconoscimento degli oggetti: le bounding box sono spesso utilizzate per identificare e delimitare gli oggetti di interesse in un'immagine. Ad esempio, in un'applicazione di riconoscimento degli animali, le bounding box possono essere usate per individuare e circoscrivere i volti degli animali all'interno di un'immagine;

2. Segmentazione delle immagini: le bounding box possono essere utilizzate anche per definire le regioni di interesse all'interno di un'immagine, in modo da consentire una segmentazione precisa degli oggetti. Questo processo permette di distinguere gli oggetti di interesse dallo sfondo circostante, contribuendo a una migliore comprensione e analisi dell'immagine;

3. Tracciamento degli oggetti: le bounding box sono utili anche nel tracciamento degli oggetti in un video.

Per la creazione delle maschere è stata utilizzata la funzione di libreria *torchvision.ops* **masks_to_boxes**. Mostrato nello snippet di codice in figura 9.

```
for i in range(len(imgs)):
  index = findIndex(i)
  img_path = imgs_path + imgs[i]
  mask_path = masks_path + Masks[index]
  img = read_image(img_path)
  mask = read_image(mask_path)
  obj_ids = torch.unique(mask)
  obj_ids = obj_ids[1:]
  masks = mask == obj_ids[:, None, None]
  boxes = masks_to_boxes(masks)
  torch.save(boxes, pathSaveBoxe(imgs[i]))
```

Figura 9: Creazione Bounding Box

Dove mask è la maschera ottenuta tramite SAM. Un esempio di bounding box estratta è rappresentato in figura 10.

Figura 10: Esempio di Bounding box

Una volta ottenuta, la box viene salvata per gli utilizzi futuri. Il codice che permette l'estrazione di box è reperibile al seguente link: *5-BoundingBox.ipynb*.

3 Metodologie utilizzate

3.1 FASTER R-CNN

L'algoritmo Faster R-CNN[1] (Region-based Convolutional Neural Network) è un approccio di rilevamento degli oggetti che combina una rete neurale convoluzionale per l'estrazione delle caratteristiche e una region proposal network per generare proposte di regioni d'interesse. L'architettura di Faster R-CNN è composta da tre principali componenti:

- Rete neurale convoluzionale (CNN): una CNN preaddestrata viene utilizzata per estrarre le caratteristiche dell'immagine di input. Questa rete è in grado di apprendere automaticamente una rappresentazione di alto livello delle immagini, rilevando pattern e strutture significative;

- Region Proposal Network (RPN): la RPN è una parte cruciale di Faster R-CNN. Utilizza la feature map prodotta dalla CNN per generare proposte di regioni d'interesse che potrebbero contenere oggetti. La RPN propone una serie di possibili regioni, indicando la loro posizione approssimativa e un punteggio di confidenza associato;

- Rete di classificazione e rilevamento: le proposte di regioni generate dalla RPN vengono utilizzate per l'elaborazione successiva. Una ROI (Region of Interest) pooling layer viene applicata per adattare le diverse dimensioni delle regioni proposte in una dimensione fissa. Successivamente, una rete neurale fully connected viene utilizzata per classificare e rilevare gli oggetti all'interno di ciascuna regione proposta;

L'algoritmo Faster R-CNN è noto per la sua accuratezza e precisione nel rilevamento degli oggetti in diverse immagini. La combinazione della CNN

per l'estrazione delle caratteristiche, della RPN per la generazione di proposte e della rete di classificazione e rilevamento rende possibile individuare e classificare efficacemente gli oggetti di interesse all'interno di un'immagine. La libreria PyTorch offre diverse varianti del modello citato, quali:

- fasterrcnn_resnet50_fpn: questo modello utilizza una rete neurale convoluzionale ResNet-50 come backbone insieme a una Feature Pyramid Network[3] (FPN). La rete ResNet-50 è una deep neural network preaddestrata sul dataset ImageNet[8] e viene utilizzata per estrarre le caratteristiche dell'immagine. L'FPN viene utilizzata per combinare le caratteristiche estratte a diverse scale per il rilevamento degli oggetti. Questa implementazione è basata sul paper "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks"[1];

- fasterrcnn_resnet50_fpn_v2: Questo modello è una versione migliorata di fasterrcnn_resnet50_fpn. Utilizza lo stesso backbone ResNet-50-FPN, ma è stato addestrato con tecniche di apprendimento automatico trasferito insieme ai Vision Transformers. L'idea principale di questo modello è combinare le caratteristiche apprese da una rete convoluzionale tradizionale con quelle apprese dai Vision Transformers, al fine di ottenere prestazioni migliori nel rilevamento degli oggetti. Questa implementazione si basa sul paper "Benchmarking Detection Transfer Learning with Vision Transformers"[4];

- fasterrcnn_mobilenet_v3_large_fpn: Questo modello utilizza un backbone MobileNetV3-Large insieme a una Feature Pyramid Network (FPN). MobileNetV3-Large è un'architettura di rete neurale progettata per l'efficienza computazionale su dispositivi mobili. L'FPN viene utilizzata per combinare le caratteristiche estratte a diverse scale per

il rilevamento degli oggetti. Questa implementazione è adatta per il rilevamento degli oggetti ad alta risoluzione;

- fasterrcnn_mobilenet_v3_large_320_fpn: Questo modello è una variante a bassa risoluzione di fasterrcnn_mobilenet_v3_large_fpn. Utilizza lo stesso backbone MobileNetV3-Large-FPN, ma è stato ottimizzato per l'utilizzo su dispositivi mobili con risorse computazionali limitate. Questa implementazione è adatta per il rilevamento degli oggetti a bassa risoluzione e viene utilizzata principalmente in scenari mobili.

In particolare è stata utilizzata l'implementazione FASTERRCNN-_RESNET50_FPN messa a disposizione dal framework PyTorch in quanto essa risulta essere la più versatile e dunque adatta ad un task di classificazione come quello presentato. PyTorch Faster R-CNN offre anche una vasta gamma di funzionalità aggiuntive, come la possibilità di utilizzare meccanismi di attenzione per migliorare la precisione del rilevamento, l'implementazione di algoritmi di non-maximum suppression per la gestione delle sovrapposizioni tra le regioni proposte e la possibilità di adattare il modello a specifici dataset mediante il transfer learning. Dopo aver addestrato e testato il modello principale sono stati effettuati dei test tramite alcuni degli altri modelli messi a disposizione dalla libreria.

3.2 Fine-Tuning

Con fine tuning, nell'ambito del machine learning, ci si riferisce a una tecnica utilizzata per adattare e migliorare i modelli di intelligenza artificiale pre-addestrati per task specifici. In particolare, quando si parla di fine tuning, ci si riferisce spesso a modelli di deep learning, come le reti neurali, che sono stati preaddestrati su un'ampia quantità di dati non specifici per un compito particolare, come ad esempio l'addestramento su grandi insiemi di

testo per la comprensione del linguaggio. Il processo di fine tuning coinvolge il training aggiuntivo di un modello preaddestrato su un insieme di dati più specifico e ristretto, noto come insieme di dati di addestramento di destinazione. Questo insieme di dati di destinazione contiene esempi e annotazioni pertinenti al compito specifico che si desidera svolgere con il modello. Il fine tuning si compone tipicamente di due fasi principali:

1. Pretraining: In questa fase, un modello di deep learning viene addestrato su un'ampia quantità di dati generici. Questo processo può richiedere molto tempo e risorse computazionali, ma può generare un modello che ha appreso una vasta gamma di rappresentazioni e conoscenze generali.

2. Fine tuning: Dopo il pretraining, il modello viene adattato al compito specifico mediante l'addestramento su un insieme di dati di destinazione. Durante questa fase, i pesi e i parametri del modello vengono regolati in base agli esempi specifici presenti nell'insieme di dati di destinazione. Il fine tuning può richiedere meno tempo e risorse rispetto al pretraining, poiché si basa sull'apprendimento generale acquisito durante il preaddestramento.

L'obiettivo del fine tuning è quello di trasferire la conoscenza generale acquisita dal modello preaddestrato al compito specifico che si desidera svolgere. Poiché il modello preaddestrato ha già appreso molte rappresentazioni e concetti generali, il fine tuning consente di ottenere prestazioni migliori e una migliore generalizzazione rispetto all'addestramento da zero, specialmente quando l'insieme di dati di destinazione è limitato.
Per il fine tuning del modello utilizzato l'approccio è stato quello di modificare esclusivamente l'ultimo layer al fine di risolvere il task di classificazione

delle monete, sfruttando le conoscenze pregresse del modello stesso. Per implementare tali operazini sono state utilizzate le librerie ufficiali messe a disposizione da PyTorch.

```
def get_object_detection_model(num_classes):

    model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)

    in_features = model.roi_heads.box_predictor.cls_score.in_features
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

    return model
```

Figura 11: Fine Tuning

3.2.1 Ottimizzazione fase di Training

Per velocizzare ulteriormente la fase di training si è utilizzato la discesa del gradiente stocastica ed è stato implementato uno scheduler.
La discesa del gradiente stocastica (Stochastic Gradient Descent, SGD) è un algoritmo di ottimizzazione ampiamente utilizzato nell'apprendimento automatico per addestrare modelli di machine learning, in particolare le reti neurali. L'obiettivo della SGD è quello di trovare i valori ottimali dei parametri di un modello, minimizzando la funzione di costo associata al task di addestramento. A differenza della discesa del gradiente classica, in cui si calcola il gradiente della funzione di costo su tutto l'insieme di addestramento, la SGD esegue l'aggiornamento dei pesi del modello utilizzando solo un sottoinsieme casuale di esempi di addestramento alla volta. In altre parole, invece di calcolare il gradiente sulla somma di tutti gli esempi di addestramento, la SGD calcola il gradiente su un mini-batch di esempi selezionati casualmente. Ciò comporta diversi vantaggi:

1. Efficienza computazionale: calcolare il gradiente su un mini-batch richiede meno risorse computazionali rispetto all'utilizzo di tutto l'insieme di addestramento. Ciò consente di addestrare modelli su insiemi di dati di grandi dimensioni in modo più rapido ed efficiente;

2. Regolarizzazione implicita: l'utilizzo di mini-batch casuali introduce una regolarizzazione implicita nel processo di addestramento. Ogni mini-batch è un sottoinsieme unico dell'insieme di addestramento, che può contribuire a ridurre l'overfitting, migliorando la capacità di generalizzazione del modello;

3. Escaping dai minimi locali: utilizzando mini-batch casuali, la SGD può saltare fuori dai minimi locali della funzione di costo, consentendo

al modello di esplorare lo spazio dei parametri in modo più dinamico. Ciò può portare a un miglioramento delle prestazioni e consentire al modello di raggiungere minimi globali migliori.

Nel contesto dell'addestramento dei modelli di machine learning, uno scheduler, o "learning rate scheduler", è una tecnica utilizzata per regolare dinamicamente il tasso di apprendimento durante il processo di ottimizzazione. Lo scheduler aiuta a determinare come il tasso di apprendimento dovrebbe cambiare nel corso dell'addestramento per migliorare le prestazioni del modello. In figura 12 è presente l'implementazione dello scheduler e dei parametri utilizzati per la SGD.

```
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005,
                            momentum=0.9, weight_decay=0.0005)

lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                               step_size=3,
                                               gamma=0.1)
```

Figura 12: Codice Ottimizzazione fase di Training

E' possibile notare tra i parametri la keyword **momentum**[5]. La tecnica del momentum fornisce uno speed-up quadratico al learning. Si sceglie un parametro solitamente settato a e si effettua l'aggiornamento tenendo uno stato :

$$\Theta^{new} = \Theta^{old} - \alpha z^{new} \; z^{new} = \beta z^{old} + \nabla J(\Theta^{old})$$

Impostando $\beta = 0$ si ottiene il gradient descent classico.

3.3 Esportazione Modello

Una volta allenato il modello è stato possibile esportarlo in modo da poterlo utilizzare direttamente in altri contesti. Il modello è stato esportato in for-

mato ONNX. ONNX (Open Neural Network Exchange)[9] è un formato open source per la rappresentazione e lo scambio di modelli di intelligenza artificiale tra diversi framework di machine learning. Le principali caratteristiche di ONNX includono:

1. Portabilità: ONNX consente di convertire modelli addestrati da un framework a un altro senza perdere le informazioni fondamentali del modello. Ciò consente agli sviluppatori di utilizzare i modelli in diversi ambienti di esecuzione senza dover ripetere l'intero processo di addestramento;

2. Interoperabilità: ONNX supporta una vasta gamma di framework di machine learning, tra cui PyTorch, TensorFlow, Caffe, Microsoft Cognitive Toolkit (CNTK) e molti altri. Ciò consente agli sviluppatori di combinare modelli da diversi framework e sfruttare le funzionalità specifiche di ciascuno;

3. Esecuzione efficiente: ONNX offre un'implementazione ottimizzata per l'esecuzione dei modelli, che può portare a un miglioramento delle prestazioni e all'utilizzo efficiente delle risorse hardware. L'ottimizzazione include ad esempio il supporto per l'inferenza accelerata su hardware specializzato come GPU e processori AI;

4. Estensibilità: ONNX offre un'architettura estensibile che consente di definire facilmente nuovi operatori o estendere le funzionalità esistenti per supportare nuove operazioni o funzioni dei modelli. Ciò favorisce la crescita dell'ecosistema ONNX e la collaborazione tra i diversi framework;

5. Comunità attiva: ONNX ha una comunità di sviluppatori molto attiva e crescente, con contributi da parte di aziende e ricercatori di tutto il

mondo. Ciò assicura un costante sviluppo e miglioramento del formato ONNX, nonché il supporto per nuovi framework e funzionalità.

I parametri di *'torch.onnx.export'* sono utilizzati per configurare e controllare il processo di esportazione di un modello PyTorch in formato ONNX. La funzione *'torch.onnx.export'* consente di convertire un modello PyTorch in un formato ONNX, consentendo di utilizzarlo in altri framework o ambienti di esecuzione compatibili con ONNX. Di seguito sono riportati alcuni dei parametri di *'torch.onnx.export'*':

1. *model*: il modello PyTorch da esportare in formato ONNX. Deve essere un'istanza di una sottoclasse di *'torch.nn.Module'*;

2. *dummy_input*: un' immagine di esempio che rappresenta gli input che il modello si aspetta;

3. *f*: il percorso del file ONNX in cui verrà salvato il modello esportato. Può essere una stringa contenente il percorso completo del file o un oggetto di tipo file aperto;

4. *export_params*: un flag booleano che indica se esportare i parametri del modello insieme alla definizione del grafo. Se impostato su 'True', i parametri del modello saranno inclusi nel file ONNX;

5. *opset_version*: la versione dell'opset ONNX da utilizzare per l'esportazione. L'opset definisce le operazioni supportate nel file ONNX e può variare tra le diverse versioni di ONNX. È consigliabile specificare la versione dell'opset corretta in base alla compatibilità con il framework o l'ambiente di esecuzione di destinazione;

6. *input_names*: un elenco di stringhe che definisce i nomi degli input del modello che saranno esportati in ONNX. Questi nomi dovrebbero

corrispondere agli identificatori degli input utilizzati nel codice PyTorch. Specificare i nomi degli input è particolarmente utile quando si desidera interfacciarsi con il modello ONNX in un altro framework o ambiente di esecuzione, in quanto consente di identificare chiaramente gli input del modello durante l'uso successivo;

7. *output_names*: un elenco di stringhe che definisce i nomi degli output del modello che saranno esportati in ONNX. Analogamente a 'input_names', questi nomi dovrebbero corrispondere agli identificatori degli output utilizzati nel codice PyTorch. Specificare i nomi degli output è utile per identificare in modo chiaro gli output del modello durante l'utilizzo successivo.

Il codice necessario per l'esportazione del modello è reperibile al seguente link: *6-Fine_Tuning.ipynb*

4 Valutazioni

Le metriche utilizzate per la fase di training sono quelle fornite da PyTorch e integrate all'interno della rete FASTER R-CNN. Di seguito una breve descrizione:

- Loss: rappresenta il valore complessivo della funzione di perdita (loss) del modello. È una misura dell'errore totale tra le previsioni del modello e i valori di riferimento (ground truth) durante l'addestramento. L'obiettivo è minimizzare questa metrica per migliorare le prestazioni del modello;

- loss_classifier: rappresenta la componente di perdita relativa alla classificazione degli oggetti. Nel contesto di Faster R-CNN, il modello deve determinare la classe degli oggetti presenti nelle immagini;

- loss_box_reg: rappresenta la componente di perdita relativa alla regressione dei bounding box (box_reg) degli oggetti. Il modello utilizza la regressione per predire le posizioni precise dei bounding box degli oggetti nell'immagine;

- loss_objectness: rappresenta la componente di perdita relativa alla classificazione dell'objectness degli anchor boxes. Gli anchor boxes sono regioni proposte dall'algoritmo RPN (Region Proposal Network) di Faster R-CNN, e questa metrica misura la capacità del modello di distinguere tra regioni contenenti oggetti e regioni di sfondo;

- loss_rpn_box_reg: rappresenta la componente di perdita relativa alla regressione dei bounding box degli anchor boxes generati dall'RPN. L'RPN propone le regioni candidate per la successiva fase di classificazione e regressione dei bounding box;

Queste metriche vengono calcolate durante l'addestramento per monitorare le prestazioni del modello e guidarne l'ottimizzazione. L'obiettivo è quello di minimizzarle tutte, in modo da migliorare la capacità del modello di classificare correttamente gli oggetti e predire con precisione i loro bounding box.

Un'altra misura utilizzata è l'**IoU**. Nel contesto dei modelli Faster CRNN (Convolutional Recurrent Neural Network) in PyTorch, l'IOU (Intersection over Union) evaluation è una metrica utilizzata per valutare le prestazioni del modello nella fase di testing o valutazione. L'IOU è una misura di sovrapposizione tra due regioni o bounding box, essa viene calcolata come il rapporto tra l'area di intersezione tra le due regioni e l'area dell'unione delle due regioni. In altre parole, misura quanto due regioni si sovrappongono tra loro. Nel contesto preso in esame, l'IOU evaluation viene utilizzata per

valutare quanto bene il modello è in grado di rilevare e localizzare oggetti di interesse all'interno di un'immagine. Durante la valutazione, il modello genera delle predizioni sotto forma di bounding box che indicano la posizione approssimata degli oggetti nell'immagine. L'IOU evaluation confronta queste predizioni con le ground truth (le vere posizioni degli oggetti nell'immagine) per calcolare l'IOU tra le predizioni del modello e le ground truth. Un punteggio IOU alto indica che il modello ha fatto predizioni accurate e ha localizzato correttamente gli oggetti di interesse. Un punteggio IOU basso indica che il modello ha fatto predizioni imprecise o non è stato in grado di localizzare correttamente gli oggetti. L'IOU evaluation viene spesso utilizzata insieme ad altre metriche di valutazione, come la precision e la recall, per ottenere una valutazione completa delle prestazioni del modello nella rilevazione e localizzazione degli oggetti.

Precision e Recall sono due metriche comunemente utilizzate per valutare le prestazioni di un modello di classificazione La Precision misura la proporzione di istanze classificate correttamente come positivi rispetto a tutte le istanze classificate come positivi dal modello. Essa è calcolata come:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$$

La recall misura la proporzione di istanze positive correttamente classificate come tali rispetto a tutte le istanze effettivamente positive. La recall rappresenta la capacità del modello di individuare correttamente tutti i veri positivi. È calcolata come:

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$$

Nel contesto della metrica IOU per le bounding box, l'attributo **"area"** può essere specificato per suddividere le bounding box in diverse categorie in base alla loro area relativa rispetto all'immagine di riferimento. Le categorie solitamente utilizzate sono:

- All, tutte le bounding box;
- Small, bounding box di piccole dimensioni;
- Medium, bounding box di dimensioni medie;
- Large, bounding box di grandi dimensioni.

Questa suddivisione in categorie diverse può essere utile per valutare le prestazioni del modello in base alle dimensioni degli oggetti rilevati. La definizione specifica di cosa rappresenta "small", "medium" e "large" può variare a seconda del contesto e del dataset utilizzato. Di solito, queste categorie vengono definite in base a soglie specifiche di area o rapporto tra l'area della bounding box e l'area totale dell'immagine. Un altro parametro utilizzato è **"maxDets"**, quest'ultimo si riferisce al numero massimo di rilevamenti (detections) considerati durante la valutazione delle prestazioni del modello. Quando si valutano le prestazioni di un modello di rilevamento degli oggetti, spesso si generano diverse predizioni (detections) per ogni oggetto nell'im-

magine. L'attributo "maxDets" viene utilizzato per specificare il numero massimo di predizioni considerate per ogni oggetto durante la valutazione. Ad esempio, se si imposta "maxDets" a 100, significa che verranno considerate solo le prime 100 predizioni più promettenti per ogni oggetto durante la valutazione. Le predizioni al di là di questo numero massimo non verranno prese in considerazione per il calcolo delle metriche. maxDets è utile per limitare il numero di predizioni considerate durante la valutazione, in modo da concentrarsi sulle predizioni più rilevanti e significative. Ciò può aiutare a ottenere una valutazione più accurata delle prestazioni del modello senza considerare predizioni ridondanti o poco attendibili.

5 Esperimenti

Come modello di partenza è stato scelto **fasterrcnn_resnet50_fpn**. I primi test eseguiti sul modello sono stati effettuati tramite il l'utilizzo di un dataset composto da circa 250 elementi e impostando il numero di epoche a 5. Utilizzando questi parametri il modello non è stato in grado di effettuare correttamente la classificazione. Esso infatti non risultava essere in grado di riconoscere i box di un'immagine e di assegnare agli elementi identificati un'etichetta. Un esempio di risultato non adeguato è rappresentato dall'immagine in figura 13.

Figura 13: Classificazione errata

Per i motivi sopra elencati nelle fasi successive si è deciso di incrementare le dimensioni del dataset, integrando i dati raccolti appositamente per il progetto, si è inoltre applicato in questa fase un processo di data augmentation applicando operazioni affini sui campioni raccolti. Inoltre il numero di epoche del training è stato fissato a 10 in quanto osservando i valori della loss, non era ancora stata raggiunta la convergenza. Nonostante ciò, osservando l'andamento dei valori si è deciso di utilizzare il modello ottenuto durante la quarta epoca di training in quanto esso presenta i risultati migliori. Oltre a queste accortezze è stato utilizzato un learning rate scheduler fornito da PyTorch che prende il nome di **StepLR**. Questo scheduler dipende da tre parametri:

- optimizer: l'optimizer che viene utilizzato per aggiornare i pesi del

modello durante l'addestramento. L'oggetto scheduler si occupa di modificare il tasso di apprendimento all'interno di questo ottimizzatore;

- step_size: il numero di epoche dopo le quali il tasso di apprendimento verrà ridotto. In altre parole, dopo step_size epoche, il learning rate sarà moltiplicato per il valore specificato dal parametro gamma;
- gamma: il fattore di riduzione del tasso di apprendimento. Dopo ogni intervallo di step_size epoche, il learning rate viene moltiplicato per gamma. Ad esempio, se gamma è impostato a 0.1, il tasso di apprendimento verrà ridotto del 90% (moltiplicato per 0.1) dopo ogni intervallo di step_size epoche.

Sperimentalmente ai fini di questo progetto si è verificato che i valori in grado di ottenere risultati migliori risultano essere 3 per la step_size e 0.2 per gamma. Inoltre come optimizer è stato utilizzato l'algoritmo SGD insieme alla tecnica del momentum con parametro impostato a 0.9.
I risultati ottenuti secondo le metriche di loss descritte nel capitolo precedente sono riportati nella seguente tabella.

Epoca	Batch	Loss	loss_classifier	loss_box_reg	loss_objectness	loss_rpn_box_reg
0	[0/49]	0.9417	0.7704	0.1618	0.0045	0.0049
0	[10/49]	0.4429	0.2735	0.1529	0.0030	0.0037
0	[0/49]	0.3154	0.1761	0.1461	0.0037	0.0023
0	[30/49]	0.1955	0.0443	0.1326	0.0053	0.0014
0	[40/49]	0.1464	0.0205	0.1206	0.0041	0.0010
0	[48/49]	0.0783	0.0160	0.0619	0.0026	0.0009
1	[0/49]	0.0480	0.0480	0.0352	0.0015	0.0006
1	[10/49]	0.0476	0.0126	0.0331	0.0007	0.0008
1	[0/49]	0.0390	0.0123	0.0246	0.0007	0.0007
1	[30/49]	0.0279	0.0104	0.0172	0.0005	0.0006
1	[40/49]	0.0254	0.0096	0.0148	0.0004	0.0006
8	[48/49]	0.0244	0.0096	0.0134	0.0003	0.0006
8	[0/49]	0.0178	0.0093	0.0079	0.0001	0.0004
8	[10/49]	0.0130	0.0068	0.0053	0.0001	0.0004
8	[0/49]	0.0130	0.0065	0.0057	0.0001	0.0003
8	[30/49]	0.0130	0.0064	0.0057	0.0001	0.0003
9	[40/49]	0.0137	0.0067	0.0057	0.0002	0.0003
9	[48/49]	0.0142	0.0067	0.0057	0.0002	0.0004
9	[0/49]	0.0003	0.0083	0.0114	0.0001	0.0005
9	[10/49]	0.0127	0.0058	0.0057	0.0001	0.0003
9	[0/49]	0.0126	0.0059	0.0054	0.0002	0.0003

Inerentemente ai risultati ottenuti osservando l'IoU sul test set, si è deciso di utilizzare il modello ottenuto durante la quarta epoca in quanto presenta valori di Average Precisione e Average Recall migliori. Nella seguente tabella sono riportati i valori relativi alle metriche prese in considerazione durante la fase di testing per ogni epoca.

Epoca	Tipo	IoU	Area	MaxDets	Valore
0	AP	0.50:0.95	all	100	0.763
0	AR	0.50:0.95	all	100	0.801
1	AP	0.50:0.95	all	100	0.981
1	AR	0.50:0.95	all	100	0.940
2	AP	0.50:0.95	all	100	0.949
2	AR	0.50:0.95	all	100	0.968
4	AP	0.50:0.95	all	100	0.967
4	AR	0.50:0.95	all	100	0.978
5	AP	0.50:0.95	all	100	0.963
5	AR	0.50:0.95	all	100	0.975
8	AP	0.50:0.95	all	100	0.961
8	AR	0.50:0.95	all	100	0.974
9	AP	0.50:0.95	all	100	0.962
9	AR	0.50:0.95	all	100	0.975

Tabella 1: Risultati delle metriche di valutazione

Nella seguente tabella sono presenti le misure inerenti all'epoca quattro:

Epoca	Tipo	IoU	Area	MaxDets	Valore
4	AP	0.50:0.95	all	100	0.967
4	AP	0.50	all	100	1.000
4	AP	0.75	all	100	1.000
4	AP	0.50:0.95	small	100	-1.000
4	AP	0.50:0.95	medium	100	0.800
4	AP	0.50:0.95	large	100	0.969
4	AR	0.50:0.95	all	1	0.978
4	AR	0.50:0.95	all	10	0.978
4	AR	0.50:0.95	all	100	0.978
4	AR	0.50:0.95	small	100	-1.000
4	AR	0.50:0.95	medium	100	0.800
4	AR	0.50:0.95	large	100	0.979

Tabella 2: Risultati quarta epoca di training

A seguito dei test effettuati il modello ottenuto è in grado di fornire risultati migliori e di portare a termine correttamente il task di detection. Un'esempio di detection corretta è riportato in figura 14 in cui osserviamo gli oggetti identificati correttamente e correlati di una label che indica la classe di appartenenza e uno score di accuratezza.

Figura 14: Risultato corretto

5.1 Esperimenti non riusciti

L'idea alla base degli esperimenti successivi durante la fase di testing è stata quella di effettuare dei confronti tra il modello ottenuto effettuando fine tuning della versione *fasterrcnn_resnet50_fpn* ed altre backbone messe a disposizone da PyTorch, le cui peculiarità sono state discusse nei capitoli precedenti. In particolare sono stati testati i modelli *fasterrcnn_resnet50_fpn_v2* e *fasterrcnn_mobilenet_v3_large_fpn*. A causa dei limiti relativi alle risorse computazionali messe a disposizione dall'ambiente di sviluppo utilizzato non è stato possibile effettuare il training di tali modelli sfruttando la GPU. Per tale motivo si è optato per effettuare il training sfruttando le risorse CPU. A causa di questa problematica è stato possibile effettuare il training dei modelli solamente per un'epoca in quanto il tempo richiesto era pari a circa 5 ore per ogni modello. Dato che non sono stati ottenuti i risultati sperati dal modello, si è preferito non effettuare confronti con il modello base ma semplicemente riportare i risultati scaturiti dal training parziale dei modelli che fanno uso delle altre backbone.

5.1.1 Fasterrcnn_resnet50_fpn_v2

In merito alla dorsale fasterrcnn_resnet50_fpn_v2, a causa dei problemi citati precedentemente, il training tramite CPU di una sola epoca ha prodotto i risultati riportati in tabella 3.

Epoca	Tipo	IoU	Area	MaxDets	Valore
0	AP	0.50:0.95	all	100	0.000
0	AP	0.50	all	100	0.000
0	AP	0.75	all	100	0.000
0	AP	0.50:0.95	small	100	0.000
0	AP	0.50:0.95	medium	100	-1.000
0	AP	0.50:0.95	large	100	-1.000
0	AR	0.50:0.95	all	1	0.000
0	AR	0.50:0.95	all	10	0.000
0	AR	0.50:0.95	all	100	0.002
0	AR	0.50:0.95	small	100	0.002
0	AR	0.50:0.95	medium	100	-1.000
0	AR	0.50:0.95	large	100	-1.000

Tabella 3: Metriche di valutazione v2

E' possibile osservare che come previsto i risultati di tale modello non risultano essere buoni ai fini della detection in quanto nulli. Un esempio è riportato in figura 15,

Figura 15: Risultati dorsale v2

dove è possibile notare che la rete non è in grado di effettuare la detection correttamente.

5.1.2 fasterrcnn_mobilenet_v3_large_fpn

Inerentemente alla dorsale light i risultati sono riportati in tabella 4.

Epoca	Tipo	IoU	Area	MaxDets	Valore
0	AP	0.50:0.95	all	100	0.000
0	AP	0.50	all	100	0.000
0	AP	0.75	all	100	0.000
0	AP	0.50:0.95	small	100	0.000
0	AP	0.50:0.95	medium	100	-1.000
0	AP	0.50:0.95	large	100	-1.000
0	AR	0.50:0.95	all	1	0.000
0	AR	0.50:0.95	all	10	0.000
0	AR	0.50:0.95	all	100	0.000
0	AR	0.50:0.95	small	100	0.000
0	AR	0.50:0.95	medium	100	-1.000
0	AR	0.50:0.95	large	100	-1.000

Tabella 4: Metriche di valutazione modello light

6 Demo

Una demo è reperebile presso una cartella condivisa di Google Drive per consentire l'esecuzione diretta del modello tramite Google Colab, sfruttando immediatamente la potenza della GPU senza la necessità di configurare il supporto per le CUDA o installare il progetto in locale. La demo consente di caricare il modello addestrato e permette all'utente finale di selezionare le immagini su cui testarlo. Di seguito sono riportati i passaggi per utilizzare la demo:

1. Seguire il *link*;
2. Aggiungere la cartella condivisa al proprio Google Drive utilizzando l'opzione "Aggiungi scorciatoia a Drive";
3. Aprire la cartella dalla propria interfaccia di Google Drive.
4. Eseguire il codice all'interno del notebook demo.ipynb;

La configurazione e il funzionamento sono illustrati nel video dimostrativo reperibile al seguente link: *video demo*.

7 Organizzazione codice

L'ambiente di sviluppo utilizzato per la realizzazione del progetto è Google Colab. Google Colab è una piattaforma di cloud computing gratuita offerta da Google. Consente agli utenti di eseguire codice Python in un ambiente basato sul web, senza dover configurare un proprio ambiente di sviluppo. Una delle caratteristiche distintive di Google Colab è la possibilità di utilizzare in modo semplice risorse GPU (unità di elaborazione grafica). Le GPU sono particolarmente efficaci nell'eseguire operazioni computazional-

mente intensive, come addestrare modelli di deep learning su grandi set di dati. La struttura del progetto è la seguente:

- Codice, questa cartella contiene i file necessari alla creazione e del funzionamento del dataset e del modello. Nello specifico:
  - Definizione_nomi, rinomina le immagini del dataset secondo la nomeclatura standard descritta nelle sezioni precedenti;
  - Creazione_CSV_Dataset, crea il CSV del dataset associando ad ogni immagine una maschera e una box;
  - Creazione_CSV_Testset, crea il CSV per il test set associando ad ogni immagine una maschera. La box non è inserita in quanto essa dovrà essere dedotta dal modello;
  - Segmentation, il codice crea un generatore di maschere automatiche utilizzando il modello SAM. Il generatore di maschere è un'istanza della classe SamAutomaticMaskGenerator presente nel modulo segment_anything;
  - BoundingBox, Le maschere vengono convertite in box di delimitazione utilizzando masks_to_boxes, e infine le box di delimitazione vengono salvate utilizzando torch.save nel percorso specificato dalla funzione pathSaveBoxe;
  - Fine_Tuning, Crea ed effettua il fine tuning del modello utilizzando il dataset creato appositamente, dopo aver effettuato il training del modello lo esporta in formato ONNX;
  - Modello, definisce le funzioni per visualizzare le box rilevate utilizzando le coordinate spaziali ed importa il modello per i test.
- DataSet, all'interno di questa cartella sono presenti le immagini, le maschere e le box estratte, necessarie alla creazione del dataset. Inoltre

sono presenti dei file CSV che permettono di aggregare i dati precedentemente descritti in un formato leggibile dal modello utilizzato. Contestualmente è presente una cartella contenente le immagini utilizzate solamente in fase di test.

- Demo, Contiene un file che permette di testare il funzionamento del modello attraverso l'upload di immagini scelte dall'utente;
- Modelli, contiene i modelli allenati in formato ONNX. In particolare sono presenti i modelli della rete standard e delle versioni V2 e Light;

8 Conclusioni

Il progetto realizzato ha evidenziato l'efficacia e la versatilità del modello Faster R-CNN, fornito da PyTorch, nell'affrontare la sfida della detection delle monete di 10 centesimi. Attraverso un processo di fine tuning, è stato possibile ottimizzare il modello per riconoscere e distinguere con precisione le caratteristiche distintive di queste monete specifiche. E' stato inoltre integrato con successo l'algoritmo di Semantic Alignment and Matching nel flusso di lavoro. Questo algoritmo ha consentito l'estrazione di maschere dettagliate delle monete, fornendo un'ulteriore informazione di contesto per migliorare la precisione della detection. Utilizzando queste maschere, sono state generate delle bounding box precise e accurate, che hanno permesso di individuare e isolare correttamente le monete nelle immagini. Uno dei punti di forza del progetto è la creazione di un dataset personalizzato. Attraverso un'attenta raccolta dei dati, è stato possibile sviluppare un set di immagini ampio e diversificato, contenente esclusivamente monete di 10 centesimi. L'utilizzo di un dataset personalizzato ha dimostrato di essere fondamentale per addestrare il modello in modo mirato e specializzato, garantendo risultati precisi e affidabili nella fase di detection che non sarebbero stati possibili sfruttando solamente il dataset reperito in rete. Riguardo ai futuri sviluppi del progetto, è stata identificata una chiara opportunità di espandere la rete per consentire la classificazione di una varietà più ampia di monete. Questo richiederà ulteriori sforzi nella raccolta di dati specifici per le nuove classi di monete e un'ulteriore fase di fine tuning del modello. L'ampliamento della rete rappresenta una prospettiva eccitante, in quanto consentirebbe di rendere il sistema ancora più versatile e in grado di affrontare diverse sfide di detection delle monete. Inoltre, come prospettiva per il miglioramento delle prestazioni, la proposta è quella di esplorare tecniche avanzate di ot-

timizzazione dei parametri del modello. Questo potrebbe coinvolgere l'uso di algoritmi di ricerca dei migliori iperparametri e l'applicazione di metodi di regolarizzazione per evitare l'overfitting dei dati. Si potrebbe inoltre valutare l'applicazione di modelli e versioni differenti della stessa rete o di architetture più sofisticate. Un'ultima osservazione va fatta sull'utilizzo di backbone diverse: si è reso noto il problema del training attraverso l'utilizzo di sole cpu che rende il processo molto più complesso e costoso in termini di tempo. Per ottenere risultati migliori e ridurre significativamente i tempi di addestramento delle deep neural networks, è necessario l'utilizzo di GPU (unità di elaborazione grafica). Le GPU sono state originariamente progettate per accelerare il rendering delle immagini e hanno dimostrato di essere altamente efficienti nell'esecuzione di operazioni matematiche parallele. Grazie alla loro architettura parallela, le GPU possono eseguire molti calcoli simultaneamente, consentendo un enorme aumento delle prestazioni rispetto alla CPU per i calcoli in quanto sono in grado di eseguire operazioni di floating-point su vettori di dati molto più velocemente rispetto alle CPU tradizionali. Visti i motivi citati ulteriori migliorie potrebbero essere fatte svolgendo le operazioni attraverso l'utilizzo di GPU.

Riferimenti bibliografici

[1] Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.
Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun

[2] Segment Anything Model.
Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, Ross Girshick

[3] Feature Pyramid Networks for Object Detection.
Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, Serge Belongie

[4] Benchmarking Detection Transfer Learning with Vision Transformers.
Yanghao Li, Saining Xie, Xinlei Chen, Piotr Dollar, Kaiming He, Ross Girshick

[5] Why Momentum Really Works.
Goh, Gabriel

[6] FASTER R-CNN PyTorch Implementations

[7] Pytorch FrameWork

[8] ImageNet dataset

[9] ONNX, Open Neural Network Exchange: The open standard for machine learning interoperability