UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Utilizzo del Deep Learning per estrarre informazioni in ambito fashion dai Social Media

Deep Learning approaches for Fashion Knowledge extraction from Social Media

Candidato:
Valerio Donnini

Relatore:
Prof. Emanuele FRONTONI

Correlatore:
Dr. Marina PAOLANTI

Anno Accademico 2020-2021

UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione

Utilizzo del Deep Learning per estrarre informazioni in ambito fashion dai Social Media

Deep Learning approaches for Fashion Knowledge extraction from Social Media

Candidato:
Valerio Donnini

Relatore:
Prof. Emanuele FRONTONI

Correlatore:
Dr. Marina PAOLANTI

Anno Accademico 2020-2021

Università Politecnica delle Marche
Facoltà di Ingegneria
Corso di Laurea in Ingegneria Informatica e dell'Automazione
Via Brecce Bianche – 60131 Ancona (AN), Italy

To my family

Sommario

Negli ultimi anni, i social network sono diventati una fonte preziosa di opinioni da parte degli utenti e l'analisi automatica dei contenuti è fondamentale. In particolare, Instagram fornisce un feedback in real time per brand e aziende del settore fashion, in quanto noti influencers riescono a creare dei veri e propri trend. Grazie alle tecniche di Intelligenza Artificiale e alle reti neurali di deep learning, le aziende che operano nel settore della moda possono individuare i pattern nei dati e costruire modelli in grado di prevedere trend futuri. In questo scenario, l'obiettivo di questa tesi è il riconoscimento automatico di fashion trend analizzando grandissime quantità di dati estratti da Instagram. Reti neurali specifiche per l'object detection, come SSD300 e YOLO, sono state addestrate ed ottimizzate e si è stati in grado di riconoscere e classificare oggetti e accessori fashion nelle immagini. Inoltre, un nuovo dataset del Made in Italy che sarà reso disponibile alla comunità internazionale, è stato raccolto ed annotato, e ad oggi rappresenta il principale dataset di riferimento nel settore.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Contesto

Il ruolo dei social media nella vita delle persone sta acquisendo sempre più importanza. Questo perchè i social media sono diventati sinonimo di:

- libertà di espressione -> ognuno può dire liberamente la sua opinione su una piattaforma piuttosto che un'altra, sempre ovviamente nei limiti del rispetto altrui;
- comunicazione globale -> persone presenti in parti opposte del globo possono comunicare senza limiti;
- fruizione di informazioni -> giornali e non mettono a disposizione le principali notizie;
- svago -> diverse persone utilizzano i social semplicemente per seguire le persone che ammirano e stimano.

Oltre all'aspetto sociale, stanno acquisendo sempre più importanza anche dal punto di vista commerciale, come dimostra la recente nascita della figura del social media manager. Per le aziende queste piattaforme sono diventate infatti la vetrina ideale per lanciare campagne di vendita e mettere in mostra le proprie creazioni e invenzioni. Addentrandosi nello specifico nel mondo fashion, settore che verrà trattato all'interno di questa tesi, ciò risulta essere ulteriormente amplificato. Ciò accade perchè persone che hanno un largo seguito, proprio per la loro influenza, riescono a creare trend a partire semplicemente da una foto postata con un indumento piuttosto che un altro. Per questo per le aziende risulta essere di vitale importanza monitorare il comportamento social degli influencer, in modo tale da captare con largo anticipo la richiesta del mercato, inteso come acquirenti.

1.2 Obiettivi

L'obiettivo principale della tesi è stato quello di riuscire a classificare e prevedere tendenze fashion a partire da dati raccolti da un social media, nello specifico Instagram.

Figura 1.1: Esempio di Object Detection con rete neurale specifica per questo task.

Per fare ciò si è utilizzato il Deep Learning, in quanto un task tipico risulta essere proprio quello della object detection, cioè l'andare a trovare all'interno di un'immagine la presenza o meno di un particolare oggetto. Nello specifico i trend che si vogliono cercare di andare a prevedere riguardano il mondo delle borse; riuscire quindi ad avere una o più reti in grado di localizzare e catalogare all'interno di specifiche immagini la presenza o meno del suddetto oggetto, ovviamente indipendentemente dalla forma e dalla dimensione. Per riuscire in questo, le reti utilizzate sono state allenate con un dataset, costruito a partire da post scaricati direttamente da Instagram con un opportuno programma che verrà analizzato in seguito.

1.3 Struttura della tesi

Parlando della struttura della tesi, nella prima parte nel capitolo 2, dopo una breve introduzione al mondo del Deep Learning e a ciò che vi sta intorno, si andrà a trattare quello che è lo stato dell'arte dell'object detection in termini di reti utilizzabili e le loro differenze principali; nella seconda parte invece si passerà ad analizzare quelli che sono i dataset attualmente disponibili in ambito fashion con le loro caratteristiche distintive in termini di tipologie di immagini, numero ecc.
Nel capitolo 3, inizialmente verrà descritto il dataset che si è utilizzato per allenare le reti, come è stato costruito e come è stato poi annotato. Verranno altresì descritti gli strumenti software preposti a questi obiettivi. Nella seconda parte verranno trattate in maniera approfondite le reti che sono state utilizzate per centrare il task di predizione trend, vedendo le differenti caratteristiche che le contraddistinguono e le diverse peculiarità. Si terminerà il terzo capitolo vedendo quelle che sono le metriche principali utilizzabili per commentare le prestazioni delle rete.
Nel capitolo 4 si discuterrano quelli che sono i risultati ottenuti e le performance delle due reti utilizzate, andando anche a vedere degli esempi pratici, ovvero l'output che si ottiene dalle reti fornendo in ingresso delle immagini target per il nostro obiettivo.

Nel capitolo 5 verranno tratte le conclusioni ed eventuali sviluppi futuri che potrebbero esserci nell'ambito della predizione di trend utilizzando il Deep Learing, ma anche con altri approcci.

Capitolo 2

Stato dell'arte

2.1 Computer Vision, Deep Learning e Object Detection

Diverse sono le reti adottate nel task di object detection, uno dei task principali del Deep Learning, branca dell'apprendimento automatico e dell'intelligenza artificiale. Prima di andare ad analizzare nello specifico queste reti, è bene andare a trattare in maniera generale i due argomenti dal quale poi la object detection deriva.
Per quel che riguarda la Computer Vision, può essere definita come l'insieme dei processi che vengono sviluppati al fine di riuscire a simulare quella che è la vista umana. Per fare ciò sono implementati dei sistemi di visione, cioè integrazione di componenti ottiche, elettriche e meccaniche che permettono di acquisire, salvare ed infine elaborare immagini. Si cerca quindi di ottenere come risultato il riconoscimento di determinate caratteristiche per varie finalità (classificazione, ecc). Compiti tipici della Computer Vision sono:

- recognition -> riconduzione a classi generiche di oggetti insieme alla loro posizione
- identification -> individuazione di un'istanza specifica di una classe
- detection -> individuazione e classificazione di particolari tipologie di oggetti all'interno di immagini

Applicazioni tipiche della Computer Vision risultano essere:

- riconoscimento difetti e rispetto delle tolleranze -> ciò permette di ridurre i costi e di effettuare un controllo della produzione fatto secondo criteri oggettivi e ripetibili
- orientamento, posizionamento e guida robot -> per gestire sistemi interfacciati a robot che si devono muovere nello spazio si deve introdurre l'acquisizione di informazioni attrvarso un sistema di visione artificiale
- controllo veicoli autonomi -> sistemi di visione artificiale possono sosituire in parte o completamente il lavoro di guida che dovrebbe essere svolto dal guidatore di un veicolo

- ambito medico -> possibilità di effettuare la diagnosi su di un paziente a partire da immagini ottenute attraverso, ad esempio, raggi X, risononanze ecc.
- riconoscimento facciale -> riuscire a riconoscere a partire da un volto se abbiamo di fronte una persona piuttosto che un'altra (basta pensare alle moderne tecnologie di sblocco degli smartphone)

Il Deep Learning è invece un sottoinsieme del Machine Learning, basato su reti neurali artificiali che non sono nient'altro che modelli di calcolo matematico-informatici che prendono spunto dal funzionamento delle reti neurali biologiche, cioè modelli identificati da interconnessione di informazioni. Si basa su una classe di algoritmi di apprendimento automatico che:

- attraverso diversi livelli di unità non linerari estraggono features e trasformano le immagini
- sono caratterizzati dal learning non supervisionato su multipli livelli gerarchici di features dei dati

Può essere utilizzato per gli obiettivi che si pone di raggiungere la Computer Vision motivo per cui classiche applicazioni risultano essere:

- classification -> classificare immagini in classi di appartenenza a seconda del soggetto principale di quest'ultima
- segmentation -> riconoscere all'interno di un'immagine le varie aree associate a soggetti diversi
- detection -> trovare e classificare all'interno di un'immagine particolari categorie di oggetti.

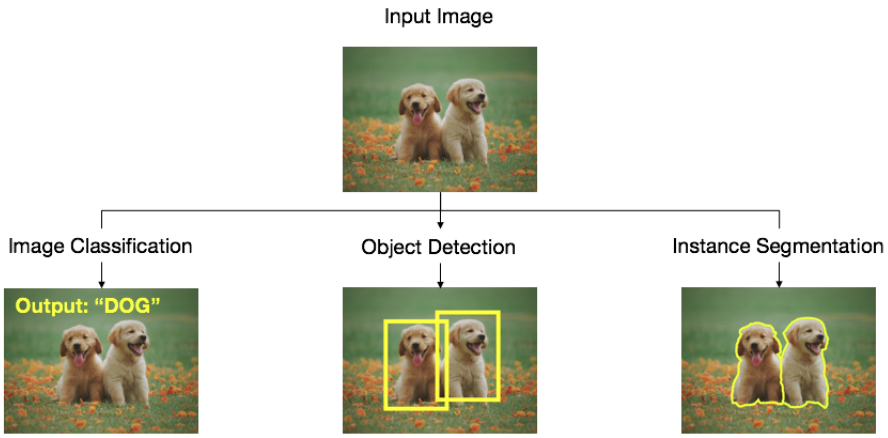

Figura 2.1: Task principali del Deep Learning

Dal punto di vista del funzionamento, con il Deep Learning si vanno a simulare i processi di apprendimento del cervello umano attraverso le reti artificiali. Queste reti venogno definite profonde proprio perchè sfruttano un maggior numero di livelli intermedi al fine di costruire più livelli di astrazione. Il passaggio da un livello ad

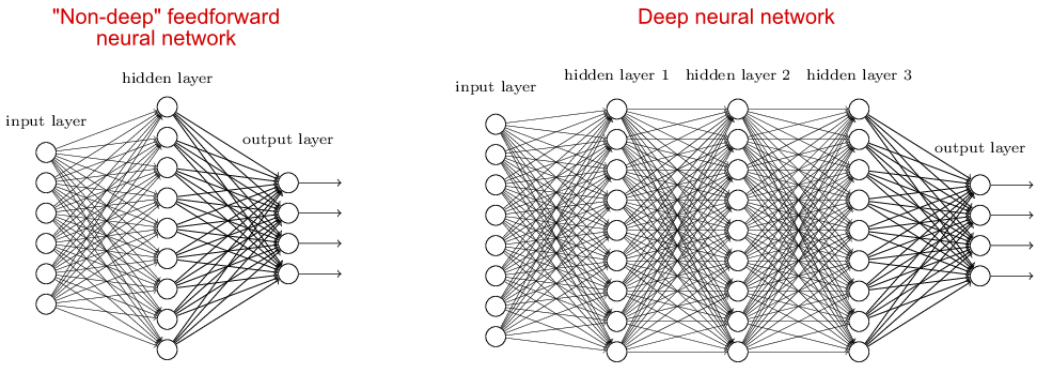

Figura 2.2: Differenza in termini di layer intermedi tra una rete profonda e una non.

un altro avviene attraverso l'operazione di convoluzione. Questa si basa su delle matrici, che prendono il nome di *kernel*, i cui valori possono essere inizializzati in diversi modi, ma che comunque si vanno a modificare durante il *training* della rete per avere le performance migliori. Durante questa fase si cerca di andare a ridurre

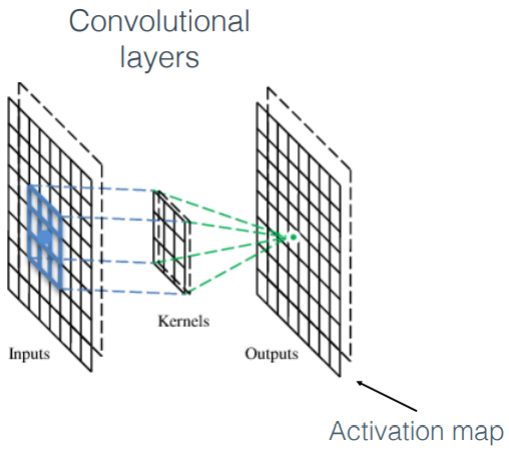

Figura 2.3: Esempio dell'operazione di convoluzione tra una feature map in input ed un kernel.

il più possibile l'errore tra il label predetto ed il label effettivo di date immagini. A seconda del task la loss può essere definita (per l'esempio i, la predizione p_i e il target t_i):

- Squared error (regression tasks) -> $L_i = (t_i - p_i)^2$
- Accuracy (accuracy tasks) -> $L_i = I(t_i = argmax_j p_{ij})^2$

- Cross entropy (durante il train per problemi di classificazione)
  -> $L_i = \sum_j t_{i,j} log p_{i,j}$

Nel passaggio da un livello intermedio ad un altro vengono aggiunte informazioni e analisi utili e ciò fornisce un enorme vantaggio per le reti profonde per andare a risolvere problemi di diversa complessità. L'altra faccia della medaglia è ovviamente il prezzo da pagare sia da un punto di vista economico che da un punto di vista computazionale. La Object Detection sfrutta le potenzialità del Deep Learning al fine di riuscire a localizzare all'interno di immagini determinate categorie di oggetti, fornendo quindi in uscita quelle che sono le coordinate che vanno ad identificare il box contenente l'elemento ricercato. Durante il train, dunque si necessiterà di un dataset che risulti essere labelizzato, in modo tale che si possa andare a definire in maniera generale la loss come differenza tra il bounding box predetto e quello reale ed inoltre, per rendere la rete insensibile alle dimensioni dell'oggetto cercato, abbia numerosi samples a diverse scale di quest'ultimo. Qualora non si abbia a disposizione un numero elevato di immagini all'interno del dataset, si possono anche andare ad utilizzare delle tecniche di data augmentation, tecniche che permettono a partire da un'immagine di ricavarne un numero n per il train, ad esempio andando a ruotarla, specchiarla, rifletterla ecc.

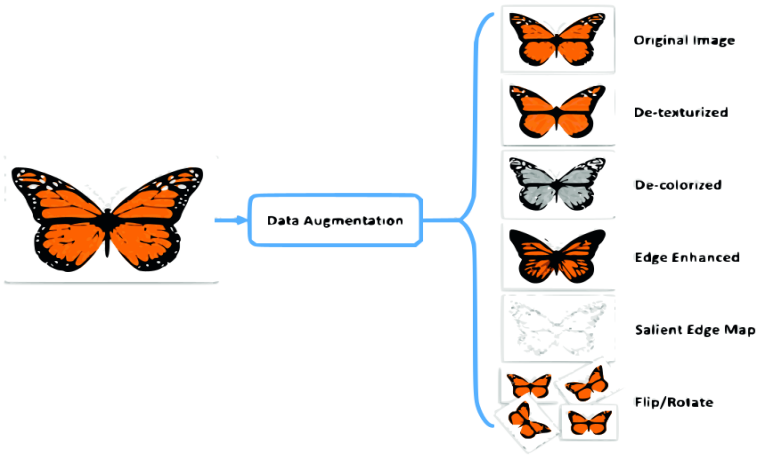

Figura 2.4: Tipologie di data augmentation

2.2 Deep Neural Networks per Object Detection

Diverse sono le reti che possono essere utilizzate per centrare il task della Object Detection. In questa sezione si andranno ad analizzare:

- R-CNN [1] [1];
- Fast R-CNN [1] [2]
- Faster R-CNN [1] [3]

- Cascade R-CNN [1] [4]
- SSD (Single Shot Detector) [5];
- YOLO (You Only Look Once) [6].

2.2.1 R-CNN

Il *paper* riguardante questa rete è stato pubblicato nel 2014 e dimostra come reti convoluzionali neurali possono essere in grado di riuscire a localizzare oggetti all'interno di immagini. La localizzazione e identificazione di oggetti avviene attraverso 3 fasi:

1. Si vanno a generare quelle che sono le regioni di interesse
2. Per ogni regione si va ad estrarre un $feature\ vector$
3. Attraverso l'ausilisio di SVM[2] specifici si vanno a classificare le varie regioni

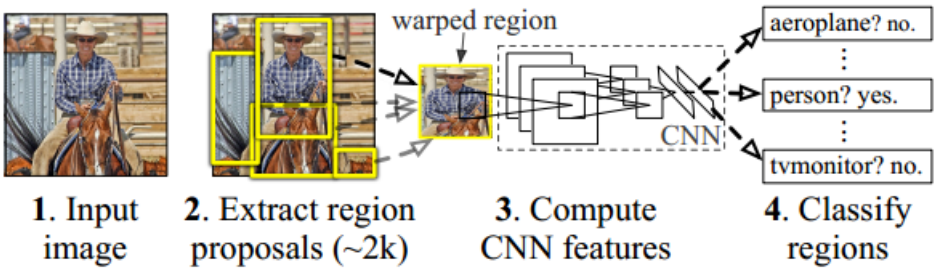

Figura 2.5: Architettura R-CNN

Le regioni di interesse che sono circa 2000 per ogni immagine, possono essere definite attraverso diverse tecniche. Nello specifico si va ad utilizzare un algoritmo che prende il nome di *selective search* che estrae quelle regioni che al loro interno si pensa presentino degli oggetti; ognuna di queste risulta essere definita da 4 valori, una coppia x e y che identifica quello che è il centro dell regione, seguita da altezza e larghezza di quest'ultima. Una volta estratte le regioni, queste vengono standardizzate

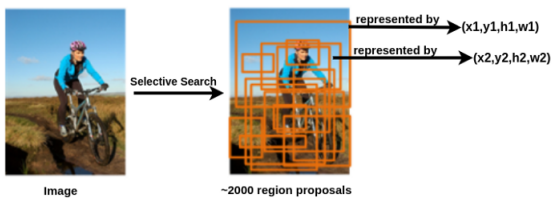

Figura 2.6: Regioni di interesse indivuduate attraverso il selective search

[1] RCN-N (Regional Convolutional Neural Network)
[2] "SVM sta per Software Vector Machine, algoritmi che possono essere utilizzati per la classificazione di immagini

in termini di dimensioni e poi date alla CNN che determina, per ognuna di esse $feature - vector$ di dimensione 4096.
Infine ogni regione è classificata attraverso l'utilizzo dei SVM che determinano lo score di appartenenza di quest'ultima ad ognuna delle classi con le quali la rete è stata allenata. Assegna quindi a quella specifica regione la classe che ha totalizzato lo score più alto.

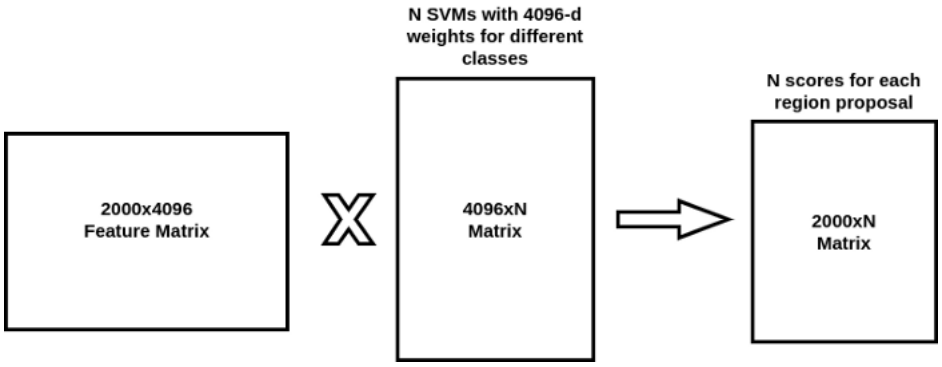

Figura 2.7: Processo di identificazione della classe delle diverse regioni

2.2.2 Fast R-CNN

La criticità principale della R-CNN è quella di dover andare ad analizzare sempre 2000 regioni per ogni immagine, sia se queste risultino essere "interessanti" o meno. Ciò ovviamente va poi ad inficiare in maniera negativa su quelli che sono i tempi di esecuzione della stessa rete. Per questo si è pensato alla Fast R-CNN che non è nient'altro che una evoluzione della rete sopracitata. L'architettura della Fast R-CNN risulta essere la seguente:

- Region proposal network
- Feature extraction usando CNN
- Roi pooling layer
- Classification e localization

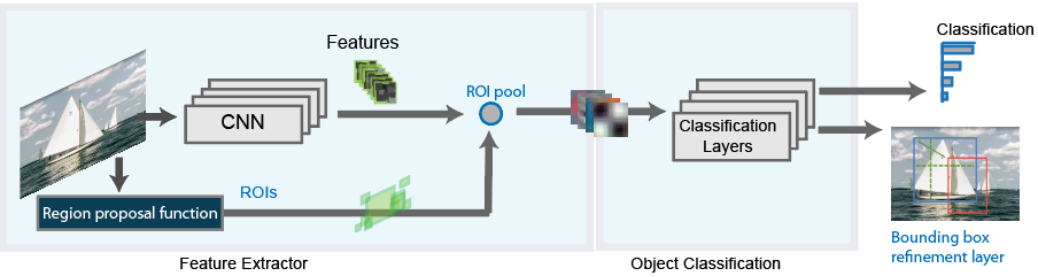

Figura 2.8: Architettura Fast R-CNN

I primi due blocchi della rete servono per estrarre una $feature\ convolutional\ map$ a partire dall'intera immagine in input, a differenza di quanto avveniva per la R-CNN dove per ognua delle regioni di interesse veniva poi estatto un $feature\ vector$. La map ottenuta si va quindi a combinare con la Region Proposal Network formando un $feature\ vectors$ di lunghezza opportuna nel Roi pooling layer. Ognuno dei $feature\ vectors$ è poi dato ai moduli di classificazione e localizzazione. Il primo va a classificare tra le K+1 classi disponibili (il +1 è dovuto al Background), mentre il secondo va a determinare il bounding box per ognuna delle K classi di oggetti.

2.2.3 Faster R-CNN

La Faster R-CNN è una rete utilizzabile per la object detection che risulta essere ancora un passo avanti rispetto alla Fast R-CNN. Nella Faster R-CNN, le regioni di interesse risultano essere generate durante le sue operazioni di convoluzione; ciò permette un boost in termini di efficienza e velocità nel task della object detection.

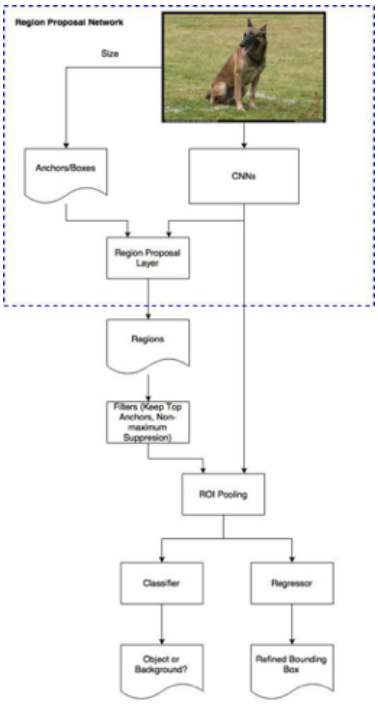

Figura 2.9: Architettura Faster R-CNN

Inizialmente si vanno a definire delle $ancors$ di base, che sono il punto di partenza per i bounding box. A partire da questi, supponendo un immagine di dimensione H X F con stride F, il centro di queste ancore può essere situato in ognuno di questi punti [(0,0), (F, 0), (0, F), (F, F) .. (H x F, W x F)]. Ciò determina un numero totale di H X W X 9 $base\ ancors$ per immagine.

L'RPN poi utilizza nei due livelli paralleli convuluzionali le $feature\ maps$ ottenute attraverso l'$image\ feature\ extractor$ ricevendo in uscita due set di output. Il primo layer convoluzionale determina il bounding box regressor outputs, che non sono direttamente le coordinate dei bounding box, bensì $offset$ e $scales$ che sono poi applicati alle ancore delle immagini per rifinire le predizioni. L'RPN in aggiunta determina in uscita anche un output di classificazione, indicando se un bounding box è in primo piano o è sullo sfondo.

A seguito troviamo il $Non-Maximun-Suppression$ layer che serve a filtrare dei bounding box che potrebbero essere ridondanti. Bounding box associati ad una stessa classe, risultano essere prima ordinati per score di appartenenza a quella classe e poi confrontati tra loro andando a determinare l'IoU ($Intersection\ over\ Union$).

Se questo parametro risulta essere al di sopra di usa soglia prefissata, il bounding box con la score di appartenenza più basso viene scartato.

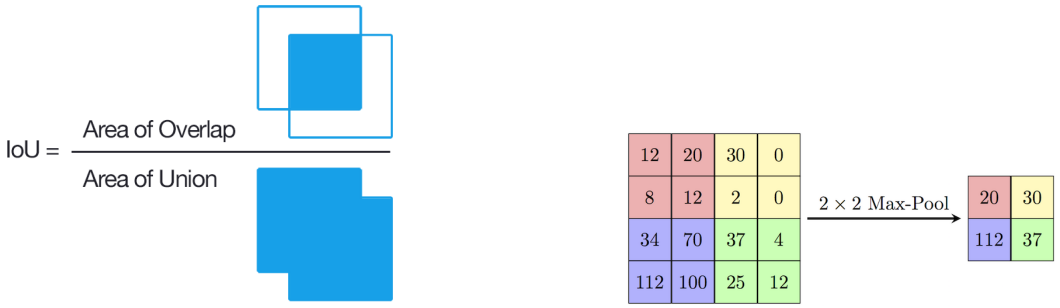

Figura 2.10: Intersection over Union e Max Pooling

Infine le regioni di interesse rimanenti vengono passate al ROI pooling layer che determina delle feature maps di dimensione opportuna che vengono splittate e su ognuna delle quali viene applicato il Max Pooling [3] (ciò viene fatto per facilitare il passaggio dell'intera immagine diirettamente al prossimo step). Nell'ultimo step la VGG-Head va a determinare gli offset/scales dei bounding box da andare ad applicare ad essi al fine di rifinire le predizioni.

2.2.4 Cascade R-CNN

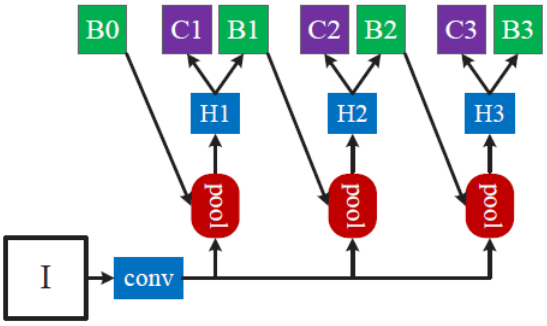

Figura 2.11: Architettura Cascade R-CNN

La Cascade R-CNN è una rete abbastanza recente, nata nel 2018, che vuole essere un miglioramento rispetto alla Faster R-CNN. Durante la localizzazione dei bounding box, utilizzando una qualunque rete utile per il task della object detection, le prestazioni calano in maniera significativa nel momento in cui si vanno ad aumentare troppo le soglie di IoU. Ciò è dovuto principalmente a due fattori:

- *Overfitting* presente durante la fase di training che fa si che all'aumentare delle soglie di IoU, il numero di positivi decresca troppo

[3] Tecnica che permette di ricavare da una matrice M X M, una matrice N X N dove $M>N$ e ogni cella della seconda matrice è caratterizzata dal massimo valore della matrice S X S presa in considerazione nella prima matrice

- Differenza in termini di soglie di IoU utilizzate tra la fase di train e la fase di test

La Cascade R-CNN cerca, attraverso la sua particolare architettura di risolvere questi problemi. Osservandola si può ben notare come siano presenti tre differenti teste (H1,H2,H3) per la detezione ($region - of - interest\ detection\ sub - network$). Ognuna di loro presenta delle soglie di IoU differenti, dalla più piccola alla più grande ed inoltre la regressione in cascata è una procedura di ricampionamento atta a fornire al livello successivo dei buoni positivi. Non sono presenti inoltre delle differenze in termini di soglie di IoU scelte per affrontare il train e poi il test della rete.

2.2.5 SSD (Single Shot Multibox Detector)

La SSD è una rete il cui paper è stato pubblicato nel 2016 riuscendo ad ottenere nuovi record in termini di performance e precisone circa quella che è la object detection. Già dal nome si può iniziare a comprendere parte di quella che è l'architettua che la caratterizza ed il suo funzionamento:

- Single Shot -> in un sol colpo, cioè "facendo passare" un'immagine in inuput attraverso la rete una sola volta, questa riesce a centrare il task di localizzazione e classificazione di un determinato oggetto target
- Mulitbox -> è la tecnica di regressione che viene utilizzata per andarea determinare i bounding box
- Detector -> la rete è in grado di localizzare oggetti, andando a specificare il bounding box di questi ultimi e allo stesso tempo è in grado anche di classificare questi oggetti

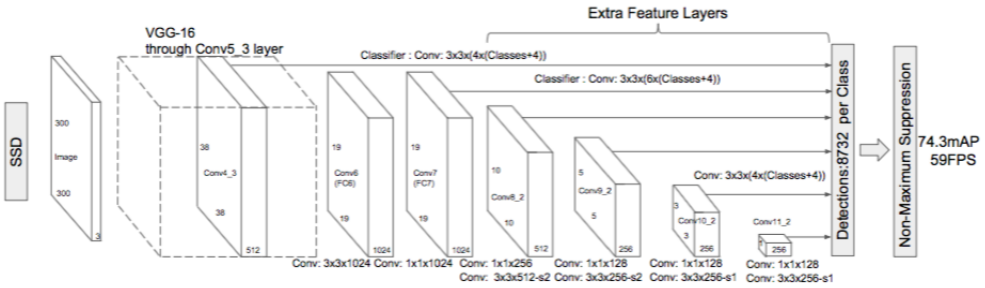

Figura 2.12: Architettura SSD

Dal punto di vista dell'architettura, questa rete basa il suo funzionamento sulla struttura della VGG16, una rete utilizzabile per la classificazione di imamgini, andando a rimpiazzare i livelli fully connected con altri livelli convoluzionali preposti per il task della object detection, in grado quindi di andare ad inidivuare il bounding box sull'oggetto interessato. Ci sono due diverse tipologie di SSD, che sono fortmente dipendenti dal formato che deve avere in ingresso l'immagine da analizzare:

- SSD300 -> la cui immagine in input deve avere dimensione 300x300 pixels
- SSD500 -> la cui immagine in input deve avere dimensione 500x500 pixels

Andando a testare le due reti sul dataset Pascal VOC2007, le principali differenze riscontrate sono in termini di performance in quanto la prima, risulta essere leggermente meno precisa (74,3% mAP), però mantiene un frame rate di 59 fps. La SSD500 invece totallizza una precison del 76,8%, a discapito della speed che risulta essere solo di 22fps. Per quel che riguarda l'allenamento, come avviene nello specifico la object detection ecc.,si intrerà nello specifico nel capitolo 3, in quanto la SSD300 riuslta essere una delle due reti utilizzate per centrare l'obiettivo della suddetta tesi.

2.2.6 YOLO (You Only Look Once)

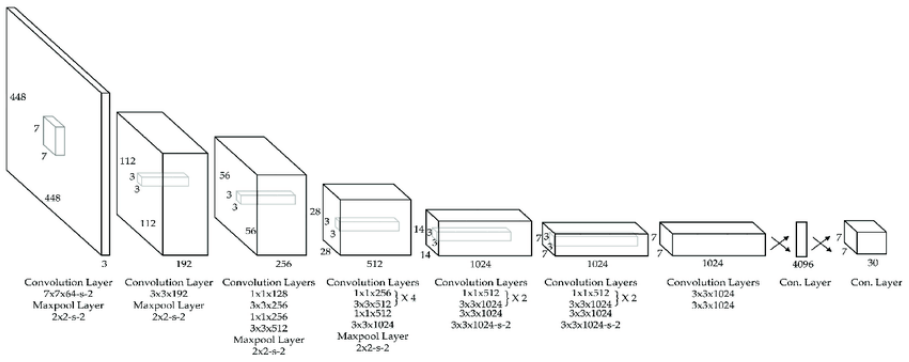

Figura 2.13: Architettura YOLO

Il paper della rete YOLO è stato publlicato nel 2015 e deve il suo nome al fatto che l'immagine che viene data in input risulta essere analizzata in un solo passaggio all'interno della stessa. A differenza di reti come la Fast R-CNN o la R-CNN, la rete non prende in esame delle regioni di interesse, bensì l'analisi viene fatta a livello dell'intera immagine. Per riuscire in questo obiettivo sfrutta le potenzialità di un'unica rete neurale e a partire dalle caratteristiche dell'immagine riesce a determinarela posizione di più oggetti, identificandone il bounding box e classificandoli, in maniera simultanea. Fatto questo si trova il $Non - Maximun - Suppression$ layer che al solito va a elidere box ridondanti. I principi vantaggi di questa rete risultano essere:

- l'estrema velocità di analisi di immagini
- l'essere in grado in fase di train e test di riuscire a vedere l'intera immagine, in modo tale da potersi salvare informazioni di contesto riguardo la classe dei vari oggetti
- il riuscire ad apprendere la rappresentazione generale di oggetti, cosa che porta poi a superare in fase di test su stessi dataset allo stato dell'arte le performance di altre reti, che non hanno questa feature

Ancher per questa rete, come per la SSD, verranno analizzati in dettaglio le metodiche del train, il funzionamento, ecc. nel prossimo capitolo in quanto risulta essere una delle due reti utilizzate nella tesi di cui comparare le performance.

2.3 Fashion Datasets

In letteratura, i dataset attualmente disponibili nel mondo fashion risultano essere i seguenti:

- Modanet [7];
- DeepFashion [8];
- DeepFashion2 [9]
- Fashion-mnist [10].

Nelle Sezioni successive saranno dettagliate le caratteristiche di questi dataset.

2.3.1 Modanet

Modanet è un dataset caratterizzato da annotazioni associate ad immagini RGB. Ogni immagine è fornita con annotazioni associate a diversi bounding box ognuno dei quali è labelizzato con una tra 13 classi disponibili. Per quel che riguarda la

Label	Description	Fine-Grained-categories
1	bag	bag
2	belt	belt
3	boots	boots
4	footwear	footwear
5	outer	coat/jacket/suit/blazers/cardigan/ sweater/Jumpsuits/Rompers/vest
6	dress	dress/t-shirt dress
7	sunglasses	sunglasses
8	pants	pants/jeans/leggings
9	top	top/blouse/t-shirt/shirt
10	shorts	shorts
11	skirt	skirt
12	headwear	headwear
13	scarf and tie	scarf and tie

Tabella 2.1: Classi degli indumenti in Modanet

struttura dei dati, risulta essere del tutto similiare rispetto a quella che si trova sul dataset di COCO.

```
{
'info' : info, 'images' : [image], 'annotations' : [annotation], 'licenses' : [license],'year': year, 'categories': [category], 'type': type
}

info{
'version' : str, 'description' : str, 'contributor' : str, 'date_created' : datetime,
}

image{
'id' : int, 'width' : int, 'height' : int, 'file_name' : str, 'license' : int
}

license{
'id' : int, 'name' : str, 'url' : str,
}

annotation{
  'area': int,
  'bbox': [x,y,width,height],
  'segmentation': [polygon],
  'image_id': int,
  'id': int,
  'category_id': int,
  'iscrowd': int
}
category{
  'supercategory': str, 'id': int, 'name': str,
}
```

Figura 2.14: Datatype Modanet

2.3.2 DeepFashion

DeepFashion è un altro dataset di caratterizzato da immagini di indumenti che vanta le seguenti peculiarità:

- contiene oltre 800'000 immagini che spaziano tra immagini di vendita ad immagini fatte da acquirenti, in modo tale da avere una varietà più ampia possibile
- è annotato con ricche informazioni sugli indumenti indossati. Ogni immagine è infatti labelizzata con 50 categorie, 1000 attributi descrittivi, bounding box e *landmarks* dei vestiti
- contiene oltre 300'000 coppie di immagini cross-pose/cross-domain

Categorie vestiti				
Anorak	Blazer	Blouse	Bomber	Button-Down
Cardigan	Flannel	Hlater	Henley	Hoodie
Jacket	Jersey	Parka	Peacot	Poncho
Sweater	tank	Tee	Top	Turtleneck
Capris	Chinos	Culottes	Cutoffs	Gauchos
Jeans	Jeggins	Jodhpurs	Joggers	Leggins
Sarong	Short	Skirt	Sweatpants	Sweatshorts
Trunks	Caftan	Cape	Coat	Coverup
Dress	Jumpsuit	Kaftan	Kimono	Nightdress
Onesie	Robe	Romper	Shirtdress	Sundress

Tabella 2.2: Classi degli indumenti in DeepFashion

2.3.3 DeepFahion2

DeepFashion2 è un dataset in ambito fashion caratterizzato da oltre 450'000 immagini, suddivise tra 13 categorie di vestiti. Nel suo insieme contiene oltre 800'000 indumenti, opportunamente labelizzati e classificati, con ognuno di essi identificato da 9 parametri. Il dataset già di per sè risulta essere suddiviso tra train, validation and test set.
Per quel che riguarda l'organizzazione deli dati, ogni immagine è caratterizzata da un ID di 6 cifre, seguito dall'estensione della stessa (es. 000001.jpg), e l'annotazione correlata ha in comune la prima parte del nome, seguita però da *.json*. Ogni file di annotazione risulta avere questa struttura:

- source -> identifica la provenienza dell'immagine (proveniente da shop o da consumatori)
- pair-id -> un numero che identifica lo shop o il consumer dal quale proviene l'immagine. Immagini fornitè da uno stesso shop/consumer presentano lo stesso pair-id
  - oggetto1
    - category_name -> la categoria di appartenenza dell'oggetto
    - category_id -> un numero associato al category_name
    - style -> un numero per distinguere immagini diverse con lo stesso pair-id
    - bounding_box -> 4 punti [x1,y1,x2,y2], dove la prima coppia rappresenta l'angolo in alto a sx del box, mentre la seconda quello in basso a dx
    - landmarks -> insieme di punti e di un parametro v per ognuno di essi che identifica la visibilità [x1,y1,v1,..,xn,yn,vn]
    - segmentation -> insieme di punti che indentificano differenti poligoni
    - scale -> numero compreso tra 1 e 3 che identifica la grandezza dell'oggetto
    - occlusion -> numero compreso tra 1 e 3 che identifica il livello di occlusione dell'oggetto
    - zoom_in -> numero compreso tra 1 e 3 che identifica il grado di zomm, se presente
    - viewpoint -> numero compreso tra 1 e 3 che identifica la tipologia di punto di vista
  - oggetto2
  -

–

– oggettoN

Il dataset inoltre viene fornito con del codice che genera annotazioni del tutto similiari alla struttura di quelle apparteneti alle immagini del dataset di COCO. Per quel che riguarda le stats del dataset, risultano essere riportate sulla seguente tabella. In

	Training	Validazione	Test	Totale
Images	390'884	33'669	67'342	491'895
Bboxes	636'624	54'910	109'198	800'732
Landmarks	636'624	54'910	109'198	800'732
Masks	636'624	54'910	109'198	800'732
Pairs	636'624	54'910	109'198	800'732

Tabella 2.3: Stats DeepFashion2

figura le statistiche di diverse varianti e il numero delgli oggetti di ognua della 13 classi.

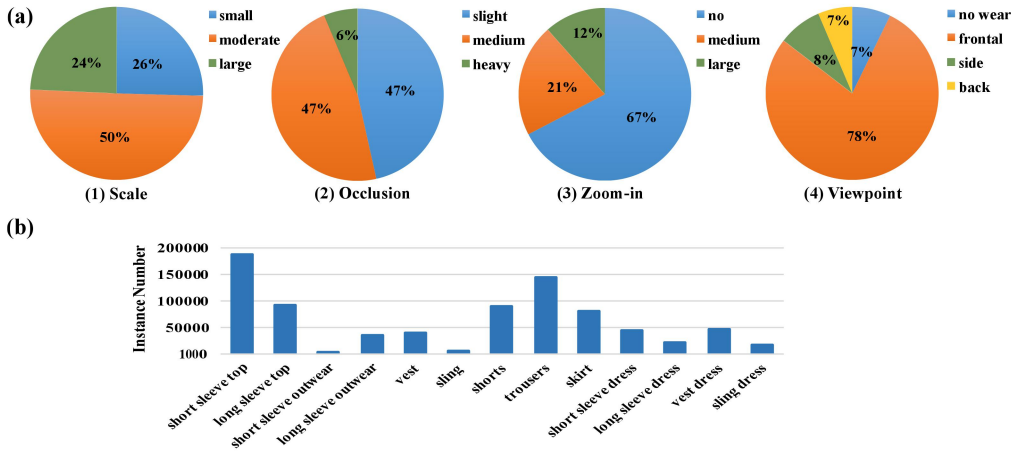

Figura 2.15: Stats delle classi e di varianti

2.3.4 Fashion-mnist

Fashion-mnist è un dataset in ambito fashion che raccoglie quelle che sono le immagini degli articoli in vendita su Zalando. In totale presenta circa 70'000 esempi suddivisi tra training (60'000) e test (10'000) ed il labelizing riguarda 13 differenti classi di vestiti. La dimensione di tutte le immagini è di 28X28 e risultano essere tutte in bianco a nero.

Label	Descrizione
0	T-shirt/Top
1	Trousers
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Sneaker
7	Bag
8	Ankle boot

Tabella 2.4: Classi Fashion-mnist

Capitolo 3

Materiali e Metodi

In questo capitolo si andranno ad affrontare tre aspetti fondamentali per lo sviluppo della tesi:

- Discussione circa il dataset interessato (creazione, descrizione, software utilizzati ecc.)
- Reti utilizzate per il task della detection delle borse (YOLO, SSD300)
- Metriche utilizzate per la descrizione delle performance delle reti

3.1 VRAI Fashion Dataset

Per poter centrare il task di localizzazione e classificazione di borse all'interno di immagini, si è dovuto allenare le reti utilizzate con un dataset proprietario, in quanto inizialmente si è pensato di utilizzare il dataset Modanet, ottenendo però scarsi risultati, ciò dovuto principalmente al fatto che all'interno del suddetto dataset, fossero presenti pochi esempi delle immagini target per lo sviluppo della tesi, ovvero immagini con borse in primo piano di diversa dimensione e tipologia.
Per questo è stato collezionato un dataset proprietario, caratterizzato interamente da foto di borse e generato a partire da immagini reali, in quanto immagini presenti su post caricati su Instagram. Per riuscire a creare questo dataset, si è scritto un particolare codice Python e sono stati utilizzati determinate chiavi di ricerca che fossero facilmente associabili ad immagini con borse. Prima di andare a vedere questo script nel dettaglio, si può fare una disamina di quelle ciò che è stato utilizzato per farlo funzionare:

- *Instaloader* -> particolare modulo Python che permette, a seguito del login su Instagram con un determinato account, il download dei post associati ad un determinato profilo o ad una determinata chiave di ricerca, impostando condizioni di ricerca
- *Heroku* -> Piattaforma cloud di programmazione che permette di avviare applicazioni online. Permette di aggirare il problema per il quale se lo script per

la creazione del dataset fosse stato avviato in multithreading sul server dell'UNIVPM, si sarebbe incorso nel ban temporaneo dell'indirizzo ip di quest'ultimo, in quanto si sarebbero fatte molte richieste ad Instagram.

Le chiavi di ricerca utilizzate per la ricerca dei post target, risultano essere stati questi hashtag:

Hashtag di ricerca				
BAULETTO	CLUTCH	fay	HOBO	Hogan
hoganbag	MARSUPIO	rogervivier	rogervivierbag	rogervivierbags
SACCA	SECCHIELLO	SHOPPING	TODS	todsbag
todsbags	todsbauletto	TRACOLLA	ZAINO	

Tabella 3.1: Hashatg utilizzati per la ricerca di post da scaricare

Di fondamentale importanza per la perfetta riuscita del download delle immagini è stato il riuscire a capire come creare numerosi account Instagram, account che nel giro di poco tempo venivano bloccati a causa delle molte richieste fatte al social. Per aggirare questo problema quello che si è fatto è stato andare a creare i diversi account, appoggiandosi a dei servizi di email temporanee e soprattutto utilizzare la procedura di creazione account di Instagram messa a disposizione sull'applicazione; questo perchè utilizzando uno smartphone, si poteva utilizzare la sua connessione dati per collegarsi ai server del social e quindi semplicemente spegnendola e riaccendendola, si avrebbe avuto un indirizzo IP pubblico diverso dal quale sarebbe arrivate la nuova richiesta di creazione al server di Instagram. Ciò non succede se si prova a fare lo stesso procedimento utilizzando ad esempio la rete Wi-fi domestica, in quanto l'indirizzo IP pubblico dato al modem di casa è fisso e ciò si traduce in un blocco del suddetto IP nella creazione di account da parte di Instagram dopo l'aver fatto un paio di volte la procedura.
Analizzando lo script utilizzato per il download delle immagini, questo risulta essere diviso nelle suddette parti:

- FASE 1: recupero del file di sincronizzazione ('sinc.csv')
- FASE 2: tentativo di connessione ad Instagram tramite Instaloader con un determinato account.
- FASE 3-A (Account selezionato funzionante): inizio download dei post che soddisfano i requisiti in termini di data del post. (Si va alla fase 4)
- FASE 3-B (Account selezionato non funzionante): sospensione e selezionamento dell'account successivo. (Si torna alla fase 2)
- FASE 4: update del file di sincronizzazione 'sync.csv'.
- FASE 5: upload post scaricati sul server.

Nella fase 1, quello che si va a fare è recuperare il file 'sync.csv' dal server. Questo file risulta essere di fondamentale importanza per il corretto funzionamento dello scritp, in quanto al suo interno contiene informazioni su:

- l'ultimo account utilzzato per il download dei post, l'ultimo hashtag utilizzato per la ricerca dei post
- l'ultimo hashtag utilizzato per la ricerca dei post
- il numero dei post scaricati con l'ultimo hashtag

Dopo aver quindi recuperato i file necessari dal server (anche i due file contenenti la lista degli account e gli hashtag da utilizzare), si vanno a recuperare le ultime informazioni informazioni inserite nel file sync.csv, riempiendo opportune variabili.

```
if __name__ == '__main__':

    # create sftp connection and define the folder of sync
    REMOTE_SYNC_FOLDER =
        '/disks/disk3/MarcoMameli/Social/FashionExtractor/SYNC/'
    REMOTE_BASE_FOLDER =
        '/disks/disk3/MarcoMameli/Social/FashionExtractor/HashtagDownloading/'
    ssh_client = paramiko.SSHClient()
    ssh_client.set_missing_host_key_policy(AutoAddPolicy())
    ssh_client.connect(hostname='193.205.130.253', port=30022,
        username='vrai', password='vrai_2019')
    sftp_client = ssh_client.open_sftp()

    # opening the sync file
    args = parser.parse_args()
    SYNC_FILE = str(args.sync_file)
    HASHTAG_FILE = str(args.hashtag_file)
    ACCOUNT_FILE = str(args.account_file)
    NUMBER_OF_POSTS = int(args.number_post)
    START_DATE = str(args.start_date)
    END_DATE = str(args.end_date)

    print('SYNC OPERATION: Get the sync file and the necessary files to
        configure instaloader')
    # download sync file from remote server
    sftp_client.get(REMOTE_SYNC_FOLDER + SYNC_FILE, './' + SYNC_FILE)
    sftp_client.get(REMOTE_SYNC_FOLDER + HASHTAG_FILE, './' + HASHTAG_FILE)
    sftp_client.get(REMOTE_SYNC_FOLDER + ACCOUNT_FILE, './' + ACCOUNT_FILE)
    print('SYNC OPERATION: download completed')

    print('CONFIGURATION: Get the actual data and update it for the future
        instaloader execution')
    if not SYNC_FILE.endswith('csv'):
```

```
        raise exit(127)
    try:
        # opening the downloaded file
        sync_dataframe = pd.read_csv(SYNC_FILE, sep=';')
        if len(sync_dataframe) == 0:
            raise pd.errors.EmptyDataError
        # loading the last information
        last_account =
            sync_dataframe.tail(1)['account_id'].values.tolist()[-1]
        last_hashtag =
            sync_dataframe.tail(1)['hashtag_id'].values.tolist()[-1]
        # time format for datetime: %m%d%Y%H%M%S
        last_UTC_time =
            str(sync_dataframe.tail(1)['last_time'].values.tolist()[-1])
        # take the number of already downloaded posts:
        last_downloaded_posts =
            int(sync_dataframe.tail(1)['downloaded_posts'].values.tolist()[-1])
    except pd.errors.EmptyDataError:
        # The sync is empty this means that it is the first time of the
            file is opening
        print('SYNC OPERATION: The sync.csv file is empty')
        print('CONFIGURATION: The data is initialized to default values')
        # start from the first account
        last_account = -1
        # start from the first hashtag
        last_hashtag = -1
        # the last search datetime is the END_DATE
        # time format for datetime: %m%d%Y%H%M%S
        last_UTC_time = END_DATE
        # the number of downloaded posts is set to 0
        last_downloaded_posts = 0


    # opening hashtag file
    hashtag_dataframe = pd.read_csv(HASHTAG_FILE, sep=',')
    # get hashtag for the current process
    try:
        if not((int(last_hashtag) + 1) >
            hashtag_dataframe.tail(1)['ID'].values.tolist()[-1]):
            actual_hashtag = hashtag_dataframe.loc[hashtag_dataframe['ID']
                == int(last_hashtag) + 1, ['hashtag']].values.tolist()[-1]
            new_hashtag = str(int(last_hashtag) + 1)
        else:
            print('INSTALOADER: The last hashtag is the last restart from
                the first')
            actual_hashtag = hashtag_dataframe.loc[hashtag_dataframe['ID']
                == 0, ['hashtag']].values.tolist()[-1]
```

```
            new_hashtag = str(0)
    except KeyError:
        print('INSTALOADER: The last hashtag is the last restart from the
            first')
        actual_hashtag = hashtag_dataframe.loc[hashtag_dataframe['ID'] ==
            0, ['hashtag']].values.tolist()[-1]
        new_hashtag = str(0)

    # opening accounts file
    account_dataframe = pd.read_csv(ACCOUNT_FILE, sep=',')
    # get the account to be used for the current process
    try:
        if not((int(last_account) + 1) >
            account_dataframe.tail(1)['ID'].values.tolist()[-1]):
            actual_account = account_dataframe.loc[account_dataframe['ID']
                == int(last_account) + 1, ['ACCOUNT',
                'PW']].values.tolist()[-1]
            new_account = str(int(last_account) + 1)
        else:
            print('INSTALOADER: The last account is the last restart from
                the first')
            actual_account = account_dataframe.loc[account_dataframe['ID']
                == 0, ['ACCOUNT', 'PW']].values.tolist()[-1]
            new_account = str(0)
    except KeyError:
        print('INSTALOADER: The last account is the last restart from the
            first')
        actual_account = account_dataframe.loc[account_dataframe['ID'] ==
            0, ['ACCOUNT', 'PW']].values.tolist()[-1]
        new_account = str(0)

    print('CONFIGURATION: Data configuration terminate')

    # collect the actual time from utc:
    actual_utc_time = datetime.utcnow().strftime('%Y%m%d%H%M%S')
```

Nella fase 2 invece, dopo aver aggiornato il file sync.csv con le ultime occorrenze (account, hashtag e ora utilizzate il download corrente) ed aver configurato in maniera oppurtuna un'istanza della classe Instaloader, si prova la connessione ad Instagram con l'account scelto e si tenta il recupero dei post associati all'hashtag selezionato.

```
future_sync: dict = {
        'account_id': new_account,
        'hashtag_id': new_hashtag,
        'last_time': actual_utc_time,
        'downloaded_post': str(last_downloaded_posts),
    }
```

```
# writing the last information on the remote file directly:
print('SYNC OPERATION: Update sync file on the remote server')
with sftp_client.open(REMOTE_SYNC_FOLDER + SYNC_FILE, 'a+',
    bufsize=2**20) as remote_sync:
    remote_sync.write(future_sync['account_id'] + ';' +
        future_sync['hashtag_id'] + ';' + future_sync['last_time']
                    + ';' + future_sync['downloaded_post'] + '\n')
    remote_sync.flush()
print('SYNC OPERATION: Updated sync file on the remote server')

# start instaloader with the actual account and the hastag.
print('INSTALOADER: Start configuration')
instaloader_instance = instaloader.Instaloader(download_videos=False,
    max_connection_attempts=1,
                                            download_geotags=True,
                                                download_comments=False,
                                                compress_json=False)
# checking the session presence on the remote server:
session_file = 'session-' + actual_account[0]
print('SYNC OPERATION: Try to download the session file')
try:
    sftp_client.get(REMOTE_SYNC_FOLDER + session_file, './' +
        session_file)
    # session file received correctly
    print('SYNC OPERATION: The session file is recovered')
except FileNotFoundError:
    print('SYNC OPERATION: The session file does not exist')

print('INSTALOADER: Connection')
try:
    print('INSTALOADER: Traying connection with username and password')
    instaloader_instance.login(user=actual_account[0],
        passwd=actual_account[1])
    # saving the session to a file
    instaloader_instance.save_session_to_file(filename=session_file)
    print('SYNC OPERATION: upload session to remote')
    sftp_client.get('./' + session_file, REMOTE_SYNC_FOLDER +
        session_file)
    print('SYNC OPERATION: uploaded session to remote')
except:
    # the login with username and password fail
    instaloader_instance.load_session_from_file(username=actual_account[0],
        filename=session_file)
    print('INSTALOADER: Connected with session file')
    print('INSTALOADER: Query for hashtag posts')
```

```
# recovering posts from instagram
try:
    instaloader_hashtag_instance =
        instaloader.Hashtag.from_name(instaloader_instance.context,
        actual_hashtag[0])
except instaloader.QueryReturnedBadRequestException:
    print('INSTALOADER: account locked please reactivate it')
    # TODO insert an email sender client to notify this lock
    print('SYSTEM: the script will be terminated because the account
        was locked')
    sys.exit(400)
except instaloader.ConnectionException:
    print('INSTALOADER: account redirection to login page')
    # assigned the number 114 to the error
    # TODO insert mail sender client to notify this state
    sys.exit(114)
```

Nella fase 3-A, fase che entra in gioco nel momento in cui l'account scelto è funzionante, si vanno a scaricare i post che soddisfano dei requisiti in termin di data, in quanto in questo modo si va ad evitare il problema del download ripetuto di uno stesso post. I vari tentativi di download vengono opportunamente intervallati da delle pause, per cercare di prevenire il blocco dell'account. Nella fase 3-B invece, lo script termina.

```
'''
    the posts are saved in a iterator structure that permit to iterate
        over withouth the necessity of index
'''
hashtag_post_iterator = instaloader_hashtag_instance.get_posts()

# iterate on the posts and get the posts in the date required
# post counter definition to control the number of posts downloaded
post_counter = last_downloaded_posts

# get the datetime format for the last time download post of the
    current hashtag
try:
    LAST_TIME = datetime.strptime(last_UTC_time, '%Y%m%d%H%M%S')
except:
    LAST_TIME = datetime.strptime(last_UTC_time, '%Y%m%d')
# define start time in datetime format (%Y%m%d):
START_TIME = datetime.strptime(START_DATE, '%Y%m%d')
# define end time in datetime format (%Y%m%d):
END_TIME = datetime.strptime(END_DATE, '%Y%m%d')

print('INSTALOADER: Searching posts in the desired data')
```

```
for post in hashtag_post_iterator:
    # get data of the post
    post_date = post.date

    # ANNOTATION: Instaloader get the post in cronological view with
        the las added as the first

    if (post_date >= START_TIME) and (post_date <= END_TIME) and
        (post_date < LAST_TIME):
        # the post can be used because in the desired data
        print('INSTALOADER: Get content of the post and save it in the
            hashtag folder')
        instaloader_instance.download_post(post, actual_hashtag[0])
        print('INSTALOADER: Contenent saved in ' + actual_hashtag[0] + '
            folder')

        # notify the actual post number
        print('INSTALOADER: Until now the number of collected post is: '
            + str(post_counter))

        # update the post counter:
        post_counter += 1

        # now it is important to add a sleep timer to make scrlike human
            behaviour
        waiting_second = randint(60, 300)
        print('INSTALOADER: Waiting to do not block the account for: ' +
            str(waiting_second) + ' seconds')
        time.sleep(waiting_second)


    else:
        # give information of about discarded posts
        print('INSTALOADER: Post discard because not in the date. The
            post date is: ' +
              post_date.strftime('%Y-%m-%d %H:%M:%S') + ' and the
                  minimum required date is: ' +
              LAST_TIME.strftime('%Y-%m-%d %H:%M:%S'))
        print('INSTALOADER: Until now the number of collected posts is:
            ' + str(post_counter))

        # to make scraping like human behaviour a sleep time was added
            but for the scraping when the
        # post is discarded the sleep time was less then the time for
            the downloaded post
        waiting_second = randint(10, 40)
        print('INSTALOADER: Waiting to do not block the account for: ' +
```

```
            str(waiting_second) + ' seconds')
        time.sleep(waiting_second)
```

Giunti alla fase 4, ovvero quella fase che entra in gioco nel momento in cui i post scaricati risultano essere in numero maggiore rispetto al numero definito da scaricare, si va ad aggiornare il file 'sync.csv' presente sul server con le occorrenze dell'ultimo download

```
if post_counter > (NUMBER_OF_POSTS + last_downloaded_posts + 1):
        # collected the number of required posts now it is important to
            save it on the remote
        print('INSTALOADER: Collected the number of desidered posts')
        # the first operation is to save the iterator posts retrieved as
            resumable information for future requests
        # and to continue the download with other accounts
        print('INSTALOADER: Creation of information to resume the
            collections of the posts')
        # saving the last post_date information
        last_date_information = post_date.strftime('%Y%m%d%H%M%S')
        # check that the last data information of a post does not today,
            in that case the last search data is the
        # passed data as parameter
        if datetime.now().strftime('%Y%m%d') !=
            last_date_information[:8]:
            last_date_information == LAST_TIME.strftime('%Y%m%d%H%M%S')
        # now I get the last version of the sync file
        sftp_client.get(REMOTE_SYNC_FOLDER + SYNC_FILE, './' + SYNC_FILE)
        # open the last version of the sync file
        sync_dataframe = pd.read_csv(SYNC_FILE, sep=';')
        # retrieve the information already saved:
        # the most important information is the hashtag that need to be
            updated
        sync_dataframe.loc[sync_dataframe['hashtag_id'] ==
            int(future_sync['hashtag_id']),
                        ['last_time']] = last_date_information
        # update the number of downloaded posts
        sync_dataframe.loc[sync_dataframe['hashtag_id'] ==
            int(future_sync['hashtag_id']),
                        ['downloaded_posts']] =
                            int(last_downloaded_posts) + NUMBER_OF_POSTS
        # saving the modified file in the original SYNC CSV file
        sync_dataframe.to_csv(SYNC_FILE, sep=';', index=False)
        # after the saving procedure the file was uploaded to the sync
            remote folder
        sftp_client.put(SYNC_FILE, REMOTE_SYNC_FOLDER + SYNC_FILE)
```

Nella fase 5 si vanno a caricare sul server i post che sono stati scaricati, controllando

nuovamente che non ci siano doppioni, in quanto hashtag diversi di ricerca potrebbero comunque essere associati a stessi post e quindi questi potrebbero essere già stati scaricati con un altro account in un momento diverso. In maniera iterativa è poi Heroku che una volta che lo script termina, si precooupa di rilanciarlo.

```
    # count the number of post syncronized
          sync_posts = 0
          # count the number of post that already exists
          no_sync_posts = 0

          # uploading the post to the remote server
          for file in glob(actual_hashtag[0] + '/*'):
              # get the remote path where the file have to be saved or
                  retrieved if it exist
              remote_file_path = REMOTE_BASE_FOLDER + actual_hashtag[0] +
                  '/' + file.split(os.sep)[-1]
              # use the try on the stat function of the sftp_client for
                  verification of the existance of the file
              try:
                  sftp_client.stat(REMOTE_BASE_FOLDER + actual_hashtag[0])
              except IOError:
                  print('SYNC OPERATION: Creating the folder ' +
                      actual_hashtag[0] + ' on the remote server')
                  sftp_client.mkdir(REMOTE_BASE_FOLDER + actual_hashtag[0]
                      + '/')
                  print('SYNC OPERATION: Created the folder on the remote
                      server')

              try:
                  sftp_client.stat(remote_file_path)
                  print('SYNC OPERATION: The file ' + str(file) + '
                      already exist on the remote path: ' +
                      remote_file_path)
                  no_sync_posts += 1
              except:
                  print('SYNC OPERATION: The file ' + str(file) + ' does
                      not exist on the remote server')
                  print('SYNC OPERATION: Start uploading the file: ' +
                      str(file) + ' ...')
                  sftp_client.put(file, remote_file_path)
                  print('SYNC OPERATION: Uploaded the file to the remote
                      path: ' + str(remote_file_path))
                  sync_posts += 1
          break
```

Con questo script si è riuscito a creare un dataset di circa 6000 immagini. Questo numero si è poi abbassato a poco più di 5200 a seguito del filtraggio delle immaigni

ottenute (rimozione di immagini non pertinenti). Di queste immagini ne risultano essere state labelizzate oltre 3000, andando ad indiviare in ognuna di esse l'oggetto e la sua giusta categoria (scelta tra 9 classi). Le stats complete del dataset risultano essere dunque:

- Numero di immagini scaricate -> 5601
- Numero di immagini labelizzate -> 3306
- Media oggetti per ogni immagine -> 1
- Numero oggetti per ogni classe:

Classe	**N° immagini**
Bauletto	532
Clutch	1256
Hobo	328
Marsupio	562
Sacca	124
Secchiello	373
Shopping	163
Tracolla	86
Zaino	6

Tabella 3.2: Numero di immagini per ogni classe

Le immagini poi risultano essere state labelizzate con un apposito software, di cui qui si riporta una diapositiva e opportunamente splittate in tre sottocartelle rispettivamente utilizzate per il training, il validation ed il test delle reti.

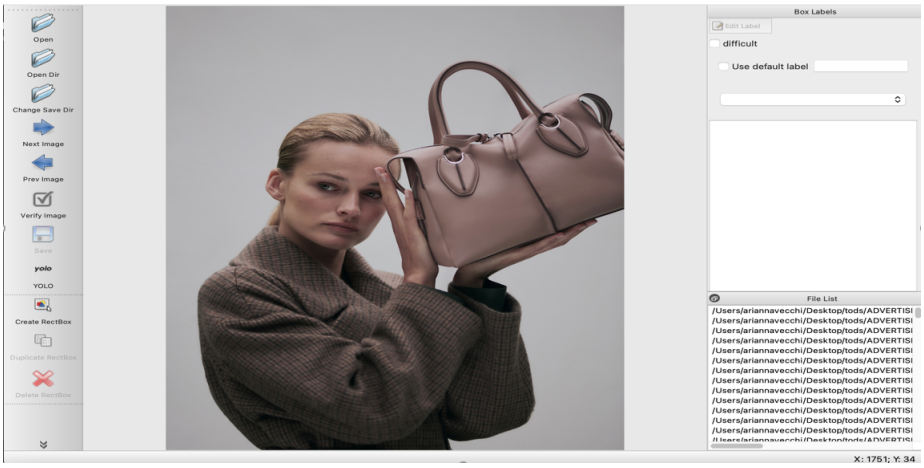

Figura 3.1: Software utilizzato per il labeling delle immagini

3.2 Modelli di Deep Learning per il settore Fashion

3.2.1 SSD300

La SSD300 è una particolare tipologia di SSD, che come descritto nell'introduzione di questa rete nel capitolo 2, prende in input delle immagini che risultaano avere dimensione 300 * 300 pixel. Analizzando il suo funzionamento, la parte di architettura che eredita dalla VGG16 risulta essere utile al fine di andare ad estrarre una feature map di una certa dimensione (38 X 38). Una volta ottenuta questa feature map, quello che la rete va a fare è definire un determinato numero di bounding box (4 in questo livello) per ognua delle celle che caratterizza la griglia e ad ognuno di essi è assegnato uno score per ognua delle classi con la quale la rete è stata allenata (introucendo anche una ulteriore classe che è il *background*). Una volta fatto questo, vista la presenza di utleriori livelli convoluzionali che la rete implementa a differenza dei fully connected che si trovano nella VGG16, queste operazioni risultano essere ripetute in maniera del tutto speculare partende però da delle feature map di dimensione diverse (19 X 19, 5 X 5, 3 X 3, 1 X 1) e definendo 6 bunding box per ognuna cella. La diversa grandezza delle feeature map dalle quali si vanno a definire i bounding box è ciò che permette alla rere di riuscire ad indivisuare oggetti di diversa grandezza, in quando una feauture map di densità maggiore, permetterà la localizzazione di oggetti più piccoli, mentre le feature map che hanno un numero minore di celle saranno preposta al riconoscimento di oggetti di dimensioni più sostenute.
Una volta che dunque l'immagine termina il suo passaggio all'interno della rete, risultano essere definiti un numero standard di bounding box per ognuna delle classi, ovvero 8732. Per evitare bounding box ridondanti, quello che si va a fare è applicare un $Non - Maximun - Suppression - Layer$, che dopo aver ordinato per classi e per score i vari bounding box, confrontandoli a due a due, elimina quello con lo score più basso nel momento in cui il valore di IoU è superiore al 50%

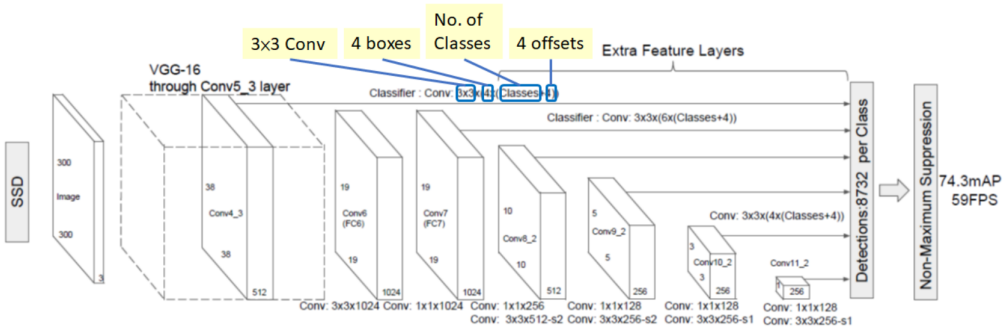

Figura 3.2: Dettaglio di ciò che avviene quando una immagine viene analizzata

Parlando dell'allenamento di questa rete, come per ogni altro training, l'obiettivo è quello di dimnuire il valore della loss il più possibile. Quest'ultima risulta essere definita in questa maniera:

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

E' caratterizzata dunque da due termini:

- L_{conf} -> è la loss associata alla *confidence* che hanno i vari bounding box per l'appartenenza ad una determinata classe
- L_{loc} -> è la loss associata alla localizzazione dell'oggetto, intesa come differenza tra il bounding box reale e bounding box predetto.

Il termine N sta ad indicare il numero di default box positivi. I default box risultano essere i bounding box dai quali parte l'allenamento, che non vengono scelti a caso, come accade nell'allenamento delle altre reti, ma vengono inizializzati manualmente, in quanto nel momento in cui si vanno a cercare determinati oggetti, questi non possono avere forme aleatorie, bensi hanno forme e strutture ben definite. Da questi default box, vengono poi determinate delle predizioni e solo nel caso in cui il default box risulti essere mathato positivamente, quel box predetto viene considerato nel calcolo della Localization Loss. Il box di default risulta essere positivo, solo nel momento in cui, presenti un IoU rispetto al bounding box reale, maggiore di 0.5 e per feature map di diversa densità risultano essere definiti default box di diversa grandezza proprio in virtù del fatto che la rete deve essere in grado di trovare l'oggetto target a

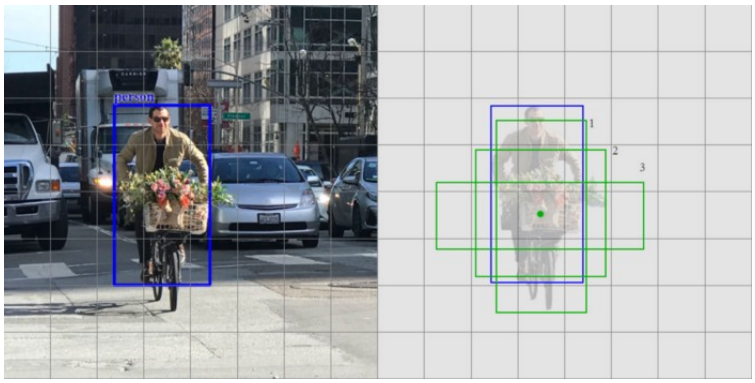

Figura 3.3: Esempio di bounding box reale e di tre default box

diverse dimensioni. Per mantenere una certa stabilità nell'allenamento, quello che la SSD va a fare inoltre è accertarsi che il rapporto tra i positive matches e i negative matches sia sempre almeno 1:3, considerando i negative matches che hanno i più alti valori di confidence loss. Un'altra parte fondamentale del training risulta esssere quella di Data Augmentation. Per ognuna delle immagini, in maniera randomica:

- Si prende l'immagine originale
- Se ne prende un sample che ha un IoU che può essere 0.1, 0.3, 0.5, 0.7 or 0.9
- Se ne prende un sample di dimensione randomica

Il sample ha un aspect ratio che oscilla tra la metà ed il doppio di quello dell'immagine di partenza. Implementando diverse tenchiche di Data Augmentation si ottengono questi miglioramenti in termini di performance:

Data augmentation	SSD300		
Flip orizziontale	✓	✓	✓
Random crop e distorsione colore		✓	✓
Espansione random			✓
VOC2007 test mAp	65.5	74.3	77.2

Tabella 3.3: Performance con Data Augmentation

3.2.2 YOLO

Analizzando il funzionamento di questa rete, quello che si può dire è che l'immagine che viene data in ingresso, risulta essere divisa in una griglia di dimensione S X S. Ognuna delle celle, nel momento in cui al suo interno ricade il centro di un oggetto, diventa quella parte di imaagine preposta per andare ad identificarlo e il task di localizzazione e classificazione avviene in maniera simultanea per ognuna delle parti nelle quali l'immagine risulta essere suddivisa. Per come la rete risulta essere costruita, si ha che per ogni cella può essere identificato al più un oggetto e ciò inevitabilmente porta a dei problemi. Inoltre vengono definiti per ognuna delle cella un determinato numero di bouding box, i quali risultano essere associati ognuno a 4 valori, cioè il centro del box (identificato da una coppia di punti) e pi largezza e altezza del box definito. In aggiunta a ciò ad ogni bounding box la rete calcola un score di confidenza che rappresenta quella che è la probabilità che all'interno del box sia presente o meno un oggetto. Infine per ognuna delle celle, la rete da in'uscita la probabilità di appartenere di quest'ultima ad ognuna delle N classi con le quali la rete risulta essere stata allenata. Le predizioni quindi vengono decodificate in tensori $S \times S \times (B \text{ X } 5 + C)$ [1]. Per quel che riguarda la funzione di loss, risulta essere

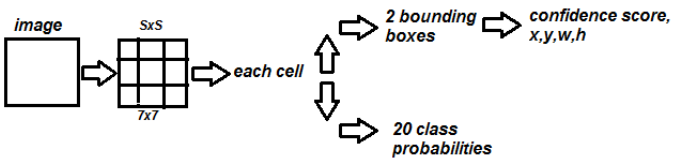

Figura 3.4: Come YOLO analizza l'immagine

caratterizzata da 5 termini:

[1] SXS = dimensione della griglia, B = bounding box per cella, 5 = 4 valori per il bounding box +1 per la confidence score, C = numero di classi con le quali la rete è stata allenata

- il primo e il secondo rappresentano la localization loss
- il terzo ed il quarto rappresentano la confidence loss
- il quinto identifica la classification loss

Loss function della rete YOLO

$$\lambda_{coord} \sum_{i=1}^{s^2} \sum_{j=0}^{B} \hat{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2\right] +$$
$$\lambda_{coord} \sum_{i=1}^{s^2} \sum_{j=0}^{B} \hat{1}_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{y_i} - \sqrt{\hat{y}_i})^2\right] +$$
$$\sum_{i=1}^{s^2} \sum_{j=0}^{B} \hat{1}_{ij}^{obj} \left(c_i - \hat{c}_i\right)^2 +$$
$$\lambda_{coord} \sum_{i=1}^{s^2} \sum_{j=0}^{B} \hat{1}_{ij}^{NoObj} \left(c_i - \hat{c}_i\right)^2 +$$
$$\sum_{i=1}^{s^2} \sum_{c \in classi} \hat{1}_{i}^{obj} \left(p_i(c) - \hat{p}_i(c)\right)^2 +$$

Questo particolare tipologia di funzione prende il nome di $sum - squared\ error$ (SSE) e specificando:

- gli errori di classificazioni penalizzano la loss solo nel momento in cui c'è un oggetto in quella particolare cella
- dei bounding box definiti per ogni cella, si prende in considerazione solo quello che ha la percentuale di IoU maggiore con il bounding box reale
- errori associati alla classificazione e alla localizzazione hanno lo stesso peso nella funzione.

Il primo termine va a determinare quella che è la differenza quadratica tra il bounding box predetto e quello reale in termini di posizione del centro del box mentre il secondo ripete lo stesso procedimento prendendo però in considerazione la differenza al quadrato tra le radici di larghezza e altezza dei box. Il terzo termine va a determinare il confidence error come differenza al quadrato tra 1 e il confidence score del box predetto mentre il quarto va a prendere in considerazione quelle celle dove non risultano essere presenti oggetti, però visto che queste risultano essere sempre in numero signivativo, per evitare che la confidence loss cresca troppo, si va a moltiplicare questo termine per un costante minore di 1. L'ultimo termine va semplicemente a sommare l'errore associato alla probabilità di appartenenza per le celle della griglia.
La rete Yolo è stata poi ulteriormente migliorata andando prima a definire la versione V2 e successivamente ancora la versione V3. Nella versione V2 le principali novità introdotte riguardano:

- l'introduzione di $BatchNormalization$ nei livelli convoluzionali
- un classificatore a maggiore risoluzione
- introduzione degli *anchor boxes* che ha permesso la predizione di più oggetti per ogni cella della griglia. La rete cerca di trovare i migliori *anchor boxes* che

vanno a descrivere la forma dell'oggetto target in modo tale da rendere più agevole l'operazione di detection. Così facendo i bounding box predetti non derivano direttamente dall'intera immagine, bensì dagli *anchor boxes*

Nella versione V3 della YOLO si vanno invece ad aggiungere features riguardo quella che è la detezione dei bounding box, in quanto oltre ad essere definti da 4 coordinate, risultano anche essere caratterizzati da un *objectness score* (confidence) che risulta essere calcolata attraverso la *logistic regression*. Viene rimosso il $Non-Maximum-Suppression\ Layer$ per far fronte alla necessità di poter labelizzre con diverse categorie uno stesso oggetto; al suo posto viene inserito un classificatore indipendente logistico, mentre durante l'allenamento viene utilizzata la $binary\ cross-entropy\ loss$ per la prediction loss. Si sono ottenute anche delle performance migliori con questa versione per quel che riguarda la detection di piccoli oggetti e ciò è dovuto alla presenza di *short cut connection* (ciò si traduce però in performance leggermente meno buone per la detezione di oggetti di media e grande dimensione). Infine un'ulteriore differenza rispetto alla prima versione e alla V2 è la capacità della rete di produrre in uscita i bounding box, non solo a seguito dell'ultimo livello, ma anche in due livelli intermedi.

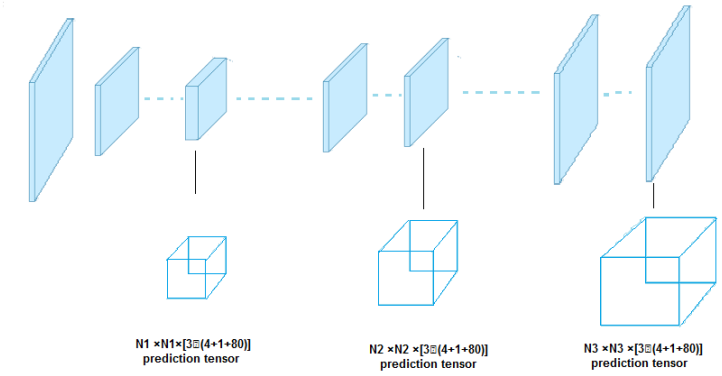

Figura 3.5: Produzione di bounding box su più livelli

3.3 Metriche per la valutazione delle performance delle reti

Le metriche utlizzate per andare a descrivere quelle che sono le performance delle due reti, risultano essere le seguenti:

- Intersection Over Union (IoU)
- Precision
- Recall
- mean Average Precision (mAP)

3.3.1 Intersection over Union

L'Intersection over Union è quella metrica che viene utilizzata per andare a determinare quella che è l'accuratezza di un object detector rispetto ad un determinato dataset. Per andare a determinarla, quello di cui si ha bisogno risulta essere il box predetto dalla rete ed il box reale che circonda l'oggetto target. E' calcolata come il rapporto tra l'area di intersezione tra i due box e l'area di unione dei due. Ciò permette di andare a distinguere la bontà delle predizioni a seconda del loro valore di IoU calcolato rispetto al box reale.

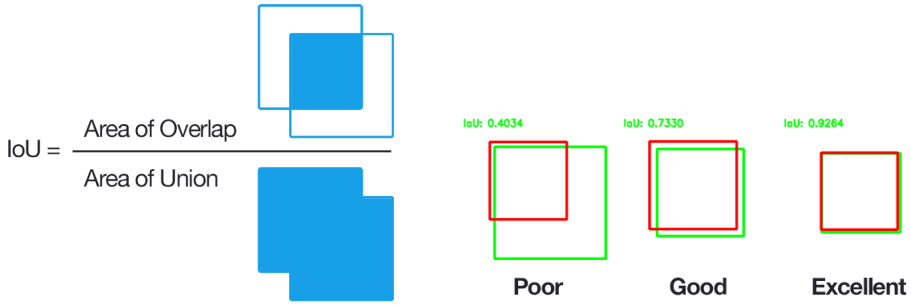

Figura 3.6: Intersection over union e come utilizzarla per distinguere le predizioni

3.3.2 Precision

La precision è un'altra metrica utile per andare a descrivere quelle che sono le performance della nostra rete, in quanto permette di determinare quella che è la capacità della rete di andare a classificare in maniera corretta. Risulta essere definita come il rapporto tra i **True Positive** e la somma tra i **True positive** e i **False positive**. Nel caso in cui non risultassero essere stati identificati dalla rete falsi positivi, la precision risulterebbe assumere valore massimo ovvero pari ad 1.

$$\textbf{Precision} = \frac{\textbf{True Positive}}{\textbf{True positive+False positive}}$$

L' appartenenza di un oggetto ad un classe presa in esame può essere contrassegnata infatti come:

- True positive -> nel caso in cui la classe vera coincida con quella predetta (classe vera: Gatto, predizione: Gatto)
- False positive -> nel caso in cui la classe associata non coincida con quella vera (classe vera: Non Gatto, predizione: Gatto)
- True negative -> nel caso in cui la classe vera non coincida con quella in esame e venga correttamente predetta (classe vera: Non Gatto, predizione: Non Gatto)

- False negative -> nel caso in cui la casse vera non coincida con quella in esame e ciò non viene correttamente predetto (classe vera: Non Gatto, predizione: Gatto)

3.3.3 Recall

La recall è una metrica del tutto speculare rispetto alla precision, solo che per andarla a determinare si utilizzano i false negativi invece dei falsi positivi nella formula. Questa metrica risponde alla domanda: quale porzione degli oggetti identificati positivamente è stata identificata correttamente? Se non fossero stati identificati falsi negativi, la recall risulterebbe assumere valore unitario.

$$\textbf{Recall} = \frac{\textbf{True Positive}}{\textbf{True positive+False negative}}$$

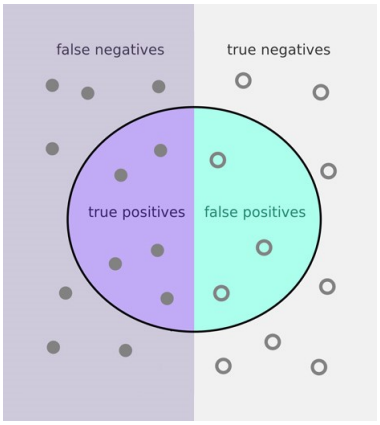

Figura 3.7: Distinzione tra i TP, FP, TN e FN

3.3.4 Mean Average Precision (mAP)

La mean Average Precision si basa su quella che è la curva Precision-Recall. Questa curva risulta mettere in relazione i due parametri, vedendo come questi variano nel momento in cui vengono scelte soglie differenti per la identificazione di una classe di un oggetto in un'immagine come positiva o meno. Questa curva quindi permette di andare a definire quella che è la soglia che consente di ottenere le migliori metriche allo stesso momento e si può notare come all'aumentare della Recall, la Precision tenda a diminuire questo perchè, tanto più il valore di recall è alto, tanto più aumentano il numero dei esempi marcati come *positivi*, motivo per il quale l'accuratezza nell'andare a marcare ognuno di essi correttamente diminuisce (e quindi la precisione si abbassa). A partire da tutte le coppie precion-recall che si possono avere si va a determinare quella che è la **Average Precision**. Questa praticamente risulta essere l'area sottesa alla curva nel grafico .
Una volta ottenuta questa per ognuna delle classi, andandone a calcolare la media si va a definire quella che è la **mean Average Precision** che quindi risulta essere la

media tra tutte le aree caratterizzanti le curve Precison-Recall per ognua delle classi.

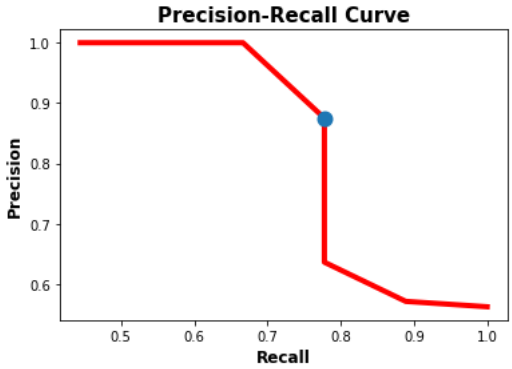

Figura 3.8: Curva Precision-Recall per una classe

Capitolo 4

Risultati e discussioni

4.1 Risultati ottenuti con la rete YOLO

La rete YOLO risulta essere stata allenata con il VRAIDataset per 150 epoche, avendo un tempo medio di train per epoca di circa 8m e 30s. La versione che è stata utilizzata è la Medium, partendo da un pre-addestramento sul dataset Coco. I risutlati sono stati calcolati dopo le prime 50 epoche e al termine dell'allenamento, utilizzando sia il validation set che il test set.

Class	**Images**	**Targets**	**P**	**R**	**mAP@.5**	**mAP@.5:.95:**
all	793	1.1e+03	0.594	0.47	0.499	0.338
Bauletto	793	143	0.574	0.641	0.616	0.386
Clutch	793	223	0.691	0.493	0.582	0.431
Hobo	793	124	0.588	0.468	0.535	0.386
Marsupio	793	112	0.65	0.527	0.559	0.336
Sacca	793	28	0.533	0.143	0.245	0.18
Secchiello	793	93	0.626	0.602	0.583	0.362
Shopping	793	96	0.629	0.354	0.406	0.286
Tracolla	793	256	0.609	0.586	0.556	0.32
Zaino	793	27	0.446	0.418	0.409	0.353

Tabella 4.1: Risultati sul validation set dopo 50 epoche

Class	**Images**	**Targets**	**P**	**R**	**mAP@.5**	**mAP@.5:.95:**
all	668	966	0.537	0.506	0.49	0.329
Bauletto	668	157	0.579	0.554	0.544	0.35
Clutch	668	169	0.611	0.566	0.583	0.414
Hobo	668	97	0.538	0.546	0.583	0.404
Marsupio	668	95	0.659	0.589	0.576	0.323
Sacca	668	18	0.499	0.278	0.267	0.218
Secchiello	668	79	0.551	0.56	0.534	0.345
Shopping	668	93	0.552	0.387	0.428	0.295
Tracolla	668	232	0.536	0.608	0.528	0.309
Zaino	668	26	0.311	0.462	0.366	0.307

Tabella 4.2: Risultati sul test set dopo 50 epoche

Class	**Images**	**Targets**	**P**	**R**	**mAP@.5**	**mAP@.5:.95:**
all	793	1.1e+03	0.705	0.501	0.577	0.391
Bauletto	793	143	0.7	0.608	0.675	0.445
Clutch	793	223	0.781	0.48	0.594	0.444
Hobo	793	124	0.833	0.46	0.577	0.407
Marsupio	793	112	0.695	0.545	0.612	0.363
Sacca	793	28	0.714	0.5	0.606	0.4
Secchiello	793	93	0.683	0.603	0.597	0.396
Shopping	793	96	0.683	0.426	0.538	0.385
Tracolla	793	256	0.69	0.48	0.605	0.353
Zaino	793	27	0.567	0.407	0.386	0.323

Tabella 4.3: Risultati sul validation set dopo 150 epoche

Class	**Images**	**Targets**	**P**	**R**	**mAP@.5**	**mAP@.5:.95:**
all	668	966	0.615	0.556	0.575	0.399
Bauletto	668	157	0.7	0.565	0.648	0.43
Clutch	668	169	0.672	0.642	0.638	0.469
Hobo	668	97	0.725	0.515	0.643	0.445
Marsupio	668	95	0.635	0.558	0.619	0.328
Sacca	668	18	0.499	0.556	0.52	0.427
Secchiello	668	79	0.716	0.709	0.659	0.458
Shopping	668	93	0.598	0.538	0.537	0.367
Tracolla	668	232	0.604	0.539	0.532	0.361
Zaino	668	26	0.39	0.385	0.376	0.304

Tabella 4.4: Risultati sul test set dopo 150 epoche

Come si può ben vedere, tutti i parametri risultano essere migliorati con il passare delle epoche di allenamento. In particolare si nota come ci siano delle classi che presentano degli ottimi valori di P (*Precision*), ad esempio *Hobo* e *Clutch* che riescono ad arrivare a quasi un 80% mentre altre si distinguono in negativo (la classe *Zaino*), ciò dovuto principalmente alla bassa presenza di esempi di quella categoria all'interno del dataset. In linea generale comunque si è ottenuta una *Precision* media di oltre il 50%, cosa che ha reso in grado la rete Yolo di riuscire a localizzare e catalogare le differenti tipologie di borse in maniera più che discreta come si può notare dagli esempi qui sotto riportati.
A distanza di 100 epoche di allenamento, a parità di gruppi presi in esame si può notare come è migliorata sia l'accuratezza in termini di detezione del bounding box, sia come risulti esserci una migliore precisione, sintomo del fatto che i dati ottenuti corrispondo alla realtà di come la rete operi.

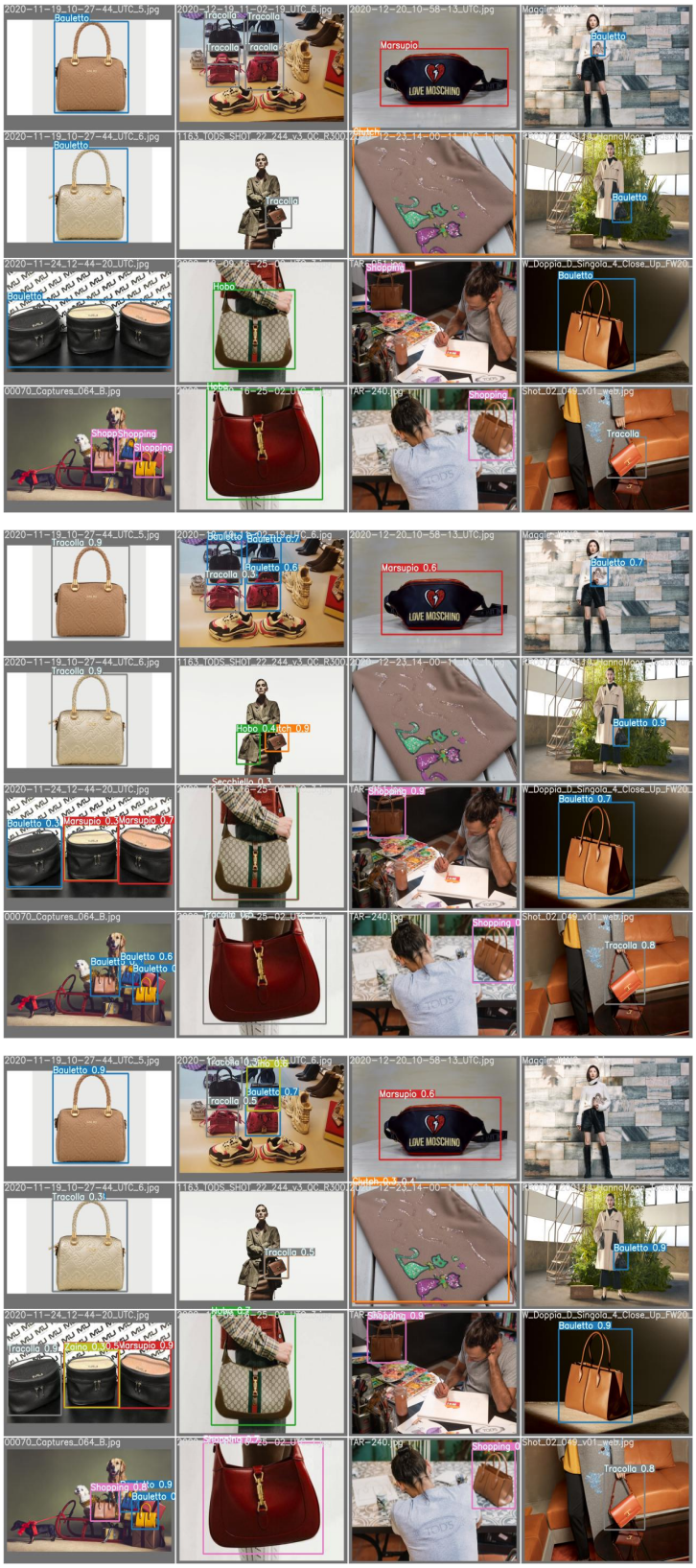

Figura 4.1: Gruppo1 del validation set e prediction dopo 50 e 150 epoche

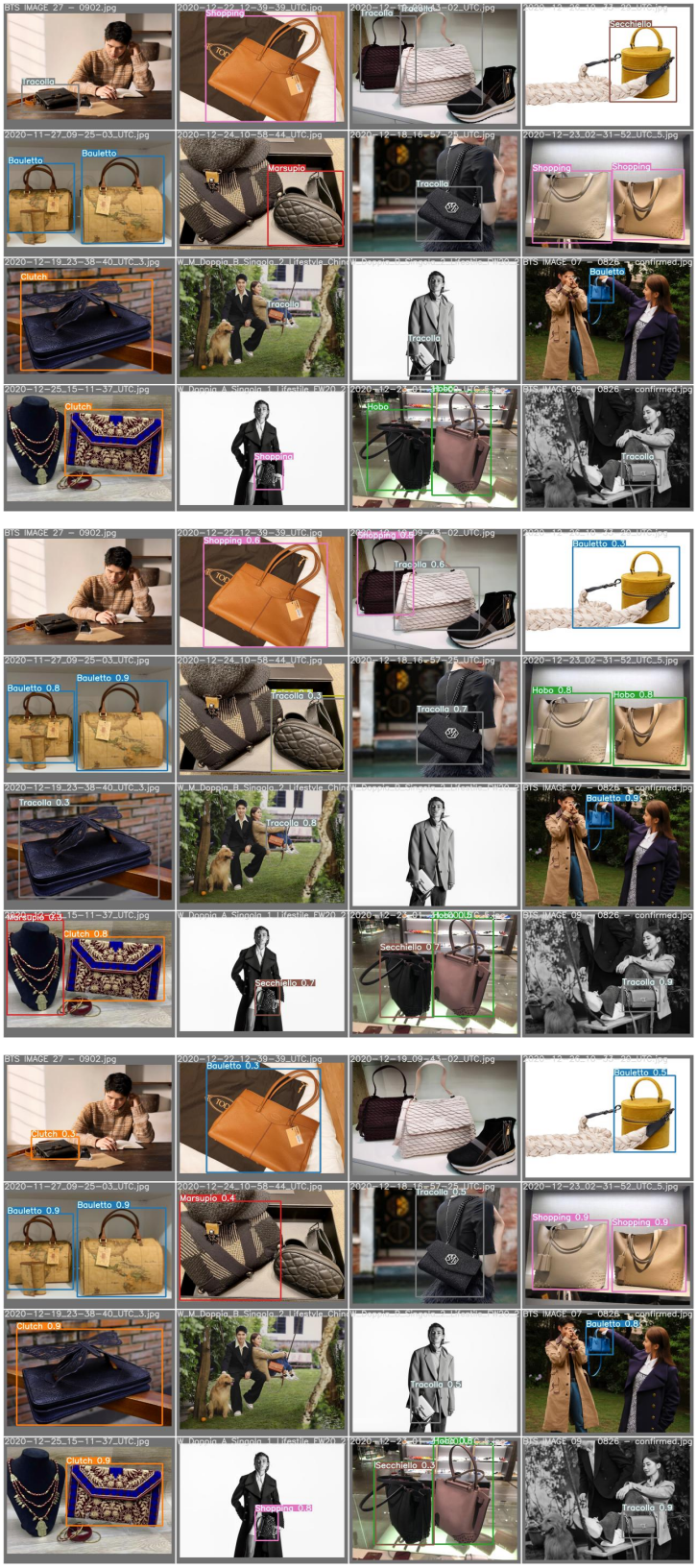

Figura 4.2: Gruppo2 del validation set e prediction dopo 50 e 150 epoche

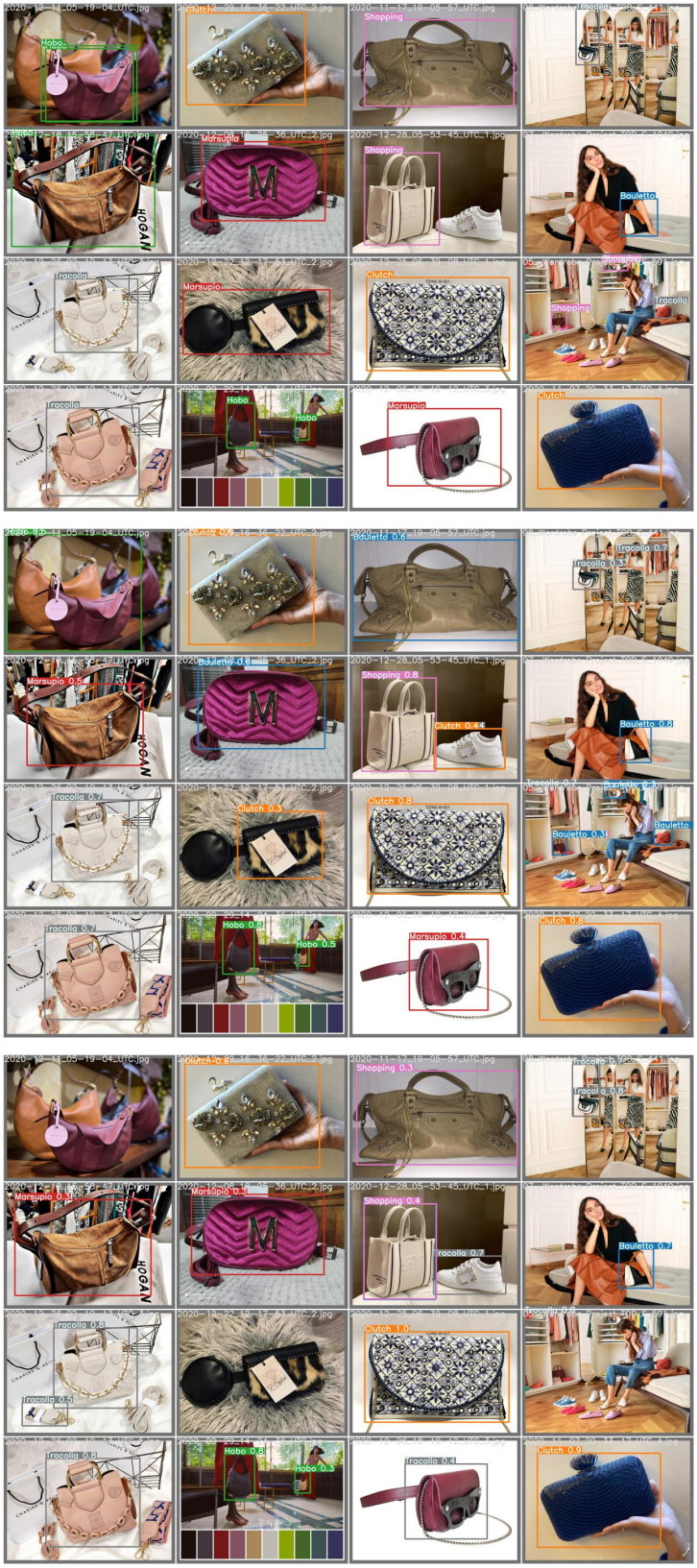

Figura 4.3: Gruppo3 del validation set e prediction dopo 50 e 150 epoche

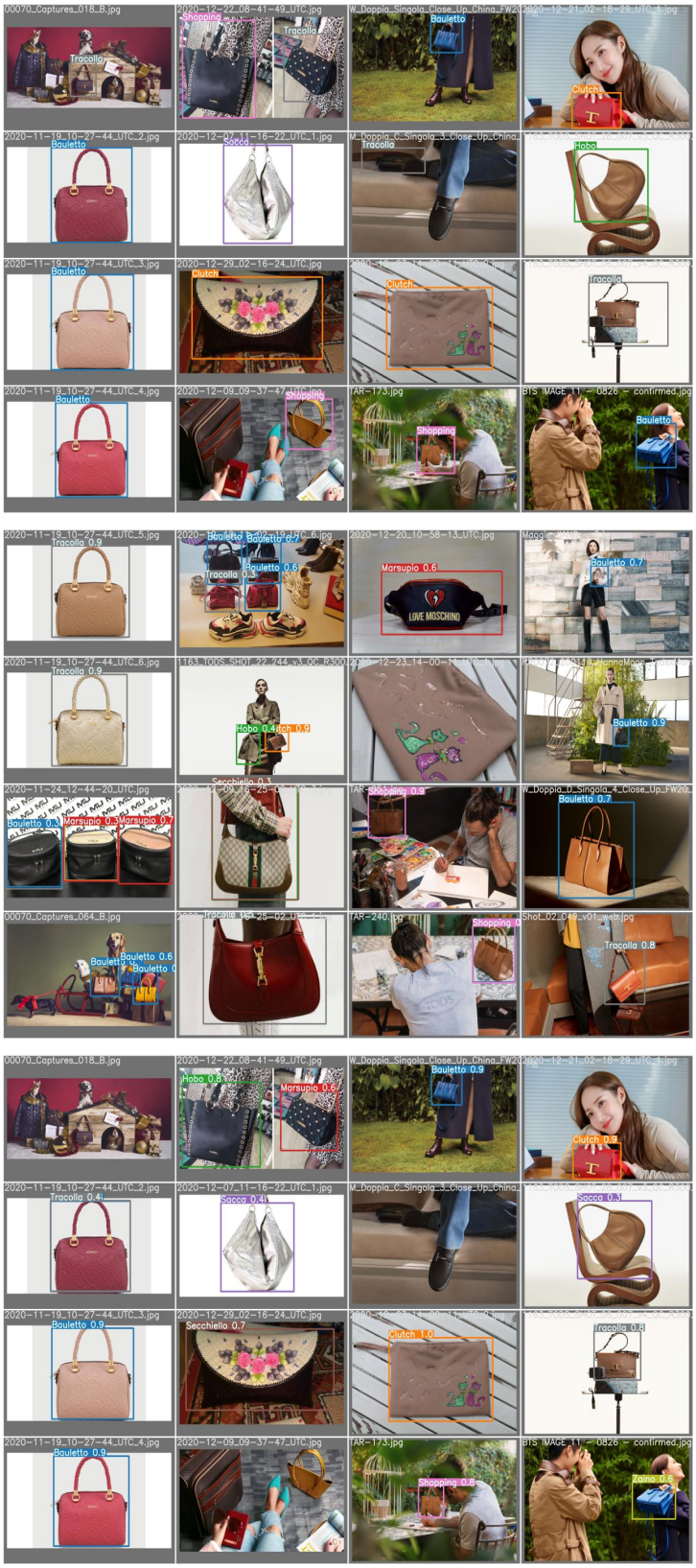

Figura 4.4: Gruppo1 del test set e prediction dopo 50 e 150 epoche

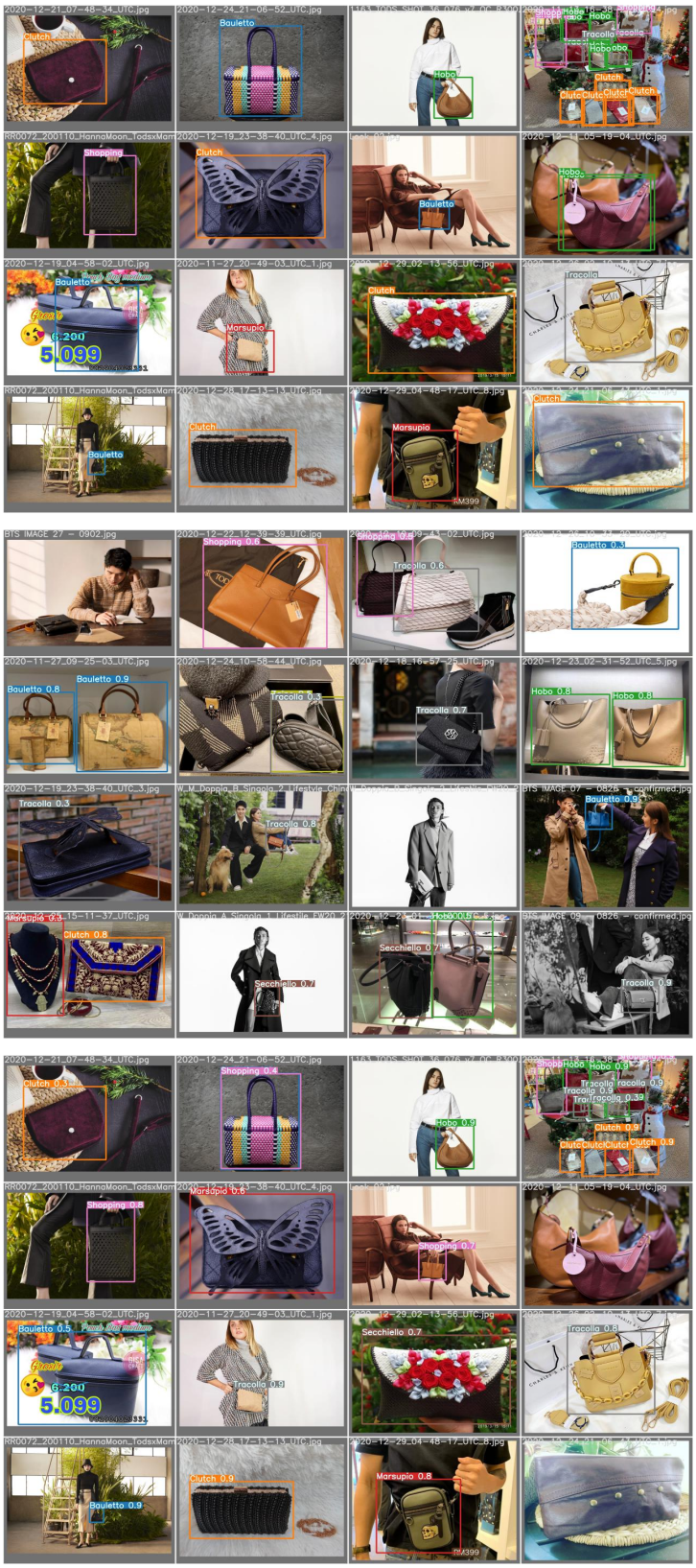

Figura 4.5: Gruppo2 del test set e prediction dopo 50 e 150 epoche

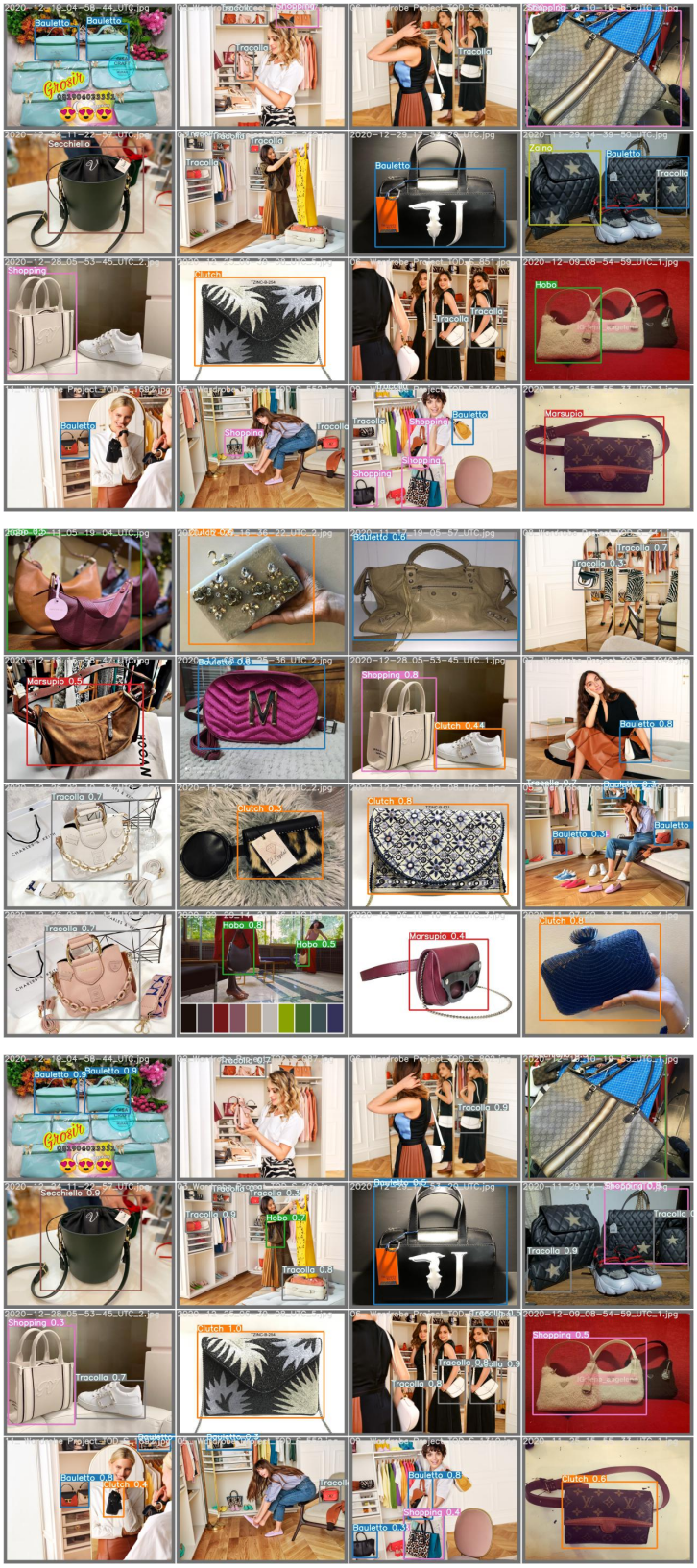

Figura 4.6: Gruppo3 del test set e prediction dopo 50 e 150 epoche

Entrando un po' più nel dettaglio si possono andare ad analizzare come sono variate le curve $Precision - Recall$ e $Precision$ alle differenti epoche di allenamento, determinate sul validation o sul test set.
Per entrambe le curve, come ci si aspettava, la media dei parametri tra le varie classi si è spostata verso l'alto e ciò denota come la rete sia diventata con il passare delle epoche di allenamento decisamente più precisa e in grado di identificare in maniera corretta le differenti tipologie di oggetti.

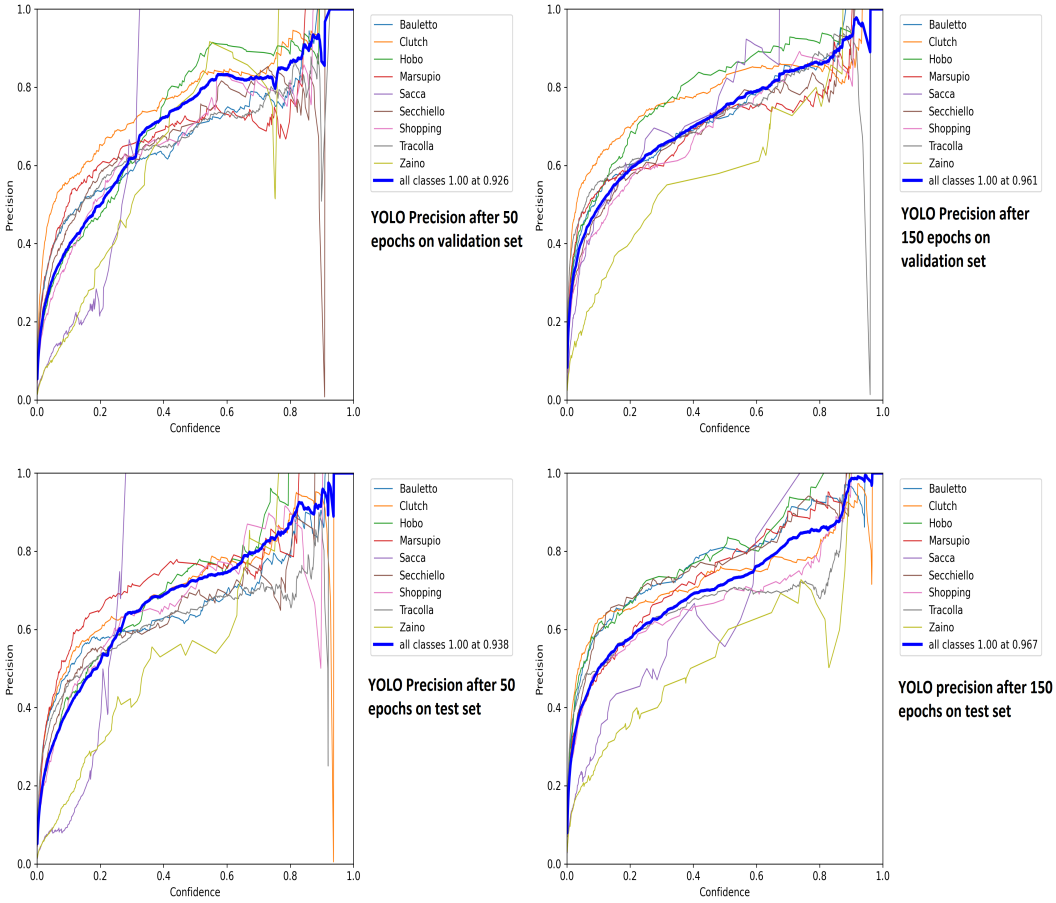

Figura 4.7: Precision della YOLO su validation e test set dopo 50 e 150 epoche

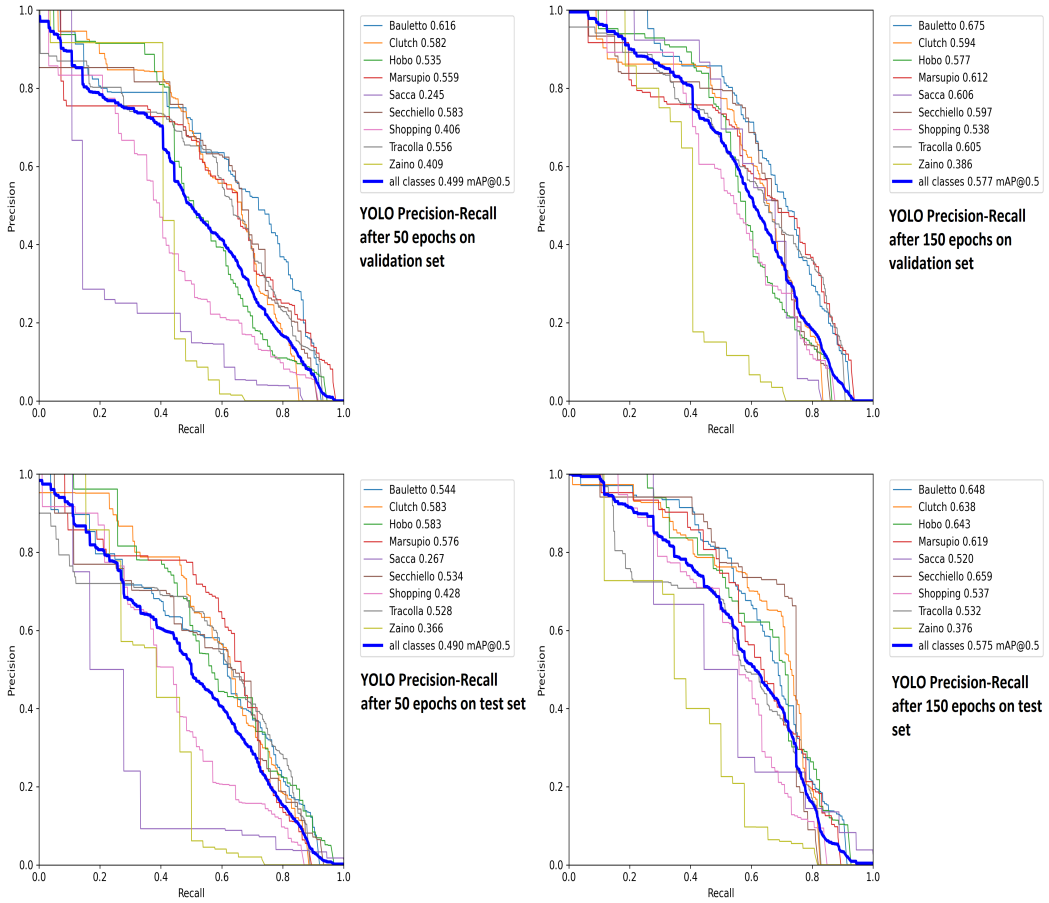

Figura 4.8: Precision-Recall della YOLO su validation e test set dopo 50 e 150 epoche

4.2 Risultati ottenuti con la rete SSD

Parlando della rete SSD, inizialmente si è provato ad allenarla utilizzando il dataset Modanet, per verificare se riuscisse ad identificare la presenza di borse in immagini. Si è optato per questo dataset in quanto presenta un numero di esempi totali decisamente maggiore rispetto a quello creato. Dopo aver fatto passare oltre 200 epoche di allenamento i risultati erano i seguenti:

Class	Precision	Class	Precision	Class	Precision
All	0.636	Occhiali da sole	0.784	Scarpe	0.247
Borsa	0.672	Pantaloni	0.887	Giacca	0.79
Cintura	0.367	Top	0.723	Cappello	0.786
Stivali	0.2	Shorts	0.797	Sciarpa	0.41
Gonna	0.787	Vestito	0.811		

Tabella 4.5: Risultati SSD300 con Modanet

Focalizzando l'attenzione sulla classe "Borsa", classe di interesse per questo elaborato, si trova un ottimo valore, prossimo al 70% di precisione. Nonostante ciò, con le

immagini target della tesi (immagini scaricate da Instagram con le borse in primo piano), la rete fa estremamente fatica a riconoscere e trovare l'oggetto in questione, come si può notare dagli esempi riportati, sintomo del fatto che Modanet presenti per il training set pochi esemplari di queste tipologie di immagini.

Figura 4.9: Immagini target date alla SSD dopo il train con Modanet

Per questo si è passati poi ad allenare la rete con il VRAIDataset. Nello specifico sono state effettuate 150 epoche di train ed il tempo medio impiegato per completare ciascuna di esse è stato di circa 55 minuti. Analizzando quelli che sono i suoi risultati al termine di questa fase si ha:

Class	**Precision**	**Class**	**Precision**
All	0.227	Sacca	0.167
Bauletto	0.397	Secchiello	0.225
Clutch	0.345	Shopping	0.153
Hobo	0.222	Tracolla	0.29
Marsupio	0.2	Zaino	0.04

Tabella 4.6: Precision della SSD300 a fine train con VRAIDataset

Quello che si può vedere è come la precisione generale sia abbastanza bassa, ciò dovuto principalmente al fatto che il dataset preso in esame risulti essere caratterizzato da un numero poco soddisfacente di immagini per classi. Nello specifico, in negativo, si sottolinea la precisione per quella che è la classe *Zaino*, che risulta essere solo dello 0,04%; ciononostante la rete presenta un comportamento decisamente migliore con le immgini target a differenza di quanto avveniniva a seguito del train con Modanet, seppur teoricamente il valore di *Precision* era nettamente migliore, come verrà mostrato in seguito
Nello specifico inoltre si possono andare ad analizzare i grafici di *Loss*, *Accouracy* e come questi risultino essere variati nel corso delle epoche, sia per quel che riguarda il

validation set che il training set, affiancandoli al grafico della *Precision* – *Recall*.

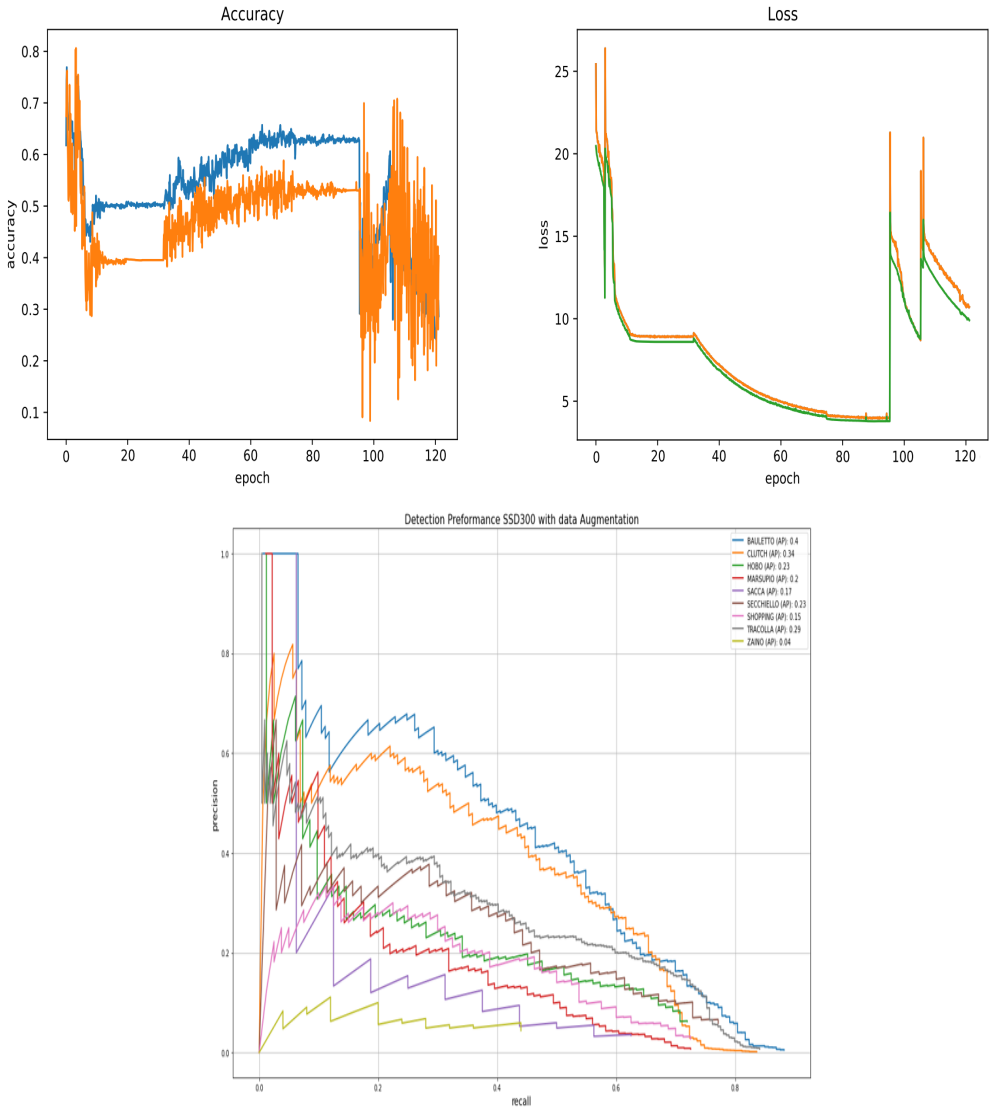

Figura 4.10: Evoluzione Accouracy e Precision su validation (arancio) e train set (blu) con a seguire grafico Precision-Recall

I due grafici presentano un andamento lineare fino alla centesima epoca, con un continuo miglioramento di ambo i parametri; dalla centesima alla centocinquantesima epoca invece, si può notare come le prestazioni in generale della rete degradino e ciò dovuto principalmente alla altalenanza della accuracy; per quel che riguarda invece il terzo grafico si può osservare come la rete a fronte di valori di *Precision* sicuramente non eccelsi, accompagni dei valori di *Recall* decisamente accettabili.
A differenza della YOLO, dove in maniera automatica sono stati svolti periodicamente delle predizioni su diversi gruppi di immagini, per la SSD, si è optato per degli esperimenti fatti alla fine di tutto il training andando ad ottenere le seguenti previsioni;

tutte e tre le prove hanno determinato la giusta classe di appartenenza delle borse e identificato in maniera abbastnaza soddisfacente la posizione di queste ultime.

Figura 4.11: Esperimenti fatti

4.3 Confronto tra le due reti

Mettendo in relazione le due reti quello che si può facilmente evincere è che:

- parlando di tempi di allenamento la YOLO è superiore alla SSD avendo un tempo medio per epoca di 9 minuti a fronte degli oltre 50 della SSD
- analizzando le curve della *Precision*, anche in questo caso la YOLO ha un andamento decisamente migliore

- prendendo in esame la *Precision* come valori assoluti, la YOLO è al di sopra della SSD in ogni classe e non di poco
- la SSD300 presenta un maggiore grado di instabilità in fase di training

A termine di questo confronto si può dire come la YOLO per riuscire a trovare e classificare borse risulti essere la scelta migliore anche se si può comunque utilizzare la SSD300, come si è potuto notare dagli esempo sopra riportati.

Capitolo 5

Conclusioni e sviluppi futuri

All'interno di questo elaborato, è stato trattato il tema della object detection inteso come riuscire a trovare e classificare all'interno di immagini diverse tipologie di borse, partendo con un analisi teorica di quello che è lo stato dell'arte per quel che concerne le reti neurali preposte a questo scopo e analizzando quelli che sono i dataset attualemente in circolazione per il mondo della moda. Si è poi passati ad analizzare ciò che è stato utilizzato per riuscire a centrare l'obiettivo preposto, ovvero andando a dettagliare quello che è il funzionamento delle reti SSD300 e YOLO in tutti i loro aspetti. Inoltre è stato discusso come sia stato creato un database ad hoc per l'oggetto target, in quanto i risultati ottenuti a seguito del training di una delle reti con Modanet, seppur sulla carta molto buoni, non trovavamo riscontro all'atto pratico con le immagini target di questo elaborato e i tool che sono stati utilizzati per popolarlo per poi andare a definire quelli che sono i bounding box reali dei vari oggetti. Si sono quindi presentati i risultati ottenuti a seguito di diverse epoche di allenamento per ambo le reti con il dataset creato, mettendo in evidenza le criticità incontrate con l'utlizzo di Modanet ed i vari parametri di valutazione. A seguito si è fatto un confronto per determinare quale tra le due risultasse essere la migliore per l'obiettivo preposto.

A partire da questa tesi si possono sviluppare diversi lavori in ambito social media, estendendo ad esempio il discorso non solo ad immagini, ma anche alla presenza di determinati oggetti in video, ovviamente andando ad utilizzare le opportune tecnologie. Quanto è stato fatto inoltre può anche essere allargato ad un qualunque settore, non solo al mondo fashion, in quanto il processo presentato per raggiungere l'obiettivo definito risulta essere ripetibile e riutilizzabile.

Bibliografia

[1] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

[2] Ross Girshick. Fast r-cnn. *CoRR*, abs/1504.08083, 2015.

[3] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[4] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018.

[5] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016.

[6] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[7] Shuai Zheng, Fan Yang, M Hadi Kiapour, and Robinson Piramuthu. Modanet: A large-scale street fashion dataset with polygon annotations. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 1670–1678, 2018.

[8] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang. Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1096–1104, 2016.

[9] Sudarshan Ghuge. Instance segmentation generator for fashion images using deepfashion-2 dataset and mask r-cnn. 2020.

[10] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.