



SAPIENZA
UNIVERSITÀ DI ROMA

Machine learning e riconoscimento automatico di opere museali

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Corso di Laurea in Informatica

Candidato

Giacomo Latini
Matricola 1868557

Relatore

Prof. Alessandro Mei

Correlatore

Dr. Massimo La Morgia

Anno Accademico 2020/2021

Tesi non ancora discussa

Machine learning e riconoscimento automatico di opere museali
Tesi di Laurea. Sapienza – Università di Roma

© 2021 Giacomo Latini. Tutti i diritti riservati

Questa tesi è stata composta con L^AT_EX e la classe Sapthesis.

Versione: 6 luglio 2021

Email dell'autore: latini.1868557@studenti.uniroma1.it

*Alla persona che più mi ha aiutato e insegnato.
Grazie mamma.*

Indice

1	Introduzione	1
2	Stato dell'arte	3
2.1	Reti Neurali Region-based CNN	3
2.1.1	RCNN	3
2.1.2	Fast RCNN	4
2.1.3	Faster RCNN	5
2.2	Reti Neurali a Singola Fase	7
2.2.1	YOLO(You Only Look Once)	7
2.2.2	YOLO9000	8
2.2.3	YOLOv3	9
3	Background	13
3.1	Apprendimento della rete neurale	13
3.1.1	Addestramento	13
3.1.2	Suddivisione dei dati	14
3.1.3	Overfitting	14
3.1.4	Cross validation	15
3.1.5	Transfer learning	15
3.1.6	Funzioni di attivazione	16
3.1.7	Metriche di COCO	17
3.2	Architetture	17
3.2.1	Rete neurale convoluzionale	17
3.2.2	Rete neurale residuale	19
3.2.3	Feature pyramid network	20
3.3	Librerie	20
3.3.1	PyTorch	21
3.3.2	Scikit-learn	21
3.3.3	Detectron2	21
4	Progetto	23
4.1	Prerequisiti del progetto	23
4.2	Dataset	23
4.2.1	Creazione del dataset	23
4.2.2	Creazione delle etichette ed ottenimento del dataset finale . .	25
4.3	Train	26

4.3.1	K-fold Cross validation	26
4.3.2	Conversione e suddivisione del dataset	26
4.3.3	Scelta del modello e relativa backbone	26
4.3.4	Addestramento	27
4.4	Test	29
4.4.1	Metriche di COCO	30
5	Risultati	31
6	Conclusioni e Sviluppi Futuri	37

Capitolo 1

Introduzione

Nel corso degli ultimi anni, con l'incremento di nuove tecniche nel campo del Machine Learning e delle Reti Neurali, è stata possibile la realizzazione di modelli sempre più precisi per risolvere problemi per il riconoscimento di oggetti.

Il riconoscimento di oggetti è una applicazione tecnologica molto importante per diversi ambiti. Grazie ad esso, è possibile localizzare e individuare più oggetti all'interno dell'immagine o di un flusso video, per poi andare a determinare la tipologia specifica di oggetto individuato (Object Recognition).

Il riconoscimento di oggetti può essere utilizzato per le auto a guida autonoma, in cui attraverso delle telecamere i software si occupano di capire l'ambiente che c'è intorno e verificare la presenza di altri veicoli, pedoni o cartelli stradali. Il riconoscimento di oggetti ha anche altri impieghi, come il rilevamento ottico dei caratteri (OCR) per il controllo stradale mediante gli autovelox. Nel momento in cui passa un veicolo in prossimità di un autovelox, viene prima cercata la targa del veicolo tramite la tecnica dell'object detection e successivamente l'OCR potrà leggerla.

Altre applicazioni per il riconoscimento di oggetti sono alla base per effettuare il tracciamento degli oggetti. Quest'ultimo è un processo per localizzare degli oggetti in movimento, usando una fotocamera in sequenze video. Un'applicazione potrebbe essere quella della sicurezza e vigilanza, in cui attraverso un sistema di videosorveglianza è possibile individuare una persona oppure un oggetto pericoloso e, in automatico, far scattare l'allarme.

In questa tesi andremo a studiare le reti neurali convoluzionali, una particolare architettura di rete neurale molto utilizzata nel campo dell'elaborazione delle immagini. Quando si vuole analizzare un'immagine tramite una rete neurale convoluzionale, un problema di object detection [19] consente di rilevare gli oggetti nell'immagine stessa e di classificarli in determinate categorie.

Durante questo studio, il lavoro di tesi è stato incentrato sul rilevamento e classificazione di opere museali, in particolare anfore etrusche risalenti al VI secolo A.C., tramite la rete neurale convoluzionale Faster RCNN [16]. Tale rete neurale convoluzionale è implementata per mezzo della libreria Detectron2 [22] per un certo dataset di immagini al fine di effettuare l'object detection, vale a dire l'identificazione delle suddette anfore. Nella pratica il sistema deve essere in grado di riconoscere ed etichettare gli oggetti contenuti nei riquadri di delimitazione, impostati dall'utente e visualizzati dal sistema stesso.

La tesi è stata svolta presso il laboratorio del Dipartimento di Informatica *RFID LAB* dell'Università degli studi di Roma *La Sapienza*, sotto la supervisione del Professore Alessandro Mei e del Dott. Massimo La Morgia.

La tesi è composta da 6 capitoli ed è strutturata nel seguente modo. Nel secondo capitolo si affronta lo studio delle reti neurali convoluzionali di interesse. Nel terzo capitolo si affrontano i vari passi per addestrare una rete neurale e si analizzano le architetture e le librerie adottate per questa tematica. Nel quarto capitolo si descrivono le fasi di train e test di una rete neurale, successive alla costruzione e strutturazione del dataset. Nel quinto capitolo vengono analizzati i risultati ottenuti. Nel capitolo finale vengono esposte le conclusioni del lavoro svolto.

Capitolo 2

Stato dell'arte

In questo capitolo vengono spiegate le reti neurali per risolvere problemi di object detection. L'object detection è un'attività che richiede la classificazione e il rilevamento di tutti gli oggetti presenti nelle immagini [19].

2.1 Reti Neurali Region-based CNN

2.1.1 RCNN

La rete **Region-based Convolutional Neural Network(RCNN)**, rappresentata nella figura 2.1, è stata presentata da Ross Girshick et al. [4] nel 2013.

2.1.1.1 Funzionamento

Data un'immagine in input, si applica l'algoritmo Selective Search [21] da cui si ottengono circa 2000 regioni d'interesse. Successivamente si esegue la rete neurale convoluzionale(CNN), dalla quale si ricavano le caratteristiche da ciascuna regione d'interesse. Infine si utilizza il Support Vector Machine(SVM) per classificare le regioni rispetto alle caratteristiche ricavate.

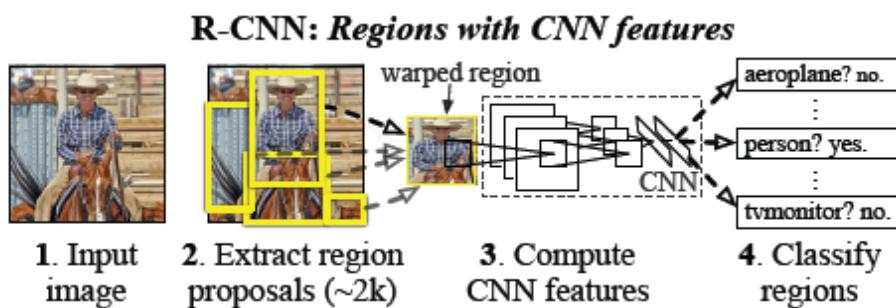


Figura 2.1. Rappresentazione del funzionamento di RCNN

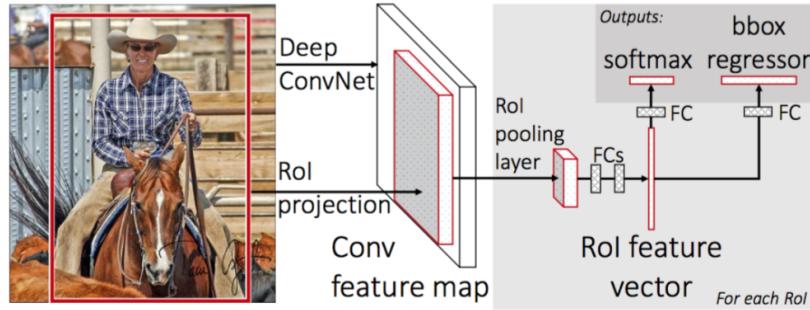


Figura 2.2. Architettura di Fast RCNN

2.1.1.2 Selective Search

Con l'algoritmo Selective Search [21] si determinano le regioni d'interesse, mediante il metodo descritto da Pedro F. Felzenszwalb et al. [2]. Tale metodo utilizza un algoritmo greedy che raggruppa iterativamente due regioni più simili e, successivamente, valuta le somiglianze tra la regione appena ottenuta e i suoi vicini. L'algoritmo greedy viene ripetuto fino a quando l'immagine non diventa un'unica regione.

2.1.2 Fast RCNN

La rete **Fast Region-based Convolutional Neural Network**(Fast RCNN) è stata presentata da Ross Girshick [3] nel 2015.

2.1.2.1 Funzionamento

La rete, mostrata nella figura 2.2, prende in input un'immagine e un insieme di proposte di oggetti, ossia aree che si suppone contengano oggetti all'interno delle immagini. La rete esegue inizialmente l'immagine tramite un insieme di strati convoluzionali e max pooling per estrarre le caratteristiche e generare una mappa di caratteristiche convoluzionale(conv feature map). Successivamente lo strato di pooling della regione d'interesse(RoI pooling layer) ricava un vettore di caratteristiche, dalla mappa di caratteristiche. Ciascun vettore di caratteristiche passa per una sequenza di strati completamente connessi(fully connected layers), per poi propagarsi in due strati di output:

- Il primo strato di output serve per stimare le probabilità su K classi di oggetti.
- Il secondo strato di output fornisce quattro numeri reali, verso ognuna delle K classi di oggetti. Questi valori definiscono le posizioni del bounding box.

La loss function, per ciascuna ROI, è la seguente:

$$\mathcal{L}(p, u, t^u, v) = \mathcal{L}_{cls}(p, u) + \lambda[u \geq 1]\mathcal{L}_{loc}(t^u, v) \quad (2.1)$$

La log loss inerente al "classificatore softmax" è pari a:

$$\mathcal{L}_{cls}(p, u) = -\log p_u \quad (2.2)$$

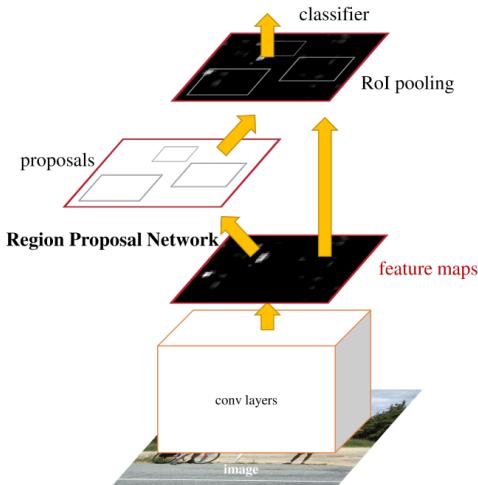


Figura 2.3. Architettura di Faster RCNN

dove p_u indica la probabilità che l'oggetto appartiene alla classe u . La log loss inerente al "bounding box regressor" è pari a:

$$\mathcal{L}_{loc}(t^u, v) = \sum_{i \in \{x,y,w,h\}} smooth_{L1}(t_i^u - v_i) \quad (2.3)$$

nel quale $smooth_{L1}$ è pari a:

$$smooth_{L1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \quad (2.4)$$

dove t^u indica il bounding box previsto per la classe u .

2.1.3 Faster RCNN

La rete **Faster Region-based Convolutional Neural Network**(Faster RCNN), mostrata nella figura 2.3, è stata presentata da Shaoqing Ren et al. [16] nel 2015. Con questa rete si modifica il procedimento per ottenere le regioni d'interesse, attraverso la Region Proposal Network(RPN). La Region Proposal Network prende in input un'immagine e restituisce le regioni d'interesse.

2.1.3.1 Funzionamento

Dall'ultimo strato convoluzionale, presente nella rete Faster RCNN, viene fatta scorrere una finestra di dimensione $n \times n$, denominata "sliding window", sulla mappa di caratteristiche con lo scopo di definire i boxes delle regioni d'interesse. Tali boxes sono denominati ancora("anchors").

Nel kernel, mostrato nella figura 2.4, si predicono contemporaneamente più regioni d'interesse. Le regioni d'interesse accettabili, per ciascuna posizione rispetto alla "sliding window", sono k . Quindi:

- Il "box regression layer(reg)" è provvisto di $4k$ output, in cui si codificano le coordinate di k boxes.

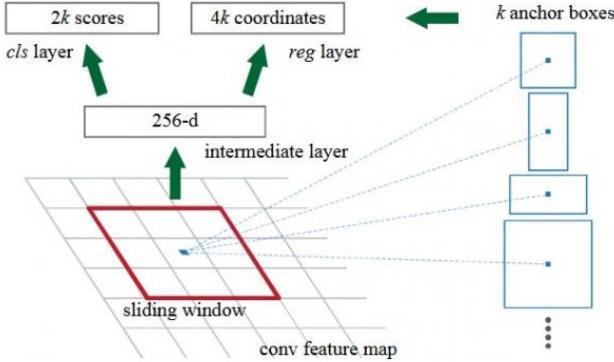


Figura 2.4. Illustrazione del kernel in una singola posizione

- Il "box classification layer(*cls*)" è provvisto di $2k$ punteggi, in cui si valuta la probabilità se c'è o meno un oggetto, per ogni bounding box proposto.

Una volta individuate le regioni d'interesse, tali regioni d'interesse vengono di nuovo eseguite dalla rete: gli strati successivi sono uguali a quelli prefissati dalla rete Fast RCNN [3], per cui sarà presente uno strato di pooling, un "classificatore softmax" e un "bounding box regressor".

Quindi la loss function in Faster RCNN, per ciascuna RoI, è la seguente:

$$\mathcal{L} = \mathcal{L}_{cls} + \mathcal{L}_{loc} \quad (2.5)$$

A differenza della rete Fast RCNN, tale equazione si differenzia per il supplemento dei termini di normalizzazione N_{cls} e N_{loc} . Quindi l'equazione (2.5) si riscrive nel seguente modo:

$$\mathcal{L}(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i \mathcal{L}_{cls}(p_i, p^*_i) + \lambda \frac{1}{N_{loc}} \sum_i p_i^* \mathcal{L}_{loc}(t_i, t^*_i) \quad (2.6)$$

dove:

- i denota l'indice di un'ancora nel kernel.
- p_i denota la probabilità prevista, cioè se l' i -esima ancora appartiene a un oggetto.
- p_i^* denota la probabilità ground truth, cioè se l' i -esima ancora è un oggetto.
- t_i denota le 4 coordinate parametrizzate del bounding box previsto.
- t_i^* denota le coordinate ground truth.
- Gli output degli strati *cls* e *reg* risultano in p_i e t_i .
- I termini *cls* e *reg* sono normalizzati da N_{cls} e N_{loc} e stimati da un parametro di bilanciamento λ .

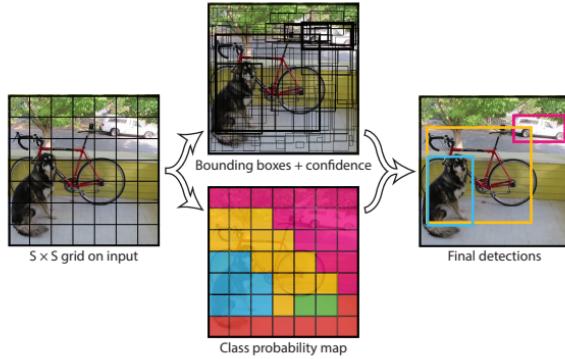


Figura 2.5. Fasi dell'esecuzione di YOLO

2.2 Reti Neurali a Singola Fase

Le reti neurali a Singola Fase sono state presentate per risolvere problemi di object detection, prevedendo bounding box, probabilità e classi degli oggetti su un'immagine di input, in un'unica fase.

2.2.1 YOLO(You Only Look Once)

La rete **You Only Look Once(YOLO)** è stata presentata da J. Redmon et al. [13] nel 2016.

2.2.1.1 Funzionamento

La rete, mostrata nella figura 2.5, suddivide l'immagine di input in una griglia di $S \times S$ celle. Se il centro di un oggetto finisce in una cella della griglia, allora tale cella si occuperà del rilevamento di quell'oggetto. Ciascuna cella della griglia prevede B bounding box e i confidence scores. I confidence scores indicano le probabilità che il bounding box includa un oggetto e anche quanto siano precise le grandezze del bounding box, riguardo l'oggetto. Ciascun bounding box, di ogni cella, è previsto dalla rete con 5 valori: le coordinate x, y, w, h e il confidence score. Le coordinate (x, y) costituiscono il centro del bounding box, rispetto ai bordi della cella della griglia, mentre la larghezza w e l'altezza h sono previste considerando tutta l'immagine.

2.2.1.2 Confronto delle prestazioni tra YOLO e Faster RCNN

Nella figura 2.6 vengono presentate le prove effettuate su YOLO, messe a confronto con il modello real time Faster RCNN [16], sul dataset PASCAL VOC 2007.

YOLO viene addestrato usando l'architettura di rete convoluzionale VGG-16, ottenendo un modello più preciso ma notevolmente più lento. Faster RCNN viene addestrato usando due architetture di rete convoluzionali: VGG-16 e Zeiler-Fergus(ZF). Con VGG-16 viene acquisita una velocità di 7 frames per secondo(fps), mentre con ZF viene acquisita una velocità di 18fps. Quindi la versione VGG-16 di Faster RCNN ha una precisione migliore di 10 punti percentuali in mean Average Precision(mAP),

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	155
YOLO	2007+2012	63.4	45
<hr/>			
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Figura 2.6. Modelli real time su dataset PASCAL VOC 2007

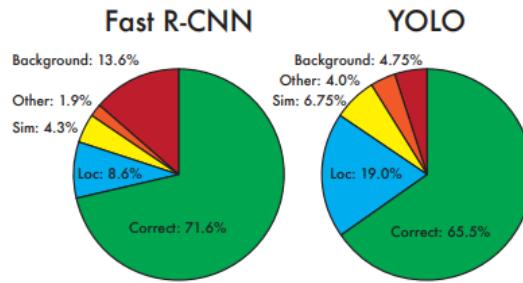


Figura 2.7. Analisi degli errori: Fast RCNN vs YOLO

ma è anche 6 volte più lenta di YOLO. La versione ZF di Faster RCNN, invece, è solo 2.5 più lenta di YOLO.

2.2.1.3 Analisi degli errori tra YOLO e Fast RCNN

J.Redmon et al. esaminano l'errore compiuto da YOLO, addestrato sul dataset PASCAL VOC 2007, confrontandolo alla rete Fast RCNN [3].

Dalla figura 2.7 YOLO fa difficoltà a rilevare correttamente gli oggetti. Infatti gli errori di rilevamento effettuati da YOLO sono più numerosi in confronto alla somma di tutti gli altri. Fast RCNN, invece, compie meno errori di rilevamento, ma più errori di background. Infatti J.Redmond et al. evidenziano che Fast RCNN ha quasi il triplo delle probabilità di prevedere i rilevamenti sul background, rispetto a YOLO.

2.2.2 YOLO9000

La rete **YOLO9000** è stata presentata da J. Redmon et al. [14] nel 2017. YOLO9000 permette di rilevare più di 9000 categorie(o classi) di oggetti.

YOLO9000 presenta vari miglioramenti rispetto alla prima versione di YOLO [13]:

- **Batch normalization.** L'inserimento della batch normalization su tutti gli strati convoluzionali porta a un aumento del 2% di mAP.
- **Classificatore ad alta risoluzione.** Nel corso dell'addestramento, il classificatore viene addestrato con immagini di dimensione pari a 224×224 . Successivamente viene effettuata un'operazione di tuning con immagini di dimensione pari a 448×448 , in maniera da sistemare i pesi per poter operare con risoluzioni migliori.
- **Anchor boxes.** Si tolgoni gli strati completamenti connessi per le anchor boxes. Le anchor boxes permettono di predire i bounding box. Con l'uso delle anchor boxes si ottiene una diminuzione del mAP dal 69.5% al 69.2% e un aumento della precisione del modello dall'81% all'88%.
- **Ridimensionamento dell'immagine in input.** Per l'addestramento si usano in input immagini di grandezza 416×416 , al posto di 448×448 , ricavando in output una mappa di caratteristiche con grandezza pari a 13×13 . L'obiettivo, infatti, è quello di avere un numero dispari di posizioni nella mappa di caratteristiche così da avere una singola cella al centro dell'immagine. La singola cella al centro dell'immagine permette di prevedere questi oggetti più rapidamente.
- **Dimension clusters.** Per determinare il numero di bounding box più opportuno, si utilizza l'algoritmo k -means clustering scegliendo $k = 5$ come soluzione più ottimale.
- **Previsione diretta della posizione.** La posizione del bounding box viene valutata mediante quella dell'anchor box, nel quale è stato rilevato l'oggetto.
- **Fine-grained features.** Si inserisce uno strato passthrough, ovvero uno strato che consente di collegare le caratteristiche ad alta risoluzione con quelle a bassa risoluzione. Quindi la mappa di caratteristiche di dimensione $26 \times 26 \times 512$ viene trasformata in una mappa di caratteristiche con grandezza pari a $13 \times 13 \times 2028$. In questa maniera è fattibile riconoscere anche gli oggetti più piccoli, riportando un aumento della precisione del modello dell'1%.
- **Multi-scale training.** Durante l'addestramento, ogni 10 epochs, la rete sceglie randomicamente la grandezza per ciascuna immagine in input, per indurre la rete ad imparare a predire immagini di grandezze differenti.

YOLO9000 utilizza un'architettura di rete per estrarre le caratteristiche, chiamata Darknet-19. Darknet-19, mostrata nella figura 2.8, è formata da 19 strati convoluzionali e 5 strati di pooling.

2.2.3 YOLOv3

La rete **YOLOv3** è stata presentata da J. Redmon et al. [15] nel 2017. Come illustrato da J. Redmon et al., con questa versione sono state adattate idee di altre persone e compiute delle piccole modifiche all'architettura del modello:

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figura 2.8. Architettura di Darknet-19

- **Bounding box prediction.** Seguendo YOLO9000 [14], YOLOv3 prevede i bounding box usando cluster dimensionali come anchor boxes, prevedendo 4 coordinate per ogni bounding box: t_x, t_y, t_w, t_h .
- **Class prediction.** Ogni box prevede sia le classi che il bounding box contenuto, utilizzando la classificazione multi etichetta(multilabel classification).
- **Predictions across scales.** YOLOv3 prevede i bounding box con 3 dimensioni diverse. Da queste dimensioni si possono estrarre le caratteristiche.
- **Dimension clusters.** Come nel caso di YOLO9000, anche YOLOv3 utilizza l'algoritmo k -means clustering per determinare il numero di bounding box.
- **Feature extractor.** YOLOv3 utilizza una nuova architettura di rete per estrarre le caratteristiche, chiamata Darknet-53. Darknet-53, mostrata nella figura 2.9, è formata da 53 strati convoluzionali e connessioni residue introdotte dall'architettura di rete ResNet. Le connessioni residue permettono di portare l'output di uno strato ad altri strati consecutivi e non solo a quello successivo.

	Type	Filters	Size	Output
	Convolutional	32	3×3	256×256
	Convolutional	64	$3 \times 3 / 2$	128×128
1x	Convolutional	32	1×1	
	Convolutional	64	3×3	
	Residual			128×128
2x	Convolutional	128	$3 \times 3 / 2$	64×64
	Convolutional	64	1×1	
	Convolutional	128	3×3	
8x	Residual			64×64
	Convolutional	256	$3 \times 3 / 2$	32×32
	Convolutional	128	1×1	
8x	Convolutional	256	3×3	
	Residual			32×32
	Convolutional	512	$3 \times 3 / 2$	16×16
8x	Convolutional	256	1×1	
	Convolutional	512	3×3	
	Residual			16×16
4x	Convolutional	1024	$3 \times 3 / 2$	8×8
	Convolutional	512	1×1	
	Convolutional	1024	3×3	
	Residual			8×8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figura 2.9. Architettura di Darknet-53

Capitolo 3

Background

In questo capitolo si discuterà delle architetture e librerie che sono state utilizzate all'interno del progetto, introducendo nella prima parte i diversi passi per l'apprendimento di una rete neurale.

3.1 Apprendimento della rete neurale

3.1.1 Addestramento

L'addestramento di una rete [23] è un procedimento di ricerca dei kernel¹ negli strati di convoluzione e dei pesi² negli strati completamente connessi. Di seguito vengono analizzati i vari procedimenti.

3.1.1.1 Loss function

La loss function [23] è un iperparametro³ che calcola le prestazioni del modello sotto specifici kernel e pesi, mediante la forward propagation. La loss function utilizzata per la classificazione multiclass è la cross entropy, mentre l'errore quadratico medio(mean squared error) è attribuito alla regressione a valori continui. L'obiettivo adesso è quello di minimizzare la loss function e per farlo si utilizza la discesa del gradiente.

3.1.1.2 Discesa del gradiente

La discesa del gradiente(gradient descent) [23] è un algoritmo di ottimizzazione che aggiorna iterativamente i parametri apprendibili della rete, così da minimizzare la loss function. Il gradiente della loss function fornisce la direzione in cui la funzione ha il tasso di aumento più ripido e ogni parametro apprendibile viene aggiornato nella direzione negativa del gradiente, mediante un iperparametro chiamato learning rate. Il gradiente è, matematicamente, una derivata parziale rispetto a ciascun

¹Con il termine “kernel” si indica un insieme di parametri apprendibili impiegati nelle operazioni di convoluzione

²I “pesi” sono parametri che vengono automaticamente appresi durante l'addestramento

³Con il termine “iperparametro” si indicano le variabili che devono essere impostate prima dell'addestramento

parametro apprendibile e un singolo aggiornamento di un parametro è enunciato come segue:

$$w := w - \alpha \frac{\partial L}{\partial w} \quad (3.1)$$

dove:

- w rappresenta ogni parametro apprendibile.
- α rappresenta il learning rate.
- L rappresenta la loss function.

3.1.1.3 Backpropagation

L'algoritmo di backpropagation è stato presentato da D.Rumelhart et al. [18] nel 1986.

Con la backpropagation, il dato in ingresso viene cessato da uno strato al consecutivo per produrre un risultato. Questo procedimento è denominato forward propagation. Successivamente i pesi vengono aggiornati affinchè la rete approssima in maniera sempre più precisa la funzione richiesta.

3.1.2 Suddivisione dei dati

I dati di un dataset sono in genere divisi in tre insiemi [23]:

- **Training set.** Il training set è utilizzato per addestrare una rete. Nel training set, i valori della loss function sono calcolati attraverso la forward propagation, mentre i parametri apprendibili sono aggiornati tramite la backpropagation.
- **Validation set.** Il validation set è utilizzato per stimare il modello durante l'addestramento e per ottimizzare gli iperparametri.
- **Testing set.** Il testing set è utilizzato per valutare le prestazioni del modello finale.

3.1.3 Overfitting

L'overfitting [23] si attiene a una condizione in cui un modello finisce per memorizzare il rumore piuttosto che apprendere il segnale e, di conseguenza, non si comporta in modo corretto su un nuovo dataset. Sono state indicate svariate tecniche per diminuire al minimo l'overfitting:

- **Addestramento di un modello su un dataset più grande.** L'addestramento di un modello su un dataset più grande comporta, in genere, una migliore generalizzazione.
- **Data augmentation.** Data augmentation è una tecnica che modifica i dati di addestramento mediante delle trasformazioni, in modo che il modello non percepisce gli stessi input. Le trasformazioni disponibili sono il capovolgimento, la traslazione, il ritaglio, la rotazione, la sfocatura e molte altre.

- **Dropout.** Dropout è una tecnica di regolarizzazione in cui le attivazioni scelte casualmente sono impostate a 0 durante l'addestramento. In questo modo il modello diventa meno sensibile a pesi specifici nella rete.
- **Weight decay(Decadimento del peso).** Weight decay, chiamata anche regolarizzazione $L2$, è una tecnica di regolarizzazione che limita l'overfitting penalizzando i pesi del modello, in modo che prendono solo valori bassi.
- **Batch normalization.** Batch normalization è un tipo di strato aggiuntivo che normalizza i valori di ingresso dello strato consecutivo, diminuendo il rischio di overfitting.

3.1.4 Cross validation

Cross validation [1] è un metodo che serve per stimare come un modello sia capace di generalizzare dati mai visti prima e per verificare la presenza di overfitting. Di seguito viene analizzato un tipo di cross validation denominato k -fold cross validation.

3.1.4.1 K-fold Cross validation

Nel **k -fold⁴ cross validation** [1] il training set viene suddiviso in k sottoinsiemi separati, di proporzioni all'incirca equivalenti. Questa suddivisione viene effettuata selezionando casualmente i dati dal training set. Il modello viene addestrato utilizzando $k - 1$ sottoinsiemi che, insieme, definiscono il training set. Successivamente il modello viene impartito al sottoinsieme restante: il validation set. Il validation set viene utilizzato per valutare le prestazioni del modello. Questo procedimento viene reiterato finché ciascuno dei k sottoinsiemi non viene usato come validation set. Infine con la media delle k prestazioni, si ottiene la prestazione finale.

3.1.5 Transfer learning

Transfer learning [7, 23] è una tecnica in cui una rete viene addestrata per un compito e successivamente riadoperata per un compito affine. Una rete viene pre-addestrata su un dataset più grande, come ImageNet. Una rete pre-addestrata può essere utilizzata attraverso due metodi:

- **Estrazione delle caratteristiche.** L'estrazione delle caratteristiche viene usata per togliere gli strati completamente connessi da una rete pre-addestrata. La restante rete, invece, viene conservata ed è formata da una successione di strati di convoluzione e di pooling, nominata base convoluzionale.
- **Fine-tuning.** Fine-tuning non viene impiegato esclusivamente per sostituire gli strati completamente connessi del modello pre-addestrato, ma viene utilizzato anche per ottimizzare tutti o alcuni dei kernel nella base convoluzionale pre-addestrata, tramite la backpropagation.

⁴con il termine “fold” si indica il numero di sottoinsiemi

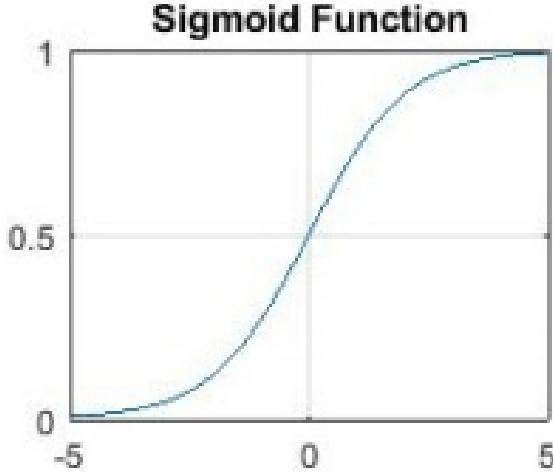


Figura 3.1. Funzione sigmoide

3.1.6 Funzioni di attivazione

Le **funzioni di attivazione** [10] sono usate nelle reti neurali per stabilire se un neurone può essere attivato o meno. Le principali funzioni di attivazione sono:

- **Funzione sigmoide(Sigmoid function).** La funzione sigmoide, mostrata nella figura 3.1, restituisce un valore compreso tra 0 e 1. La funzione sigmoide è specificata nel seguente modo:

$$f(x) = \left(\frac{1}{1 + \exp^{-x}} \right) \quad (3.2)$$

- **Funzione tangente iperbolica.** La funzione tangente iperbolica, mostrata nella figura 3.2, restituisce valori che sono compresi tra -1 e 1. La funzione tangente iperbolica è specificata nel seguente modo:

$$f(x) = \left(\frac{e^x - e^{-x}}{e^x + e^{-x}} \right) \quad (3.3)$$

- **Funzione softmax.** La funzione softmax genera un output che consiste in un intervallo di valori compresi tra 0 e 1, con la somma delle probabilità uguale a 1. La funzione softmax è specificata nel seguente modo:

$$f(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \quad (3.4)$$

- **Funzione ReLU(Rectified Linear Unit).** Con la funzione ReLU i valori minori di 0 sono posti a 0. La funzione ReLU è specificata nel seguente modo:

$$f(x) = \max(0, x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \quad (3.5)$$

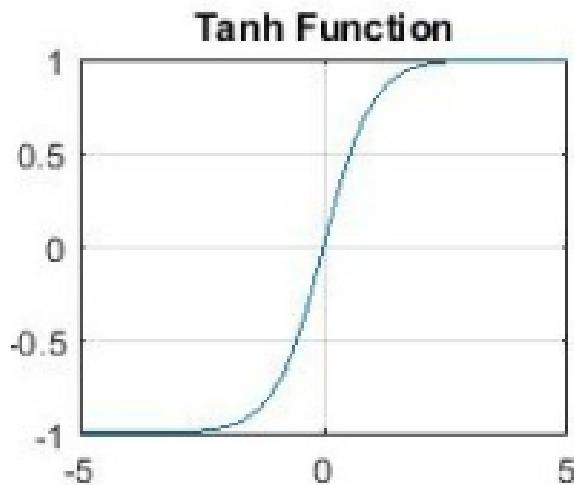


Figura 3.2. Funzione tangente iperbolica

3.1.7 Metriche di COCO

"Le metriche" utilizzate su un dataset in formato COCO(Common Objects in Context) [9] sono 12 e sono mostrate nella figura 3.3, in cui:

- "AP $^{IoU=.50}$ " e "AP $^{IoU=.75}$ " sono gli "Average Precision(AP)" calcolati prendendo solo le istanze che hanno "Intersection-over-Union(IoU) maggiore o uguale a 0.50 e 0.75".
- "AP a IoU=.50:.05:.95 è l'AP che modifica il valore di IoU. La modifica avviene incrementando di 0.05 ciascuna valutazione del dataset e restituendo la media tra le valutazioni".

3.2 Architetture

In questa sezione andiamo ad analizzare le architetture utilizzate per svolgere il lavoro di tesi. In particolare andiamo ad analizzare le reti neurali convoluzionali(CNN), reti neurali residuali(ResNet) e Feature Pyramid Network(FPN).

3.2.1 Rete neurale convoluzionale

Una **rete neurale convoluzionale(CNN: Convolutional Neural Network)** [23] viene utilizzata per eseguire le immagini. La CNN è formata da tre tipi di strati: strati di convoluzione(convolutional layers), strati di pooling(pooling layers) e strati completamente connessi(fully connected layers).

3.2.1.1 Strato di convoluzione

Uno strato di convoluzione effettua l'estrazione delle caratteristiche. L'estrazione delle caratteristiche consiste in un'unione di operazioni lineari e non lineari come l'operazione di convoluzione e la funzione di attivazione, dove:

```

Average Precision (AP):
    AP                      % AP at IoU=.50:.05:.95 (primary challenge metric)
    APIoU=.50            % AP at IoU=.50 (PASCAL VOC metric)
    APIoU=.75            % AP at IoU=.75 (strict metric)

AP Across Scales:
    APsmall              % AP for small objects: area < 322
    APmedium              % AP for medium objects: 322 < area < 962
    APlarge               % AP for large objects: area > 962

Average Recall (AR):
    ARmax=1              % AR given 1 detection per image
    ARmax=10             % AR given 10 detections per image
    ARmax=100            % AR given 100 detections per image

AR Across Scales:
    ARsmall              % AR for small objects: area < 322
    ARmedium              % AR for medium objects: 322 < area < 962
    ARlarge               % AR for large objects: area > 962

```

Figura 3.3. "Metriche di COCO, ricavate da: <https://cocodataset.org>"

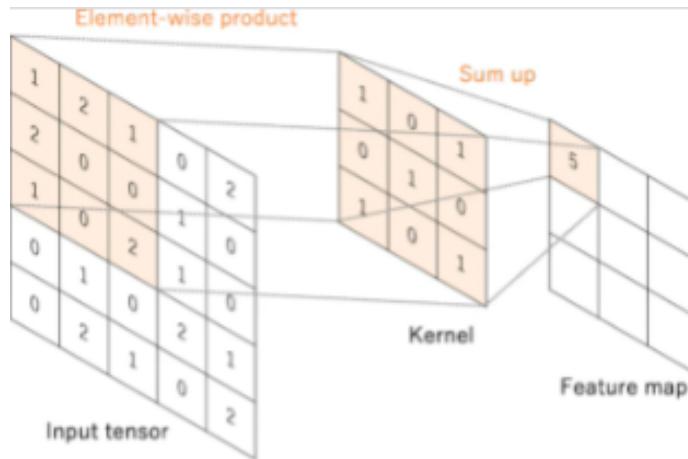


Figura 3.4. Esempio di operazione di convoluzione

- L'operazione di convoluzione, mostrata nella figura 3.4, viene utilizzata per l'estrazione delle caratteristiche. L'estrazione delle caratteristiche avviene assegnando il kernel a un tensore di input, mediante un procedimento chiamato prodotto scalare. Il prodotto scalare permette di far ottenere un valore in una posizione del tensore di output, denominata mappa di caratteristiche(feature map). Questo procedimento viene ripetuto impiegando più kernel, con lo scopo di ottenere un numero arbitrario di mappe di caratteristiche.
Ci sono due iperparametri che definiscono l'operazione di convoluzione: la dimensione e il numero di kernel.
La dimensione è solitamente 3x3, 5x5 o 7x7.
Il numero di kernel è arbitrario e stabilisce la profondità delle mappe di caratteristiche di output.
Lo spazio tra due posizioni consecutive del kernel è denominato stride. La scelta prevalente per uno stride è 1.

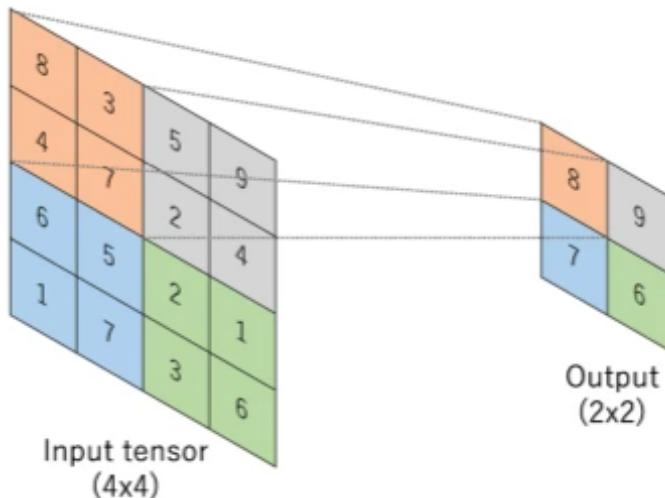


Figura 3.5. Esempio di operazione di max pooling

- Le uscite di un'operazione lineare, come la convoluzione, sono successivamente passate a una funzione di attivazione. La funzione di attivazione più utilizzata è la ReLU.

3.2.1.2 Strato di pooling

Uno **strato di pooling** esegue un'operazione di downsampling. L'operazione di downsampling permette di diminuire la dimensione delle mappe di caratteristiche. Ci sono due tipologie di operazioni di pooling:

- Max pooling.** Max pooling, mostrata nella figura 3.5, consente di ricavare delle patch da ciascuna mappa di caratteristiche, ritornando il valore massimo per ogni patch.
- Global average pooling.** Global average pooling è un'operazione di pooling in cui una mappa di caratteristiche di grandezza $w \times h$ viene ridotta in un array 1×1 , prendendo la media di ciascun elemento in ogni mappa di caratteristiche.

3.2.1.3 Strato completamente connesso

Gli **strati completamente connessi(fully connected layers)** hanno l'obiettivo di produrre la raffigurazione finale. Gli strati completamente connessi sono più volte collegati tra di loro e sono inoltre conosciuti come dense layers. Nei dense layers ogni input è connesso a ciascun output.

Sull'ultimo strato può essere inserita anche una funzione di attivazione. Nel caso della classificazione multiclasse, si utilizza la funzione softmax.

3.2.2 Rete neurale residuale

Con una rete neurale profonda, la difficoltà rilevata da K.He et al. [5] nel 2014 è che aumentando la profondità della rete, la precisione del modello si riduce. La

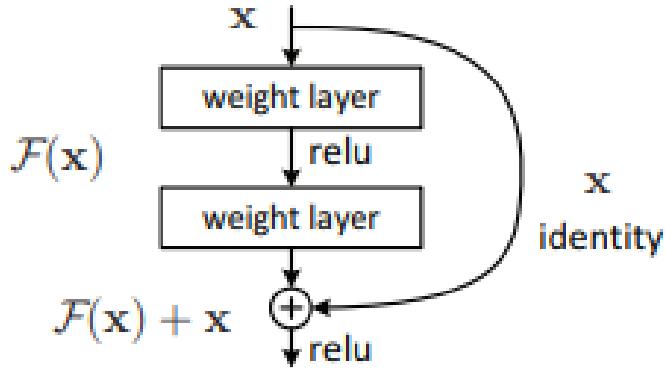


Figura 3.6. Raffigurazione di un building block

difficoltà non è originata dall’overfitting, ma è dovuta all’incremento dell’errore di addestramento. Per risolvere questa difficoltà, è stata presentata nel 2015 la **rete neurale residuale(ResNet)** da K.He et al. [6]. Con una rete neurale convoluzionale, ciascuno strato è connesso al suo successivo. Quindi lo strato successivo prende in carico l’ingresso x ritornato dal precedente output, ovvero $\mathcal{F}(x)$ mostrato nella figura 3.6.

Nella rete neurale residuale, invece, l’output prende il valore del suo residuo, ovvero $\mathcal{H}(x)=\mathcal{F}(x) + x$ mostrato nella figura 3.6.

Il blocco di operazioni della figura 3.6 è denominato building block.

3.2.3 Feature pyramid network

Nel 2017 T.Lin et al. presentano una rete denominata **Feature Pyramid Network(FPN)** [8], mostrata nella figura 3.7.

Il procedimento definito da T.Lin et al. è quello di produrre mappe di caratteristiche che permettono di conservare il valore semantico di quelle con minor risoluzione.

Si inizia dalla mappa di caratteristiche di minore risoluzione e maggiore valore semantico, presente nell’ultimo strato convoluzionale. Questa mappa di caratteristiche viene concatenata alla mappa di caratteristiche di maggiore risoluzione e minore valore semantico, dello strato successivo della piramide.

Il procedimento di concatenazione viene eseguito per ogni strato della piramide, fino ad arrivare all’ultima mappa di caratteristiche.

L’intero procedimento avviene mediante un “percorso top-down”, che scende la piramide dal basso verso l’alto attraverso le “connessioni laterali”. Il risultato è una piramide di caratteristiche ricca di semantica ad ogni livello, costruita prendendo in input un’immagine senza che essa venga ridimensionata.

3.3 Librerie

In questa sezione andiamo ad analizzare le librerie utilizzate per svolgere il lavoro di tesi. In particolare andiamo ad analizzare PyTorch, Scikit-learn e Detectron2.

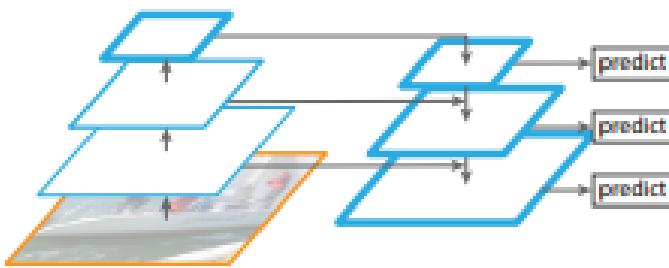


Figura 3.7. Feature Pyramid Network(FPN)

3.3.1 PyTorch

PyTorch [11] è una libreria Python per il machine learning, sviluppata da A.Paszke et al. nel 2019.

Con questa libreria si agevola:

1. L'implementazione di un'architettura di rete neurale, in cui gli strati di tale architettura sono rappresentati come classi Python. In ciascuna classe, i costruttori creano e inizializzano i parametri della rete, mentre i metodi forward elaborano un'attivazione in ingresso.
2. L'elaborazione dei modelli, rappresentati come classi.
3. Il caricamento dei dati, per poter inserire nuovi dataset.
4. L'addestramento e l'esecuzione degli ottimizzatori, per poter calcolare automaticamente i gradienti dei modelli.

Nello sviluppo della tesi è stata utilizzata la libreria PyTorch tramite Anaconda. Anaconda è una distribuzione Python per lo sviluppo di programmi per il machine learning. Anaconda fornisce ambienti virtuali come Jupyter e vari pacchetti, tra cui il pacchetto conda per l'utilizzo di PyTorch.

3.3.2 Scikit-learn

Scikit-learn [12], sviluppato da F.Pedregosa et al., è un modulo Python che integra algoritmi di machine learning per problemi supervisionati [20], ovvero problemi in cui tutti i campioni di addestramento sono etichettati con la classe corretta, e problemi non supervisionati [20], ovvero problemi in cui tutti i campioni di addestramento non sono etichettati.

In Scikit-learn l'oggetto principale è uno stimatore. Lo stimatore, attraverso la sua valutazione, permette di analizzare i dati e di dargli una stima.

L'altro oggetto fondamentale in Scikit-learn è il "cross validation iterator". Il "cross validation iterator" viene usato per valutare le prestazioni di uno stimatore e per fornire coppie di indici di train e test per dividere i dati.

3.3.3 Detectron2

Detectron2 [22] è una libreria per il machine learning scritta in PyTorch e rilasciata nel 2019 da Facebook Artificial Intelligence Research(FAIR).

Detectron2 supporta un certo numero di architetture e fornisce un addestramento veloce su server GPU(Graphics Processing Unit) singoli o multipli per vari tipi di modelli. Detectron2 dispone del proprio Model Zoo, ossia una repository che raccoglie vari modelli pre-addestrati sul dataset COCO. Per ciascun modello pre-addestrato sono presenti le relative backbone, per l'estrazione delle caratteristiche da un'immagine di input.

Oltre agli innumerevoli vantaggi, la libreria presenta anche dei vincoli ricavabili dai requisiti di installazione:

1. Può essere utilizzata in locale solo con i sistemi operativi Linux e macOS, con la versione di Python ≥ 3.6 .
2. Dipende dalla libreria PyTorch.
3. Serve necessariamente di una GPU per essere utilizzata, poiché da CPU(Central Processing Unit) si riducono i casi d'uso.
4. Durante la configurazione dell'addestramento si possono utilizzare solo modelli pre-addestrati, con le relative backbone, presenti sul Model Zoo.
5. Per l'addestramento non sono supportate immagini con dimensioni maggiori di $1024 \times 1024 \times 3$.

Capitolo 4

Progetto

In questo capitolo si illustrerà lo svolgimento del progetto, mostrando i software impiegati e le fasi che lo hanno composto.

4.1 Prerequisiti del progetto

La parte principale del progetto consiste nell'addestramento di una rete neurale per il rilevamento degli oggetti. Gli oggetti in questione sono dei vasi. Per effettuare l'addestramento si utilizza l'operazione di transfer learning con dei pesi già addestrati, poiché con l'addestramento da zero sarebbe stato molto costoso e avrebbe difficilmente prodotto risultati in breve tempo. In fase di train e test è stata utilizzata una GPU Tesla K80 tramite Google Colab. Google Colab è una piattaforma che permette l'esecuzione del codice su cloud usando i Jupyter Notebook, ovvero dei documenti nei quali è possibile elaborare righe di codice. Con tale piattaforma è stato possibile effettuare anche l'installazione della libreria Detectron2.

Altri passi basilari che sono stati svolti prima dell'addestramento sono:

- Creazione del dataset.
- Creazione delle etichette, mediante il tool Yolo mark.
- Implementazione dell'algoritmo k -fold cross validation per valutare l'accuratezza del dataset, mediante la libreria PyTorch e Scikit-learn.

Per lo svolgimento di queste fasi si è favorito lavorare in locale.

4.2 Dataset

4.2.1 Creazione del dataset

Inizialmente mi è stato fornito un dataset di 22 immagini, raffiguranti dei vasi. Le immagini sono state suddivise in 5 classi, in base al tipo di vaso in questione. Successivamente su di esse sono state applicate un insieme di trasformazioni in 4 modi diversi, tramite il package torchvision della libreria PyTorch, ottenendo 88 immagini con dimensioni pari a $463 \times 463 \times 3$. Le trasformazioni, mostrate nella tabella 4.1, sono:

Tabella 4.1. L'insieme di trasformazioni applicate alle 22 immagini del dataset iniziale

1'Modo	2'Modo	3'Modo	4'Modo
Resize CenterCrop ToTensor	RandomRotation RandomResizedCrop RandomHorizontalFlip ToTensor	Resize RandomCrop RandomVerticalFlip GaussianBlur ToTensor	Resize CenterCrop ColorJitter ToTensor

**Figura 4.1.** Le immagini ottenute mediante le trasformazioni del package torchvision, della libreria PyTorch

- **Resize.** Resize effettua il ridimensionamento dell'immagine.
- **CenterCrop.** CenterCrop effettua il ritaglio, al centro dell'immagine.
- **RandomRotation.** RandomRotation effettua la rotazione dell'immagine.
- **RandomResizedCrop.** RandomResizedCrop effettua il ritaglio dell'immagine, a dimensioni e proporzioni casuali.
- **RandomHorizontalFlip.** RandomHorizontalFlip capovolge orizzontalmente l'immagine.
- **RandomVerticalFlip.** RandomVerticalFlip capovolge verticalmente l'immagine.
- **RandomCrop.** RandomCrop effettua il ritaglio, in una posizione casuale dell'immagine.
- **GaussianBlur.** GaussianBlur effettua la sfocatura gaussiana su un'immagine.
- **ColorJitter.** ColorJitter cambia in maniera casuale la luminosità di un'immagine.
- **ToTensor.** ToTensor effettua la conversione delle immagini in un tensore.

Le trasformazioni, per poter essere elaborate, devono essere inserite all'interno della trasformazione `Compose`. `Compose` è una trasformazione del package `torchvision` che combina le trasformazioni in una pipeline di trasformazioni. L'output così ottenuto è mostrato nella figura 4.1. Su ciascuna immagine è presente anche un numero, che indica la classe a cui appartiene.

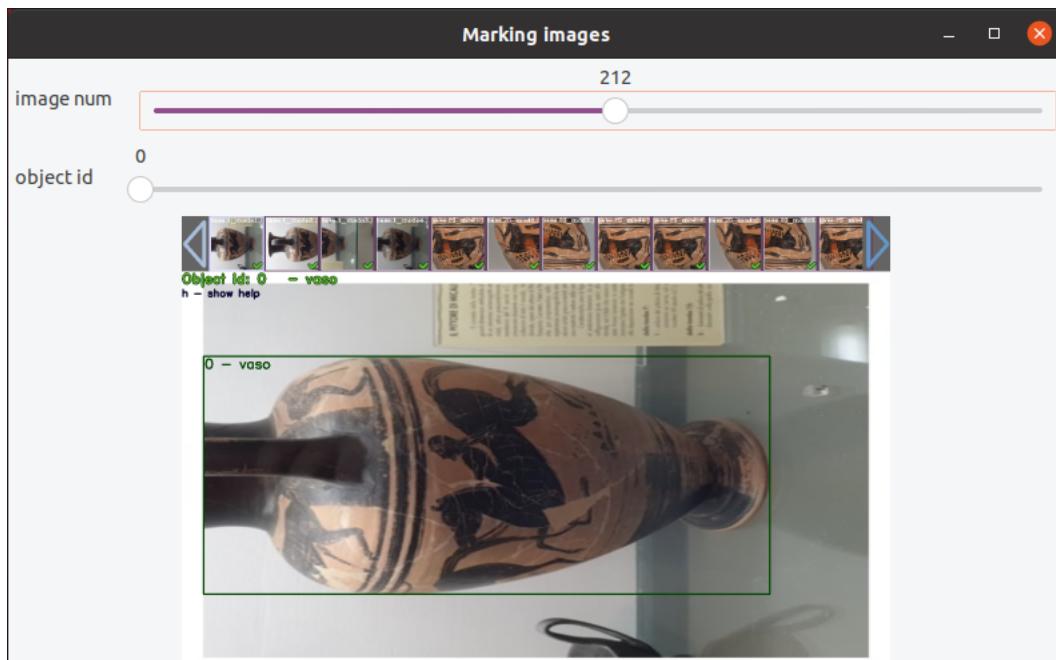


Figura 4.2. Localizzazione ed etichettatura tramite Yolo mark

Infine queste trasformazioni sono state applicate anche in maniera manuale, rispetto alle 22 immagini del dataset iniziale, utilizzando il tool GNU Image Manipulation(GIMP).

4.2.2 Creazione delle etichette ed ottenimento del dataset finale

Parte essenziale è la strutturazione del dataset. Ciò che è stato fatto, è stato quello di assegnare delle etichette su determinate aree(o zone) dell'immagine, chiamati bounding box, utilizzando il tool Yolo mark.

Partendo dal dataset ottenuto nella sottosezione 4.2.1, per prima cosa sono state aggiunte altre 128 immagini casuali del dataset COCO e 104 immagini raffiguranti sempre dei vasi. Successivamente tramite Yolo mark sono state saltate tutte le immagini sulle quali non sono stati individuati gli oggetti d'interesse, quindi non localizzando ed etichettando nessuna area dell'immagine. Sulle restanti immagini, invece, sono state effettuate le relative localizzazioni ed etichettature su una o più aree. Un esempio è mostrato nella figura 4.2.

Tramite Yolo mark, per ciascuna immagine, viene creato un file. Il file potrebbe essere vuoto, oppure contenere una o più coordinate. Ciascuna coordinata è impostata nel seguente modo: "ID X Y WIDTH HEIGHT", in cui

- ID è l'identificazione attribuita alle diverse classi definite. Nel nostro caso sono state definite 5 classi comprese tra ID=0 e ID=4.
- X indica la coordinata X del centro dell'oggetto.
- Y indica la coordinata Y del centro dell'oggetto.

- WIDTH indica la larghezza dell'oggetto.
- HEIGHT indica l'altezza dell'oggetto.

Rispetto alla figura 4.2, si ha la seguente coordinata: "0 0.430469 0.511806 0.800000 0.598611". Infine una volta applicate tutte le localizzazioni ed etichettature, il dataset finale è formato da 204 immagini e 204 etichette.

4.3 Train

4.3.1 K-fold Cross validation

Una volta ottenuto il dataset, è stato applicato l'algoritmo *k*-fold cross validation per analizzare la sua accuratezza. L'implementazione dell'algoritmo è avvenuta mediante l'utilizzo di due librerie:

- **Scikit-learn.** Scikit-learn è stata utilizzata per impostare il numero di fold da applicare sul dataset. E' stato scelto k=5 come numero di fold, ottenendo una suddivisione in cui l'80% fa riferimento al training set, con 163 immagini, mentre il restante 20% fa riferimento al testing set, con 41 immagini.
- **PyTorch.** PyTorch è stata utilizzata per l'addestramento del training set, con un numero di epoch pari a 500. Al termine delle 500 epoch, è stata valutata l'accuratezza utilizzando il testing set. Il testing set contiene i dati che non sono stati addestrati. Il processo di addestramento e di valutazione avviene alternando ogni volta l'80% del training set e il 20% del testing set, in base al numero di fold scelti tramite la libreria Scikit-learn.

Una volta eseguito l'algoritmo, l'accuratezza ottenuta è del 99%.

4.3.2 Conversione e suddivisione del dataset

Essendo che la libreria Detectron2 supporta solo dataset in formato COCO, l'obiettivo è stato quello di convertire le etichette dal formato YOLO TXT al formato COCO JSON(JavaScript Object Notation). La conversione è stata effettuata tramite Roboflow. Roboflow è un convertitore che permette di effettuare il caricamento e l'esportazione delle annotazioni in un qualsiasi formato, tra cui da YOLO TXT a COCO JSON.

In seguito è stato diviso il dataset per eseguire l'addestramento tramite Detectron2. Nella tabella 4.2 è mostrata la partizione del dataset, con 145 immagini per il training set, 34 immagini per il validation set e 25 immagini per il testing set.

4.3.3 Scelta del modello e relativa backbone

Per quanto riguarda la scelta del modello, Detectron2 mette a disposizione una repository chiamata Model Zoo. Il Model Zoo contiene diversi modelli pre-addestrati sul dataset COCO che, successivamente, potranno essere addestrati su un determinato dataset.

Il modello utilizzato è Faster RCNN. Rispetto alla figura 4.3, è stata scelta la

Tabella 4.2. Partizione del dataset per l'addestramento

Set	Numero di immagini
Training set	145
Validation set	34
Testing set	25
Total	204

Name	lr sched	train time (s/iter)	inference time (s/im)	train mem (GB)	box AP	model id	download
R50-C4	1x	0.551	0.102	4.8	35.7	137257644	model metrics
R50-DC5	1x	0.380	0.068	5.0	37.3	137847829	model metrics
R50-FPN	1x	0.210	0.038	3.0	37.9	137257794	model metrics
R50-C4	3x	0.543	0.104	4.8	38.4	137849393	model metrics
R50-DC5	3x	0.378	0.070	5.0	39.0	137849425	model metrics
R50-FPN	3x	0.209	0.038	3.0	40.2	137849458	model metrics
R101-C4	3x	0.619	0.139	5.9	41.1	138204752	model metrics
R101-DC5	3x	0.452	0.086	6.1	40.6	138204841	model metrics
R101-FPN	3x	0.286	0.051	4.1	42.0	137851257	model metrics
X101-FPN	3x	0.638	0.098	6.7	43.0	139173657	model metrics

Figura 4.3. Model Zoo di Detectron2, per il modello pre-addestrato Faster RCNN

backbone R50-FPN-3x. Tale backbone, in termini di velocità ed accuratezza per l'estrazione delle caratteristiche, è la migliore tra quelle elencate. Tale backbone è formata da una ResNet con 50 livelli di convoluzione e da una FPN per l'estrazione delle caratteristiche.

In Detectron2, inoltre, si utilizzano i pesi in riferimento al task di classificazione ImageNet.

4.3.4 Addestramento

Una volta definito il dataset e scelto il modello, è possibile iniziare l'addestramento utilizzando le Application Programming Interface(API) di Detectron2. E' indispensabile prima la realizzazione di una configurazione impostando i vari parametri ed iperparametri più essenziali per regolare le prestazioni della rete. Successivamente è possibile eseguire l'addestramento del modello pre-addestrato rispetto al dataset definito.

La configurazione per il train è la seguente:

```

1 #Configurazione per il train
2 from detectron2.config import get_cfg
3 from detectron2 import model_zoo
4 from detectron2.evaluation import COCOEvaluator
5 import os
6
7 cfg=get_cfg()
8
9 cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/
    faster_rcnn_R_50_FPN_3x.yaml"))
10
11 cfg.DATASETS.TRAIN=("my_dataset_train",)
12 cfg.DATASETS.TEST=("my_dataset_val",)
13
14 cfg.DATALOADER.NUM_WORKERS=4
15
16 cfg.MODEL.WEIGHTS=model_zoo.get_checkpoint_url("COCO-Detection/
    faster_rcnn_R_50_FPN_3x.yaml")
17
18 cfg.SOLVER.IMS_PER_BATCH=4
19 cfg.SOLVER.BASE_LR=0.001
20
21 cfg.SOLVER.MAX_ITER=500
22
23 cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE=64
24 cfg.MODEL.ROI_HEADS.NUM_CLASSES=5
25
26 cfg.TEST.EVAL_PERIOD=100
27 os.makedirs(cfg.OUTPUT_DIR,exist_ok=True)
28 trainer=COCOEvaluator(cfg)
29 trainer.resume_or_load(resume=False)
30 trainer.train()

```

- Con **get_cfg** si acquisisce una copia della configurazione scelta.
- **model_zoo** contiene una raccolta di funzioni per creare architetture di modelli elencati nel Model Zoo.
- **cfg.merge_from_file** unisce la configurazione scelta, per eseguire l'addestramento.
- Con **cfg.DATASETS.TRAIN** viene analizzato il training set durante l'addestramento.
- Con **cfg.DATASETS.TEST** viene analizzato il validation set durante l'addestramento.
- **DATALOADER.NUM_WORKERS** denota il numero di thread utilizzati per il caricamento dei dati.
- **cfg.MODEL.WEIGHTS** denota i pesi in riferimento al task di classificazione ImageNet.
- **cfg.SOLVER.IMS_PER_BATCH** denota il numero di immagini che vengono addestrate, per batch.

- **cfg.SOLVER.BASE_LR** denota il learning rate.
- **cfg.SOLVER.MAX_ITER** denota il numero di iterazioni.
- **cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE** denota il numero di regioni d'interesse.
- **cfg.MODEL.ROI_HEADS.NUM_CLASSES** denota il numero di classi rispetto al dataset inserito. Nel nostro caso sono state definite 5 classi. Ciascuna classe contiene le istanze delle immagini.
- **cfg.TEST.EVAL_PERIOD** denota il periodo per valutare il modello durante l'addestramento. Nel nostro caso la valutazione viene effettuata ogni 100 iterazioni tramite il validation set.
- Con **COCOEvaluator** si valutano le proposte di oggetti e il rilevamento delle istanze, mediante le metriche di COCO.

Durante l'addestramento vengono esposti vari valori tra cui il numero dell'iterazione che si sta eseguendo e il valore aggiornato della loss function. Ogni 100 iterazioni avviene la valutazione del modello, stampando tutte le metriche di COCO e salvando il risultato in un checkpoint. Una volta terminato l'addestramento, viene presentato il valore totale della loss function. Inizialmente si è partiti con un valore pari a 2.993, per arrivare a un valore pari a 0.233.

4.4 Test

Dopo aver completato l'addestramento, è stata svolta la fase di test. La fase di test verifica se le classi di oggetti presenti su ciascuna immagine vengono individuate o meno, con i relativi bounding box. Nella fase di test sono stati utilizzati dati mai visti prima, ovvero quelli appartenenti al testing set.

La configurazione per il test è la seguente:

```

1 #Configurazione per il test
2 from detectron2.config import get_cfg
3 from detectron2.engine import DefaultPredictor
4 from detectron2.data import build_detection_test_loader
5 from detectron2.evaluation import COCOEvaluator, inference_on_dataset
6 import os
7
8 cfg.MODEL.WEIGHTS=os.path.join(cfg.OUTPUT_DIR,"model_final.pth")
9 cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST=0.85
10 predictor=DefaultPredictor(cfg)
11
12 evaluator=COCOEvaluator("my_dataset_test",cfg,False,output_dir="./
    output/")
13
14 val_loader=build_detection_test_loader(cfg,"my_dataset_test")
15 inference_on_dataset(trainer.model,val_loader,evaluator)
```

- **cfg.MODEL.WEIGHTS** denota i pesi in riferimento al task di classificazione ImageNet, con **os.path.join** che indica il percorso verso il modello appena addestrato.

- `cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST` denota il "threshold test" per misurare la sovrapposizione di un bounding box previsto, rispetto a quello effettivo per un oggetto.
- Con **DefaultPredictor** si crea un predittore end-to-end, per produrre un solo output al posto di un batch di immagini.
- Con **COCOEvaluator** si valutano le proposte di oggetti e il rilevamento delle istanze, mediante le metriche di COCO.
- Con **build_detection_test_loader** si producono un insieme di campioni.
- Con **inference_on_dataset** si esegue il modello, su tutti i campioni del dataset.

4.4.1 Metriche di COCO

Sia in fase di train che di test, il modello è stato valutato tramite la "metrica di COCO con AP a IoU=.50:.05:.95". Con questa metrica si considerano i valori di IoU relativi ai bounding box, in cui:

- Nel caso del train è stata ottenuta la seguente valutazione: AP=79.907%
- Nel caso del test è stata ottenuta la seguente valutazione: AP=81.760%

Capitolo 5

Risultati

Dopo aver completato sia il train che il test, è stata effettuata l'inferenza per visualizzare i risultati della previsione rispetto alle immagini del testing set. Le immagini in discussione sono riportate di seguito e si possono trarre da esse delle considerazioni:

1. Sull'**immagine 1**:

- C'è un caso di "falso positivo" sull'oggetto a sinistra, poiché viene individuato un bounding box etichettato con la classe numero 4 e invece non dovrebbe esserci nulla su di esso.
- C'è un caso di "vero negativo" sull'oggetto centrale, poiché è stata sbagliata l'etichetta.

2. Sull'**immagine 11** c'è un caso di "falso negativo", poiché non viene rilevato l'oggetto centrale etichettato con la classe numero 3.

3. Sull'**immagine 18** c'è un caso di "falso positivo" sull'oggetto a sinistra, poiché viene individuato un bounding box etichettato con la classe numero 3 e invece non dovrebbe esserci nulla su di esso.

4. Su tutte le altre **immagini** sono stati individuati correttamente gli oggetti d'interesse, con le relative classi. L'unica difformità è presente su due bounding box dell'**immagine 4** e **immagine 10**: i due bounding box non coprono precisamente l'area dell'oggetto di interesse.

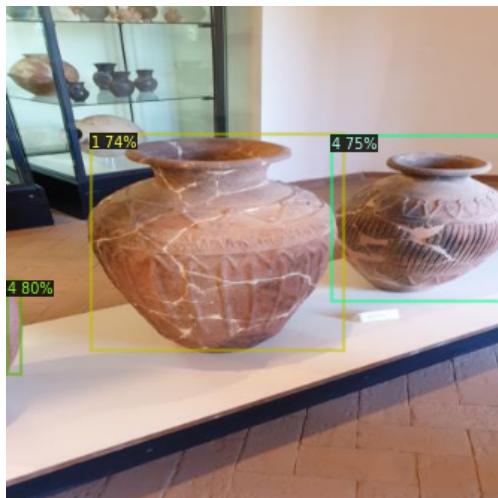


Figura 5.1. Immagine 1



Figura 5.2. Immagine 2

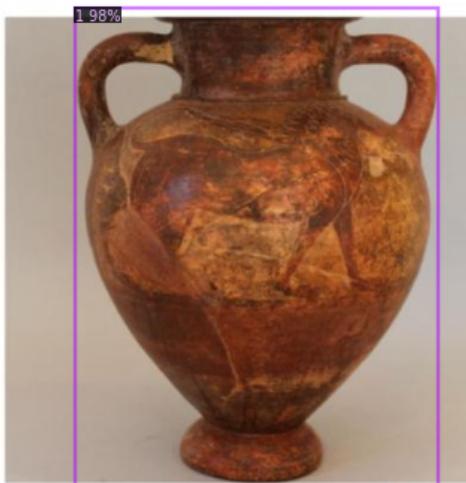


Figura 5.3. Immagine 3

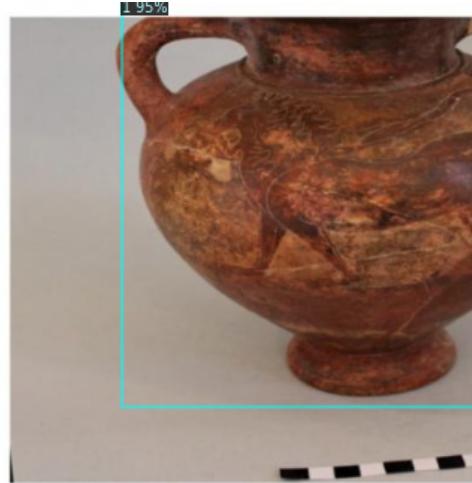


Figura 5.4. Immagine 4

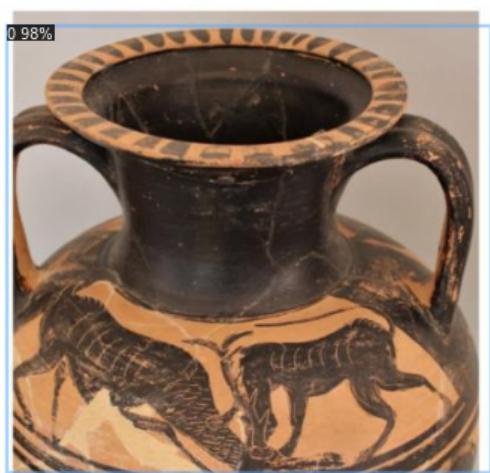


Figura 5.5. Immagine 5



Figura 5.6. Immagine 6



Figura 5.7. Immagine 7



Figura 5.8. Immagine 8

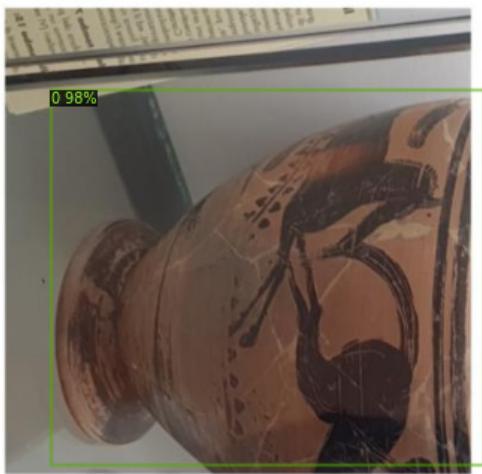


Figura 5.9. Immagine 9



Figura 5.10. Immagine 10



Figura 5.11. Immagine 11

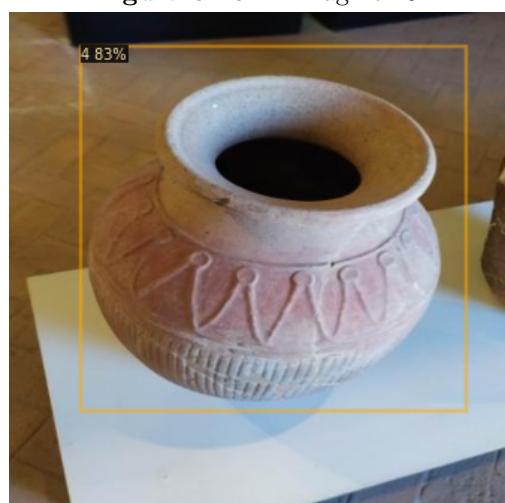


Figura 5.12. Immagine 12

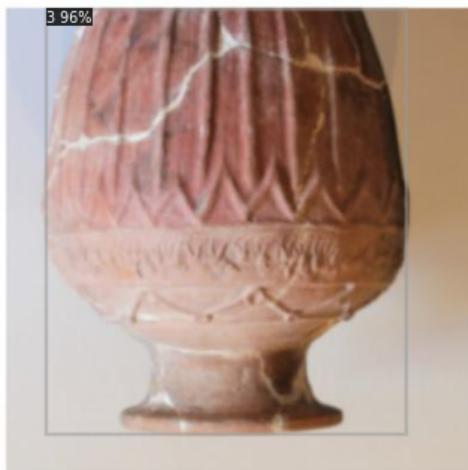


Figura 5.13. Immagine 13

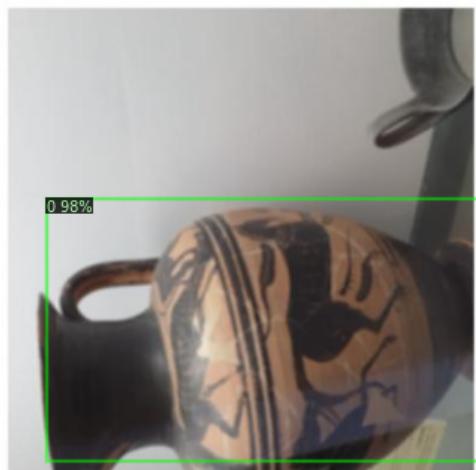


Figura 5.14. Immagine 14



Figura 5.15. Immagine 15

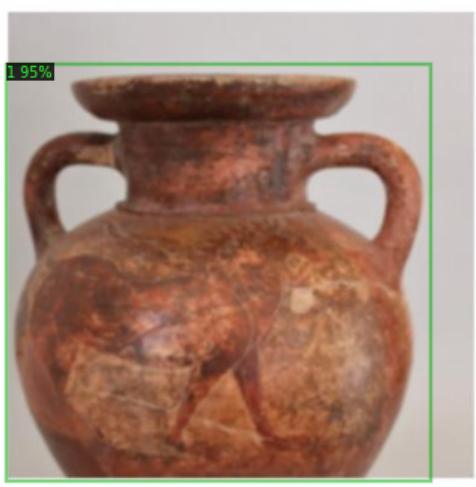


Figura 5.16. Immagine 16

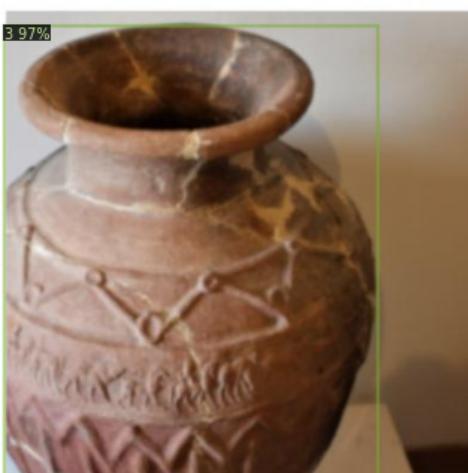


Figura 5.17. Immagine 17



Figura 5.18. Immagine 18

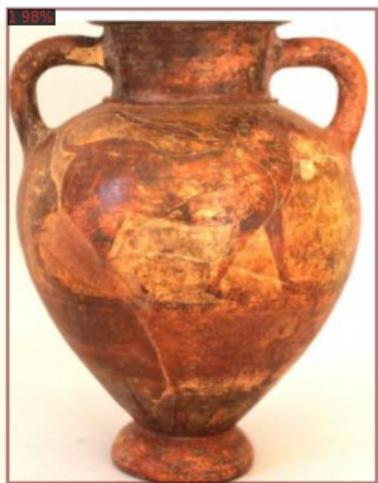


Figura 5.19. Immagine 19



Figura 5.20. Immagine 20



Figura 5.21. Immagine 21

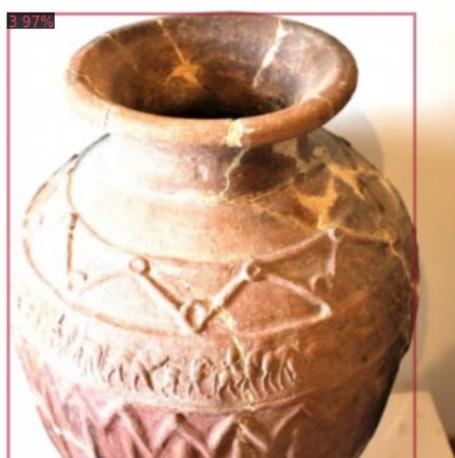


Figura 5.22. Immagine 22



Figura 5.23. Immagine 23



Figura 5.24. Immagine 24



Figura 5.25. Immagine 25

Capitolo 6

Conclusioni e Sviluppi Futuri

In questa attività di tirocinio è stato realizzato un sistema per il rilevamento e classificazione di opere museali.

Il lavoro è cominciato con lo studio dello stato dell'arte su metodologie di reti neurali per fare object detection. In particolare, sono state approfondite le reti neurali region-based CNN e le reti neurali a singola fase, focalizzandoci sui loro funzionamenti e sui risultati ottenuti. Una volta effettuato questo studio, si è scelto di utilizzare la libreria Detectron2 per l'implementazione della rete neurale Faster RCNN, per il riconoscimento di opere museali. Nella fase iniziale è stata eseguita la costruzione del dataset mediante l'applicazione di trasformazioni. Finalizzato il dataset, si è passati alla fase di strutturazione del dataset con l'etichettatura dei bounding box sugli oggetti d'interesse delle immagini. Successivamente è stato effettuato lo sviluppo per il riconoscimento dei bounding box.

Alla luce dei risultati ottenuti si è dimostrato che, con il dataset addestrato, si è ottenuta un'Average Precision(AP) per il train del circa 80% e un'AP per il test del circa 81%.

Poichè lo stato dell'arte è in continuo aggiornamento, uno sviluppo futuro sarà quello di implementare una rete neurale più recente, con lo scopo di migliorare le prestazioni.

Un ulteriore sviluppo futuro che si potrà effettuare è aumentare il dataset con altre classi, così da avere più tipologie di dati a disposizione.

Bibliografia

- [1] BERRAR, D. Cross-validation. In *Encyclopedia of Bioinformatics and Computational Biology* (edited by S. Ranganathan, M. Gribskov, K. Nakai, and C. Schönbach), pp. 542–545. Academic Press, Oxford (2019). ISBN 978-0-12-811432-2. Available from: <https://www.sciencedirect.com/science/article/pii/B978012809633820349X>, doi:<https://doi.org/10.1016/B978-0-12-809633-8.20349-X>.
- [2] FELZENZWALB, P. F. AND HUTTENLOCHER, D. Efficient graph-based image segmentation. *International Journal of Computer Vision*, **59** (2004), 167.
- [3] GIRSHICK, R. Fast r-cnn (2015). [arXiv:1504.08083](https://arxiv.org/abs/1504.08083).
- [4] GIRSHICK, R., DONAHUE, J., DARRELL, T., AND MALIK, J. Rich feature hierarchies for accurate object detection and semantic segmentation (2014). [arXiv:1311.2524](https://arxiv.org/abs/1311.2524).
- [5] HE, K. AND SUN, J. Convolutional neural networks at constrained time cost (2014). [arXiv:1412.1710](https://arxiv.org/abs/1412.1710).
- [6] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition (2015). [arXiv:1512.03385](https://arxiv.org/abs/1512.03385).
- [7] HUSSAIN, M., BIRD, J. J., AND FARIA, D. R. A study on cnn transfer learning for image classification. In *UKCI* (2018).
- [8] LIN, T.-Y., DOLLÁR, P., GIRSHICK, R., HE, K., HARIHARAN, B., AND BELONGIE, S. Feature pyramid networks for object detection (2017). [arXiv:1612.03144](https://arxiv.org/abs/1612.03144).
- [9] LIN, T.-Y., ET AL. Microsoft coco: Common objects in context (2015). [arXiv:1405.0312](https://arxiv.org/abs/1405.0312).
- [10] NWANKPA, C., IJOMAH, W., GACHAGAN, A., AND MARSHALL, S. Activation functions: Comparison of trends in practice and research for deep learning (2018). [arXiv:1811.03378](https://arxiv.org/abs/1811.03378).
- [11] PASZKE, A., ET AL. Pytorch: An imperative style, high-performance deep learning library (2019). [arXiv:1912.01703](https://arxiv.org/abs/1912.01703).
- [12] PEDREGOSA, F., ET AL. Scikit-learn: Machine learning in python (2018). [arXiv:1201.0490](https://arxiv.org/abs/1201.0490).

- [13] REDMON, J., DIVVALA, S., GIRSHICK, R., AND FARHADI, A. You only look once: Unified, real-time object detection (2016). [arXiv:1506.02640](https://arxiv.org/abs/1506.02640).
- [14] REDMON, J. AND FARHADI, A. Yolo9000: Better, faster, stronger (2016). [arXiv:1612.08242](https://arxiv.org/abs/1612.08242).
- [15] REDMON, J. AND FARHADI, A. Yolov3: An incremental improvement (2018). [arXiv:1804.02767](https://arxiv.org/abs/1804.02767).
- [16] REN, S., HE, K., GIRSHICK, R., AND SUN, J. Faster r-cnn: Towards real-time object detection with region proposal networks (2016). [arXiv:1506.01497](https://arxiv.org/abs/1506.01497).
- [17] REZATOFIGHI, H., TSOI, N., GWAK, J., SADEGHIAN, A., REID, I., AND SAVARESE, S. Generalized intersection over union: A metric and a loss for bounding box regression (2019). [arXiv:1902.09630](https://arxiv.org/abs/1902.09630).
- [18] RUMELHART, D. E., HINTON, G. E., AND WILLIAMS, R. J. Learning Representations by Back-propagating Errors. *Nature*, **323** (1986), 533. Available from: <http://www.nature.com/articles/323533a0>, doi:10.1038/323533a0.
- [19] RUSSAKOVSKY, O., ET AL. Imagenet large scale visual recognition challenge (2015). [arXiv:1409.0575](https://arxiv.org/abs/1409.0575).
- [20] SZE, V., CHEN, Y.-H., YANG, T.-J., AND EMER, J. Efficient processing of deep neural networks: A tutorial and survey (2017). [arXiv:1703.09039](https://arxiv.org/abs/1703.09039).
- [21] UIJLINGS, J., SANDE, K., GEVERS, T., AND SMEULDERS, A. Selective search for object recognition. *International Journal of Computer Vision*, **104** (2013), 154. doi:10.1007/s11263-013-0620-5.
- [22] WU, Y., KIRILLOV, A., MASSA, F., LO, W.-Y., AND GIRSHICK, R. Detectron2. <https://github.com/facebookresearch/detectron2> (2019).
- [23] YAMASHITA, R., NISHIO, M., DO, R., AND TOGASHI, K. Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, **9** (2018). doi:10.1007/s13244-018-0639-9.

Ringraziamenti

A conclusione di questo lavoro desidero menzionare tutte le persone che hanno contribuito, con il loro instancabile supporto, alla realizzazione di questa tesi e che mi hanno accompagnato in questi anni di carriera universitaria.

Un ringraziamento al professore Alessandro Mei che mi ha permesso di svolgere questo tirocinio e ai ragazzi del RFID LAB che sono stati eccezionali, in particolare: Massimo e Alberto.

Ringrazio di cuore i miei genitori. Grazie per avermi sempre sostenuto e che avete fatto di tutto per farmi completare questo percorso universitario.

In particolare voglio ringraziare mia madre che mi è stata sempre accanto, aiutandomi e confortandomi nei momenti più difficili, cercando sempre di darmi consigli nelle mie scelte.

In particolare voglio ringraziare mio padre. Anche se abbiamo avuto sempre molti battibecchi per l'università, so che in fondo sei fiero di chi sono.

Un ringraziamento speciale va a mio nonno Giorgio, per essermi stato sempre vicino con le tue parole rassicuranti.

Un ringraziamento speciale va al mio amico Lorenzo, un fratello. In questo percorso universitario sei sempre stato una spalla a cui appoggiarmi e con cui liberarmi sia delle incazzature che dei momenti belli. Lo ricorderò per sempre.

Un ringraziamento speciale va al mio amico Vincenzo. La nostra amicizia nata qui alla Sapienza va al di fuori del semplice collega. Se sono arrivato fin qui, è anche merito tuo.

Un ringraziamento speciale va al mio amico Gabriele. Anche se il destino ha voluto che io me ne andassi da Tor Vergata, la nostra amicizia fortunatamente non è mai terminata.

Ringrazio il mio amico Gioele che è stato sempre disponibile, in qualsiasi situazione.

Un pensiero va anche ai miei amici: Valerio, Gianluca e Lorenzo. Con voi ci siamo conosciuti al momento del trasferimento eabbiamo scambiato qui alla Sapienza molte sofferenze, ma anche tantissime gioie. Grazie per esserci sempre.

Un pensiero va anche ai miei amici, non colleghi: Stefano, Alessandro, Luca, Gabriel, Lorenzo, Domenico, Edoardo e Matteo.

Un pensiero va anche a mia cugina Aurora, che sei stata sempre disponibile a sopportare le mie ansie per l'università.

Un pensiero va anche a mia zia Marisa, che sei stata sempre vicina a rincuo-

rarmi prima e dopo un esame e per questo lavoro di tesi.

Un pensiero va anche ai miei amici di sempre, che mi hanno sostenuto.

Un ultimo pensiero va a chi è sempre al mio fianco, anche se non è più in questo mondo. Mi manchi nonna.