



UNIVERSITÀ
DEGLI STUDI
FIRENZE

Scuola di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Magistrale in Informatica
Curriculum: *Data Science*

TECNICHE DI APPRENDIMENTO
PROFONDO PER RICONOSCIMENTO DI
OGGETTI IN VIDEO TERMICI

DEEP LEARNING TECHNIQUES FOR
OBJECT DETECTION IN THERMAL VIDEOS

FEDERICO SCHIPANI

Relatore: Prof. *Marco Bertini*

Anno Accademico 2018-2019

Federico Schipani: *Tecniche di apprendimento profondo per riconoscimento di oggetti in video termici*, Corso di Laurea Magistrale in Informatica, © Anno Accademico 2018-2019

INDICE

Introduzione	9
1 Object Detection	11
1.1 Storia della object detection	11
1.1.1 Evoluzione delle tecniche	11
1.1.2 Dataset	20
1.2 Detector basati su metodi tradizionali	26
1.3 Detector basati su Deep Learning	28
1.3.1 Estrattori di feature	28
1.3.2 Two Stage Detector	30
1.3.3 One Stage Detector	33
1.4 RetinaNet	37
1.4.1 Focal Loss	38
1.4.2 Struttura del detector	40
2 SISTEMA	43
2.1 Dataset	43
2.1.1 KAIST Multispectral Pedestrian Dataset	43
2.1.2 FLIR Thermal Starter Dataset	46
2.2 Addestramento iniziale di RetinaNet	48
2.2.1 Transfer Learning	48
2.2.2 Addestramento sulle immagini RGB	52
2.2.3 Passaggio alle immagini termiche su KAIST	53
2.3 Data Augmentation	56
2.3.1 Auto Augment	57
2.3.2 Rand Augment	59
3 Esperimenti	61
3.1 Organizzazione dei dataset	61
3.1.1 KAIST Multispectral Pedestrian Dataset	61
3.1.2 FLIR Thermal Starter Dataset	63
3.1.3 Video di Rete Ferroviaria Italiana	64
3.2 Esperimenti iniziali su immagini termiche	64
3.3 Data Augmentation	75
3.4 Esperimenti su video di RFI	80
3.5 IoU sul tempo	81
4 Conclusioni	85
Appendice	87

2 Indice

- A Codice di IoU over time 89
- B Implementazione di RandAugment e AutoAugment su Retina-Net 91
- Acronimi 95

ELENCO DELLE FIGURE

Figura 1	Schema di un neurone	10
Figura 2	Storia della Object Detection [1]	12
Figura 3	Evoluzione della detection multiscala [1]	13
Figura 4	Evoluzione della regressione basata su Bounding Box (BB) [1]	15
Figura 5	Evoluzione del context priming [1]	17
Figura 6	Evoluzione della Non Maximum Suppression [1]	18
Figura 7	Evoluzione di Hard Negative Mining [1]	19
Figura 8	Statistiche riassuntive di Microsoft - Common Object in COntext (MS-COCO) [2]	25
Figura 9	Esempio di estrazione feature HOG, a sinistra immagine originale, a destra dopo estrazione delle feature	28
Figura 10	In (a) la struttura di un detector a doppio stadio, in (b) la struttura di un detector a singolo stadio [3]	29
Figura 11	Struttura di Spatial Pyramid Pooling Networks (SPPNet) [4]	32
Figura 12	Struttura di You Only Look Once (YOLO) [5]	34
Figura 13	Architetture di SSD e DSSD. In blu i layer di SSD, in rosso quelli di DSSD.	37
Figura 14	Struttura di m2det [6]	37
Figura 15	Focal Loss al variare di γ	39
Figura 16	Struttura di RetinaNet	40
Figura 17	Pattern a scacchiera usato per la calibrazione di KAIST	45
Figura 18	Heatmap riguardante la posizione dei pedoni	47
Figura 19	Differenze tra apprendimento tradizionale e transfer learning	49
Figura 20	Transfer learning da COCO a KAIST	53
Figura 21	Esempio di predizioni, in verde la <i>ground truth</i> in rosso le predizioni.	54
Figura 22	Transfer learning da KAIST Visibile a KAIST Termico	55

4 Elenco delle figure

Figura 23	Schema di funzionamento di AutoAugment	57
Figura 24	Esempio di policy con 5 sub-policy, immagine tratta da [7]	57
Figura 25	Suddivisione del dataset KAIST MPD	62
Figura 26	MAP al variare del valore di soglia su FLIR	65
Figura 27	Esempio di predizioni, in verde la <i>ground truth</i> in rosso le predizioni.	66
Figura 28	Precision-Recall per classe person	66
Figura 29	Addestramento di RetinaNet su FLIR partendo dai pesi di COCO	67
Figura 30	Mean Average Precision (mAP) su FLIR addestrato dai pesi di COCO	68
Figura 31	Curva P/R di FLIR addestrato partendo da COCO (person)	69
Figura 32	Addestramento su FLIR partendo dai pesi di KAIST	70
Figura 33	Grafico P/R dell'addestramento su FLIR partendo dai pesi di KAIST (person)	71
Figura 34	Risultati dell'addestramento su FLIR partendo dai pesi di KAIST	71
Figura 35	Test su KAIST del risultato migliore su FLIR	72
Figura 36	Fine tuning di KAIST partendo da FLIR	74
Figura 37	P/R su KAIST all'epoca 8	75
Figura 38	Fine tuning di KAIST partendo da FLIR	76
Figura 39	Training su FLIR con policy di AutoAugment V2	79
Figura 40	mAP al variare degli iperparametri	81
Figura 41	Rilevazioni su video RFI con soglia 0.30	82
Figura 42	Rilevazioni su video RFI con soglia 0.80	82

LISTINGS

2.1	Algoritmo di RandAugment in Python [8]	59
3.1	Script di conversione per dataset FLIR	63
3.2	Policy Vo di AutoAugment	75
3.3	Policy V1 di AutoAugment	77
3.4	Policy V2 di AutoAugment	77
A.1	Algoritmo di IoU over time in Python	89
B.1	Funzioni per applicare AutoAugment	92
B.2	Funzione compute_input_output del generatore	92
B.3	Funzione per implementare RandAugment	94

ELENCO DELLE TABELLE

Tabella 1	Riassunto dei dataset analizzati, dove presente lo '/' indica la suddivisione tra train e test	20
Tabella 2	Schema riassuntivo delle categorie di Transfer Learning	52
Tabella 3	Test complessivo delle performance dopo l'addestramento	53
Tabella 4	Risultati della valutazione separata	54
Tabella 5	Test complessivo delle performance dopo l'addestramento	55
Tabella 6	Risultati della valutazione separata	56
Tabella 7	Suddivisione giorno notte di KAIST	62
Tabella 8	Risultati del primo esperimento su FLIR	65
Tabella 9	Risultati di FLIR da COCO con soglia 0.3	67
Tabella 10	Risultati migliori dell'addestramento su FLIR partendo dai pesi di KAIST, soglia 0.3	72

6 Elenco delle tabelle

Tabella 11	Baseline per la rilevazione di vetture su KAIST	73
Tabella 12	Risultati del fine tuning su kaist annotato manualmente	73
Tabella 13	Incremento rispetto alla baseline dopo fine tuning	74
Tabella 14	Risultati complessivi di AutoAugment su KAIST	78
Tabella 15	Incremento rispetto alla baseline dopo fine tuning	78
Tabella 16	Test di transfer learning da FLIR addestrato con AA V2 a KAIST (soglia 0.3)	79
Tabella 17	Test dopo ulteriore fine tuning su KAIST	79

"Perché Javascript

...

Javascript inoltre funziona molto bene con HTML e CSS, fondamentali per realizzare la parte front end di un'applicazione web."
— *Un'applicazione web browser per la visualizzazione e analisi di sequenze dell'enciclopedia OEIS, Marco Vignini*

INTRODUZIONE

Il *Machine Learning* detto anche Apprendimento Automatico nasce come diramazione dell'intelligenza artificiale e sta prendendo sempre più piede nel mondo odierno. In passato questa disciplina era limitata dall'hardware disponibile, ma ora grazie all'avvento di Graphics Processing Unit (GPU) sempre più potenti o addirittura hardware dedicato si riesce ad usare modelli sempre più complessi e precisi in diversi ambiti della vita quotidiana.

Il trend attuale prevede l'utilizzo sempre maggiore di Reti Neurali artificiali. Una Rete Neurale è un modello matematico che per molti versi cerca di ricalcare la funzionalità di un cervello umano al solo scopo di sostituirlo per quei compiti che possono essere definiti *ripetitivi*.

Il cervello umano è composto da neuroni connessi tra di loro che formano una rete, uno di questi componenti è possibile riassumerlo molto schematicamente in Figura 1. All'interno di un neurone scorrono impulsi elettrici che arrivano fino all'assone, e da quest'ultimo sono modulati dalle sinapsi ed infine arrivano ai dendriti degli altri neuroni connessi ad esso. Una rete neurale artificiale, come il cervello umano è composta da tanti neuroni artificiali che sono del tutto analoghi come schema ad un neurone biologico. Hanno un input dei dati che vengono modulati da dei *pesi* ed infine passando per una *funzione di attivazione* vengono restituiti in output ad altri neuroni.

Un esempio di utilizzo può essere la guida autonoma: sempre più case costruttrici adottano reti neurali per realizzare veicoli sempre più sicuri e per ridurre l'intervento umano in casi di emergenza o durante lunghi viaggi, riuscendo quindi a integrare hardware molto potente in situazioni critiche. Lo stato dell'arte in ambito consumer sulla guida autonoma è stato raggiunto dall'americana *Tesla* con il suo *AutoPilot* che grazie a nove telecamere posizionate attorno alla vettura riesce a raggiungere un livello di autonomia mai visto prima. Altri utilizzi possono essere la videosorveglianza, dove riconoscere intrusioni di esseri umani o vetture all'interno di un perimetro diventa un compito critico e molto importante che fino a pochi anni fa era prerogativa solamente di esseri umani. Le reti neurali sono usate anche per controllo qualità, *speech recognition*, *sentiment analysis* e via discorrendo.

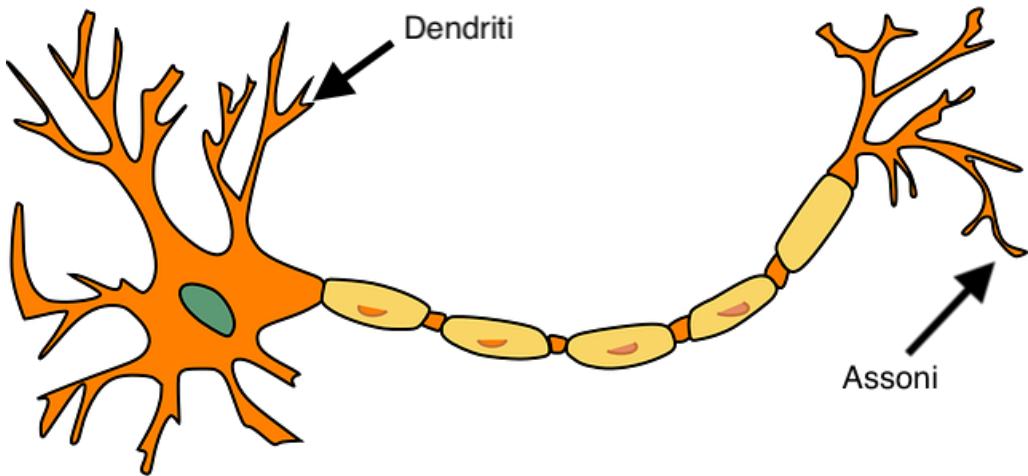


Figura 1.: Schema di un neurone

Il nostro interesse sarà focalizzato sul miglioramento del riconoscimento su immagini che fanno parte dello spettro termico in quanto offrono un vantaggio in condizioni di scarsa visibilità rispetto alle immagini a colori o in bianco e nero. Il resto dell'elaborato è strutturato come segue:

- nel Capitolo 1 è stata data un'introduzione all'argomento riguardante la rilevazione di oggetti, fornendo una panoramica delle metodologie, tecniche e modelli più importanti e di come si sono evolute nel corso del tempo.
- all'interno Capitolo 2 si parlerà di come è stato organizzato il sistema su cui abbiamo lavorato per lo sviluppo della tesi, in particolare si affronterà da un punto di vista più descrittivo i dataset e le tecniche utilizzate.
- il Capitolo 3 riguarda invece gli esperimenti effettuati usando come base ciò che è stato descritto nel precedente capitolo.

1

OBJECT DETECTION

L'Object Detection è un *task* legato al mondo della *computer vision* che consiste nel rilevare e classificare istanze di oggetti in immagini o video.

Negli ultimi anni, grazie soprattutto all'avvento delle GPU, c'è stato un incremento notevole del potere computazionale. Questo ha portato a sviluppare tecniche sempre più raffinate allo scopo di raggiungere prestazioni sempre migliori.

Lo sviluppo di hardware sempre più potente ha portato l'interesse verso modelli di *Deep Learning*. In questo capitolo cercheremo di contestualizzare, anche storicamente, tecniche, modelli ed evoluzioni nel campo della *Object Detection*. La letteratura sui detector è molto disomogenea e variegata, prenderemo quindi come riferimento i lavori di Licheng Jiao *et al.* [3] e Zhengxia Zou *et al.* [1].

1.1 STORIA DELLA OBJECT DETECTION

Una prima, ma importante distinzione va fatta tra il periodo pre e post *deep learning*. Il primo periodo va dagli inizi degli anni 2000 fino al 2014. Il secondo periodo, dove hanno preso il sopravvento tecniche basate sul *deep learning*, va dal 2014 fino ai nostri giorni. Quest'ultime tecniche possono essere a loro volta divise in altre due categorie, *One Stage Detector* e *Two Stage Detector* (sarà il caso di tradurre in italiano?) il cui sviluppo procede in maniera parallela. In Figura 2 è presente uno schema riassuntivo con tutte le pietre miliari raggiunte durante lo sviluppo di tecniche per il rilevamento di oggetti.

1.1.1 Evoluzione delle tecniche

Durante questo ventennio i detector più famosi sono stati costruiti usando come mattoncini delle tecniche sviluppate ed affinate via via nel tempo.

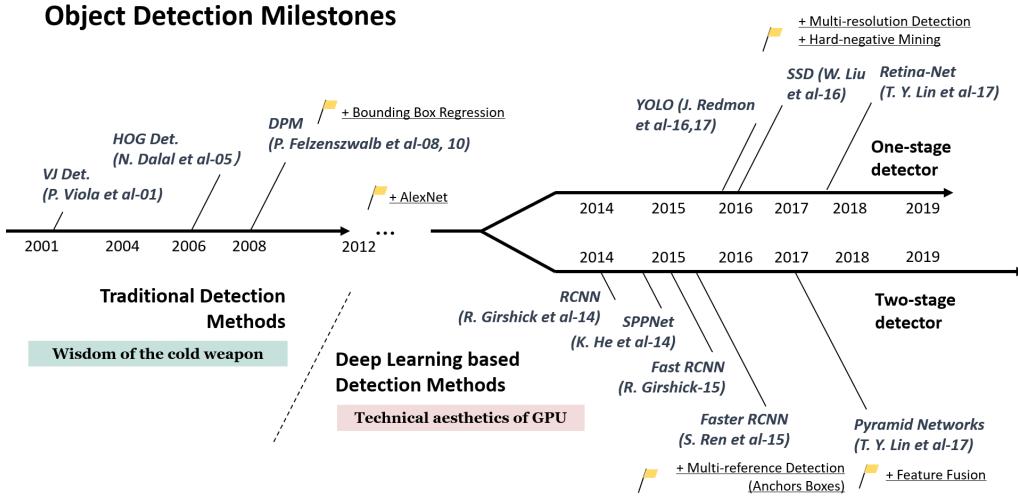


Figura 2.: Storia della Object Detection [1]

Queste tecniche sono di diverso tipo ed hanno subito evoluzioni di cui faremo una disamina nel proseguo di questa sottosezione.

Prime tecniche

Storicamente una delle prime tecniche si basava sul una teoria cognitiva chiamata *Recognition by Components* [9], ed è stata per molto tempo la base di alcuni lavori riguardanti il riconoscimento di immagini e la rilevazione di oggetti [10, 11, 12].

Nel passato alcuni ricercatori hanno formulato soluzioni al problema usando misure di similarità tra le componenti di un oggetto, tra la forma o i contorni, tra cui *Distance Transforms* [13], *Shape Contexts* [14] e *Edgelet* [15].

I risultati iniziali erano molto promettenti, tuttavia quando la rilevazione è diventata più complicata queste tecniche hanno iniziato a mostrare i propri limiti, motivo per cui il passaggio al Machine Learning è stato quasi naturale. Le prime metodologie basate su questo approccio risalgono ad un periodo inquadrabile prima del 1998, in questo caso la detection si basava su modelli statistici costruiti sopra le caratteristiche visibili che accomunano gli oggetti da rilevare. Il primo di questi modelli statistici, nato nel 1991 e chiamato *Eigenfaces* [16, 17], riesce in laboratorio a riconoscere volti in tempo reale.

Successivamente, fino al 2005, l'evoluzione ha portato a tecniche in cui si cambiava radicalmente la rappresentazione dell'immagine, detta anche insieme delle feature. Inizialmente lo scopo era apprendere come

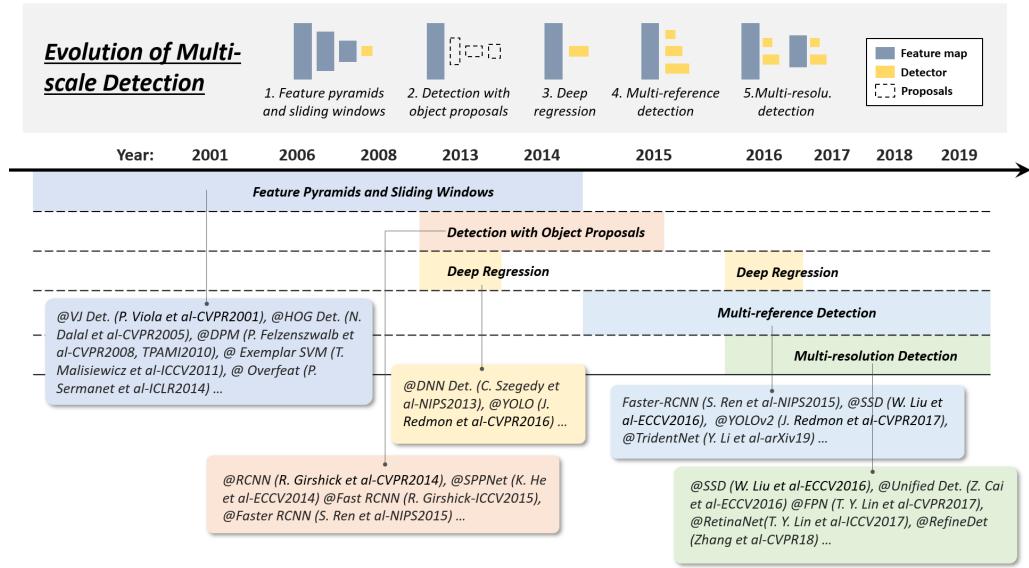


Figura 3.: Evoluzione della detection multiscala [1]

trasformare un'immagine da insieme di pixel a insieme di coefficienti *wavelet*. Grazie alla sua efficienza, tra tutte le trasformate, quella a prendere piede fu la *Haar wavelet*. Dal 2005 al 2012 c'è stato un passaggio a rappresentazioni basate sul gradiente.

Intorno al 1990 iniziano a fare capolino le prime Convolutional Neural Network (CNN) [18] le quali però non hanno avuto grandi applicazioni per via dell'elevato costo computazionale rispetto alle risorse disponibili ai tempi. I modelli realizzati con CNN non potevano essere quindi molto profondi, e per questo avevano forti limitazioni. Un lavoro che ai tempi tentava di ridurre il l'elevato costo computazionale è *space displacement network* [19]. L'idea era di estrarre una rappresentazione dell'immagine in un solo passaggio ed è stata realizzata estendendo i layer della CNN. Storicamente le CNN di cui si parla nel lavoro di LeCun *et al.* possono essere considerate un po' le antenate di quelle che attualmente chiamiamo Fully Convolutional Network (FCN) [20] [21]

Detection multiscala

Uno degli aspetti più interessanti della ricerca si basa sulla rilevazione di oggetti con diverse misure o diverse proporzioni. Come è possibile vedere in Figura 3 la soluzione a questo problema ha attraversato varie fasi.

FEATURE PIRAMIDALI E FINESTRE SCORREVOLI L'idea dietro questa tecnica è abbastanza basilare, infatti dopo aver estratto le feature da un'immagine quello che viene fatto è far scorrere una finestra rettangolare di dimensione generalmente fissa per effettuare il rilevamento e la classificazione di oggetti.

Dal 2004 al 2014 sono stati creati un sacco di detector basati su questa filosofia, il problema è che erano stati disegnati con l'intento specifico di rilevare oggetti con proporzioni fisse. Ricercatori come R. Girshick *et al.* iniziarono a cercare soluzioni a questo problema, arrivando a formulare un modello mistura [22] composto da più modelli addestrati su oggetti con differenti proporzioni. Sono state sviluppate anche altre soluzioni, basate questa volta sull'addestrare modelli separati per ogni istanza di oggetto dell'insieme di addestramento [23, 24]. Le limitazioni di tutte queste tecniche risiedono nel fatto che i dataset più moderni sono molto diversificati, quindi nel corso del tempo sono diventate sempre meno precise ed utilizzabili. Ciò ha portato allo sviluppo di *Object Proposal*.

OBJECT PROPOSAL Il primo avvistamento di *Object Proposal* risale al 2010 in un task di rilevazione di oggetti [25]. Possiamo definire una regione come un'area di un'immagine contenente pixel che hanno caratteristiche comuni tra di loro. L'idea dietro questa tecnica è creare regioni non etichettate con classi che potenzialmente possono contenere qualunque tipo di oggetto, e lo scopo è riuscire a rilevare oggetti di varie misure e scale pur non dovendo necessariamente svolgere una ricerca esaustiva con finestre scorrevoli.

Per ottenere queste regioni di pixel su un'immagine ci sono vari modi, discussi in parte da J. Hosang *et al.* in [26].

DEEP REGRESSION Questa tecnica, sviluppata dal 2013 al 2016 si basa sull'idea di predire direttamente le coordinate della BB contenente l'oggetto usando come feature quelle estratte da un modello di *deep learning* [5]. Il vantaggio fondamentale di questo approccio è l'efficienza e la velocità di implementazione, mentre uno svantaggio è la bassa accuratezza di localizzazione specialmente su piccoli oggetti.

MULTI REFERENCE DETECTION Questo approccio è il più usato per il rilevamento di oggetti con scale differenti e si basa sull'uso di un insieme di finestre rettangolari, che possono variare in dimensione e proporzioni, applicate sull'immagine, dette anche *Anchor Boxes* [27, 28, 29]. Sulla base di queste regioni rettangolari viene poi effettuata una predizione della

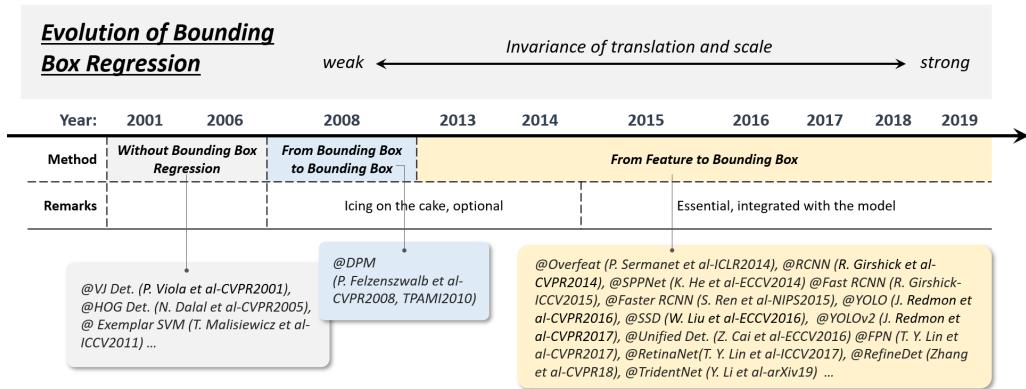


Figura 4.: Evoluzione della regressione basata su BB [1]

BB.

MULTI RESOLUTION DETECTION Negli ultimi anni un altro approccio che ha preso piede si basa sul rilevare oggetti con dimensioni differenti a layer differenti, sfruttando quindi la struttura di un modello a strati [28, 30, 31, 32]. Basti pensare alle CNN che nel corso della propagazione dell’immagine in input, grazie alla loro composizione, formano una rappresentazione piramidale dell’input. Diventa quindi più facile rilevare oggetti grandi nei layer più profondi e viceversa diventa più facile rilevare oggetti piccoli nei layer meno profondi.

Regression basata su Bounding Box

Questo insieme di metodologie ha lo scopo di affinare la posizione delle BB basandosi sulle rilevazioni effettuate tramite *Object Proposal* o *Anchor Boxes*, descritte precedentemente. Uno schema riassuntivo dell’evoluzione tecnica di questi approcci è in Figura 4.

I primi detector non raffinavano in alcun modo la posizione delle BB, anzi molte volte usavano direttamente l’output derivato da un algoritmo basato su finestre scorrevoli. L’unico modo per ottenere rilevazioni più precise era quindi costruire modelli piramidali molto densi e assicurarsi di far scorrere la finestra lungo tutta l’immagine.

DA BOUNDING BOX A BOUNDING BOX I primi ad usare una forma di regressione per aumentare la precisione sulle BB sono stati P. F. Felzenszwalb *et al.* in DPM [33] formulando la soluzione con il metodo dei minimi quadrati. Per scendere più nel dettaglio dobbiamo considerare un modello con feature piramidali. In breve nel modello proposto in [33]

L'implementazione è effettuata tramite una funzione $g(z)$ che associa ad un vettore di feature le coordinate (x_1, y_1) e (x_2, y_2) della BB. Dopo la fase di addestramento del modello viene usato l'output di $g(z)$ per un'ulteriore fase di addestramento nella quale tramite il metodo dei minimi quadrati si impara ad effettuare predizioni più corrette su x_1, y_1, x_2 e y_2 partendo da $g(z)$. Bisogna però specificare che questo tipo di ottimizzazione è stata implementato a livello di post-processing, quindi risulta del tutto opzionale.

DA FEATURE A BOUNDING BOX A differenza del tipo di ottimizzazione proposto in precedenza, con l'introduzione delle *Faster RCNN* [27] nel 2015 la regressione è implementata allo stesso livello nel quale viene effettuata anche la rilevazione stessa dell'oggetto. L'addestramento quindi non è più una fase separata ed opzionale, bensì diventa parte fondamentale e procede in parallelo con l'addestramento del modello. Inoltre, sempre confrontandolo con quanto detto prima, vengono usate anche diverse funzioni di loss da minimizzare che risultano più robuste rispetto a quella usata nei minimi quadrati. Degli esempi possono essere la *smooth-L1* o la *root-square*.

Valutazione del contesto

Generalmente gli oggetti che vengono rilevati dai sistemi di detection sono immersi in un contesto. Il cervello umano durante la fase cognitiva trae vantaggio dal riconoscere un contesto, motivo per cui le tecniche spiegate nel proseguito di questa sottosezione provano a emulare questa capacità degli umani. Una breve storia di come queste tecniche si sono evolute è in Figura 5.

CONTESTO LOCALE Per contesto locale si intendono tutte le informazioni visive che fanno parte dell'area più prossima ai contorni di un oggetto. Sin dagli anni 2000 Sinha e Torralba in [34] hanno provato che includere del contesto locale migliora le prestazioni ai fini del rilevamento di volti. Dalal e Triggs in [35] hanno dimostrato che introdurre una piccola porzione di sfondo migliora i risultati anche nel rilevamento dei pedoni.

CONTESTO GLOBALE Il contesto globale può essere considerato come una fonte di informazione aggiuntiva riguardante la scena in cui l'oggetto da rilevare è immerso. Un metodo usato all'inizio per inglobare nella

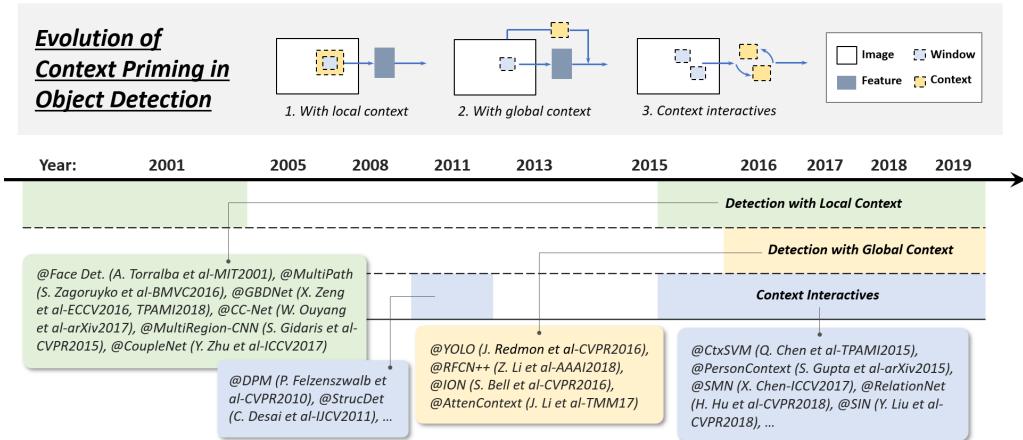


Figura 5.: Evoluzione del contesto priming [1]

detection informazioni sul contesto globale era realizzare delle statistiche che riassumevano la totalità degli elementi compresi nella scena [36]. In lavori più moderni catalogabili come modelli di *deep learning* sono state intraprese due strade, la prima è quella di inglobare il contesto tramite campi recettivi sempre più ampi, a volte anche più dell'immagine stessa [5] o usare l'operazione di *pooling* delle CNN [37]. La seconda via per ottenere informazioni dal contesto globale è pensare ad esso come un flusso di informazioni sequenziali ed usare Recurrent Neural Network (RNN) [38, 39].

CONTESTO DERIVATO DALLE INTERAZIONI L'ultima contestualizzazione che si può dare ad un oggetto riguarda le sue interazioni con ciò che lo circonda. Per interazioni si possono intendere tutte quei vincoli o dipendenze che può avere l'obbiettivo della rilevazione. Recentemente in alcuni lavori sono state analizzate questo tipo di informazioni contestuali ai fini del miglioramento della detection. Questi miglioramenti possono essere divisi in due macrocategorie, la prima di queste è quella in cui si esplorano le relazioni tra oggetti individuali [22, 40, 41, 42, 43]. Nella seconda categoria fanno parte i lavori nei quali vengono prese in considerazione le relazioni che ci sono tra gli oggetti e la scena che li circonda [44, 45].

Non Maximum Suppression

Per Non-Maximum Suppression (NMS) si fa riferimento a tutte quelle tecniche di post processing per ridurre il fenomeno delle BB duplicate.

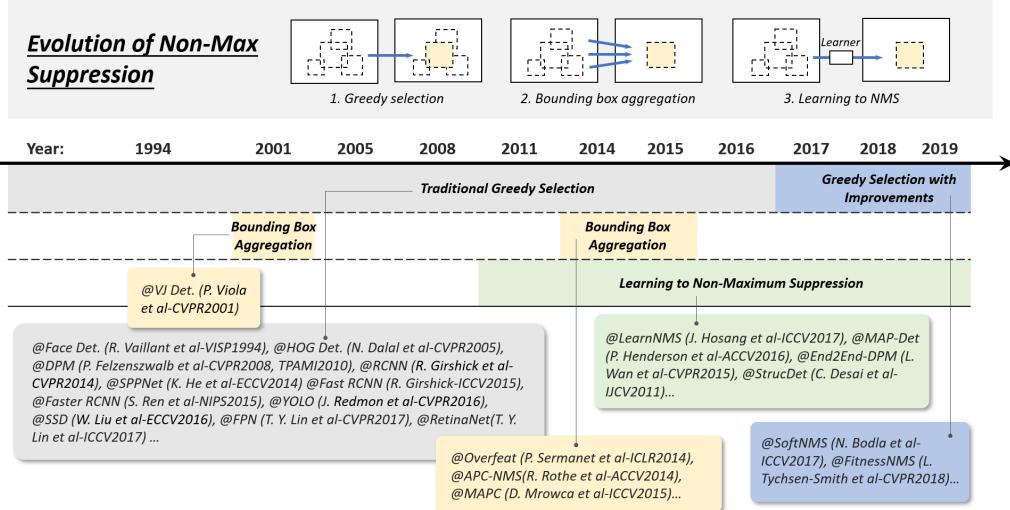


Figura 6.: Evoluzione della Non Maximum Suppression [1]

Questo fenomeno si concretizza quando, per la stessa rilevazione, ci sono più BB che circondano l'oggetto con confidenze molto simili tra di loro. L'evoluzione di queste tecniche nel corso del tempo è possibile vederla in Figura 6.

GREEDY La maniera più semplice per attuare la NMS è con un algoritmo di tipo *greedy*, infatti per un insieme di BB sovrapposte si considera solamente quella con la confidenza massima, mentre le altre vengono scartate. La sua semplicità, che da un certo punto di vista può anche essere vista come un punto di forza, può anche essere fonte di debolezze in quanto un algoritmo *greedy* non sempre porta all'ottimalità. Si possono infatti verificare casi in cui la BB con massima confidenza non ricopre tutto l'oggetto, o ancora peggio le BB di oggetti vicini tra di loro possono essere scartate erroneamente.

AGGREGAZIONE DI BOUNDING BOX L'aggregazione di BB vicine tra di loro è un altro approccio per attuare la NMS [46, 47, 48, 49]. L'aggregazione può essere fatta sia attraverso algoritmi di *clustering*, sia combinando le BB sovrapposte in un'unica singola detection.

IMPARARE AD APPLICARE NON-MAXIMUM SUPPRESSION Approcci più recenti per migliorare le già sopracitate tecniche di NMS riguardano l'apprendimento automatico [50, 40, 51, 52]. L'idea dietro questi metodi è trattare la NMS alla stessa stregua di un filtro che assegna nuovi valori di

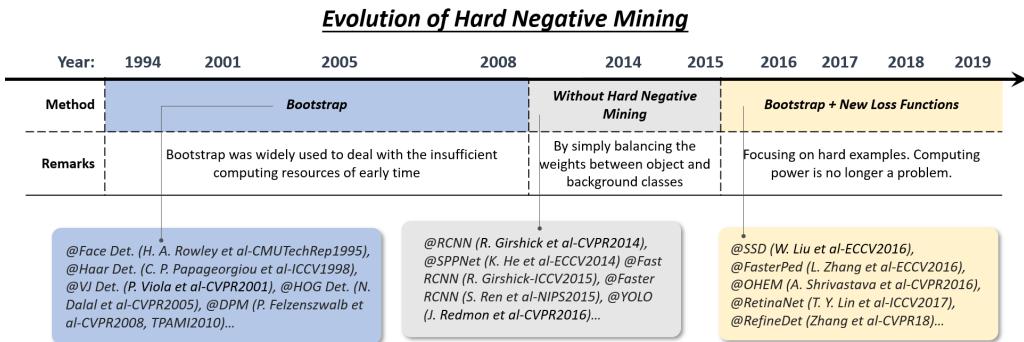


Figura 7.: Evoluzione di Hard Negative Mining [1]

confidenza a tutte le detection e quindi bisognoso anch'esso di una fase di addestramento.

Hard Negative Mining

Uno dei problemi a cui bisogna far fronte quando si tratta la *Object Detection* è lo sbilanciamento in cardinalità tra gli oggetti che vogliamo rilevare e conseguentemente classificare e tutto quello che non ci interessa. Visto che generalmente lo sfondo ricopre una buona parte dell'immagine la prima soluzione che potrebbe venire in mente per risolvere questo problema è addestrare il modello a riconoscere lo sfondo, ma questo porta a risultati pessimi durante l'addestramento in termini di efficienza. Tecniche di Hard Negative Mining (HNM) servono proprio a risolvere queste problematiche. Una breve storia è possibile vederla in Figura 7.

BOOTSTRAP Per Bootstrap si fa riferimento ad un gruppo di tecniche attraverso le quali si fa iniziare la fase di addestramento con piccoli esempi di sfondo. Successivamente quando si rilevano esempi di sfondo classificati erroneamente si aggiungono al processo di addestramento. Questo approccio risulta efficiente in quanto si evita di addestrare il modello su milioni di esempi di sfondi [46, 53, 54].

HNM IN DETECTOR BASATI SU DEEP LEARNING Recentemente modelli come Faster RCNN e YOLO per risolvere questo problema bilanciano i pesi tra finestre con esempi negativi e positivi. Nonostante tutto però non viene risolto completamente il problema di sbilanciamento, quindi c'è stato un ritorno al *bootstrap*. Un altro modo per risolvere lo sbilanciamento è l'introduzione di nuove funzioni di Loss, come ad esempio la

Dataset	Immagini	Totale BB	Classi	Risoluzione	Fonte	Altre informazioni
Mit ped	500/200	ND	1	64 × 128	ND	È stato il primo
INRIA	1805	ND	1	64 × 128	ND	Forma rudimentale di Data Augmentation
Pascal VOC (2012)	11540/10991	27450 in train	20	Variabile	Web	Challenge
Caltech	124K/124K	350K totali	3	640 × 480	Vettura in movimento	Ambiente cittadino
KITTI	194/195	160K totali	6	1382 × 512	Vettura in movimento	3D, moltissime vetture
ILSVRC	458K/46K	ND	200	Variabile	Web	Challenge
CityPersons	5K	35K	4	ND	Altro dataset	BB con stesse proporzioni, 50 città diverse
MS-COCO	123K/40K	896K in train	91	Variabile	Web	Ottima contestualizzazione
OPEN IMAGES (2018)	1,784M/125K	14M in train	600	Variabile	Web	Molte classi, molto ampio, alta media di istanze per immagine
EuroCity	28K/19K	250K in totale	9	1920 × 1024	Vettura in movimento	Varietà nell'ambiente, qualità delle immagini

Tabella 1.: Riassunto dei dataset analizzati, dove presente lo '/' indica la suddivisione tra train e test

Focal Loss in RetinaNet [55], descritta con maggior dettaglio in Sezione 1.4.1.

1.1.2 Dataset

L’insieme dei dati con cui addestrare e testare le performance dei modelli via via sviluppati nel corso del tempo ha subito un’evoluzione. Costruire dataset sempre più grandi e con meno bias è sempre stato un obiettivo principale che si ponevano i ricercatori, tutto ciò per realizzare dei benchmark che mettessero sempre di più a dura prova i nuovi modelli.

Nel seguito di questa sezione analizzeremo in breve alcuni dei dataset più famosi nell’ambito della computer vision, con particolare attenzione per alcuni incentrati sulla rilevazione di pedoni. In Tabella 1 è presente uno schema riassuntivo dei dataset presi in analisi.

MIT PED. Risale all’inizio del nuovo millennio ed è uno dei primi dataset che ha come scopo il riconoscimento di pedoni. Rispetto agli standard odierni risulta molto piccolo in quanto contiene circa 509 immagini usabili per l’addestramento e 200 usabili per la fase di test. [56]

INRIA Risalente al 2005, nasce dall’esigenza di creare un dataset dove la detection diventasse più complicata rispetto a quello fatto dal MIT [35]. Contiene 1805 immagini ad una risoluzione di 64×128 pixel di esseri umani. Le immagini usate per l’insieme di training sono 2478, ovvero 1239 immagini, contenenti esempi positivi, prese dal totale più le stesse immagini ma specchiate secondo l’asse delle y.

PASCAL VOC Pascal Visual Object Classes (VOC) è collocabile in un periodo che va dal 2005 al 2012 [57, 58]. VOC consiste di due parti complementari, la prima è un dataset pubblico e disponibile per esperimenti e benchmark, la seconda è una sfida annuale. Nel corso degli anni ne sono state sviluppate diverse versioni, individuabili dal pattern VOCANNO. I task effettuabili su questo dataset spaziano classificazione di immagini alla object detection passando anche per il rilevamento di azioni.

Per la sfida nel 2007 sono state raccolte solo immagini dal social network Flickr. Le immagini sono molto variegate, ad esempio come scritto nell'articolo di presentazione del dataset [57] ci possono essere motociclette in diverse pose e forme, come può essere il veicolo per strada o come soggetto principale del fotogramma. Un altro esempio è la classe "*person*", mentre in molti dataset con questa classe ci si riferisce ad un pedone in VOC2007 con questa classe si fa riferimento ad un essere umano in diversi contesti.

Per realizzare il dataset sono state definite delle keyword con cui effettuare query su Flickr. Queste suddette keyword sono state definite sulla base delle classi degli oggetti che si desiderava annotare. Tramite queste query sono state recuperate 500.000 immagini, non prendendo in considerazione la data di acquisizione, il nome del fotografo, la location e via discorrendo. Le query venivano effettuate a gruppi di 100.000 immagini alla volta, di cui venivano selezionate casualmente solamente fotografie che venivano effettivamente inserite nel dataset. Questa operazione è stata ripetuta fino ad ottenere la quantità desiderata di file. L'operazione successiva è stata quella di eliminare i duplicati o comunque immagini molto somiglianti tra di loro, una volta fatto questo sono passati ad annotarle. Di queste 500.000 immagini agli annotatori ne sono state presentate 44.269. Gli annotatori avevano la facoltà di scartare alcune immagini se le ritenevano non adatte ad essere annotate o avevano una confidenza bassa sull'eventuale annotazione da effettuare.

Nonostante ciò è stato scoperto un'elemento che portava a del bias all'interno del dataset. L'elemento in questione riguardava il modo in cui le immagini sono state recuperate. Quando si effettua una query su Flickr il server restituisce le immagini in ordine cronologico di upload sulla piattaforma. Il dataset è stato realizzato nel Gennaio 2007, quindi buona parte delle immagini erano ambientate in un contesto natalizio o perlopiù invernale. Con VOC2008 il problema è stato risolto aggiungendo una data casuale all'interno della query per recuperare le immagini.

CALTECH PEDESTRIAN DATASET Il *Caltech Pedestrian Dataset* [59] nasce nel 2009 ed è molto più ampio di tutti gli altri dataset visti in precedenza. Le immagini sono state ricavate da circa 10 ore di video girato a 30 frame al secondo in un ambiente urbano con traffico regolare. È quindi presente un numero di frame che è nell'ordine di grandezza di 10^6 . La telecamera con cui sono state acquisite le registrazioni è stata piazzata su un'autovettura con un guidatore che attraversava le strade di Los Angeles guidando in maniera normale. La risoluzione delle immagini è di 640×480 e come conseguenza del fatto che il sistema è stato più volte smontato e rimontato ci sono piccole variazioni nella posizione della telecamera. Il dataset è stato creato da 11 sessioni dove in ognuna di queste venivano esplorati 5 quartieri. Dopodiché le prime sei sessioni sono state destinate all'addestramento, mentre le rimanenti cinque sono state adibite a test set.

Sono stati annotati 250.000 fotogrammi, per un totale di 350.000 BB e 2300 pedoni univoci. Di tutti i fotogrammi, circa il 50% non hanno pedoni, mentre il 30% del totale ne ha almeno due. In media un pedone è visibile per un tempo di 5 secondi. Come su dataset più recenti i pedoni sono raggruppati secondo la loro distanza dal guidatore. In particolare abbiamo che i pedoni sono vicini se la loro altezza è maggiore di 80 pixels, sono ad una distanza media se la loro altezza è compresa tra 30 ed 80 pixels, mentre sono considerati lontani se la loro altezza è al più 30 pixels. Questa discretizzazione riguardante la distanza dei pedoni è stata realizzata seguendo la distribuzione delle altezze degli stessi all'interno del dataset. Sono state inoltre prese in considerazione statistiche riguardanti l'occlusione dei pedoni e la loro posizione rispetto alla telecamera.

KITTI [60] Questo dataset è realizzato a partire da hardware usato per la guida autonoma. La particolarità è che offre informazioni tridimensionali dell'ambiente grazie a dei sensori laser dedicati che mappano il territorio circostante. Per la realizzazione sono state utilizzate due telecamere a colori con una risoluzione di 1392×512 pixels, uno scanner laser ed un localizzatore GPS con un'unità di correzione RTK, il tutto orchestrato da un calcolatore su cui girava un database in real time. Tutto questo hardware è stato montato su una station wagon che è stata guidata per dei normali scenari cittadini. Inoltre le annotazioni non sono BB in due dimensioni, ma dei parallelepipedi che avvolgono gli oggetti. Per la loro realizzazione sono stati presi annotatori umani che hanno posizionato BB tridimensionali su oggetti come auto, furgoni, camion, tram, pedoni e

ciclisti. Inoltre gli annotatori sono stati istruiti per marcare ogni BB come visibile, semi occlusa, occlusa o troncata.

ILSVRC Come per VOC anche ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [61] è una challenge organizzata annualmente dal 2010 al 2017. Inoltre, sempre come per VOC è presente un dataset pubblico disponibile per addestramenti e test. Come per alcuni dei dataset precedentemente introdotti le immagini state prese in parte da Flickr, con l'aggiunta però di altri motori di ricerca. In particolare andremo ad analizzare la costruzione della parte di dataset riguardante la *object detection*, però il dataset contiene anche una parte dedicata alla classificazione di immagini ed un'altra parte dedicata alla rilevazione singola di oggetti all'interno dell'immagine.

Le classi di oggetti presenti sono 200 e sono stati scelte partendo dalle 1000 classi usate per la classificazione di immagini. Una prima riduzione per arrivare a 494 classi è stata inizialmente effettuata tramite l'eliminazione di label corrispondenti ad oggetti che occupavano gran parte del fotogramma o semplicemente non adatti alla *object detection*. Un ulteriore operazione di unificazione tra classi simili ha portato il numero a 200.

Per il training set le immagini hanno tre provenienze differenti. La prima sorgente di immagini è la parte di dataset dedicata alla rilevazione singola di oggetti. Per aggiungere ulteriori esempi negativi sono state prese ulteriori immagini da altri motori di ricerca. Le immagini provenienti da queste due prime fonti sono state annotate solamente con un sottinsieme delle classi totali. L'ultima sorgente di immagini è la piattaforma Flickr su cui sono state fatte query generiche. Riguardo invece la porzione di dataset dedicata alla validazione e al test la situazione è molto simile in quanto la fonte primaria (77%) è la porzione di ILSVRC2012 adibita alla rilevazione di oggetti singoli, a cui però sono state sottratte le immagini nel quale gli oggetti occupavano più del 50% dell'area totale del fotogramma. Per aggiungere ulteriori esempi, come per il training set, sono state effettuate interrogazioni generiche su Flickr. A differenza dell'insieme di immagini di addestramento, per la validazione e il test, sono state usate tutte e 200 le classi disponibili. In totale per il training set sono presenti circa 458.000 immagini, mentre per il validation e test set sono presenti 46.000 fotogrammi.

CITYPERSONS Zhang *et al.* [62] propongono un nuovo dataset derivante da Cityscapes [63], ma invece che focalizzarsi sulla segmentazione

delle scene in contesti urbani CityPersons è incentrato sulla rilevazione di pedoni. Infatti per ogni frame di Cityscapes sono stati annotati esseri umani tramite BB.

Le classi usate in questo dataset sono quattro, e variano a seconda della postura del pedone. Con `pedestrian` vengono indicate le persone in piedi, che corrono o camminano, mentre con `rider` si indicano quelle persone che sono alla guida di un mezzo a due ruote. Sono presenti altre due classi per indicare le persone sedute (`sitting person`) e persone con pose inusuali (`other person`).

Viene standardizzato il processo di annotazione dei `pedestrian` e `riders` in quanto le BB hanno tutte la stessa proporzione (0.41), è quindi sufficiente tracciare una linea che va dalla testa ai piedi del pedone per generare una BB delle giuste dimensioni. In questo modo si perfeziona l'allineamento della regione con l'oggetto da rilevare, e quindi si migliorano le prestazioni dell'eventuale modello. Per le rimanenti due classi invece non è stato standardizzato il processo, quindi vengono semplicemente disegnate BB che contengono la persona. Un'altra operazione che è stata effettuata è stata ricercare tra le immagini tutte quelle persone `false` (manichini, statue, riflessi) per marcarli come regioni da ignorare.

Il dataset è composto da 5000 immagini, con un totale di circa 35000 annotazioni e 13000 regioni da ignorare. A differenza di KITTI e Caltech c'è una densità di persone per frame sette volte superiore ed il numero di individui distinti è prossimo a 20.000, quindi rispettivamente 15 e 3 volte superiore. La suddivisione tra test e train set è la stessa di Cityscapes.

ms-coco [2] Attualmente MS-COCO grazie alla difficoltà di ottenere buone prestazioni è considerato uno dei più interessanti dataset per la *object detection*. Rispetto a ILSVRC ha meno classi, ma contiene molte più immagini ed annotazioni. Il punto di forza di questo dataset è la grande molteplicità di contesti in cui sono state catturate le immagini e soprattutto la densità di oggetti che rispecchia in maniera del tutto simile quella del mondo reale. Inoltre una proprietà importante è che gli oggetti si trovano quasi sempre in contesti appropriati.

Andando più nello specifico ci si trova davanti ad un dataset che nella sua ultima versione ha, per la *object detection*, circa 165.000 immagini dedicate all'addestramento, e circa 160.000 dedicate alla validazione ed al test. Le classi in totale sono 91. In media ci sono 3.5 categorie diverse e 7.7 oggetti per immagine. In Figura 8 è presente uno schema riassuntivo delle statistiche di questo dataset, ed un'altra cosa che si può notare è che, rispetto ad esempio a ILSVRC e VOC, ci sono molte meno immagini

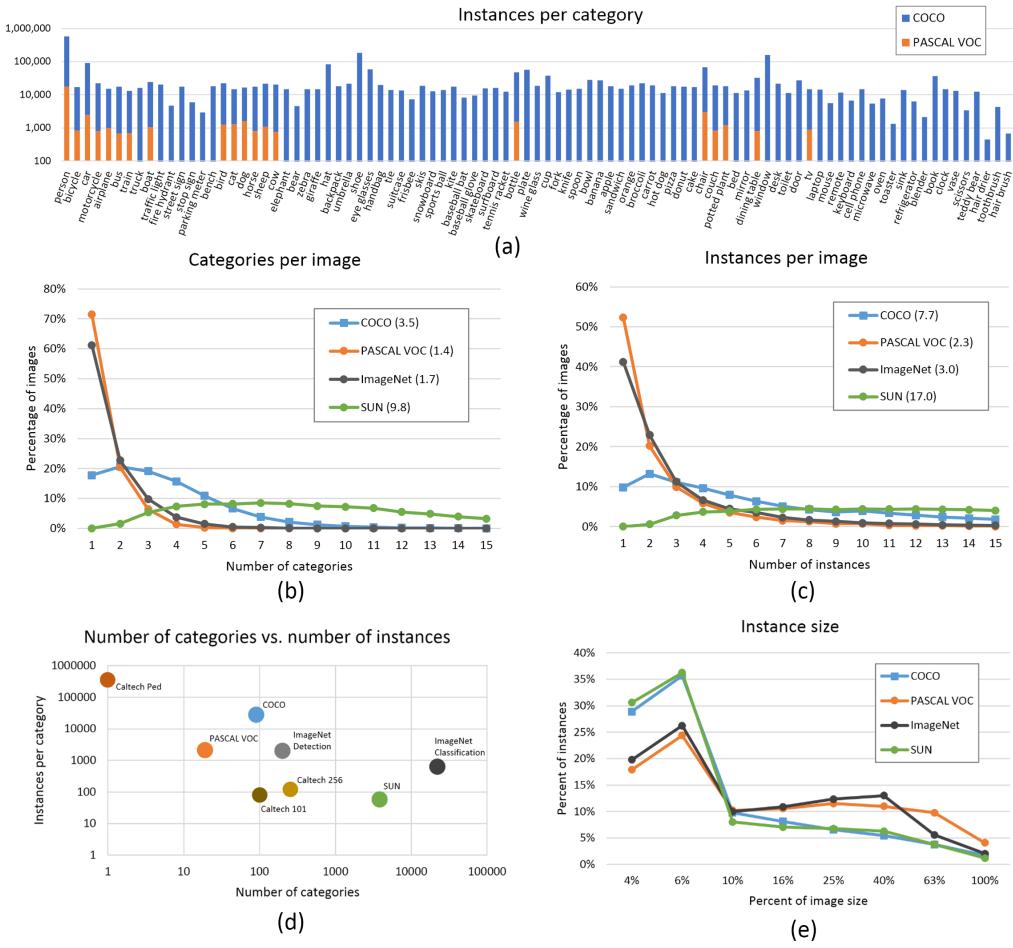


Figura 8.: Statistiche riassuntive di MS-COCO [2]

con una sola BB, infatti in MS-COCO solamente il 10% rientra in questa categoria.

OPEN IMAGES Open Images Detection (OID) [64], nella versione 5, è un dataset composto da 9.2 milioni di immagini. Ai giorni d'oggi rappresenta una delle sfide più interessanti insieme a ILSVRC e MS-COCO. Come per gli altri dataset anche questo si adatta a più task poiché le annotazioni sono state effettuate a livello di singola immagine, di *object detection*, segmentazione e interazioni tra oggetti. Analizzando l'aspetto riguardante la rilevazione di oggetti ha ancora più classi rispetto a ILSVRC in quanto si arriva a 600 differenti label su 1.9 milioni di immagini con BB realizzate per la maggior parte da annotatori professionali interni a Google. In aggiunta i contesti e le ambientazioni in cui sono state catturate le immagini sono molto variegati.

EUROCITY EuroCity è un dataset proposto da Braun *et al.* nel 2019 [65]. Le immagini sono state acquisite da un veicolo in movimento in 31 diverse città europee appartenenti a 12 differenti stati. L'arco temporale attraversa le quattro stagioni, ed anche le condizioni metereologiche, fatta eccezione per forte pioggia e tempeste di neve, sono state tutte acquisite.

La risoluzione è di 1920×1024 con un framerate di 20 immagini al secondo. Una particolarità delle immagini acquisite da queste telecamere è lo spazio colore che arriva a 16 bit. Quest'alta gamma dinamica si traduce in una più alta qualità dei fotogrammi in condizioni di luce non ottimale, come può essere un controsole o una notturna.

Per ogni città in media sono state catturate 1.7 ore di video e le immagini sono state campionate ogni 4 secondi (80 frame). Questo porta ad avere una ripetizione minore di pedoni o oggetti uguali, soprattutto in condizioni trafficate. Le classi in questo dataset si differenziano in base al veicolo su cui si trovano i pedoni. Sono stati annotati gli esseri umani a piedi, sulle moto, scooter, tricicli, sedie a rotelle e buggy. L'occlusione è stata gestita decidendo di annotare l'oggetto nella sua totalità dando una misura spaziometrica per le dimensioni della BB. Riguardo i veicoli invece sono state fatte BB separate per il veicolo e la persona al di sopra. C'è anche una parte di persone scartate che sono quelle che hanno un'altezza inferiore a 20 pixels.

1.2 DETECTOR BASATI SU METODI TRADIZIONALI

In questa sezione verranno analizzati in breve i detector più famosi che hanno segnato la storia della *Object Detection* prima dell'avvento del *Deep Learning*.

VIOLA JONES DETECTORS Nel 2001 P. Viola e M. Jones [46, 66] sono riusciti a realizzare un modello, chiamato Viola Jones (VJ) Detector, capace di riconoscere volti umani in condizioni non vincolate. L'hardware che fu usato era al passo con i tempi, si parla infatti di un processore Intel Pentium III, ed i risultati erano impressionanti. Molti altri algoritmi di rilevazione di volti infatti giravano decine, se non centinaia, di volte più lenti rispetto ad VJ e l'accuratezza era del tutto paragonabile.

L'approccio usato dai due ricercatori in fase di sviluppo è anche uno dei più basilari, ovvero la finestra scorrevole. Nonostante il compito fosse molto al di là della portata dell'hardware dei tempi questo detector tramite alcune tecniche è riuscito a migliorare drasticamente la sua efficienza in termini di velocità e accuratezza. La prima di queste tecniche è un

algoritmo chiamato *integral image* che serve a computare efficientemente la somma di valori in un sottoinsieme rettangolare di una griglia. In questo modo si è riusciti a velocizzare le operazioni di filtraggio e convoluzione. Il secondo mattoncino con cui è stato costruito VJ riguarda la scelta delle feature delle immagini. Le feature venivano estratte grazie alla *Wavelet Haar* ed invece che selezionare manualmente un sottoinsieme di filtri è stato usato Adaboost [67]. Infine la rilevazione veniva effettuata a cascata, in questo modo si riusciva a ridurre lo spreco di risorse evitando di passare più tempo del dovuto su porzioni di immagine dove l'algoritmo aveva una ragionevole certezza fossero sfondo.

HISTOGRAM OF ORIENTED GRADIENTS Histogram of Oriented Gradients (HOG) è un descrittore di feature per immagini presentato nel 2005 da N. Dalal e B. Triggs [35]. Il suo scopo era quindi l'estrazione di informazioni utili da un'immagine, cercando di scartare il più possibile informazioni inutili.

La motivazione per la realizzazione è stata la rilevazione di pedoni, tuttavia è facilmente adattabile alla rilevazione di altre classi. L'idea dietro queste feature si basa sul gradiente secondo una direzione. Possiamo spiegarlo intuitivamente come la variazione di colore che si ha andando in una determinata direzione di una regione rettangolare dell'immagine. L'estrazione procede dividendo l'immagine in celle non disgiunte e per ognuna di esse osservando i pixel che la compongono si calcola un istogramma monodimensionale dei gradienti secondo le varie direzioni. Prendendo tutti gli istogrammi derivanti dalle celle si ottiene una rappresentazione sottoforma di feature HOG dell'immagine. In Figura 9 è possibile vedere un'esempio di estrazione di questo tipo di feature.

DEFORMABLE PART-BASED MODEL La realizzazione di Deformable Part-based Model (DPM) parte da P. Felzenszwalb nel 2005 come estensione di HOG [10]. Questo detector è di tipo "*Divide et Impera*" in quanto la fase di addestramento può semplicemente essere considerata come l'apprendere la decomposizione di un oggetto in più parti, mentre la fase di inferenza può essere vista intuitivamente come il rilevamento delle parti di un oggetto. Successivamente R. Girshick ha esteso questo modello, detto *star-model*, ad un modello mistura [68, 22, 69, 70]. In questo modo si è riusciti ad applicare DPM a casi reali. Per molti anni è stato considerato un punto di riferimento in quanto vincitore della sfida VOC dal 2007 al 2009.

DPM è formato da un filtro posizionato alla radice, più tanti altri fil-

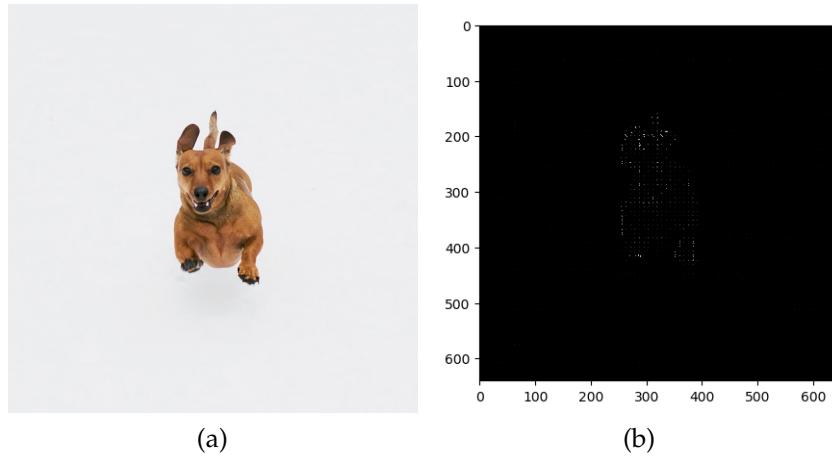


Figura 9.: Esempio di estrazione feature HOG, a sinistra immagine originale, a destra dopo estrazione delle feature

tri sottostanti adibiti al riconoscimento delle varie parti che formano un oggetto. Nel modello mistura di Girshick, invece che specificare manualmente cosa dovessero rilevare questi sottofiltri, venivano implementati come variabili latenti e come tali avevano bisogno di una fase di apprendimento supervisionato.

1.3 DETECTOR BASATI SU DEEP LEARNING

Come è possibile vedere da Figura 2 gli anni tra il 2012 ed il 2014 hanno segnato un punto di svolta nella *object detection* grazie a nuovi modelli basati su tecniche di apprendimento profondo. L'era attuale in cui il deep learning ha preso il sopravvento vede una diramazione nelle tecniche sviluppate per la rilevazione degli oggetti, nonostante ciò i due rami sono complementari e sviluppati in parallelo in quanto oggetto di studi recenti. La prima diramazione è quella dei detector *One Stage*, questi detector ottengono prestazioni in inferenza molto elevate, a discapito però di accuratezza e precisione nella localizzazione più scarse rispetto ai detector *Two Stage*, che però sono più lenti in fase di inferenza.

1.3.1 Estrattori di feature

Nell'era dell'apprendimento profondo anche gli estrattori di feature si sono evoluti. Vengono chiamati in gergo *Backbone Network* e sono reti neurali che prendono in input un'immagine e ne restituiscono le feature

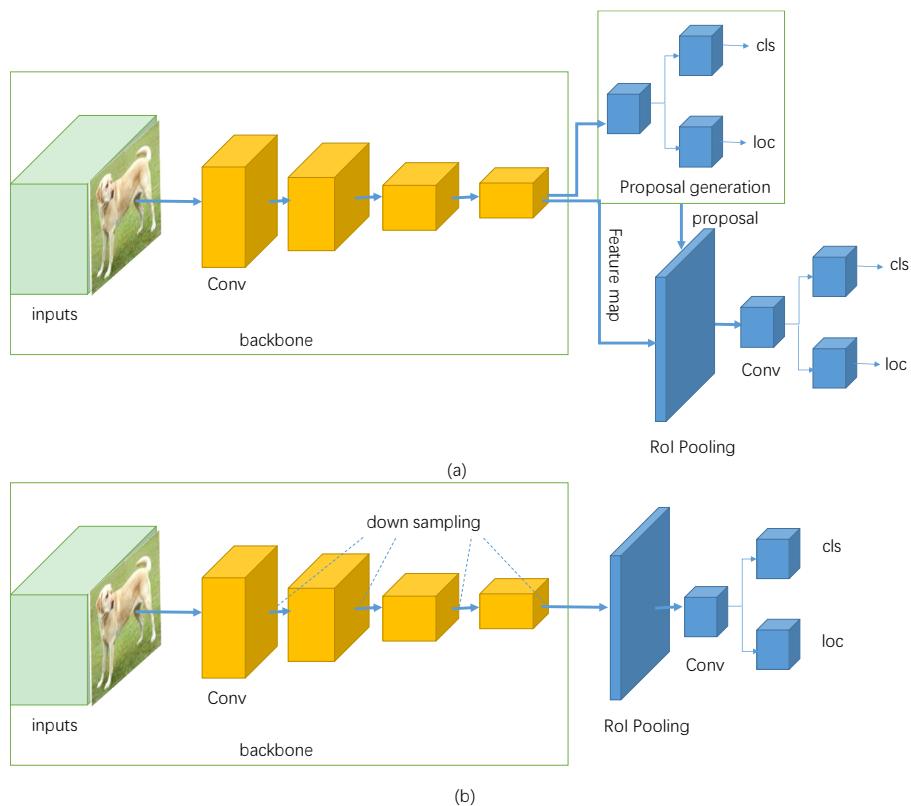


Figura 10.: In (a) la struttura di un detector a doppio stadio, in (b) la struttura di un detector a singolo stadio [3]

che poi andranno in input al modello che si occuperà della rilevazione di oggetti.

Molte delle reti di backbone usate attualmente sono reti create per la classificazione di immagine, ma senza l'ultimo strato usato come classificatore. Come per molti compiti anche per le estrazioni delle feature possiamo effettuare delle scelte che ci portano ad avere più accuratezza, ma minore velocità di esecuzione o viceversa. Reti come ResNet[66], ResNeXt [4], AmoebaNet [71] sono profonde e hanno un elevato grado di connessione tra i vari neuroni, risultano quindi generalmente più accurate, ma conseguentemente avere una struttura così densa porta ad avere effetti negativi sulle prestazioni. Molte volte però si può sacrificare un po' di accuratezza per migliorare la velocità, un ambito ad esempio è quello dei dispostivi portatili come possono essere gli smartphone. Reti come MobileNet [27], ShuffleNet [5], SqueezeNet [28] o Xception [30] sono dei backbone leggeri che possono essere usati per questo scopo.

1.3.2 Two Stage Detector

Uno schema basico dell'architettura di un rilevatore a doppio stadio è mostrata in Figura 10 (a). I cubi gialli sono strati convoluzionali, chiamati blocchi, presenti nella rete di backbone. Essendo nella rete di backbone da questi blocchi si otterrà una mappa delle feature, che verrà passata al Region of Interest (ROI) pooling layer per ottenere le feature definitive. Le proprietà estratte a quest'ultimo passaggio, nel caso dei detector a doppio stadio, passano generalmente per una Region Proposal Network (RPN) che propone delle regioni in cui potenzialmente potrebbe essere contenuto un candidato alla rilevazione. L'output della RPN a questo punto andrà come input in dei layer convoluzionali aggiuntivi (i cubi blu di Figura 10) per la classificazione. Vedremo qui di seguito alcuni modelli a doppio stadio.

R-CNN Girshick nel 2014 propone il primo detector a doppio stadio chiamato R-CNN [72]. Il modello è composto da quattro moduli. Il primo modulo genera le cosiddette region proposal, ovvero delle regioni che potenzialmente possono contenere oggetti. Il secondo modulo, partendo da queste regioni genera dei vettori di feature di dimensione fissata (4096 elementi) usando cinque strati convoluzionali e due completamente connessi. Un problema con cui si sono scontrati gli autori in questa fase è stata che l'input di una CNN è di dimensione fissata, mentre gli oggetti possono avere dimensioni e proporzioni differenti. È stato quindi

fissato l'input della CNN come una regione di dimensione 227×227 e per far combaciare le regioni proposte del primo modulo sono state applicate trasformazioni. La parte di classificazione è invece appannaggio del terzo modulo, ovvero un insieme di Support Vector Machines (SVM). Le BB vengono infine generate tramite regressione dal quarto ed ultimo modulo. Tra le CNN i parametri sono condivisi, mentre le SVM usate per la classificazione sono totalmente indipendenti l'una dall'altra.

La fase di addestramento di R-CNN si fa su ogni singolo componente in maniera separata. Come prima cosa viene realizzata una fase di pre-addestramento, seguita da una fase di addestramento fine. Dopodiché si vanno ad addestrare in maniera separata i classificatori SVM e gli strati di regressione per generare le BB. Riguardo la classificazione un aspetto interessante di cui tenere conto è l'applicazione della Intersection over Union (IoU) o indice di Jaccard, ovvero un valore che indica la sovrapposizione tra due regioni. Varia tra 0 e 1, con 0 quando non c'è alcuna sovrapposizione ed 1 quando la sovrapposizione è totale. La IoU si calcola prendendo in considerazione la BB *vera* e la BB predetta in fase di inferenza. In particolare è definita come il rapporto tra l'area di intersezione delle due regioni e l'unione delle due aree.

SPATIAL PYRAMID POOLING NETWORKS In R-CNN uno dei problemi era nel secondo modulo quando le CNN richiedevano un input di dimensione fissata. Con SPPNet [4] si risolve il problema introducendo un layer di pooling piramidale che permette alle CNN di lavorare anche con input di dimensione differente, senza necessità di applicare trasformazioni alla regione. In Figura 11 possiamo vedere come è stato realizzato questo nuovo layer di pooling piramidale.

Con l'uso di questo nuovo strato le feature possono essere estratte solamente una volta e dall'intera immagine, il che porta SPPNet ad essere circa 20 volte più veloce di R-CNN, senza perdere in accuratezza. Nonostante ciò sono presenti lo stesso alcune problematiche infatti come per R-CNN la fase di addestramento è ancora realizzata in più passaggi.

FAST R-CNN Fast R-CNN, proposta sempre da R. Girshick, [71] è una versione migliorata di R-CNN. Uno dei grandi difetti di R-CNN era la velocità in quanto per ogni regione candidata era necessario un passaggio sulle CNN per estrarre le feature. Con Fast R-CNN le feature vengono estratte in un unico momento e dall'intera immagine e successivamente passate ad un ROI pooling layer che restituisce in uscita vettori di dimensione fissata da passare agli strati di classificazione e regressione. A

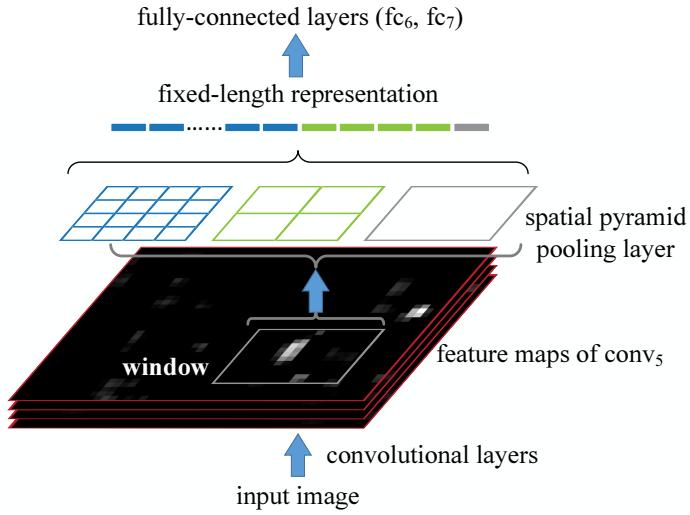


Figura 11.: Struttura di SPPNet [4]

differenza di R-CNN e SPPNet l’addestramento è composto da una sola fase, e questo è reso possibile da una misura di loss comune a tutta la struttura. Un altro miglioramento rispetto a R-CNN, e che accomuna questo detector a SPPNet, è che vengono preservate le informazioni spaziali delle regioni candidate in quanto non c’è più necessità di avere input di dimensione fissa per le CNN.

Tramite queste migliorie si ottengono vantaggi notevoli rispetto a R-CNN, sia in fase di training che di inferenza. In addestramento si ha uno speedup di circa 9 volte, mentre si ha un incremento di prestazioni in fase di inferenza che arriva fino a 223 volte.

FASTER R-CNN Tre mesi dopo la presentazione di Fast R-CNN venne presentata un’ulteriore versione migliorata chiamata Faster R-CNN [27]. Il collo di bottiglia di Fast R-CNN è la generazione delle regioni candidate a contenere oggetti. Con Faster R-CNN viene risolto con l’utilizzo di una RPN, ovvero una rete convoluzionale in grado di generare efficientemente regioni di diverse dimensioni e proporzioni. Viene quindi scartata la parte iniziale di Fast R-CNN e sostituita con questa nuova RPN che in poche parole dice a Fast R-CNN *dove guardare*.

L’input di una RPN è un’intera immagine e l’output sono regioni rettangolari con un indice di appartenza ad una determinata classe. In più in Faster R-CNN, grazie all’implementazione di una RPN, vengono utilizzate anche le *Anchor Boxes* che semplificano la rilevazione di oggetti con dimensioni differenti.

FPN Nel 2017 Lin *et al.* hanno proposto Feature Pyramid Network (FPN) [30] sulla base di Faster R-CNN. Prima di FPN l’idea era di eseguire la rilevazione di oggetti basandosi solamente sull’output dell’ultimo livello dell’estrattore di feature. Tuttavia però alcune informazioni contenute nei layer più profondi della CNN possono essere utili, per questo motivo è stata fatta un’architettura top down con connessioni laterali sfruttando la piramidalità intrinseca delle CNN. Ora FPN è utilizzato in molti modelli, tra cui anche RetinaNet.

1.3.3 One Stage Detector

Lo schema architetturale di un detector a singolo stadio è visibile in Figura 10 (b). Inizialmente è sempre presente un backbone che servirà ad estrarre le feature dall’immagine in input, però a differenza di 1.3.2 non è più presente una eventuale RPN, bensì dal ROI pooling layer si passa direttamente a dei layer convoluzionali che si occuperanno di rilevare e classificare l’oggetto di interesse.

Come detto all’inizio della Sezione 1.3 la struttura dei modelli a singolo stadio li porta ad avere una accuratezza ed una precisione nella localizzazione minore rispetto alla controparte a doppio stadio. Una struttura più semplice dal punto di vista degli strati però porta ad avere maggiore velocità di elaborazione. Di seguito vedremo alcuni dei più famosi modelli a singolo stadio. In particolare in Sezione 1.4 vedremo in dettaglio RetinaNet, ovvero la rete neurale usata durante il lavoro di ricerca di questa tesi.

YOLO Redmon *et al.* proposero un nuovo detector chiamato YOLO con lo scopo di effettuare rilevazioni in tempo reale su video tratti da webcam [5]. Il flusso di YOLO inizialmente prevede la divisione dell’immagine in input tramite una griglia di dimensione $S \times S$. Ogni elemento della griglia è responsabile di predire B BB ed i loro punteggi di confidenza. Questo punteggio viene calcolato per rispecchiare la confidenza che ha il modello nell’affermare che una certa BB contenga un oggetto e per affermare anche quanto è precisa questa predizione. Viene infatti calcolato come $\text{Pr}(\text{Oggetto}) * \text{IOU}_{\text{pred}}^{\text{thruth}}$. Quindi la struttura di una BB è rappresentabile come una quintupla $(x, y, w, h, \text{confidenza})$ dove i primi quattro valori rappresentano le coordinate nello spazio, e l’ultimo la confidenza. Il singolo elemento della griglia inoltre calcola un vettore C dimensionale che rappresenta la probabilità condizionale $P(\text{Classe}|\text{Oggetto})$ per ogni classe. In Figura 12 è possibile vedere come è strutturata la rete nella sua

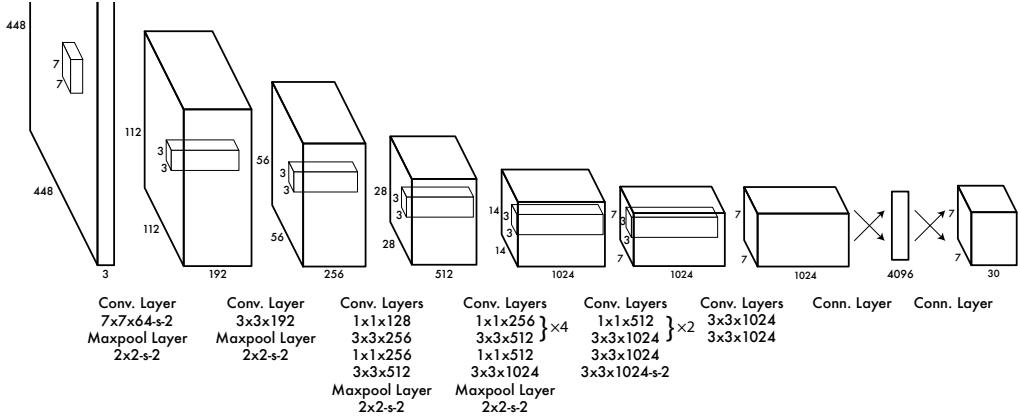


Figura 12.: Struttura di YOLO [5]

totalità.

YOLO v2 YOLO V2 come intuibile dal nome è la seconda versione del detector YOLO che lo migliora in alcuni aspetti utilizzando tecniche prese da lavori precedenti [29]. Innanzitutto è stata adottata la *Batch Normalization* davanti ad ogni layer convoluzionale [73]. Questi strati di regolarizzazione favoriscono la fase di training rendendola più veloce. Inoltre si ottiene un miglioramento del 2% sulla mAP. Il secondo miglioramento è stato aumentare la risoluzione dell'input del classificatore. Nella prima versione la risoluzione è di 224×224 , che viene incrementata 448×448 quando si effettua la rilevazione. In YOLO V2 viene prima effettuato un fine tuning di 10 epoche del classificatore usando come risoluzione 448×448 . La prima versione di YOLO per predire la posizione delle BB usava degli strati totalmente connessi posizionati dopo i layer convoluzionali. Ora invece in YOLO V2 viene usato l'approccio delle *Anchor Boxes* rimuovendo quindi questi ultimi layer insieme al layer di pooling. In questo modo si mantiene una feature map ad alta risoluzione adatta allo scopo. È stata modificata anche la rete per far sì che operi alla risoluzione di 416×416 . Il numero 416 è stato scelto per via del fatto che si possa mappare con feature 13×13 , ed avendo un numero di pixel dispari si ha un unico pixel centrale.

Le predizioni riguardanti dimensioni e proporzioni delle BB fino ad ora veniva fatto in maniera empirica, anche in detector come Fast R-CNN. La seconda versione di YOLO usa l'algoritmo di clustering *K-Means* sull'insieme dei dati per l'addestramento per ottenere delle buone probabilità a priori. La particolarità è che non viene usata la distanza euclidea, ma una misura basata sull'IoU. Un altro aspetto migliorato riguarda le feature.

L'estrazione come detto in precedenza prevede finestre più piccole di dimensione 13×13 , questo porta ad un miglioramento nella rilevazione di oggetti piccoli. È presente anche un layer che concatena feature a dimensione 26×26 con quelle a 13×13 su canali differenti. Il training ora viene fatto usando una risoluzione di input differente che varia da 10 batch in 10 batch. Il modello effettua una riduzione di un fattore 32 dalla risoluzione standard di 416×416 ad una feature map di dimensione 13×13 . Per mantenere questa struttura quindi la risoluzione di input varia da minimo di 320×320 fino ad un massimo di 608×608 attraversando esclusivamente i multipli di 32. Da YOLO V2 deriva anche YOLO 9000 che riesce a rilevare, a discapito di un degrado delle performance, fino a 9000 oggetti.

YOLO v3 YOLO V3 è l'ultima versione, ed ottenuta tramite miglioramenti dalla V2 [74]. La prima miglioria effettuata è sulla predizione dei label, in YOLO V3 è possibile assegnare ad una BB più classi. Non si usa più un classificatore *softmax*, bensì un classificatore logistico coadiuvato dalla loss *cross-entropy*.

All'estrattore di feature sono stati aggiunti alcuni layer convoluzionali che permettono di predirre BB con 3 scale differenti. L'ultimo di questi layer restituisce in output un tensore 3d che rappresenta la predizione in termini di coordinate della BB, misura di confidenza e predizioni delle classi. Gli esperimenti effettuati dagli autori sono stati fatti sul dataset MS-COCO e sono state predette 3 BB ad ogni scala differente, perciò il tensore di output è di dimensione $N \times N \times [3 \cdot (4 + 1 + 80)]$, dove l'ultima dimensione è strutturata in questo modo in quanto le scale sono 3 differenti, per ogni BB ci sono 4 coordinate, più 1 misura di confidenza, più 80 valori di probabilità sulle classi. Terza novità è la rete di backbone, che cambia passando a *Darknet-53*

SSD Single-Shot Detector (SSD) [28] è detector basato su una rete neurale convoluzionale che produce un insieme di dimensione fissata composto da BB e indici di confidenza per indicare la presenza o meno di oggetti al loro interno. Infine è presente una fase di NMS per produrre le rilevazioni finali.

La rete è divisa fondamentalmente in due parti, la parte iniziale è un'architettura standard usata per la classificazione di immagini ma troncata in fondo in maniera da rimuovere gli ultimi layer che si usano per la classificazione. Successivamente è presente una struttura ausiliaria che si divide a sua volta in altre tre fasi:

- Feature multi scala: alla fine della rete troncata vengono aggiunti una serie di layer convoluzionali che via via diminuiscono in dimensione e perciò permettono di effettuare rilevazioni su scale multiple.
- Predittori convoluzionali: Ogni layer aggiunto alla fase precedente produce un insieme fisso di predizioni usando un gruppo di filtri convoluzionali. Considerando un layer con dimensione $m \times n$ a p canali abbiamo un filtro di dimensione $3 \times 3 \times p$.
- BB di default: è un approccio simile a quanto già visto con le *Anchor Boxes*, tuttavia sono applicate a diverse mappe di feature a diverse risoluzioni.

Durante l’addestramento di SSD si cerca di far combaciare il più possibile le BB di default con le BB che rappresentano la *ground truth*. Per fare ciò bisogna determinare quale BB di default ha maggior intersezione con le BB rappresentanti la verità. Questo si fa attraverso l’indice di Jaccard (o IoU) e prendendo quelle BB di default con gli indici maggiori. Dopo questa fase di matching delle BB ci sono molte predizioni negativi che introducono sbilanciamento tra esempi positivi e negativi in fase di training, perciò si adotta una tecnica di HNM ordinando gli esempi negativi secondo il loro livello di confidenza e prendendo solo i più alti. In questo modo si riesce a ridurre il rapporto tra esempi negativi e positivi a circa 3 su 1.

DECONVOLUTIONAL SSD Deconvolutional Single-Shot Detector (DSSD) [75] è una versione modificata di SSD che aggiunge un modulo di predizione e di deconvoluzione, oltre ad usare ResNet-101 come backbone. In Figura 13 è possibile vedere un confronto delle due architetture.

I moduli di predizione aggiunti via via nel corso della computazione servono a rilevare oggetti di varie dimensioni e proporzioni, mentre i layer di deconvoluzione portano ad avere delle feature più rappresentative.

M2DET [6] Nasce per andare incontro all’esigenza di rilevare oggetti uguali, ma con dimensioni differenti. Come si può vedere in Figura 14 dopo la rete di backbone per l’estrazione delle feature è presente una Multi-Level Feature Pyramid Network (MLFPN) composta a sua volta di tre moduli:

- Feature Fusion Module (FFM): serve ad fondere le feature provenienti da livelli diversi, usano layer convoluzionali di dimensione 1×1 per comprimere i canali di input e aggregare le feature map.

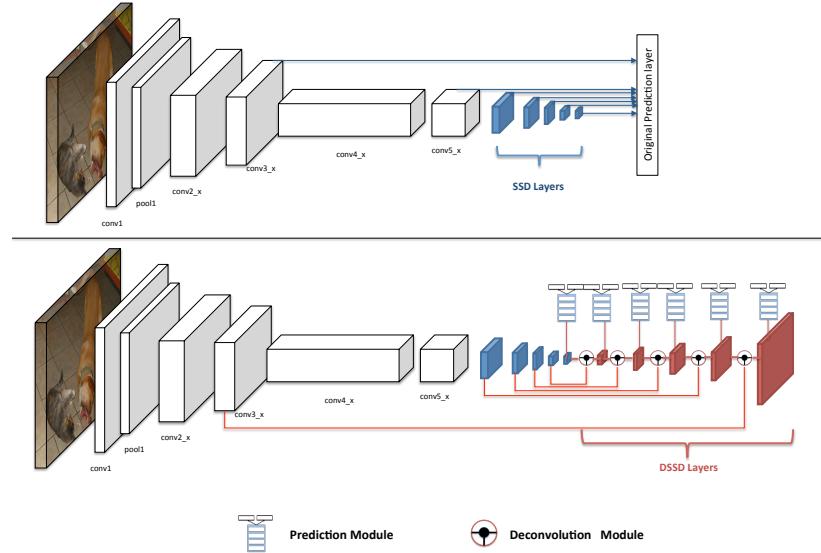


Figura 13.: Architetture di SSD e DSSD. In blu i layer di SSD, in rosso quelli di DSSD.

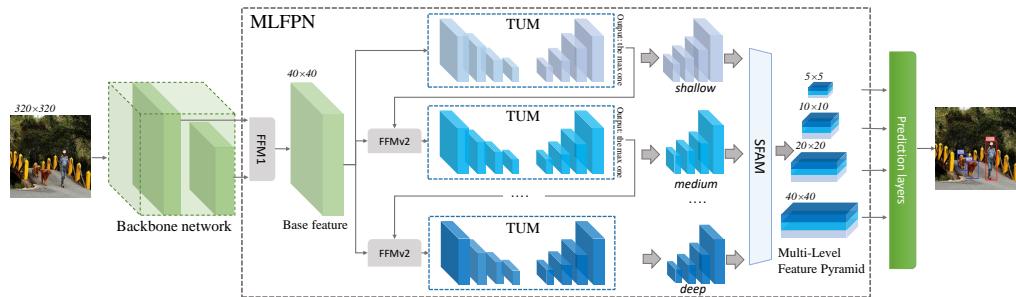


Figura 14.: Struttura di m2det [6]

- Thinned U-shape module (TUM): genera un gruppo di feature multi scala. È formato da una serie di layer convoluzionali di dimensione 3×3 con stride 2.
- Scale-wise Feature Aggregation Module (SFAM): aggrega le feature generate da TUM in strutture piramidali tramite concatenazione.

1.4 RETINANET

Il trade-off tra precisione e velocità è sempre stato un punto cruciale che ha determinato la scelta tra detector a singolo stadio ed a doppio stadio. Con RetinaNet [55] si tenta di risolvere questo problema. Secondo

i ricercatori è da imputare al forte sbilanciamento che si incontra in fase di training tra esempi di sfondo ed esempi di oggetti. Quindi è stata introdotta una versione modificata della classica *cross-entropy* loss chiamata *Focal Loss* che migliora la situazione.

1.4.1 Focal Loss

Prima di introdurre la *Focal Loss* è necessario ricordare la forma della *Cross-Entropy* per problemi di classificazione binaria:

$$\text{CE}(p, y) = \begin{cases} -\log(p) & \text{se } y = 1 \\ -\log(1-p) & \text{altrimenti.} \end{cases}$$

con $y \in \{\pm 1\}$ che specifica la classe di *ground truth* e p valore di probabilità per la classe con $y = 1$. Per comodità si definisce un'indice p_t che indica se un esempio è classificato bene o meno:

$$p_t = \begin{cases} p & \text{se } y = 1 \\ 1-p & \text{altrimenti,} \end{cases}$$

un valore di p_t superiore a 0.5 indica che l'esempio è stato classificato correttamente, viceversa indica un errore nella classificazione. Con p_t è poi possibile riscrivere $\text{CE}(p, y) = \text{CE}(p_t) = -\log(p_t)$. In Figura 15 la curva blu rappresenta la *Cross Entropy* ed è possibile notare una proprietà riguardante gli esempi considerati facili da classificare, ovvero quelli con un valore p_t molto superiore a 0.5. Come è possibile vedere la loss di questi esempi ha comunque un valore elevato, nonostante siano considerati facili. Quindi sommando questa serie di valori si ottiene una loss comunque alta che non permette di concentrarsi sull'addestramento degli esempi considerati positivi e dare minor peso agli esempi negativi quali lo sfondo. Questo forte sbilanciamento tra sfondo e oggetti è di circa 1 a 10000 e per evitare effetti disastrosi sulla funzione di loss i ricercatori hanno cercato innanzitutto di bilanciare la *Cross Entropy* con un fattore $\alpha \in [0, 1]$ per la classe 1 e $1 - \alpha$ per la classe -1. Per comodità, analogamente a p_t è possibile definire anche α_t come:

$$\alpha_t = \begin{cases} \alpha & \text{se } y = 1 \\ 1 - \alpha & \text{altrimenti} \end{cases}$$

Quindi partendo da α_t è possibile definire la nuova *Cross Entropy* bilanciata $\text{CE}_b(p_t) = -\alpha_t \log(p_t)$.

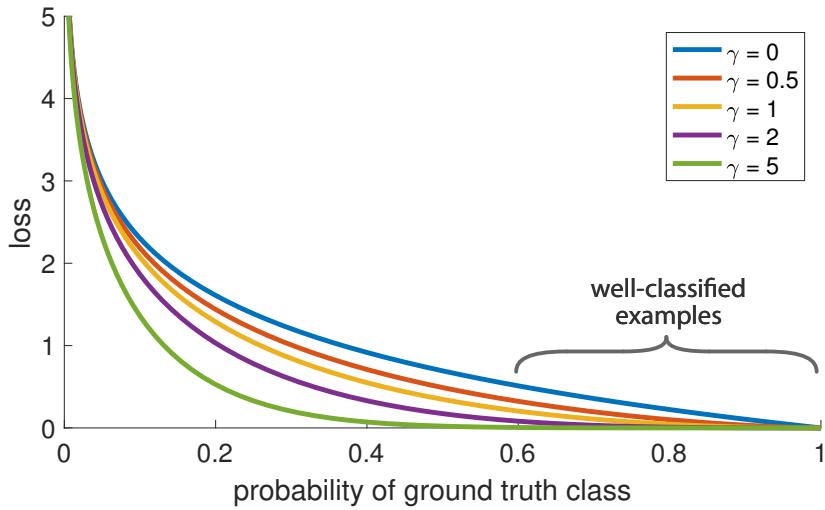


Figura 15.: Focal Loss al variare di γ

Con il termine α si è riusciti quindi a bilanciare gli esempi negativi da quelli positivi, però rimane il problema di bilanciare tra esempi la cui detection è facile o difficile, ovvero quelli per cui abbiamo un valore p_t elevato o basso. Lo scopo quindi è dare poca rilevanza agli esempi facilmente classificabili e molta rilevanza a quelli difficilmente classificabili. Si aggiunge quindi alla *Cross Entropy* un nuovo termine $(1 - p_t)^\gamma$ con $\gamma \geq 0$ ottenendo una nuova funzione di loss detta *Focal Loss*:

$$FL(p_t) = -(1 - p_t)^\gamma \log(p_t)$$

. Realizzando un'analisi di questa funzione possiamo affermare che avendo un esempio classificato male ($p_t < 0.5$), il fattore modulativo $(1 - p_t)^\gamma$ è vicino a 1 e la loss rimane più o meno invariata, mentre più p_t tende ad 1 e più il fattore modulativo sarà vicino a 0, rendendo così la loss per gli esempi classificati bene meno efficace. Possiamo inoltre immaginare γ come un parametro che permette di gestire questa riduzione di efficacia. Ponendo $\gamma = 0$ ci si ricongiunge alla *Cross Entropy*, mentre più si aumenta e più si ottiene la riduzione di efficacia per gli esempi classificati bene.

Unendo la *Focal Loss* e la *Cross Entropy* loss bilanciata si ottiene la forma di *Focal Loss* utilizzata all'interno di RetinaNet:

$$FL(p_t) = \alpha_t(1 - p_t)^\gamma \log(p_t)$$

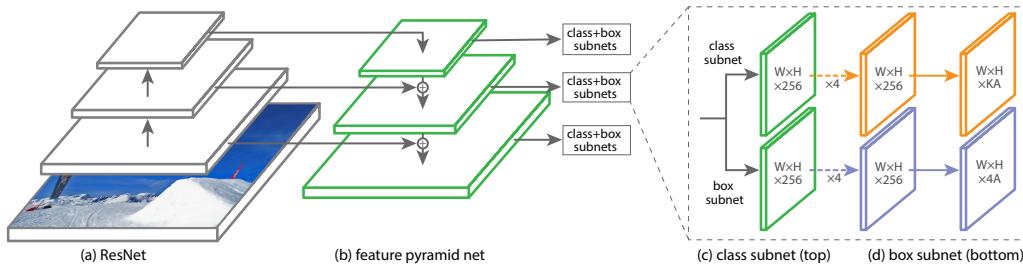


Figura 16.: Struttura di RetinaNet

1.4.2 Struttura del detector

RetinaNet è composta da tre sottoreti, la prima è il backbone, le altre due sono una rete convoluzionale adibita alla classificazione e l’ultima una rete convoluzionale per la regressione sulle BB. In Figura 16 è possibile vedere questa struttura appena descritta. Descriveremo di seguito le componenti del modello.

FEATURE PYRAMID NETWORK BACKBONE Una parte del backbone di RetinaNet è una FPN [30] che costruisce una struttura di feature piramidali a partire da un’immagine in input. Ogni livello della piramide può essere utilizzato per rilevare oggetti con dimensioni differenti. La FPN utilizzata è stata costruita al di sopra di una architettura ResNet [76]. È stata quindi costruita una struttura prendendo i livelli che vanno da P_3 a P_7 tale per cui il livello P_l con $l \in [3, 7]$ ha una risoluzione minore di un fattore 2^l rispetto all’input. Tutti i livelli hanno $C = 256$ canali. Inoltre a differenza di [30] i livelli che vanno da P_3 a P_5 sono calcolati direttamente a partire dai corrispondenti livelli C_3, C_4 e C_5 di ResNet. Riguardo P_6 è ottenuto tramite un layer convoluzionale con filtri 3×3 e stride 2 da C_5 . P_7 è ottenuto applicando ReLU alla stessa operazione effettuata per ottenere P_6 , ma applicata su C_6 .

ANCHOR Le *Anchor Boxes* sono simili a quelle utilizzate in [30]. Hanno un’area che può variare da 32×32 a 512×512 rispettivamente sui livelli della piramide delle feature che vanno da P_3 a P_7 . Come in [30] ogni livello della piramide usa *Anchor Boxes* di tre differenti proporzioni ($1 : 2, 1 : 1, 2 : 1$). Inoltre per aumentare la copertura ad ogni livello vengono aggiunte *Anchor Boxes* che hanno dimensioni scalate di $2^{\frac{1}{3}}$ e $2^{\frac{2}{3}}$ rispetto alle originali. Ogni layer della piramide ha quindi $A = 9$ *Anchor Boxes*, e per ognuna di esse si associano due vettori

- un vettore one-hot di dimensione K come le K classi usate nel task di rilevazione
- un vettore di dimensione 4 contenente le coordinate delle BB ottenute per regressione

Le *Anchor Boxes* vengono assegnate ad una BB che fa parte della *ground truth* sulla base dell'indice di Jaccard. Se questo indice è inferiore a 0.4 allora la *Anchor Boxes* è considerata sfondo, mentre se il valore è compreso tra 0.4 e 0.5 è ignorata durante il training. Nel caso in cui invece l'indice di Jaccard di una *Anchor Boxes* rispetto ad una BB è superiore a 0.5 viene assegnato il valore 1 alla posizione corrispondente alla classe nel vettore di dimensione K.

SOTTORETE DI CLASSIFICAZIONE Questa sottorete si occupa di calcolare la probabilità di presenza di un oggetto per ognuna delle *Anchor Boxes* dell'insieme A e per ognuna delle K classi. È realizzata tramite una FCN attaccata ad ogni livello della FPN. I parametri della rete convoluzionale sono condivisi tra i livelli.

La sottorete prende in input una feature map con C canali presa da un livello della piramide e ci applica quattro layer convoluzionali di dimensione 3×3 con C filtri, ognuno con una funzione di attivazione ReLU, seguita a sua volta da un altro layer convoluzionale di dimensione 3×3 con KA filtri. Infine la funzione di attivazione scelta è la *Sigmoid*.

SOTTORETE DI REGRESSIONE In parallelo alla rete di classificazione lavora la rete di regressione che attacca un'altra FCN ad ogni livello della piramide per calcolare, partendo dalle *Anchor Boxes* le coordinate della BB contenente l'oggetto. La struttura è del tutto simile alla rete usata per la classificazione, ma alla fine i filtri non sono KA bensì 4A.

2

SISTEMA

In questa sezione sarà descritta la struttura del sistema utilizzata al fine di realizzare gli esperimenti. Dopo una introduzione ai dataset utilizzati verranno descritte le tecniche utilizzate, in particolare il Transfer Learning e la Data Augmentation.

2.1 DATASET

Inizialmente i dataset usati per gli esperimenti sono due. Il primo è KAIST Multispectral Pedestrian Dataset (KAIST MPD) [77], per cui è disponibile un'ampia documentazione, il secondo è un dataset gratuito realizzato dalla FLIR [78] per cui è disponibile una documentazione molto stringata. Successivamente è stato reso disponibile per l'utilizzo ai fini di questa tesi un insieme di video termici di Rete Ferroviaria Italiana (RFI).

2.1.1 KAIST Multispectral Pedestrian Dataset

Il Korea Advanced Institute of Science and Technology (KAIST) propone in [77] un dataset che fornisce coppie di immagini termiche e a colori. La particolarità che offre questo dataset è che le due immagini sono allineate. Inoltre sono state raccolte sufficienti immagini sia diurne che notturne.

SPECIFICHE HARDWARE KAIST ha sviluppato una piattaforma basata su una camera a colori, una termica ed un *Beam Splitter*, oltre che ad un supporto a tre assi chiamato *camera jig*. Un *Beam Splitter* è un dispositivo ottico di forma cubica formato in molti casi da due prismi che divide la luce in due parti. In questo caso viene utilizzato per l'allineamento delle due immagini in quanto permette il passaggio dello spettro termico mentre quello visibile viene riflesso. Il dispositivo usato per la realizzazione del dataset è stato costruito a partire da un wafer di silicio zinato.

Le telecamere utilizzate sono una *PointGrey Flea3* per la parte a colori ed una *FLIR-A35* per la parte termica. La prima acquisisce immagini ad una risoluzione di 640x480 pixels con un Field of View (FOV) di 103.6°, mentre la seconda ha una risoluzione di 320x256 con un FOV di 39°. Come si può notare il campo visivo della telecamera visibile è più ampio di quello della telecamera termica, motivo per cui viene sacrificata parte dell'immagine visibile al fine di allineare i due fotogrammi. Il *framerate* è di 20 FPS.

CALIBRAZIONE L'idea per la realizzazione di questa architettura hardware è stata ripresa dal lavoro di Bienkowski *et al.* [79]. Sempre in questo lavoro però non si fa riferimento alla metodologia usata per la calibrazione. Parleremo in questo paragrafo dell'approccio utilizzato per la realizzazione di questo dataset.

Innanzitutto è stata calcolata la traslazione fra le due telecamere, applicando un tipo di calibrazione stereo. Si può osservare che gli assi ottici delle telecamere al di là della divisione del fascio di luce sono paralleli per via della struttura hardware su cui sono montate. Di conseguenza, fra i due domini dell'immagine, è presente unicamente una traslazione ed è necessario solamente aggiustare la posizione tramite *camera jig* finché la traslazione non diventa nulla. Dopo l'aggiustamento, i due domini sono rettificati fino ad avere la stessa distanza focale virtuale. Al termine di queste procedure, oltre alla focale, i domini condividono i punti principali. Il dominio dell'immagine, virtualmente allineato, ha 640x512 pixel di risoluzione spaziale e analogamente a quella umana ha un FOV di 39°. Un pattern a scacchiera convenzionale non è osservabile con telecamera termica, quindi viene invece utilizzata una tavola di calibrazione speciale, con un certo numero di buchi. Quando viene scaldata, si ottiene una differenza di temperatura fra la tavola e i buchi, che possono essere osservati nel termico. In Figura 17 è raffigurato un esempio della tavola usata per la calibrazione. A sinistra abbiamo l'immagine termica, centralmente invece è presente l'immagine a colori con distorsioni dovute al *beam splitter*, mentre a destra si ha l'immagine a colori dopo la correzione.

CORREZIONE DEI COLORI Per via del passaggio all'interno dei prismi del *Beam Splitter* le immagini catturate, soprattutto nello spettro del visibile, mostrano distorsioni piuttosto evidenti dei colori. Per gestire questo problema è stato deciso di acquisire un fotogramma di riferimento completamente bianco che mostrava distorsioni di colore. Per motivi legati al sensore utilizzato all'interno della telecamera visibile la distorsione del

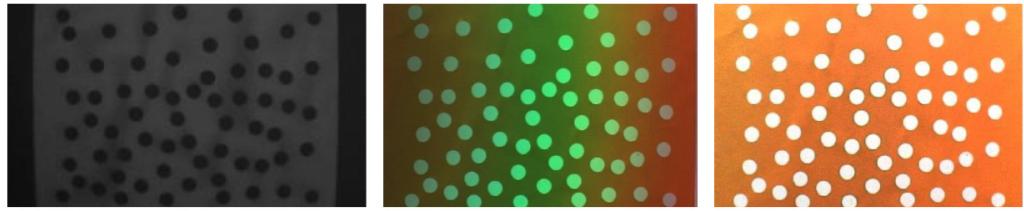


Figura 17.: Pattern a scacchiera usato per la calibrazione di KAIST

colore può essere considerata come una funzione lineare. Quindi ogni pixel dell'immagine di riferimento può essere usato come coefficiente di correzione per le altre immagini, dividendo il livello di intensità di queste immagini per questi coefficienti.

ACQUISIZIONE DEI DATI E ANNOTAZIONI Tutto il marcheggiamento è composto dalle due telecamere, il *Beam Splitter* ed il *Camera jig* è stato montato sul tetto di un'automobile al fine di realizzare immagini egocentriche del traffico. In particolare, come già accennato in precedenza, sono state realizzate raccolte di dati sia di giorno che di notte.

Il numero totale delle coppie di immagini catturate sono 95328 che sono state annotate manualmente con un totale di 103128 BB. Per realizzare le annotazioni è stata usata una versione modificata del Piotr's Computer Vision Toolbox [80]. Le BB sono state annotate con quattro differenti label:

- person: individuo singolo ben individuabile
- people: individui non distinguibili
- cyclist: persone che stanno utilizzando una bicicletta
- person?: individuo non ben identificabile per via di fotogrammi molto densi

Inoltre, anche se per lo scopo della tesi non sono stati presi in considerazione, ogni BB ha una corrispondenza temporale che identifica il singolo individuo attraverso i vari frame.

TRAIN E TEST SET Per dividere tra *train set* e *test set* è stato usato un criterio ben definito:

- Il numero di pedoni nei due set è simile
- Il numero di frame notturni e diurni nei due set è simile
- I due set non si sovrappongono

PROPRIETA

- Attributo di scala: per ogni BB è associato un valore di scala. Questo valore dipende dalla distanza che ha il pedone dall'automobile, ed è giustificato dalla seguente logica. Supponendo che una vettura in area urbana viaggia ad una velocità compresa tra 30 e 50 km/h lo spazio di arresto varia tra gli 11 ed i 28 metri. Questo intervallo, scalato opportunamente rispetto alla risoluzione dell'immagine, e considerando anche che l'altezza media di un pedone è di 1.7 metri corrisponde ad un range che va da 45 a 115 pixel. All'interno di questo range le rilevazioni vengono poste come *medium*, al di sopra *far* ed al di sotto *near*.
- Occlusione: questo attributo associa ad ogni BB un valore che rappresenta l'occlusione del pedone. I valori possibili sono *no occlusion*, *partial occlusion*, *heavy occlusion*. I primi sono circa il 78.6%, i secondi circa il 12.6% e gli ultimi 8.8%.
- Posizione: l'impostazione dell'hardware rispecchia il più possibile quello di un essere umano, motivo per cui questo particolare setup concentra il rilevamento di pedoni nell'area centrale dell'immagine, in particolare nel lato destro. Questo è motivato dal fatto che nel paese dove sono stati acquisiti questi dati la guida è sulla destra. In Figura 18 è possibile vedere questo fenomeno.
- Cambio d'aspetto: l'aspetto dei pedoni all'interno del dataset è molto variabile. In condizioni di pieno sole i pedoni sono ben visibili e con dei contorni ben definiti, mentre la differenza di temperatura tra l'ambiente circostante ed il pedone è meno marcata. Quindi nello spettro a colori sono presenti pedoni ben definiti, mentre nel termico no. Di notte invece, per via delle temperature ambientali più basse e per l'assenza di luce si verifica il contrario.

2.1.2 *FLIR Thermal Starter Dataset*

Come già accennato in precedenza la documentazione riguardante questo dataset è molto stringata, limitandosi dunque ad una sola pagina web molto riassuntiva. Cercheremo in questa sezione di parlare degli aspetti che caratterizzano questo dataset.

Il dataset in questione offre immagini termiche con annotazioni e l'equivalente a colori non annotato. A differenza di 2.1.1 le due immagini

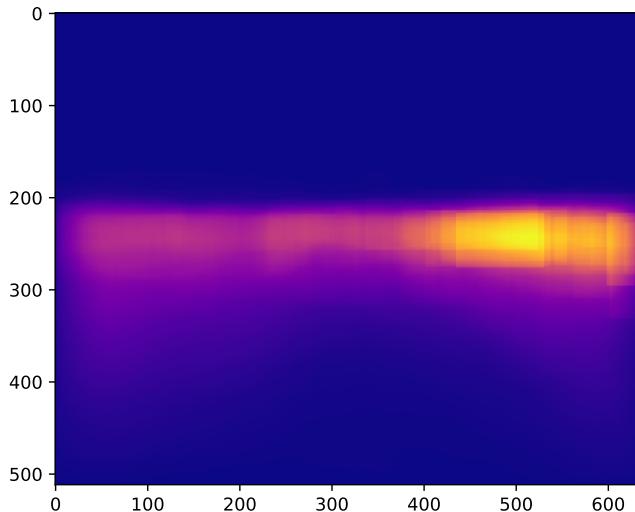


Figura 18.: Heatmap riguardante la posizione dei pedoni

non sono allineate, quindi non è possibile portare le annotazioni delle immagini termiche sulle immagini a colori. Le immagini sono state acquisite tramite telecamere montate su una vettura e contiene un totale di 14453 immagini, di cui 10228 campionate da video di breve durata e 4224 provenienti da video di 144 secondi. Tutte le immagini sono state acquisite su strade ed autostrade a Santa Barbara, in California. L'arco temporale varia da Novembre a Maggio, nella stessa quantità di giorno e notte. Il meteo è generalmente buono.

Le immagini termiche sono state scattate con una *FLIR Tau2*, mentre quelle RGB con una *FLIR BlackFly*. Entrambi i device sono stati impostati in maniera tale da avere lo stesso FOV e per quanto riguarda il resto sono state lasciate entrambe alle impostazioni di default. Le videocamere sono state posizionate sullo stesso supporto a distanza di circa 1.9 pollici (circa 4.8 centimetri) l'una dall'altra. Il *framerate* è di 2 frame al secondo in scenari densi di annotazioni, mentre in scenari più tranquilli è stato deciso di scendere ad un frame al secondo.

Le annotazioni dove possibile ricalcano i codici adottati dal dataset COCO, ed hanno i seguenti codici:

- 1 People: esseri umani.
- 2 Bicycles: biciclette e motocicli. Questa è l'unica categoria non consistente con il formato adottato da COCO.

- 3 Cars: automobili e veicoli piccoli.
- 18 Dogs: cani
- 91 Other Vehicle: camion, rimorchi e imbarcazioni.

Le annotazioni sono state fatte manualmente da esseri umani ai quali è stato comunicato di fare BB il più piccole possibili e che omettessero piccole parti di oggetti, accessori personali e parti occluse. Inoltre è stato comunicato di non annotare oggetti di piccole dimensioni o molto occlusi e persone delle quali si vede solo braccia o gambe.

2.2 ADDESTRAMENTO INIZIALE DI RETINANET

In questa sezione presenteremo i risultati iniziali dell'addestramento di RetinaNet sui due dataset descritti in sezione 2.1. Per lo scopo è stata usata una versione di *RetinaNet* implementata tramite *Keras* reperibile in forma originale al seguente link. Durante lo sviluppo del lavoro di tesi le modifiche al codice originale sono state molteplici, tanto da aver richiesto un *fork* della *repository* originale reperibile al seguente link. Per tenere traccia dell'addestramento è stato usato *Weight & Biases*.

2.2.1 Transfer Learning

Inizialmente è stata usata la tecnica del *Transfer Learning*. Una rapida spiegazione del significato di questa espressione ce la fornisce il libro *Deep Learning* di *Goodfellow et al* [81]

Transfer learning and domain adaptation refer to the situation where what has been learned in one setting is exploited to improve generalization in another setting.

Per un'introduzione più dettagliata su cos'è il *transfer learning* e sui vari tipi è stato preso spunto da *A survey on Transfer Learning* di *S. J. Pan e Q. Yang* [82].

La necessità di attuare tecniche di *transfer learning* deriva dal fatto che molti modelli di Machine Learning lavorano bene solo sotto determinate assunzioni, soprattutto quella che i dati di addestramento e di test derivino dallo stesso spazio delle *feature* e dalla stessa distribuzione. I problemi sorgono quando cambia la distribuzione, in quanto è necessario procedere ad una nuova fase di training.

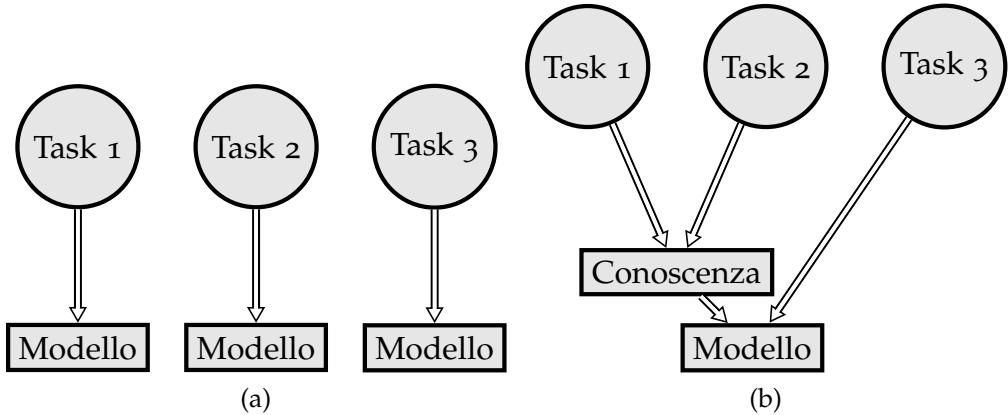


Figura 19.: Differenze tra apprendimento tradizionale e transfer learning

Le casistiche in cui il *transfer learning* è applicabile sono molteplici, ad esempio l’analisi dei sentimenti, dove il compito è classificare le recensioni di un determinato prodotto in positive o negative. Per un compito del genere il primo passo da effettuare è la raccolta e l’annotazione di recensioni. Successivamente è necessaria una fase di addestramento di un modello usando come dati le recensioni precedentemente raccolte ed annotate. Lo scopo è poter usare lo stesso modello per vari prodotti, in questo caso però si va incontro al problema che le distribuzioni dei dati su diversi prodotti possono differire anche di molto. La soluzione sarebbe quindi di annotare altre recensioni, ma richiederebbe uno sforzo notevole. L’idea è quindi di adattare un modello di classificazione, addestrato su alcuni prodotti, per aiutare la fase di addestramento su articoli differenti. Le differenze tra processi di apprendimento tradizionali e *transfer learning* sono mostrate in Figura 19.

NOTAZIONE PRELIMINARE Per introdurre un po’ più nello specifico le varie tipologie di *transfer learning* è necessario definire alcuni concetti. Il primo di questi è il *Dominio* \mathcal{D} , definito come una tupla $\mathcal{D} = \{\mathcal{X}, P(X)\}$, dove \mathcal{X} è lo spazio delle *feature* e $P(X)$ con $X = \{x_1, x_2, \dots, x_n\} \in \mathcal{X}$ è una distribuzione di probabilità marginale. In generale due domini possono essere considerati differenti se hanno differenti spazi delle *feature* o distribuzioni differenti.

Dato uno specifico dominio $\mathcal{D} = \{\mathcal{X}, P(X)\}$ è possibile definire il *task*. Un *task* \mathcal{T} è una tupla $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$ con \mathcal{Y} spazio dei *label* e $f(\cdot)$ funzione di predizione. La particolarità del *task* è il fatto che non è osservabile ma può essere appreso dai dati di addestramento, che consistono di una coppia

$\{x_i, y_i\}$ con $x_i \in X$ e $y_i \in Y$. Una volta completata la fase di addestramento dovrebbe essere possibile usare la funzione $f(\cdot)$ per predirre il label $f(x)$ corrispondente ad una nuova istanza x .

Chiameremo il dominio sorgente \mathcal{D}_S ed il dominio target \mathcal{D}_T . In particolare avremo a disposizione anche i dati D_S del dominio sorgente, definiti come $D_S = \{(x_{S_1}, y_{S_1}), \dots, (x_{S_n}, y_{S_n})\}$ tali che $x_{S_i} \in \mathcal{X}_S$ è l'istanza del dato e $y_{S_i} \in \mathcal{Y}_S$ è il corrispondente label. In maniera similare definiamo anche i dati del dominio target $D_T = \{(x_{T_1}, y_{T_1}), \dots, (x_{T_n}, y_{T_n})\}$ tali che $x_{T_i} \in \mathcal{X}_T$ è l'istanza del dato e $y_{T_i} \in \mathcal{Y}_T$ è il corrispondente label.

Dire che due domini \mathcal{D}_S e \mathcal{D}_T sono differenti implica che o $\mathcal{X}_S \neq \mathcal{X}_T$ oppure $P_X(X) \neq P_T(X)$. In maniera del tutto analoga è possibile definire la differenza tra due task T_S e T_T . Nel caso in cui i due domini ed i due task sono uguali ci si riconduce ad un tradizionale problema di apprendimento. Quando invece c'è una relazione, implicita o esplicita, tra gli spazi delle *feature* dei due domini si dice che il dominio sorgente e target sono in relazione tra di loro.

TIPOLOGIE DI TRANSFER LEARNING Prima di introdurre le varie tipologie di *transfer learning* è necessario definire formalmente questo concetto e fare alcune premesse.

Definizione 2.2.1. (Transfer Learning) Dato un dominio sorgente \mathcal{D}_S , un task di apprendimento T_S , un dominio target \mathcal{D}_T ed un task di apprendimento T_T , il transfer learning tenta di migliorare l'apprendimento di $f(\cdot)_T \in \mathcal{D}_T$ usando la conoscenza in \mathcal{D}_S e T_S , con $\mathcal{D}_S \neq \mathcal{D}_T$ o $T_S \neq T_T$.

Quando si parla di *transfer learning* bisogna mettere in conto tre problemi: *cosa*, *come* e *quando* trasferire.

- Cosa trasferire: quale parte della conoscenza bisogna trasferire tra domini o task sorgenti e target. In particolare possiamo dire che alcune conoscenze sono comuni tra i diversi domini, mentre altre sono specifiche.
- Come trasferire: a questo problema pone una soluzione l'algoritmo di trasferimento della conoscenza.
- Quando trasferire: ci chiediamo in quali situazioni è realmente necessario applicare tecniche di *transfer learning* ed in quali non è assolutamente necessario. Ad esempio in casi in cui i due domini sorgente e target non sono in relazione tra di loro il *transfer learning* potrebbe non portare ad alcun risultato positivo. Si tende

quindi sempre a parlare di *transfer learning* dando per scontato che i due domini siano in qualche modo relazionati tra di loro, in quanto altrimenti non avrebbe senso andare oltre l'apprendimento tradizionale.

Possiamo ora descrivere le tre categorie di *transfer learning*:

- *Transfer Learning Induttivo*: il task target è differente dal task sorgente e non importa se il dominio sorgente ed il dominio target sono uguali o meno. Per indurre un modello predittivo oggettivo, da usare nel dominio target, sono necessari alcuni dati etichettati nel dominio sorgente. A seconda della tipologia ed alla quantità di annotazioni possiamo dividere questo tipo di *transfer learning* in ulteriori due sottocategorie.
 - Molti dati annotati nel dominio sorgente: ci si riconduce al caso del *multitask learning*, tuttavia mentre lo scopo di quest'ultimo è operare bene in entrambi i domini, lo scopo del *transfer learning* induttivo è operare bene solamente sul dominio obiettivo.
 - Nessun dato annotato nel dominio sorgente: le similarità in questo caso ci portano a pensare al *self-taught learning*, dove le annotazioni tra il dominio sorgente ed obiettivo sono totalmente differenti, e quindi non direttamente utilizzabili.
- *Transfer Learning Trasduktivo* (*si traduce così "transductive"*?): in questo caso il task obiettivo ed il task sorgente sono i medesimi, mentre i domini sono differenti. Abbiamo quindi molti dati annotati nel dominio sorgente e nessuna annotazione nel dominio obiettivo. Possiamo a sua volta dividere questa categoria in ulteriori due sottocategorie.
 - Gli spazi delle feature tra sorgente e obiettivo sono differenti, più formalmente abbiamo $\mathcal{X}_S \neq \mathcal{X}_T$
 - Gli spazi delle feature tra sorgente e obiettivo sono gli stessi, ma cambia la distribuzione di probabilità marginale, quindi $P(X_S) \neq P(X_T)$. Quest'ultimo caso è chiamato anche *Domain Adaptation*.
- *Transfer Learning non supervisionato*: il task obiettivo è differente dal task sorgente, ma hanno una qualche tipo di relazione tra di loro. Tuttavia l'attenzione si focalizza sul risolvere compiti di apprendimento non supervisionati nel dominio obiettivo, come

Tipologia	Similarità	Annotazioni Sorgente	Annotazioni Target	Campo di applicabilità
Induttivo	Multitask Learning	SI	SI	Regressione e Classificazione
	Self-taught Learning	NO	SI	Regressione e Classificazione
Trasduttivo	Domain adaptation	SI	NO	Regressione e Classificazione
Non supervisionato	46501	NO	NO	Clustering, dimensionality reduction

Tabella 2.: Schema riassuntivo delle categorie di Transfer Learning

possono essere il *clustering*, *dimensionality reduction* o *density estimation*. Non abbiamo quindi annotazioni né nel dominio sorgente, né nel dominio obiettivo.

In Tabella 2 sono riassunte tutte le caratteristiche principali delle varie categorie di *transfer learning*.

2.2.2 Addestramento sulle immagini RGB

Per il primo esperimento è stato deciso di effettuare un training di *Retina-Net* partendo dai pesi della rete precedentemente addestrata sul dataset di COCO. Il dataset utilizzato è KAIST MPD, descritto precedentemente in 2.1.1. Il motivo è legato al fatto che è l'unico dataset a nostra disposizione a disporre di annotazioni sulle immagini RGB.

Questa prima fase di addestramento è durata circa 40 ore, e come si può vedere dal grafico in Figura 20 è proceduta senza particolari problemi fino ad arrivare a convergenza intorno all'epoca 45. Le classi usate per l'addestramento sono solamente *person* e *cyclist*. È stato deciso di non prendere in considerazioni le rimanenti classi in quanto sono persone non ben distinguibili. Tutti gli esperimenti sono stati eseguiti su una macchina remota dotata di una GPU Nvidia Titan X.

Dopo la fase di addestramento sono stati eseguite le valutazioni sulla parte di dataset adibita ai test. Inizialmente le classi utilizzate per i test sono le stesse usate per l'addestramento. I risultati complessivi vengono riassunti in Tabella 3. La mAP mostrata nell'ultima riga della tabella è stata calcolata come media pesata secondo il numero di esempi all'interno del *test set*.

La valutazione è stata anche effettuata in maniera separata sia sulla parte di *test set* diurna che notturna. I risultati sono mostrati in Tabella 4a e Tabella 4b. Come lecito aspettarsi, avendo visibilità limitata in notturna si ottengono risultati mediamente peggiori. In Figura 21 è possibile vedere un esempio di predizioni fatte sul *test set*.

Il successivo step della valutazione è stato effettuato su più classi, per questa comparazione delle performance è stata presa in considerazione

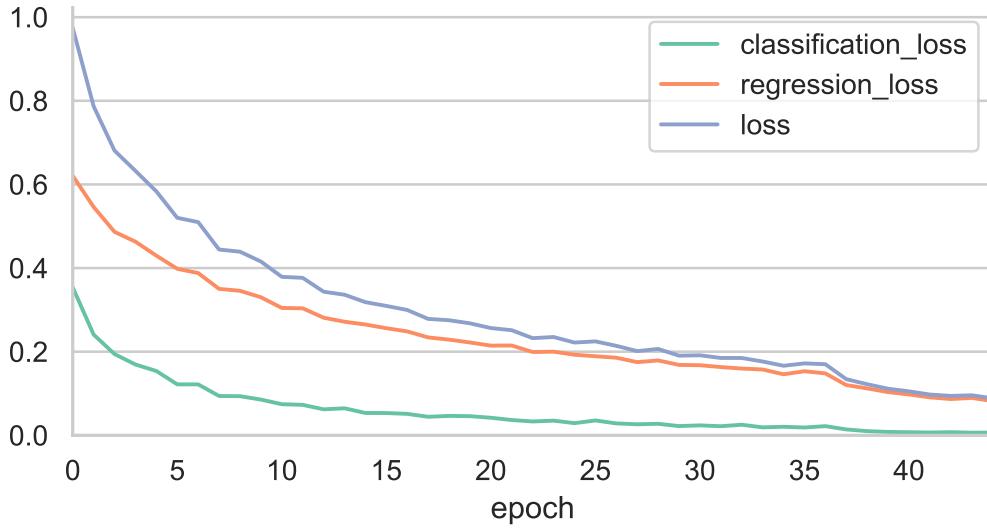


Figura 20.: Transfer learning da COCO a KAIST

	# Istanze	mAP
Person	45195	0.4184
Cyclist	1396	0.1154
Complessivo	46501	0.4093

Tabella 3.: Test complessivo delle performance dopo l'addestramento

anche la classe `people`, ma rinominandola in `person` in maniera tale da poter riusare i pesi della rete già addestrata.

2.2.3 Passaggio alle immagini termiche su KAIST

Dopo la fase di addestramento descritta in Sezione 2.2.2 è stato effettuato il passaggio all’addestramento sulle immagini termiche del dataset di KAIST MPD. La base di partenza per questa fase di *training* sono i pesi derivanti dall’addestramento effettuato in Sezione 2.2.2, è quindi stata attuata una tecnica di *Transfer Learning* (2.2.1) in maniera da ridurre i tempi di addestramento. In questo caso, operando su due tipologie di immagini differenti cambia il dominio, più in particolare lo spazio delle feature. La distribuzione di probabilità sulle varie immagini resta la medesima in quanto il training è svolto sulle stesse immagini, come rimane invariato anche il task, ovvero il riconoscimento di pedoni e ciclisti. La fase di apprendimento è proceduta senza particolari intoppi,

	# Istanze	mAP		# Istanze	mAP
Person	33688	0.4493	Person	11507	0.3281
Cyclist	818	0.2015	Cyclist	578	0.0029
Complessivo	34506	0.4434	Complessivo	12085	0.3125

(a) Giorno

(b) Notte

Tabella 4.: Risultati della valutazione separata

Figura 21.: Esempio di predizioni, in verde la *ground truth* in rosso le predizioni.

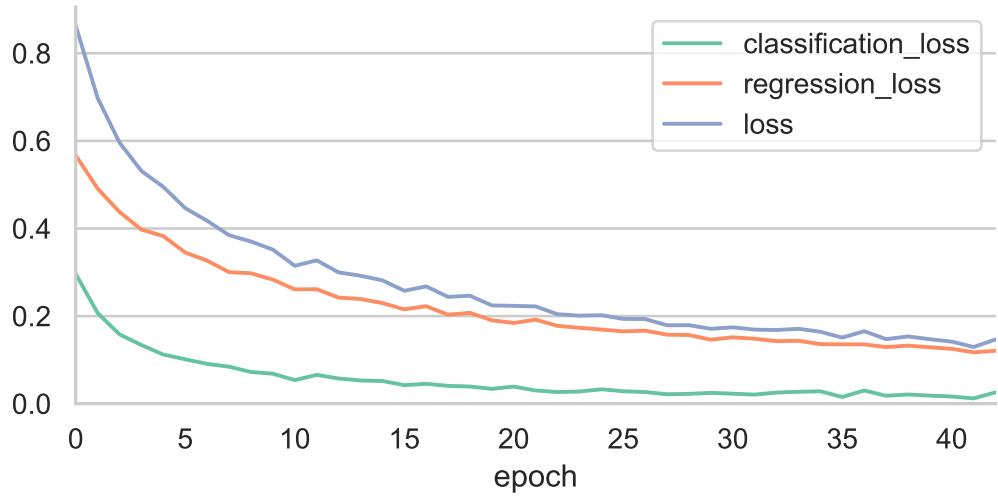


Figura 22.: Transfer learning da KAIST Visibile a KAIST Termico

	# Istanze	mAP
Person	53443	0.3663
Cyclist	1396	0.0392
Complessivo	54839	0.3580

Tabella 5.: Test complessivo delle performance dopo l'addestramento

come si può vedere in Figura 22 le varie loss scendono in maniera stabile e controllata. In maniera del tutto similare a quanto si può vedere in Figura 20.

I risultati della valutazione sul test set sono mostrati in maniera complessiva in Tabella 5. Il test è stato effettuato usando i label *person*, *cyclist* e *people*, quest'ultimi opportunamente rinominati in *person* per farli riconoscere correttamente a RetinaNet. La prima cosa che si nota è un calo delle performance rispetto al test sulle immagini visibili (Tabella 3). La spiegazione di questo calo delle performance si ottiene analizzando i risultati della valutazione separata effettuata sulla parte di dataset diurna e notturna, presente in Tabella 6. Guardando la Tabella 6a si osserva un drastico calo della mAP nella parte diurna, mentre in Tabella 6b abbiamo un leggero incremento. Il motivo è dovuto alla differenza di temperatura tra il pedone e quello che lo circonda. Di giorno, essendo la temperatura atmosferica più alta rispetto alla notte questa differenza di temperatura è conseguentemente più bassa, rendendo così il pedone, o chi per lui, più difficilmente riconoscibile. Di notte invece accade l'esatto contrario,

	# Istanze	mAP
Person	38802	0.3415
Cyclist	818	0.0387
Complessivo	39620	0.3353

	# Istanze	mAP
Person	14641	0.4440
Cyclist	578	0.0440
Complessivo	15219	0.4288

(a) Giorno

(b) Notte

Tabella 6.: Risultati della valutazione separata

la temperatura atmosferica è generalmente più bassa, quindi aumenta il delta di temperatura tra il pedone e l’ambiente circostante, rendendolo più facilmente riconoscibile. Considerazioni simili possono essere fatte sull’analisi svolta sulla parte di dataset RGB in quanto di notte essendoci poca luce si fatica di più a riconoscere un oggetto che non è illuminato, di giorno invece accade l’esatto contrario.

2.3 DATA AUGMENTATION

Nel Deep Learning avere un dataset ampio è un prerequisito fondamentale per far apprendere al proprio modello le caratteristiche che portano ad ottenere buoni risultati ed evitare il fenomeno dell’overfitting. Ad esempio, nel caso di Sezione 3.1.1 per ovvie questioni di tempo e praticità le annotazioni effettuate sul dataset sono molte poche, rendendolo così appena sufficiente ad i nostri scopi. Per ovviare a questo problema esiste la *Data Augmentation*. Con questo termine si fa riferimento a tutte quelle tecniche che permettono di creare nuovi dati tramite manipolazioni dei dati originali.

Restando nel dominio delle immagini, la più semplice tecnica di *Data Augmentation* è applicare casualmente alle immagini in input trasformazioni come possono essere la rotazione o l’inversione secondo un asse.

Recentemente l’interesse è virato più verso una *Data Augmentation* mirata al dataset su cui verrà applicata, ovvero con una serie di regole che comprendono il tipo di trasformazione da applicare, l’intensità e la probabilità con cui verrà applicata. Sono necessarie quindi nuove fasi di addestramento per apprendere queste regole che a tutti gli effetti possono essere considerate alla stessa stregua di iperparametri del nostro modello.

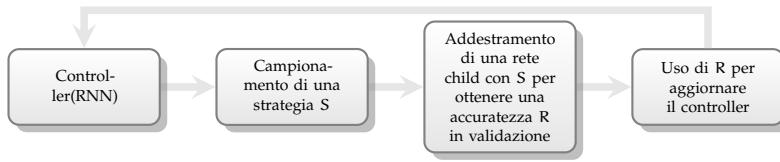


Figura 23.: Schema di funzionamento di AutoAugment

	Original	Sub-policy 1	Sub-policy 2	Sub-policy 3	Sub-policy 4	Sub-policy 5
Batch 1						
Batch 2						
Batch 3						
	ShearX, 0.9, 7 Invert, 0.2, 3	ShearY, 0.7, 6 Solarize, 0.4, 8	ShearX, 0.9, 4 AutoContrast, 0.8, 3	AutoContrast, 0.8, 3 Equalize, 0.6, 3	Invert, 0.9, 3 Equalize, 0.6, 3	ShearY, 0.8, 5 AutoContrast, 0.7, 3

Figura 24.: Esempio di policy con 5 sub-policy, immagine tratta da [7]

2.3.1 Auto Augment

Lo scopo di *AutoAugment* [7] è automatizzare il processo che porta ad ottenere delle regole di *Data Augmentation* per un determinato dataset obiettivo. La ricerca delle migliori policy per il dataset viene trattato alla pari di un problema di ricerca discreta, in Figura 23 è possibile vedere uno schema riassuntivo della struttura di *AutoAugment*. I componenti principali di *AutoAugment* sono due: un algoritmo di ricerca ed uno spazio di ricerca. L'algoritmo di ricerca, implementato tramite una RNN campiona una policy S. Questa policy al suo interno contiene le informazioni riguardanti quale trasformazione applicare, con quale probabilità applicarla, e con quale potenza. Un esempio di policy contenuta all'interno dello spazio di ricerca è possibile vederla in Figura 24. La subpolicy 1 definisce una specifica sequenza di operazioni, la prima è *Shear X* con probabilità di essere applicata pari a 0.9 e potenza di 7/10. La seconda operazione che verrà effettuata è *Invert* con probabilità 0.8.

In totale le operazioni nello spazio di ricerca sono 16, ed ognuna di esse ha un intervallo di potenza discretizzato con cui può essere applicata che varia da 1 a 10. In maniera del tutto similare vengono discretizzati anche la probabilità in un intervallo di 11 valori. Trovare quindi una subpolicy

è un problema di ricerca in uno spazio di dimensione $(16 \times 10 \times 11)^2$. Considerando che una policy contiene 5 subpolicy lo spazio di ricerca arriva a $(16 \times 10 \times 11)^{10}$, che è nell'ordine di grandezza di 10^{32} .

L'algoritmo di ricerca prende spunto dalle tecniche di apprendimento per rinforzo. Le sue componenti sono due, il controller RNN e l'algoritmo di addestramento, che in questo caso è il *Proximal Policy Optimization* [83]. L'addestramento del controller RNN è realizzato con un segnale di ricompensa che rappresenta il miglioramento in termini di generalizzazione della rete child addestrata con la policy S. Nel caso di Figura 23 la ricompensa è R, ovvero il risultato della valutazione sul validation set. Alla fine di questa fase di ricerca vengono trovate le migliori 5 policy, che vengono concatenate in un'unica grande policy contenente 25 subpolicy.

AUTOAUGMENT PER OBJECT DETECTION Fino ad ora abbiamo parlato di *AutoAugment* usata allo scopo di migliorare la classificazione di un'immagine. Il lavoro di tesi è però focalizzato sulla *Object Detection*. Gli stessi autori di [7] in [84] hanno evoluto questa tecnica per far sì che operi anche su questo tipo di compiti.

Come in precedenza il problema viene trattato come una ricerca in uno spazio discreto. Sempre come prima vengono definite policy come insiemi di K subpolicy, con queste ultime composte da N operazioni da applicare all'immagine in maniera sequenziale. In maniera del tutto analoga ogni operazione ha una probabilità con cui sarà applicata, a differenza della classificazione è stata resa più personalizzabile la scelta dei possibili valori discretizzati, chiamandoli M per la potenza e L per la probabilità.

Una importante differenza invece riguarda le operazioni che è possibile applicare all'immagine, in quanto in aggiunta alle classiche operazioni di trasformazione dell'intero frame si aggiungono ulteriori 6 operazioni applicabili solamente a livello di BB, raggiungendo così un totale di 22 operazioni. La dimensione dello spazio di ricerca di una subpolicy raggiunge quindi le dimensioni di $(22 \times L \times M)^2$, e più in particolare lo spazio di ricerca di una policy arriva a $(22 \times L \times M)^{10}$.

Gli autori in esperimenti preliminari hanno trovato che porre $L = M = 6$ è un buon *trade-off* tra performance e costi computazionali, in quanto si arriva ad uno spazio di ricerca nell'ordine di 10^{28} . Purtroppo però rimane comunque computazionalmente molto oneroso in quanto con 400 Tensor Processing Unit (TPU) sono state impiegate circa 48 ore di tempo computazionale.

Tuttavia sono state rilasciate delle policy apprese da dataset differenti

con cui sono stati eseguiti esperimenti che verranno successivamente descritti.

2.3.2 Rand Augment

Il più grande svantaggio di *AutoAugment* è l'elevato costo computazionale per una fase di ricerca separata su un task proxy. *RandAugment* [8] è un nuovo modo di realizzare *Data Augmentation* proposto dagli stessi autori di [7] e [84] che risolve questa problematica. La fase di ricerca di *AutoAugment* porta ad avere, nella sua versione orientata alla classificazione di immagini, più di 30 parametri, motivo per cui il lavoro qui descritto mira a ridurne drasticamente il numero fino addirittura a portarli a due soli. La selezione delle policy di augmentation viene infatti realizzata in maniera totalmente casuale, ciò vuol dire che se abbiamo K operazioni possibili la probabilità di selezionarne una è di $\frac{1}{K}$. Inoltre applicando N operazioni ad un'immagine si possono esprimere tutte le possibili policy come le N disposizioni con ripetizione di K operazioni, quindi K^N .

L'ultima variabile da prendere in considerazione è la forza con cui vengono applicate queste operazioni. Viene ripresa la stessa scala usata in [7], ovvero un intero che varia da 0 a 10. Al fine di ridurre il numero dei parametri gli autori sono intervenuti anche in questo definendo un singolo valore M per parametrizzare la potenza con cui vengono applicate le operazioni.

Riassumendo quindi gli iperparametri sono due: il numero N di operazioni da applicare e la forza M con cui verranno applicate. Essendo così pochi può essere un usato un classico algoritmo di *GridSearch* per ottimizzarli. Nel caso di questa tesi è stato usato un ottimizzatore bayesiano presente sulla piattaforma Comet.ml di cui non vengono però rilasciati dettagli tecnici o implementativi.

Risulta quindi un algoritmo molto semplificato rispetto a quello descritto in Sezione 2.3.1. Come è possibile vedere in Codice 2.1 sono solo poche righe di codice in cui vengono selezionate casualmente N operazioni tra quelle disponibili e vengono applicate successivamente con una forza pari a M .

Listing 2.1: Algoritmo di RandAugment in Python [8]

```
transforms = [
    'Identity', 'AutoContrast', 'Equalize',
    'Rotate', 'Solarize', 'Color', 'Posterize',
    'Contrast', 'Brightness', 'Sharpness',
```

```
'ShearX', 'ShearY', 'TranslateX', 'TranslateY']\n\ndef randaugment(N, M):\n    """Generate a set of distortions.\n\n    Args:\n        N: Number of augmentation transformations to apply\n            sequentially.\n        M: Magnitude for all the transformations.\n    """\n\n    sampled_ops = np.random.choice(transforms, N)\n    return [(op, M) for op in sampled_ops]
```

Un altro svantaggio di *AutoAugment* è che le policy vengono apprese su un sottoinsieme che può non rispecchiare le caratteristiche dell'intero dataset. In *RandAugment* essendo la fase di ottimizzazione molto più semplificata è possibile operare direttamente sull'intero dataset, portando così ad ottenere risultati potenzialmente migliori.

3

ESPERIMENTI

In questo capitolo verranno descritti ed analizzati gli esperimenti svolti durante lo sviluppo di questa tesi. La prima fase ha richiesto un'analisi dal punto di vista pratico dei dataset descritti in Sezione 2.1 seguita da una fase di preprocessing. Infine dopo aver reso i dataset usabili sono stati effettuati gli esperimenti usando varie tecniche allo scopo di incrementare sempre di più la Mean Average Precision.

3.1 ORGANIZZAZIONE DEI DATASET

I dataset utilizzati nello sviluppo del progetto di tesi sono stati descritti in Sezione 2.1, descriveremo ora gli stessi insiemi di dati ma dal punto di vista delle operazioni preliminari che sono state necessarie per renderli utilizzabili e conformi tra di loro in maniera da poter effettuare gli esperimenti.

3.1.1 *KAIST Multispectral Pedestrian Dataset*

La prima analisi effettuata su questo dataset è stata capire com'è stato suddiviso tra le varie cartelle. La situazione a cui ci si trova di fronte sono due cartelle, una contenente le immagini ed una contenente le annotazioni. All'interno di queste due cartelle la struttura è identica e sono presenti 12 sottocartelle contenenti a loro volta un numero variabile di video già suddivisi in immagini. Un'esempio di cosa contiene una delle due sottocartelle è possibile visualizzarlo in Figura 25. Il dataset in questione ha già una divisione predefinita tra train set e test set, ottenibile eseguendo uno script presente nella cartella del dataset.

La successiva operazione che è stata effettuata è la stesura di uno script per convertire questi suddetti file in un formato accettato dalla specifica implementazione di RetinaNet utilizzata, ovvero un CSV compo-

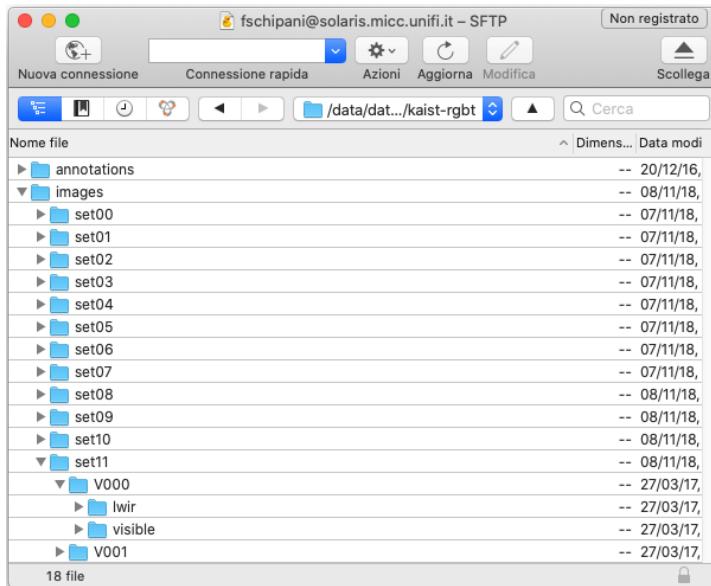


Figura 25.: Suddivisione del dataset KAIST MPD

Diurne	SET ₀₀	SET ₀₁	SET ₀₂	SET ₀₆	SET ₀₇	SET ₀₈
Notturne	SET ₀₃	SET ₀₄	SET ₀₅	SET ₀₉	SET ₁₀	SET ₁₁

Tabella 7.: Suddivisione giorno notte di KAIST

sto da colonne contenenti in ordine il path dell’immagine, le coordinate dell’eventuale e la classe dell’eventuale BB.

In questo dataset, come detto in Sezione 2.1.1, sono presenti le classi *person*, *people*, *cyclist* e *person?*. A scopo di test sono state create due varianti dei file CSV poco sopra descritti. La prima variante comprende solamente le BB che contengono *person* e *cyclist*, mentre la seconda variante, usata soprattutto in fase di test comprende anche la classe *people* opportunamente rinominata alla necessità in *person*.

ANNOTAZIONI DELLE AUTO A differenza del dataset di FLIR, descritto in Sezione 2.1.2, mancano le annotazioni delle autovetture. Per effettuare altri esperimenti, descritti nelle sezioni a venire, è stato deciso di realizzarle a mano su circa 1000 immagini.

Al fine di rispecchiare il più possibile la divisione originale tra giorno e notte del dataset è stata necessaria un’analisi più approfondita della struttura stessa, da cui è derivato che ci sono in numero molto simile immagini diurne e notturne. Per il dettaglio della suddivisione si può far riferimento a Tabella 7. Dopodiché sono state selezionate casualmente 200

immagini per ognuna delle cartelle SET00/V000, SET05/V000, SET06/V001, SET08/V000 e SET09/V000.

Tramite lo strumento VGG Image Annotator (VIA) [85, 86] messo a disposizione gratuitamente sono state annotate tutte le auto, furgoni e bus non eccessivamente occlusi. Infine sono state prese le immagini di SET00/V000 e SET05/V000 per formare un piccolo train set, mentre quelle di SET08/V000 e SET09/V000 sono state usate per il test set. Il validation set è stato fatto con il SET06/V001.

3.1.2 FLIR Thermal Starter Dataset

Il dataset di FLIR ha richiesto molto poco preprocessing in quanto è stato sufficiente scrivere uno script che convertisse le annotazioni fornite con il dataset in un formato usabile da RetinaNet. In Codice 3.1 è possibile vedere il suddetto script. L'unica accortezza usata è stata di selezionare solamente un sottoinsieme delle annotazioni, ovvero `people`, `bicycles` e `cars`. Questo è stato fatto per far coincidere il più possibile le annotazioni di questo dataset con quelle del dataset di Sezione 3.1.1.

Listing 3.1: Script di conversione per dataset FLIR

```
import json
import glob
import csv
import itertools
import progressbar
if __name__ == "__main__":
    annotations_path = '/data/datasets/FLIR_ADAS/FLIR_ADAS/training/Annotations/*.json'
    id_to_name = {
        '1': 'People',
        '2': 'Bicycles',
        '3': 'Cars',
        '18': 'Dogs',
        '91': 'Other'
    }
    tot = len(glob.glob(annotations_path))
    bar = progressbar.ProgressBar(maxval=tot, widgets=[progressbar.Bar('=', '[', ']'), ' ', progressbar.Percentage()])
    i = 0
    bar.start()
    with open('thermal_validation_KAIST_TRAIN_WCARS.csv', 'w') as csvfile:
        filewriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.QUOTE_MINIMAL)
        for file_name in glob.iglob(annotations_path):
            data = json.load(open(file_name))
            if len(data['annotation']) > 0:
                for i in range(len(data['annotation'])):
                    if data['annotation'][i]['category_id'] in ['1', '2', '3']:
                        x1, y1 = data['annotation'][i]['segmentation'][0][0], data['annotation'][i]['segmentation'][0][1]
                        x2, y2 = data['annotation'][i]['segmentation'][0][4], data['annotation'][i]['segmentation'][0][5]
                        filewriter.writerow([file_name.replace('Annotations', 'PreviewData')] + [x1, y1, x2, y2] + [id_to_name[data['annotation'][i]['category_id']]])
            else:
                filewriter.writerow([file_name, '', '', '', ''])
            i = i+1
            bar.update(i)
    bar.finish()
```

3.1.3 Video di Rete Ferroviaria Italiana

I video termici forniti per il progetto di ricerca della tesi sono stati girati da telecamere *Foshvision FS-UV535R104A Thermal Bi-spectrum* e *HikVision DS-2TD2866-25 Thermal Bi-spectrum Network Bullet Camera* sul circuito di test di RFI di San Donato a Bologna. Il formato in cui sono stati forniti è mp4, quindi per renderli usabili da RetinaNet è stato reso necessario l'utilizzo del software `ffmpeg` per estrarre i singoli frame.

Per realizzare una fase di fine tuning sulla rete, analogamente a quello fatto in Sezione 3.1.1, sono stati annotati manualmente degli operai a lavoro su un unico video. Il video in questione, della durata di circa 15 minuti può essere segmentato in tre parti. La prima, che arriva circa fino al minuto 8 non contiene alcun operaio o oggetto su cui è possibile realizzare annotazioni. La seconda parte, che inizia dal minuto 8 e si conclude quasi alla fine, vede degli operai che lavorano sulla linea ferroviaria. L'ultima parte è simile alla prima in quanto gli operai hanno finito il loro lavoro e non sono più nella scena ripresa dalla telecamera. Sempre come in Sezione 3.1.1 sono state selezionate casualmente delle immagini. Dalla prima parte del video sono state selezionate 200 immagini, dalla seconda 400 e dall'ultima che è la più corta 100. Le annotazioni anche in questo caso sono state realizzate con VIA, solamente sulla seconda parte di video e solamente con la classe person.

3.2 ESPERIMENTI INIZIALI SU IMMAGINI TERMICHE

Riprendendo il discorso di Sezione 2.2 si parlerà ora dei vari esperimenti effettuati con i vari dataset a disposizione.

TEST DEI PESI DI KAIST SU FLIR La prima di queste sperimentazioni è stata provare i risultati di Sezione 2.2.3 sul dataset di FLIR. Il test è stato effettuato provando diverse soglie di rilevazione che variano tra 0.3 a 0.8 con passo di 0.1. Il grafico di Figura 26 riassume i risultati. I risultati sono abbastanza deludenti, soprattutto sul fronte della rilevazione dei ciclisti. La motivazione principale è la carenza di esempi sui ciclisti in quanto ce ne sono molti pochi rispetto ad esempio i pedoni, un'altra motivazione degli scarsi risultati ottenuti è imputabile al diverso formato dell'annotazione in quanto su FLIR la BB viene posta solamente sul ciclista, mentre su KAIST MPD sul ciclista più il mezzo. I risultati migliori comunque si ottengono con una soglia pari a 0.3, in Figura 27 mostriamo alcuni esempi di immagini annotate e classificate da Retinanet, mentre in

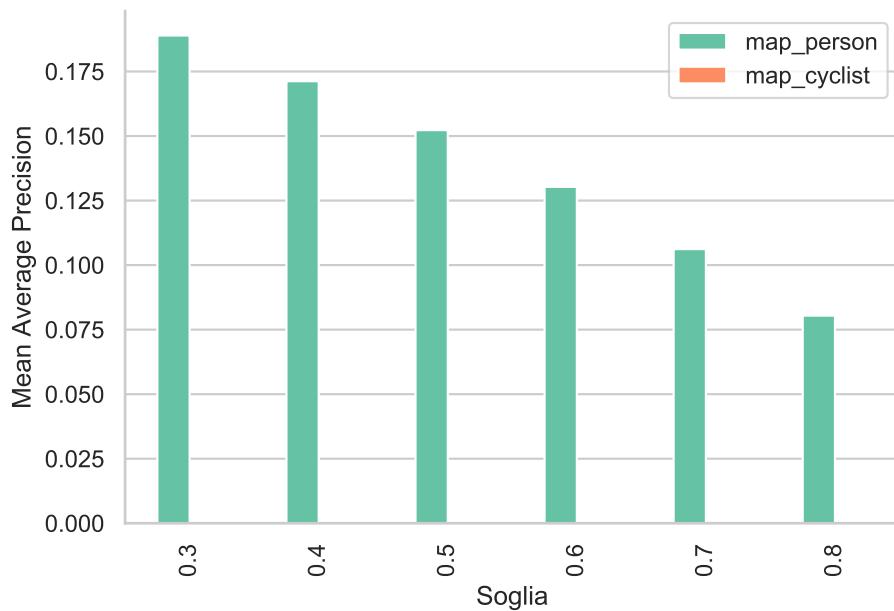


Figura 26.: MAP al variare del valore di soglia su FLIR

	Istanze	Map
Person	5779	0.1889
Cyclist	471	0.0001
Complessivo	6250	0.1747

Tabella 8.: Risultati del primo esperimento su FLIR

Tabella 8 sono presenti i valori più dettagliati rispetto al miglior risultato. La Figura 28 mostra la curva di precision-recall al variare della soglia, e si nota immediatamente una mancanza sia di precisione che di richiamo. Infatti se abbiamo maggior esempi classificati per una soglia più bassa otteniamo una scarsa precisione, situazione contraria si verifica invece quando si aumenta la soglia di rilevamento.

ADDESTRAMENTO SU FLIR PARTENDO DA COCO In questo esperimento si effettua una fase di addestramento di RetinaNet partendo dai pesi della rete preaddestrata su MS-COCO. Dare questo tipo di imprinting al nostro modello permette di effettuare un'operazione considerabile alla stessa stregua del Transfer Learning, e di ridurre quindi sensibilmente i tempi di addestramento. Il grafico della fase di addestramento è in Figura 29 e come è possibile vedere le varie loss sono scese fino ad

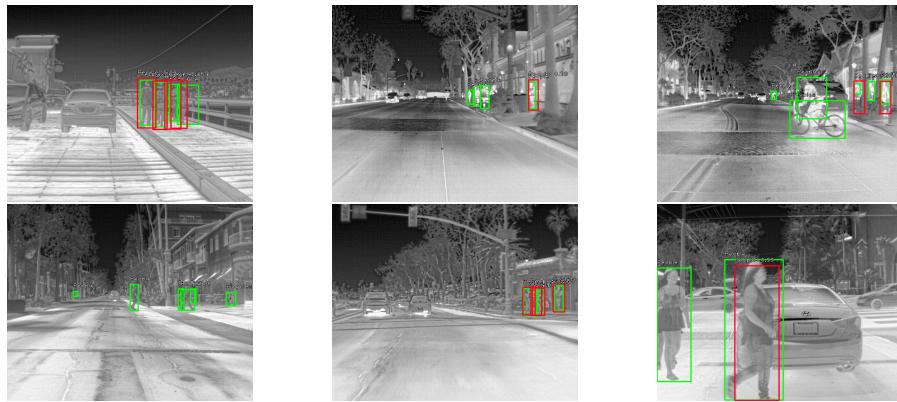


Figura 27.: Esempio di predizioni, in verde la *ground truth* in rosso le predizioni.

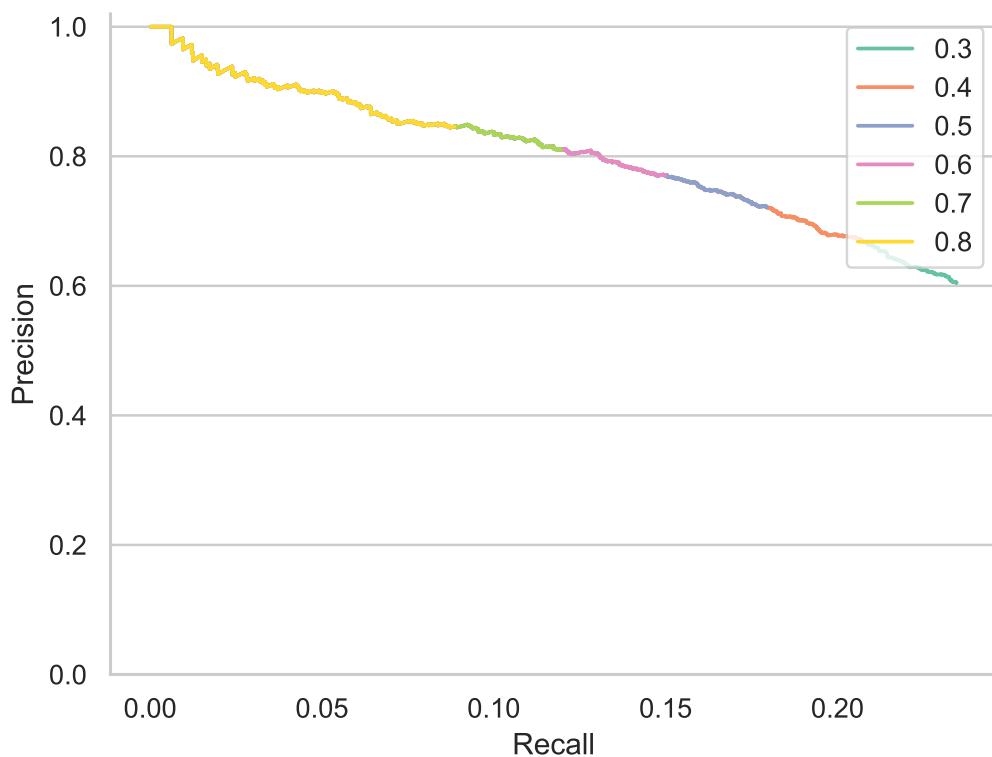


Figura 28.: Precision-Recall per classe person

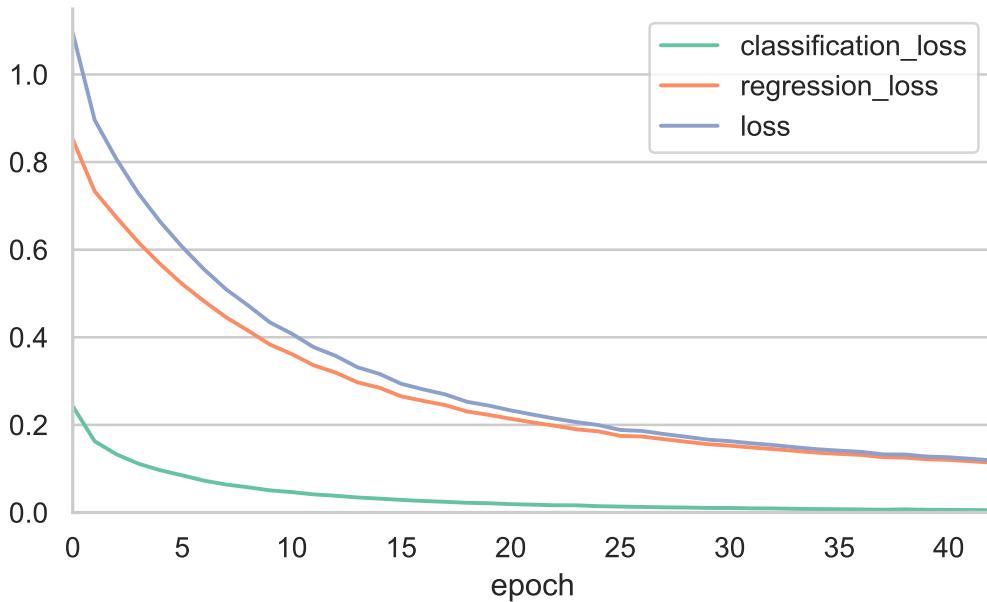


Figura 29.: Addestramento di RetinaNet su FLIR partendo dai pesi di COCO

	Istanze	Map
Person	5779	0.575442
Cyclist	471	0.769743
Complessivo	6250	0.672592

Tabella 9.: Risultati di FLIR da COCO con soglia 0.3

arrivare quasi a convergenza verso l'epoca 45. La discesa è più stabile rispetto a quella vista in Sezione 2.2.3 e questo è probabilmente dovuto alla migliore qualità di immagini di FLIR rispetto a KAIST MPD. La fase di test dei pesi risultati di questo modello sono stati testati sul dataset di FLIR ottenendo i risultati in Figura 30. Rispetto ai risultati mostrati in precedenza si ha un incremento notevole della mAP. Come prima si verifica anche lo stesso fenomeno per cui si ottiene maggior mAP con una soglia più bassa, ma a discapito della qualità delle rilevazioni. Si può osservare questo fenomeno guardando la curva di precision-recall in Figura 31. In Tabella 9 sono presenti i risultati migliori ottenuti in termini di mAP.

ADDESTRAMENTO DI FLIR PARTENDO DA KAIST L'idea dietro questo esperimento fonda le sue basi dietro al fallimento riguardante un test

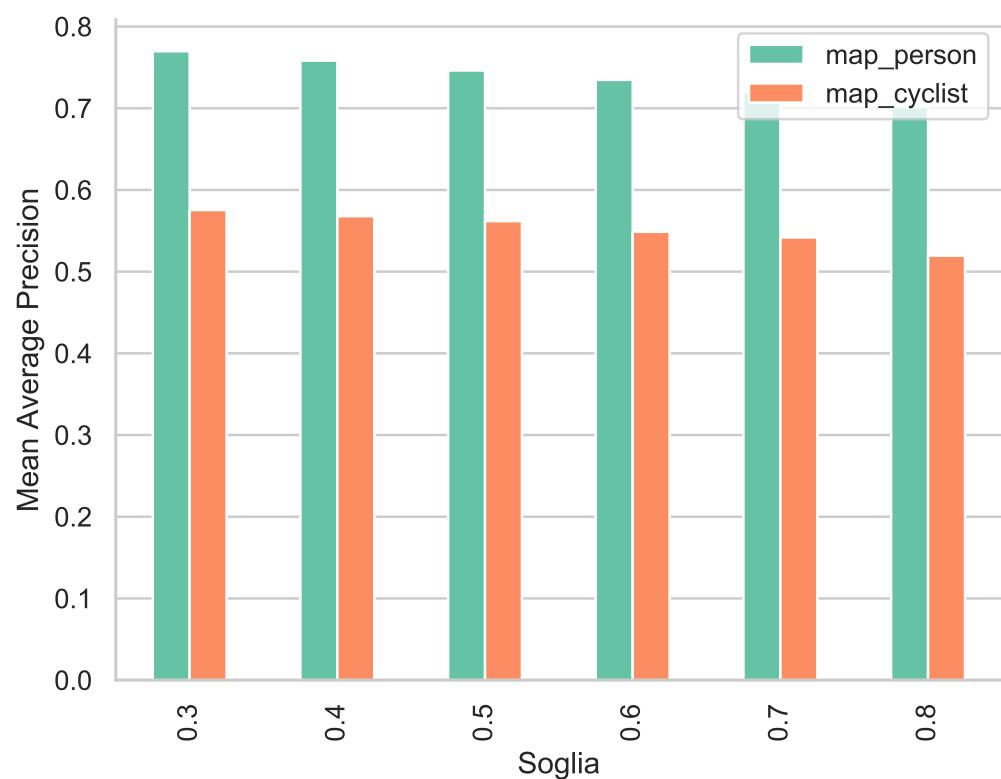


Figura 30.: mAP su FLIR addestrato dai pesi di COCO

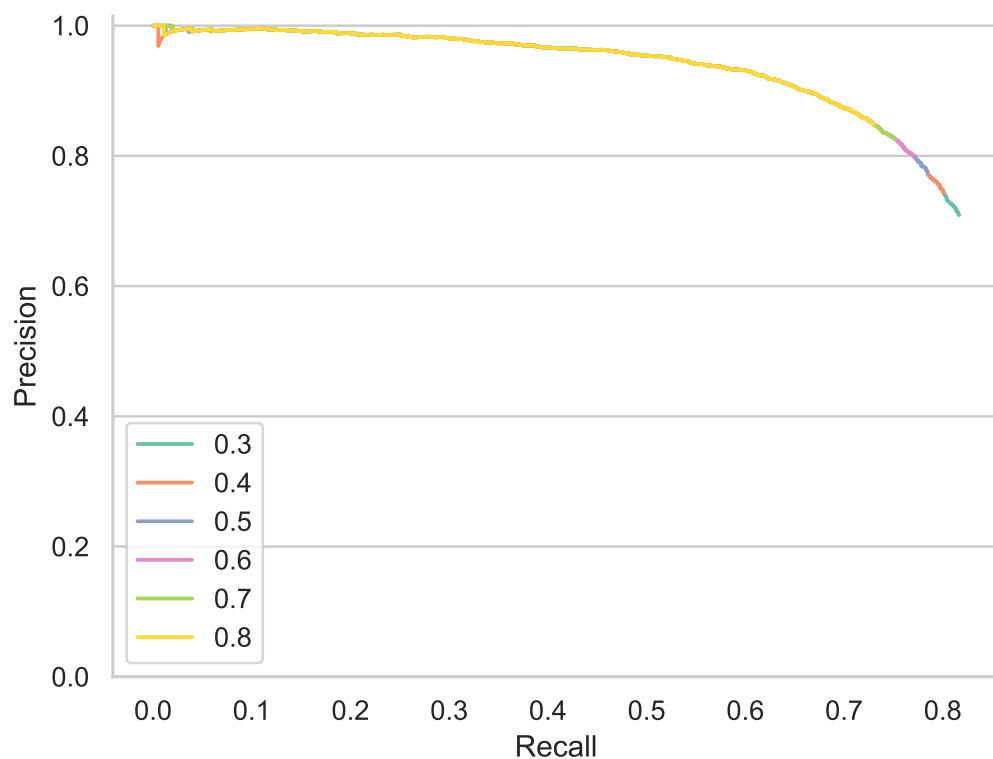


Figura 31.: Curva P/R di FLIR addestrato partendo da COCO (person)

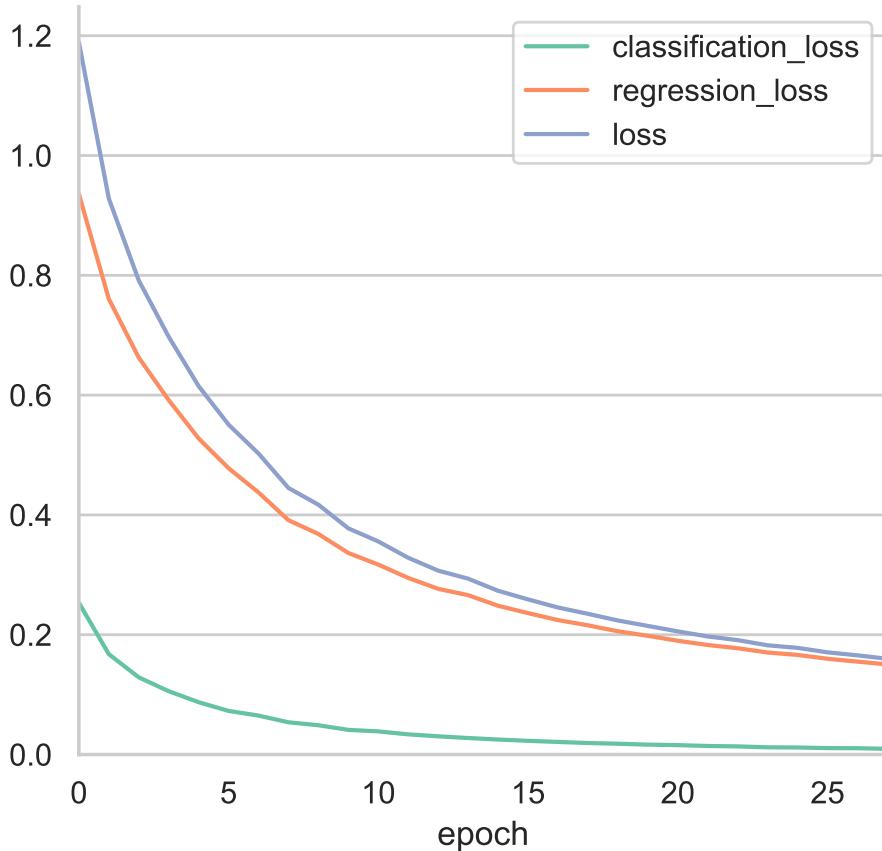


Figura 32.: Addestramento su FLIR partendo dai pesi di KAIST

precedente, ovvero quello in cui si testano i pesi di KAIST MPD sul dataset di FLIR. L’evoluzione delle misure di loss durante l’addestramento è visibile in Figura 32. La prima cosa che si nota è l’andamento molto simile rispetto all’esperimento precedente, successivamente si può notare anche come si arriva ai medesimi risultati, ma con qualche epoca di anticipo. Infatti, nonostante i valori di partenza delle loss sono sensibilmente più elevati si ha una discesa più repentina. Grafici analoghi ai precedenti è possibile vederli in Figura 33 e 34. L’andamento è comunque analogo al precedente, ma come si può vedere da Tabella 10 si ottiene un calo delle performance sotto il punto di vista della mAP.

TEST DEI PESI MIGLIORI SU KAIST Per ora i pesi migliori ottenuti in termini di mAP sulle persone è stato ottenuto addestrando RetinaNet sul dataset di FLIR partendo dai pesi della rete preaddestrata su MS-COCO. Proviamo ora a vedere cosa succede testando questi pesi sul dataset di

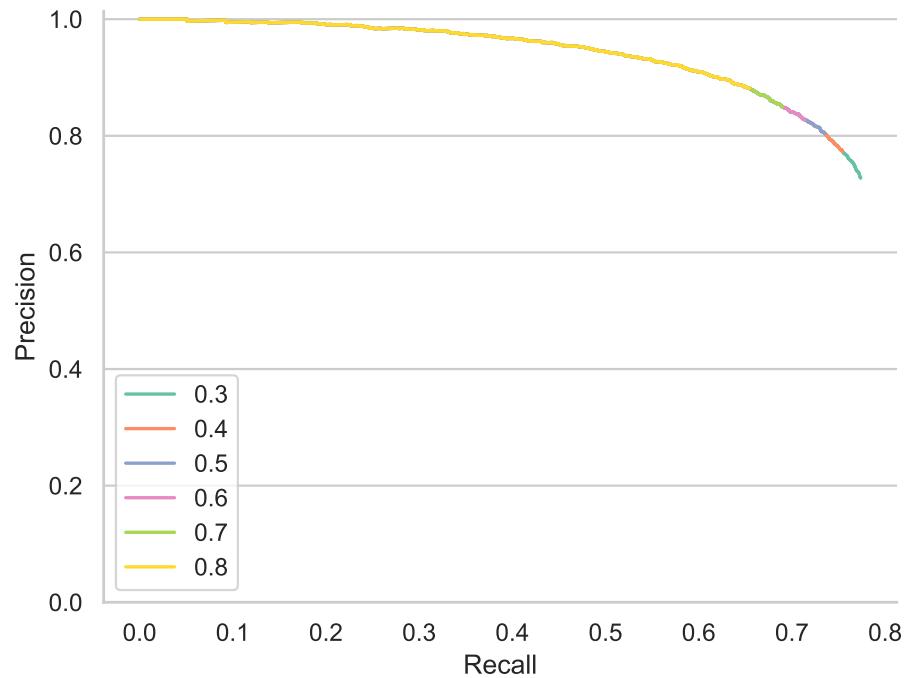


Figura 33.: Grafico P/R dell’addestramento su FLIR partendo dai pesi di KAIST (person)

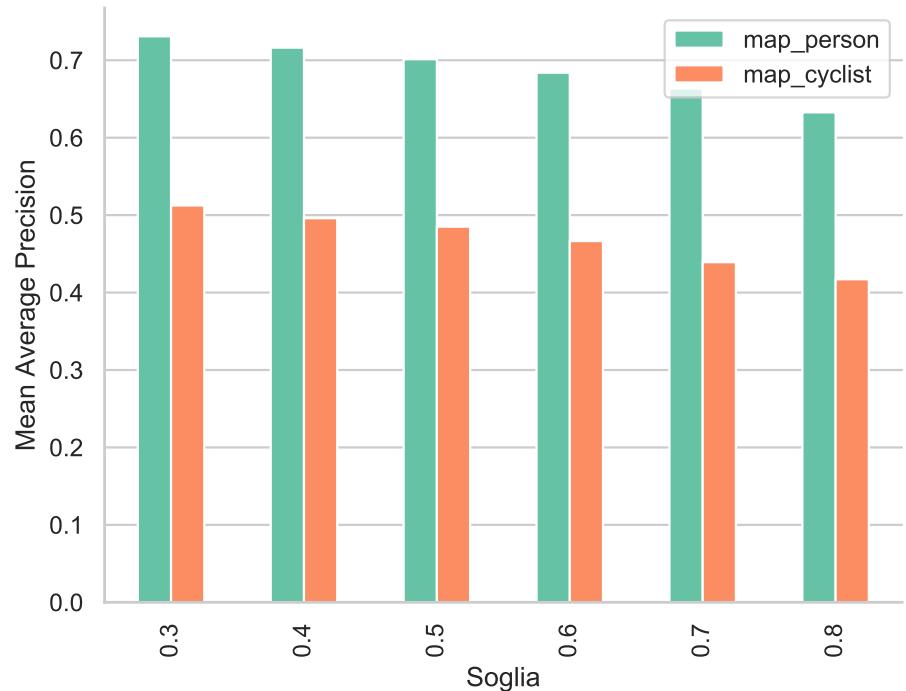


Figura 34.: Risultati dell’addestramento su FLIR partendo dai pesi di KAIST

	Istanze	Map
Person	5779	0.512462
Cyclist	471	0.730898
Complessivo	6250	0.621680

Tabella 10.: Risultati migliori dell'addestramento su FLIR partendo dai pesi di KAIST, soglia 0.3

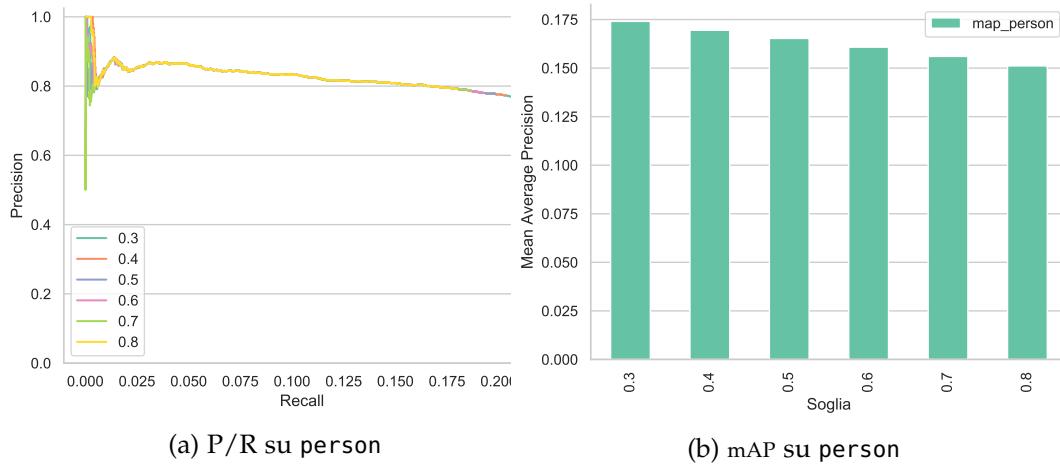


Figura 35.: Test su KAIST del risultato migliore su FLIR

KAIST MPD in un caso che è già stato visto essere favorevole alla detection tramite immagini termiche, ovvero quando le acquisizioni sono state effettuate di notte.

In Figura 35 è possibile visionare un sunto dei risultati ottenuti. Si può osservare in particolare nella parte 35b che la mAP ottenuta è molto bassa. Questa scarsa precisione in fase di inferenza porta a non avere grandi risultati all'atto pratico, basti vedere anche la Figura 35a per notare come la curva di Precision/Recall indichi una forte penuria nella Recall. Dati alla mano si può dire sicuramente che le rilevazioni effettuate sono corrette, ma in pratica ne vengono trovate molte poche rispetto al totale.

RILEVAZIONE DELLE AUTO La rilevazione delle auto su entrambi i dataset ha richiesto un lavoro di preprocessing sul dataset KAIST MPD descritto in precedenza nella Sezione 3.1.1. Il primo test effettuato è fissare una baseline data dall'addestramento su FLIR ed il successivo test sulla parte di dataset annotata manualmente di KAIST MPD cercando di vedere il comportamento con le autovetture. Il risultato è in Tabella 11.

	Istanze	Map
Person	755	0.3730
Cyclist	15	0
Cars	1088	0.5716
Complessivo	6250	0.4863

Tabella 11.: Baseline per la rilevazione di vetture su KAIST

	EP 18	EP 10	EP 09	EP 08	EP 05
Person (755)	0.4964	0.5050	0.5018	0.5137	0.5048
Cyclist (15)	0.0333	0.0444	0.0667	0.1000	0.0667
Cars (1088)	0.6767	0.6929	0.6854	0.6787	0.6870
Complessivo	0.5983	0.6113	0.6058	0.6070	0.6080

Tabella 12.: Risultati del fine tuning su kaist annotato manualmente

Il passo successivo è stata una fase di *fine tuning* di questi pesi sulla parte di train di KAIST MPD annotata manualmente con le vetture. Come si può vedere in Figura 36 l’addestramento ha avuto una durata relativamente corta in quanto con solamente 15 ore e 18 epochhe è arrivato al termine. Vista la repentina discesa delle loss e la scarsità di dati a disposizione è facile incorrere in un fenomeno di overfitting, ed infatti è quello che è accaduto. In Tabella 12 sono mostrati i risultati per alcune epochhe di questa fase di addestramento, in grassetto ci sono i risultati migliori per ogni categoria. Come si può vedere nell’epoca 18, ovvero l’ultima, non si raggiungono picchi di mAP, il che è un sintomo di scarsa capacità di generalizzazione. Riguardo la classe person e cyclist il risultato migliore si ha all’epoca 8. Invece riguardo la rilevazione di automobili si ha il risultato migliore all’epoca 10. Si può notare inoltre che sia riguardo le persone che le automobili c’è stato un incremento nelle prestazioni della detection (Tabella 13).

In particolare possiamo andare ad analizzare le performance ottenute all’epoca 8 in quanto risultano mediamente le migliori. In Figura 37 sono presenti le curve di Precision/Recall sulla classe person e cars, mentre in Figura 38 sono presenti i grafici di mAP al variare della soglia di rilevamento. Non è stata analizzata in maniera più dettagliata la classe cyclist in quanto gli esempi sono troppi pochi per ottenere risultati significativi.

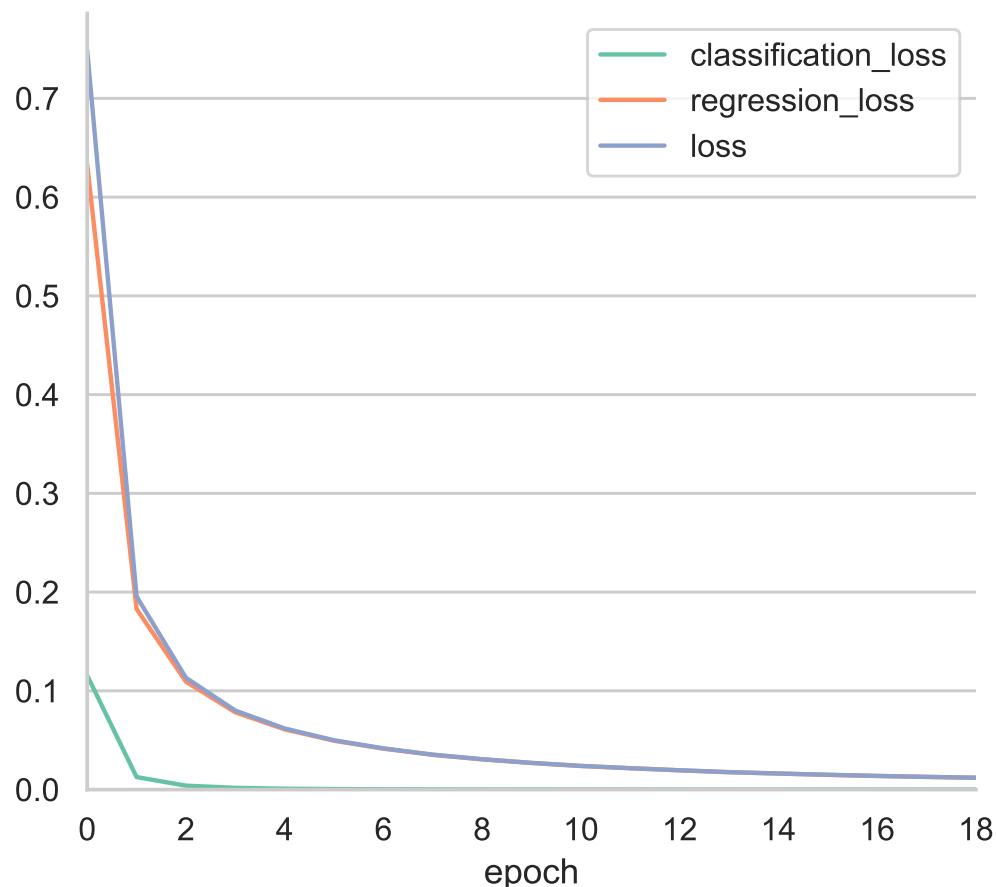


Figura 36.: Fine tuning di KAIST partendo da FLIR

	Post	Pre	Incremento
Person (755)	0.5137	0.3730	+37%
Cyclist (15)	0.1000	0.000	+
Cars (1088)	0.6929	0.5716	+21%
Complessivo	0.6113	0.4863	+25%

Tabella 13.: Incremento rispetto alla baseline dopo fine tuning

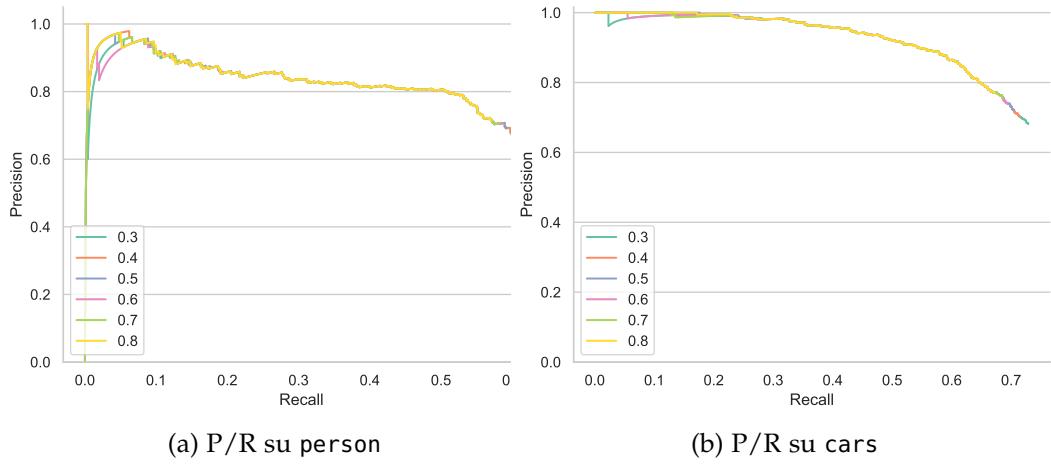


Figura 37.: P/R su KAIST all'epoca 8

Andando a confrontare le curve di P/R di Figura 37 si nota come sulle automobili si hanno in media rilevazioni più precise ed anche un richiamo maggiore, il che porta ad avere detection sulle vetture in generale migliori rispetto a quelle ottenute sui pedoni. Il motivo presumibilmente può essere imputato ad una maggior somiglianza tra l'aspetto delle vetture in immagini termiche rispetto a quello che possono avere dei pedoni.

3.3 DATA AUGMENTATION

Al fine di migliorare la detection degli esseri umani un ulteriore diramazione di esperimenti è stata svolta usando tecniche di Data Augmentation (Sezione 2.3). Le tecniche utilizzate in particolare sono AutoAugment, RandAugment ed infine delle immagini termiche generate da una Generative Adversarial Network (GAN).

AUTOAUGMENT AutoAugment per via dell'elevata richiesta computazionale, non alla portata dell'hardware utilizzato per lo sviluppo della tesi, non è stato sfruttato adeguatamente, bensì sono state usate policy derivanti dagli addestramenti effettuati su altri dataset. In Codice 3.2, 3.3 e 3.4 sono presenti le tre politiche di AutoAugment con cui sono stati effettuati gli esperimenti.

Listing 3.2: Policy Vo di AutoAugment

```
def policy_vo():
    policy = [
        [ ('TranslateX_BBox', 0.6, 4), ('Equalize', 0.8, 10)],
        [ ('TranslateY_Only_BBoxes', 0.2, 2), ('Cutout', 0.8, 8)],
        [ ('Sharpness', 0.0, 8), ('ShearX_BBox', 0.4, 0)],
```

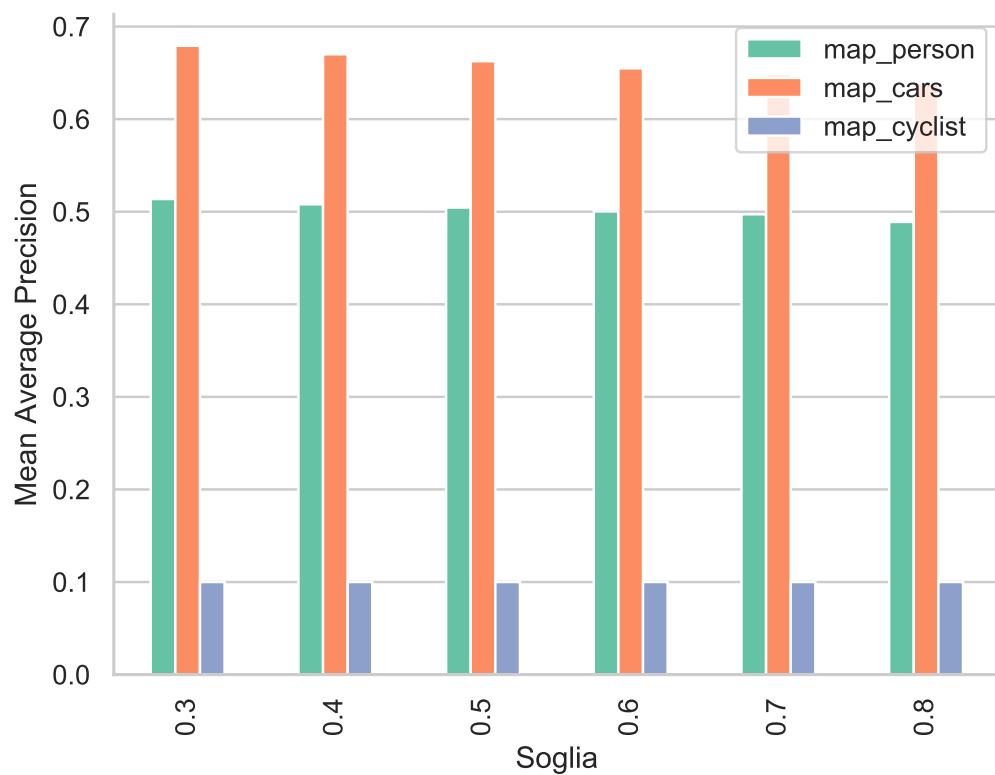


Figura 38.: Fine tuning di KAIST partendo da FLIR

```

    [ ('ShearY_BBox', 1.0, 2), ('TranslateY_Only_BBoxes', 0.6, 6)],
    [ ('Rotate_BBox', 0.6, 10), ('Color', 1.0, 6)],
]
return policy

```

La prima policy è la più semplice tra le tre, contiene al suo interno solamente cinque subpolicy formate a loro volta da due operazioni.

Listing 3.3: Policy V1 di AutoAugment

```

def policy_v1():
    policy = [
        [ ('TranslateX_BBox', 0.6, 4), ('Equalize', 0.8, 10)],
        [ ('TranslateY_Only_BBoxes', 0.2, 2), ('Cutout', 0.8, 8)],
        [ ('Sharpness', 0.0, 8), ('ShearX_BBox', 0.4, 0)],
        [ ('ShearY_BBox', 1.0, 2), ('TranslateY_Only_BBoxes', 0.6, 6)],
        [ ('Rotate_BBox', 0.6, 10), ('Color', 1.0, 6)],
        [ ('Color', 0.0, 0), ('ShearX_Only_BBoxes', 0.8, 4)],
        [ ('ShearY_Only_BBoxes', 0.8, 2), ('Flip_Only_BBoxes', 0.0, 10)],
        [ ('Equalize', 0.6, 10), ('TranslateX_BBox', 0.2, 2)],
        [ ('Color', 1.0, 10), ('TranslateY_Only_BBoxes', 0.4, 6)],
        [ ('Rotate_BBox', 0.8, 10), ('Contrast', 0.0, 10)],
        [ ('Cutout', 0.2, 2), ('Brightness', 0.8, 10)],
        [ ('Color', 1.0, 6), ('Equalize', 1.0, 2)],
        [ ('Cutout_Only_BBoxes', 0.4, 6), ('TranslateY_Only_BBoxes', 0.8, 2)],
        [ ('Color', 0.2, 8), ('Rotate_BBox', 0.8, 10)],
        [ ('Sharpness', 0.4, 4), ('TranslateY_Only_BBoxes', 0.0, 4)],
        [ ('Sharpness', 1.0, 4), ('SolarizeAdd', 0.4, 4)],
        [ ('Rotate_BBox', 1.0, 8), ('Sharpness', 0.2, 8)],
        [ ('ShearY_BBox', 0.6, 10), ('Equalize_Only_BBoxes', 0.6, 8)],
        [ ('ShearX_BBox', 0.2, 6), ('TranslateY_Only_BBoxes', 0.2, 10)],
        [ ('SolarizeAdd', 0.6, 8), ('Brightness', 0.8, 10)],
    ]
return policy

```

La seconda, a differenza della prima introduce un numero di subpolicy più alto, ma mantiene il numero di operazioni applicabili ad immagine.

Listing 3.4: Policy V2 di AutoAugment

```

def policy_v2():
    policy = [
        [ ('Color', 0.0, 6), ('Cutout', 0.6, 8), ('Sharpness', 0.4, 8)],
        [ ('Rotate_BBox', 0.4, 8), ('Sharpness', 0.4, 2),
         ('Rotate_BBox', 0.8, 10)],
        [ ('TranslateY_BBox', 1.0, 8), ('AutoContrast', 0.8, 2)],
        [ ('AutoContrast', 0.4, 6), ('ShearX_BBox', 0.8, 8),
         ('Brightness', 0.0, 10)],
        [ ('SolarizeAdd', 0.2, 6), ('Contrast', 0.0, 10),
         ('AutoContrast', 0.6, 0)],
        [ ('Cutout', 0.2, 0), ('Solarize', 0.8, 8), ('Color', 1.0, 4)],
        [ ('TranslateY_BBox', 0.0, 4), ('Equalize', 0.6, 8),
         ('Solarize', 0.0, 10)],
        [ ('TranslateY_BBox', 0.2, 2), ('ShearY_BBox', 0.8, 8),
         ('Rotate_BBox', 0.8, 8)],
        [ ('Cutout', 0.8, 8), ('Brightness', 0.8, 8), ('Cutout', 0.2, 2)],
        [ ('Color', 0.8, 4), ('TranslateY_BBox', 1.0, 6), ('Rotate_BBox', 0.6, 6)],
        [ ('Rotate_BBox', 0.6, 10), ('BBox_Cutout', 1.0, 4), ('Cutout', 0.2, 8)],
        [ ('Rotate_BBox', 0.0, 0), ('Equalize', 0.6, 6), ('ShearY_BBox', 0.6, 8)],
        [ ('Brightness', 0.8, 8), ('AutoContrast', 0.4, 2),
         ('Brightness', 0.2, 2)],
        [ ('TranslateY_BBox', 0.4, 8), ('Solarize', 0.4, 6),
         ('SolarizeAdd', 0.2, 10)],
        [ ('Contrast', 1.0, 10), ('SolarizeAdd', 0.2, 8), ('Equalize', 0.2, 4)],
    ]
return policy

```

La terza invece diminuisce leggermente il numero di subpolicy rispetto alla seconda, ma aumenta le operazioni applicabili ad immagine a 3.

Le seguenti politiche sono state usate per addestrare RetinaNet sulla parte di dataset di KAIST MPD con i label sulle vetture. Il motivo è legato al

Policy	Vo	Vo	Vo	V1	V1	V1	V1	V1	V1	V2	V2	V2	V2	V2	V2
Epoca	5	8	11	5	10	15	20	30	33	05	10	15	20	30	37
Person (755)	0.4808	0.4850	0.4982	0.5157	0.5005	0.4896	0.4876	0.4975	0.3873	0.5241	0.5032	0.5007	0.4959	0.5064	0.5064
Cyclist (15)	0.0067	0.0167	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Cars (1088)	0.7195	0.6942	0.6792	0.6805	0.6728	0.6605	0.6703	0.6693	0.5603	0.6506	0.6564	0.6505	0.6254	0.6264	0.6265
Complessivo	0.6167	0.6037	0.6002	0.6080	0.5973	0.5857	0.5906	0.5941	0.4855	0.5940	0.5888	0.5844	0.5677	0.5726	0.5726

Tabella 14.: Risultati complessivi di AutoAugment su KAIST

	Prima di AA	Dopo AA	Variazione
Person (755)	0.5137	0.5241	+2%
Cyclist (15)	0.1000	0.0167	-0.4%
Cars (1088)	0.6929	0.7195	+3.8%
Complessivo	0.6113	0.6167	+0.8%

Tabella 15.: Incremento rispetto alla baseline dopo fine tuning

fatto che essendo le annotazioni poco numerose si prova ad aumentarne virtualmente il loro numero variandone l'aspetto tramite trasformazioni sull'immagine stessa.

Effettuando tre diverse fasi di addestramento ognuna con una policy diversa si ottengono in fase di inferenza i risultati riassunti in Tabella 14. Rispetto ai risultati ottenuti in precedenza si ha un miglioramento nella rilevazione delle automobili usando la policy Vo all'epoca 5, mentre per i pedoni si ha un miglioramento nella rilevazione usando la politica V2 sempre all'epoca 5. Le variazioni tra i risultati migliori ottenuti in precedenza e quelli migliori ottenuti con AutoAugment sono riassunti in Tabella 15.

Un ulteriore esperimento effettuato con AutoAugment è relativo al miglioramento della rilevazione dei pedoni. In pratica è stata usata la politica che ha fatto ottenere un miglioramento nella rilevazione dei pedoni per effettuare un training sul dataset di FLIR. Lo scopo era di ottenerne miglior capacità di generalizzazione e quindi passare da un dataset all'altro con risultati migliori. In Tabella 16 sono presenti i risultati dell'esperimento appena descritto, mentre in Figura 39 è presente il grafico riguardante la progressione delle Loss durante la fase di addestramento.

L'ultimo esperimento effettuato con AutoAugment parte dai pesi di quello precedentemente descritto (Figura 39, Tabella 16) realizzando una fase di *fine tuning* sul dataset di KAIST. In Tabella 17 sono presenti i risultati di questa ultima fase, e come si può vedere non si ottiene alcun miglioramento.

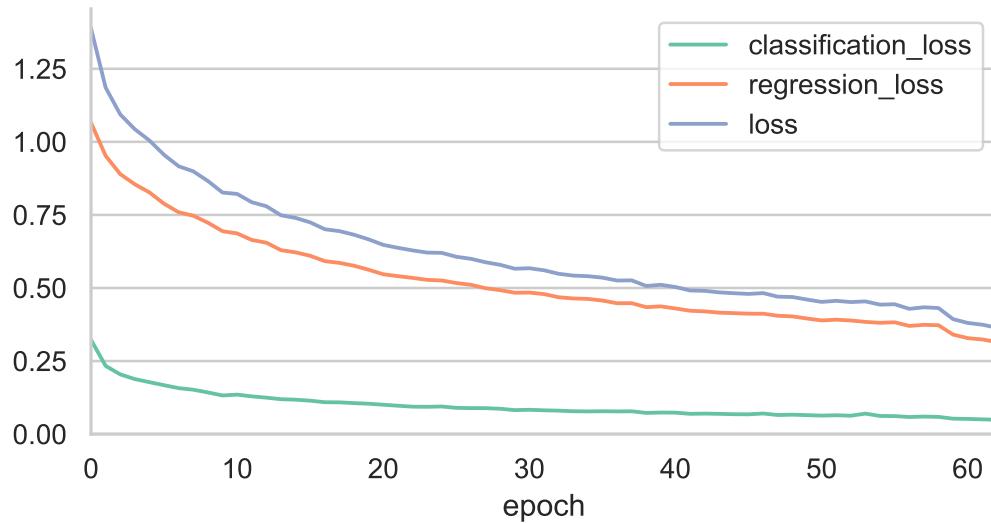


Figura 39.: Training su FLIR con policy di AutoAugment V2

	EP 05	EP 20	EP 20	EP 50	EP 63
Person (755)	0.3342	0.2516	0.3619	0.3270	0.3531

Tabella 16.: Test di transfer learning da FLIR addestrato con AA V2 a KAIST (soglia 0.3)

	EP 05	EP 10	EP 15	EP 20	EP 25	EP 30	EP 35	EP 40	EP 44
Person (755)	0.5061	0.4899	0.4913	0.4959	0.4822	0.4847	0.5035	0.4859	0.4911
Cyclist (15)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Cars (1088)	0.5250	0.5395	0.5360	0.5383	0.5462	0.5427	0.5454	0.5387	0.5399
Complessivo	0.5131	0.5150	0.5135	0.5167	0.5158	0.5148	0.5239	0.5129	0.5157

Tabella 17.: Test dopo ulteriore fine tuning su KAIST

RANDAUGMENT RandAugment, come precedentemente descritto in Sezione 2.3.2, riduce notevolmente il costo computazionale rispetto ad AutoAugment per far sì che si adatti al dataset su cui sarà applicato.

In questo caso ci si hanno solo due parametri N ed M con il primo che indica il numero di operazioni da applicare, ed il secondo la potenza con cui applicarle. Ci si riconduce in breve ad un problema di ottimizzazione di iperparametri risolvibile anche con un algoritmo GridSearch. In questa tesi però è stata usata la piattaforma Comet.ml che implementa una serie di algoritmi per realizzare ottimizzazione. L'algoritmo scelto è stato quello che loro definiscono il migliore, un algoritmo di tipo *bayesiano* su cui non vengono rilasciati ulteriori dettagli.

Il target dell'ottimizzatore è la minimizzazione della mAP moltiplicata per -1 , in pratica quindi è la massimizzazione della metrica. L'intero processo ha avuto una durata di circa 130 ore consecutive in quanto l'ambiente è stato impostato per eseguire una fase di training di 7 epoch seguita da un test per la valutazione delle performance, partendo dai pesi di RetinaNet preaddestrata su MS-COCO. Il tutto è stato realizzato sulla parte di dataset di KAIST MPD con le annotazioni delle automobili. Un sommario dell'ottimizzazione è in Figura 40 e come si può vedere i risultati migliori si ottengono ponendo M con valori compresi tra 26 e 30 e con N uguale a 3.

Purtroppo però non ha portato a miglioramenti probabilmente perché sette epochhe non sono sufficienti partendo da pesi generici come quelli di MS-COCO. Si ottiene una mAP sulle persone pari a 0.4769, mentre per le vetture si arriva a 0.6646.

GAN

3.4 ESPERIMENTI SU VIDEO DI RFI

I video termici di RFI forniti sono tre, e su tutti e tre il task è la rilevazione di persone. Inizialmente non essendo annotati l'unica operazione possibile è stata dare in pasto a RetinaNet i frame del video per realizzare inferenza usando i pesi migliori per la rilevazione di persone, ovvero quelli ottenuti tramite l'addestramento su KAIST MPD tramite la terza policy all'epoca 5 (Tabella 14).

I risultati iniziali, pur non potendo calcore metriche appaiono da una valutazione puramente soggettiva promettenti, in quanto le rilevazioni sembrano abbastanza precise.

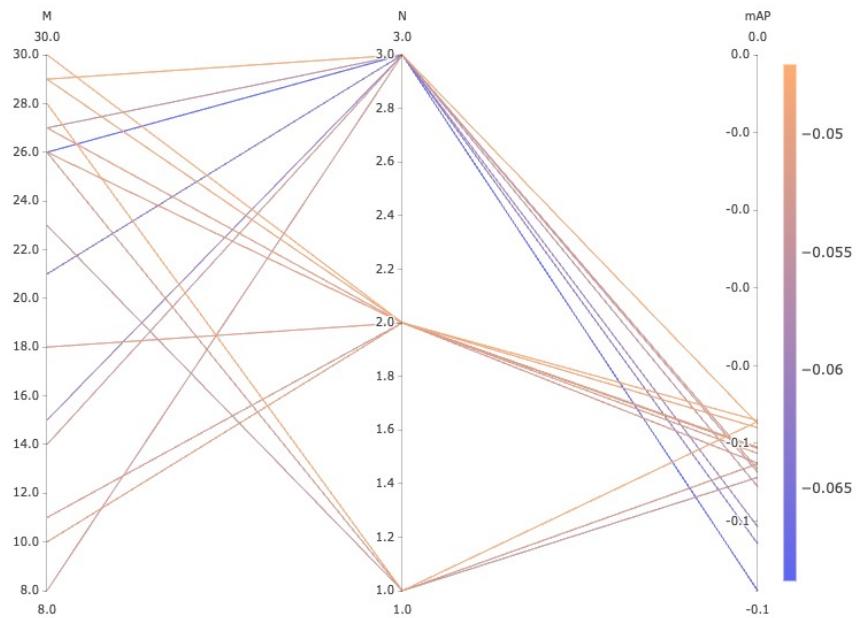


Figura 40.: mAP al variare degli iperparametri

I primi test sono stati effettuati tenendo una soglia di rilevazione pari a 0.30, questo ha portato ad una rilevazione sempre accurata degli operai, ma anche a molte rilevazioni false, come ad esempio quelle mostrate in Figura 41.

Questo problema è facilmente risolvibile aumentando la soglia di rilevamento, infatti portandola a 0.8 si rimuovono la maggior parte dei falsi positivi (Frame in Figura 42).

Un altro problema che rimane però è la rilevazione dei lavoratori in posizioni non usuali, ad esempio quando sono chinati. In Figura 42b vi è un'esempio in quanto l'operaio sembra stia raccogliendo qualcosa per terra e non è stato rilevato. Per risolverlo sono stati annotati alcuni frame di quel video per poi effettuare test su altri due video a nostra disposizione.

3.5 IOU SUL TEMPO

Allo scopo di migliorare precision e recall è stato sviluppato un algoritmo per eliminare rilevazioni fasulle e validare quelle effettivamente presenti. L'algoritmo in questione è in Algoritmo 1, mentre l'implementazione in

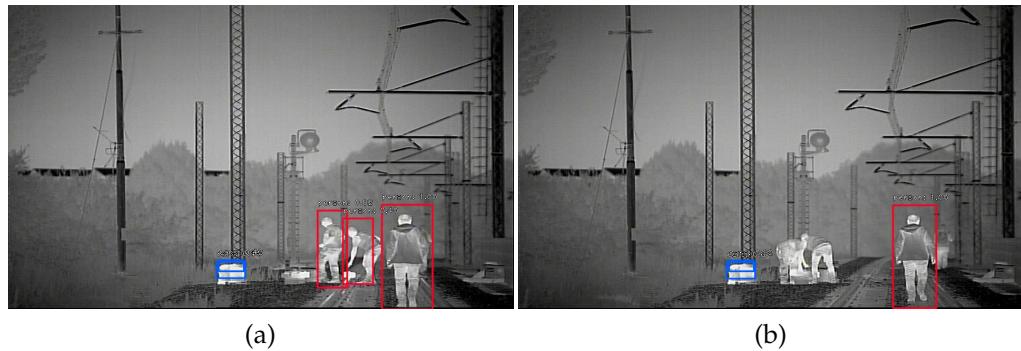


Figura 41.: Rilevazioni su video RFI con soglia 0.30



Figura 42.: Rilevazioni su video RFI con soglia 0.80

Python è reperibile in Appendice A.

Algorithm 1: Algoritmo di IoU calcolata su diversi frame

```

Input : Detection di N immagini, Threshold
Output: Detection di N immagini
Data: detections, T
Result: detections'
max_detections ← max_detections_in_image(detections) ;
R_tree ← new_r_tree();
for d ∈ (detections – max_detections) do
    | R_tree.insert(d);
end
iou_values ← empty_array(len(max_detections));
for di ∈ max_detections do
    | intersected_dets ← R_tree.find_intersection(di);
    | for dj ∈ intersected_dets do
        | | iou_values[di] += compute_iou(di, dj);
    | end
    | iou_values[di] /= len(intersected_dets);
end
detections' ← empty_array(len(detections));
for di ∈ max_detections do
    | if iou_values[di] > T then
        | | detections'.insert(di);
    | end
end
return detections'
```

L'input dell'algoritmo sono le rilevazioni effettuate su N immagini ed una soglia T. Il primo passo effettuato è la ricerca dell'immagine contenente più rilevazioni in quanto è quella intuitivamente più soggetta ad avere qualche BB trovata erroneamente durante la fase di inferenza. Una volta trovata si inseriscono le detection di tutte le altre immagini all'interno di una struttura che ci permetta di effettuare ricerche su coordinate spaziali in maniera efficiente, ovvero un albero R. Dopodiché, per ogni immagine che contiene più BB, si calcola un valore di IoU rispetto a tutte le BB delle altre immagini che si intersecano con essa. La IoU è un valore compreso tra 0 e 1, quindi si normalizza usando il numero di BB che intersecano. Dopo questa fase avremo quindi un array di lunghezza pari al massimo numero di rilevazioni su un'immagine che contiene valori di IoU. Nell'ultima parte dell'algoritmo si confronta ogni valore di

questo array per vedere se è superiore alla soglia T passata in input: se lo è la BB corrispondente verrà inserita come detection *valida* per ogni immagine nel batch, altrimenti viene scartata.

4

CONCLUSIONI

Appendici

A

CODICE DI IOU OVER TIME

Listing A.1: Algoritmo di IoU over time in Python

```
def compute_intersection_over_union(box1, box2):
    x1 = max(box1[0], box2[0])
    y1 = max(box1[1], box2[1])
    x2 = min(box1[2], box2[2])
    y2 = min(box1[3], box2[3])
    intersection_area = max(0, x2 - x1 + 1) * max(0, y2 - y1 + 1)
    box_1_area = (box1[2] - box1[0] + 1) * (box1[3] - box1[1] + 1)
    box_2_area = (box2[2] - box2[0] + 1) * (box2[3] - box2[1] + 1)
    intersection_over_union = intersection_area / float(box_1_area + box_2_area - intersection_area)
    return intersection_over_union

def calc_bbox_size(bbox):
    """
    Get the size of bbox in input
    # Arguments
        bbox      : coordinates of bbox (x_min, y_min, x_max, y_max).
    # Returns
        size of bbox
    """
    return (bbox[2]-bbox[0])*(bbox[3]-bbox[1])

def iou_evaluation(detections, threshold):
    index_max_detections = np.argmax(list(map(lambda x: len(x), detections)))
    id_tree = 0
    idx = index.Index(interleaved=True)
    iou_values = [0]*len(detections[index_max_detections])
    for_debug = np.concatenate(np.delete(detections, index_max_detections, axis = 0))
    for detection in for_debug:
        idx.insert(id_tree, detection[0:4], obj = detection[4:6])
        id_tree = id_tree + 1
    for i in range(len(detections[index_max_detections])):
        intersection_bbox = idx.intersection(detections[index_max_detections][i][0:4], objects=True)
        num_elem = 0
        for item in intersection_bbox:
            num_elem = i+1
            iou_values[i] += compute_intersection_over_union(item.bbox, detections[index_max_detections][i][0:4])
        if num_elem is not 0:
            iou_values[i] /= num_elem
    iou_values = np.divide(iou_values, len(detections)-1)
    bbox = []
    labels = []
    scores = []
    for det, iou in zip(detections[index_max_detections], iou_values):
        if iou > threshold:
            intersection_bbox = idx.intersection(det[0:4], objects=True)
            gen = list(intersection_bbox)
            max_bbox_index = np.argmax(list(map(lambda x: calc_bbox_size(x), list(map(lambda x: x.bbox, gen)))))
            bbox.append(gen[max_bbox_index].bbox)
            scores.append(max(list(map(lambda x: x.object[0], gen))))
            labels.append(gen[max_bbox_index].object[1])
    return np.array(bbox), np.array(scores), np.array(labels, dtype=int)
```


B

IMPLEMENTAZIONE DI RANDAUGMENT E AUTOAUGMENT SU RETINANET

Implementare RandAugment e AutoAugment su RetinaNet usato durante lo sviluppo del lavoro di tesi ha richiesto del lavoro di reverse engineering. Il punto di partenza per vedere come lavorava questa implementazione è stato lo script di train. La parte interessante ai fini della scrittura di una forma di Data Augmentation era come venissero lette e passate alla rete le immagini, in quanto in mezzo a questo processo andava piazzato un modulo che permetesse di effettuare operazioni sull'input della rete.

Le immagini vengono passate a RetinaNet tramite un oggetto, completamente configurabile, che funge da generatore. Il generatore viene creato attraverso diverse opzioni che lo andranno a configurare secondo le esigenze. Tra queste è presente anche una rudimentale forma di data augmentation che applica casualmente un'operazione selezionata da un insieme molto limitato.

Inizialmente è stato implementato AutoAugment, di cui era già disponibile un'implementazione parziale reperibile a questo link. Con poche modifiche, dovute perlopiù al passaggio da Tensorflow 1.1x a Tensorflow 2.0 si è arrivati a questa implementazione, dove tramite la chiamata della funzione `distort_image_with_autoaugment` si riesce ad applicare una policy all'immagine.

Le modifiche più evidenti invece hanno riguardato il generatore. L'input di questa versione di RetinaNet prevede vari formati, motivo per cui gli autori hanno deciso di implementare i generatori in maniera Object Oriented. Abbiamo quindi una superclasse `Generator` e le varie sottoclassi `CocoGenerator`, `CSVGenerator`, `KittiGenerator`, `OpenImagesGenerator` e `PascalVocGenerator` ognuna corrispondente ad i vari formati di input accettati da questa implementazione. L'operazione di modifica di un'immagine non è specifica del formato di input, ma dev'essere comune a tutte. Quindi l'intervento è stato circoscritto alla superclasse `Generator`. Seguendo quanto stato fatto dagli autori per implementare la prima

forma rudimentale di DataAugmentation sono stati implementate due funzioni del tutto simili alle controparti originali.

Listing B.1: Funzioni per applicare AutoAugment

```

def auto_augment_group_entry(self, image, annotations):
    """
    Randomly auto-augment image and annotation.
    """
    if self.auto_augment is not None:
        image_width = image.shape[1]
        image_height = image.shape[0]
        if annotations['bboxes'].shape[0] is 0:
            return image, annotations
        normalized_annotations = np.zeros(annotations['bboxes'].shape)
        normalized_annotations[:, 0] = annotations['bboxes'][:, 0] / image_width
        normalized_annotations[:, 2] = annotations['bboxes'][:, 2] / image_width
        normalized_annotations[:, 1] = annotations['bboxes'][:, 1] / image_height
        normalized_annotations[:, 3] = annotations['bboxes'][:, 3] / image_height
        normalized_annotations[:, [0, 1]] = normalized_annotations[:, [1, 0]]
        normalized_annotations[:, [3, 2]] = normalized_annotations[:, [2, 3]]
        normalized_annotations = tf.compat.v2.convert_to_tensor(normalized_annotations, dtype=tf.float32)
        image = tf.compat.v2.convert_to_tensor(image, dtype=tf.float32)
        augmented_img, augmented_annotation = distort_image_with_autoaugment(
            image, normalized_annotations, self.auto_augment)
        augmented_annotation = augmented_annotation.numpy()
        augmented_annotation[:, [0, 1]] = augmented_annotation[:, [1, 0]]
        augmented_annotation[:, [3, 2]] = augmented_annotation[:, [2, 3]]
        augmented_annotation[:, 0] = augmented_annotation[:, 0] * image_width
        augmented_annotation[:, 2] = augmented_annotation[:, 2] * image_width
        augmented_annotation[:, 1] = augmented_annotation[:, 1] * image_height
        augmented_annotation[:, 3] = augmented_annotation[:, 3] * image_height
        augmented_img = augmented_img.numpy()
        new_annotations = {
            'labels': annotations['labels'],
            'bboxes': augmented_annotation
        }
        return augmented_img, new_annotations
    return image, annotations

def auto_augment_group(self, image_group, annotations_group):
    """
    Apply AutoAugment policy to each image and its annotations.
    """
    assert(len(image_group) == len(annotations_group))
    for index in range(len(image_group)):
        # transform a single group entry
        image_group[index], annotations_group[index] = self.auto_augment_group_entry(
            image_group[index],
            annotations_group[index])
    return image_group, annotations_group

```

La prima delle due funzioni in Codice B.1 applica la trasformazione ad una singola immagine. Per via del formato di input differente è stata necessaria una piccola fase di preprocessing di dati dove si normalizzavano tra 0 e 1 le posizioni delle BB e si invertivano le coordinate. La seconda funzione richiama la prima su gruppi di immagini. Successivamente l'ultima modifica effettuata è stata intervenire sulla funzione `compute_input_output` della classe Generator. Questa funzione, in base agli argomenti dati in input alla creazione dell'oggetto, applica le dovute trasformazioni. In codice B.2 è possibile vedere come è stata aggiunta la chiamata necessaria ad AutoAugment.

Listing B.2: Funzione `compute_input_output` del generatore

```

def compute_input_output(self, group):
    """
    Compute inputs and target outputs for the network.
    """

```

```

# load images and annotations
image_group      = self.load_image_group(group)
annotations_group = self.load_annotations_group(group)
# check validity of annotations
image_group, annotations_group = self.filter_annotations(image_group, annotations_group, group)

# randomly apply visual effect
image_group, annotations_group = self.random_visual_effect_group(image_group, annotations_group)

# randomly transform data
image_group, annotations_group = self.random_transform_group(image_group, annotations_group)

# apply auto augment
image_group, annotations_group = self.auto_augment_group(image_group, annotations_group)

# perform preprocessing steps
image_group, annotations_group = self.preprocess_group(image_group, annotations_group)

# compute network inputs
inputs = self.compute_inputs(image_group)

# compute network targets
targets = self.compute_targets(image_group, annotations_group)

return inputs, targets

```

Per RandAugment la questione è stata un po' più tediante. In Tensorflow 2.0 una libreria, fondamentale per il funzionamento di queste operazioni, è stata eliminata e sostituita da una libreria installabile a parte chiamata Tensorflow Addons.

Usandola con le policy predefinite di AutoAugment questa libreria non causava problemi, ma una particolare sequenza di trasformazioni applicate all'immagine generata casualmente da RandAugment portava molte volte ad una terminazione inaspettata del programma. Per risolvere questo incidente di percorso è necessario applicare un flag in alcune operazioni della libreria Tensorflow Addons e ricompilerla con gli stessi toolkit NVidia usati per compilare Tensorflow, pena il fallimento nell'import della libreria stessa. Avendo fallito nel compito di compilare la libreria ho deciso di fare un nuovo ambiente di Anaconda con Tensorflow 1.14 ed una vecchia versione di RetinaNet. Questa via apparentemente ha portato ad una soluzione funzionante, ma il tutto era incredibilmente lento. La colpa è da inputare all'utilizzo improprio di Tensorflow, le conversioni tra tensori Numpy e tensori portavano via un sacco di tempo. Questo comportamento però in Tensorflow 2.0 non si osserva in quanto di default è abilitata la Eager Execution, cosa che non è possibile fare con Tensorflow 1.14 in quanto richiederebbe modifiche piuttosto pesanti a RetinaNet.

La soluzione è arrivata da un utente di GitHub che ha implementato tutte le operazioni di AutoAugment (e conseguentemente RandAugment) usando Numpy e Python Image Library (PIL). La repository contenente questo codice è reperibile al seguente link. È stata quindi aggiunta una funzione che implementa la trasformazione dell'immagine, visibile in Codice B.3. Seguendo l'articolo di riferimento di RandAugment è stato

sufficiente selezionare N operazioni in maniera casuale ed eventualmente applicarle con una probabilità p casuale con una forza M. Successivamente, in maniera del tutto analoga ad AutoAugment, è bastato definire le opportune funzioni nella classe Generator.

Listing B.3: Funzione per implementare RandAugment

```
def distort_image_with_rand_augment(image, bboxes, N, M):
    """Applies randaugment policy to input image.
    Paper: https://arxiv.org/abs/1909.13719

    Args:
        'image': 'Tensor' of shape [height, width, 3] representing an image.
        'N': integer, Number of transformation to apply to an image.
        'M': integer, shared Magnitude for all augmentation operations.

    Returns:
        A tuple containing the augmented versions of 'image' and 'bboxes'.
    """
    augmentation_hparams = {
        "cutout_max_pad_fraction": 0.75,
        "cutout_bbox_replace_with_mean": False,
        "cutout_const": 100,
        "translate_const": 250,
        "cutout_bbox_const": 50,
        "translate_bbox_const": 120
    }
    replace_value = [128] * 3
    tf.compat.v1.logging.info('Using RandAug.')
    available_ops = ['AutoContrast', 'Equalize', 'Solarize', 'SolarizeAdd', 'Contrast', 'Brightness',
                     'Sharpness', 'Cutout', 'BBox_Cutout', 'Rotate_BBox', 'TranslateX_BBox', 'TranslateY_BBox',
                     'ShearX_BBox', 'ShearY_BBox', 'Rotate_Only_BBoxes', 'ShearX_Only_BBoxes', 'ShearY_Only_BBoxes',
                     'TranslateX_Only_BBoxes', 'TranslateY_Only_BBoxes', 'Flip_Only_BBoxes', 'Solarize_Only_BBoxes',
                     'Equalize_Only_BBoxes', 'Cutout_Only_BBoxes']
    #available_ops = ['BBox_Cutout', 'TranslateY_BBox']
    for layer_num in range(N):
        op_to_select = np.random.randint(0, len(available_ops), size=1)
        random_magnitude = float(M)
        for (i, op_name) in enumerate(available_ops):
            prob = np.random.uniform(0.2, 0.8, 1)
            if i == op_to_select:
                func = NAME_TO_FUNC[op_name]
                args = level_to_arg(augmentation_hparams)[op_name](random_magnitude)
                if 'prob' in inspect.getargspec(func)[0]:
                    args = tuple([prob] + list(args))
                if 'replace' in inspect.getargspec(func)[0]:
                    assert 'replace' == inspect.getargspec(func)[0][-1]
                    args = tuple(list(args) + [replace_value])
                if 'bboxes' not in inspect.getargspec(func)[0]:
                    func = bbox_wrapper(func)
                image, bboxes = (lambda selected_func=func, selected_args=args: selected_func(
                    image, bboxes, *selected_args))(func, args)
            else:
                image, bboxes = image, bboxes
    return image, bboxes
```

ACRONIMI

mAP Mean Average Precision

KAIST MPD KAIST Multispectral Pedestrian Dataset

GPU Graphics Processing Unit

KAIST Korea Advanced Institute of Science and Technology

FOV Field of View

BB Bounding Box

RNN Recurrent Neural Network

TPU Tensor Processing Unit

CNN Convolutional Neural Network

FCN Fully Convolutional Network

RNN Recurrent Neural Network

NMS Non-Maximum Suppression

HNM Hard Negative Mining

YOLO You Only Look Once

VOC Pascal Visual Object Classes

ILSVRC ImageNet Large Scale Visual Recognition Challenge

MS-COCO Microsoft - Common Object in COntext

OID Open Images Detection

VJ Viola Jones

HOG Histogram of Oriented Gradients

DPM Deformable Part-based Model

- ROI** Region of Interest
RPN Region Proposal Network
SVM Support Vector Machines
IoU Intersection over Union
SPPNet Spatial Pyramid Pooling Networks
SSD Single-Shot Detector
DSSD Deconvolutional Single-Shot Detector
FPN Feature Pyramid Network
MLFPN Multi-Level Feature Pyramid Network
FFM Feature Fusion Module
TUM Thinned U-shape module
SFAM Scale-wise Feature Aggregation Module
RFI Rete Ferroviaria Italiana
VIA VGG Image Annotator
GAN Generative Adversarial Network

BIBLIOGRAFIA

- [1] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *CoRR*, abs/1905.05055, 2019. (Cited on pages 3, 11, 12, 13, 15, 17, 18, and 19.)
- [2] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. (Cited on pages 3, 24, and 25.)
- [3] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. A survey of deep learning-based object detection. *CoRR*, abs/1907.09408, 2019. (Cited on pages 3, 11, and 29.)
- [4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015. (Cited on pages 3, 30, 31, and 32.)
- [5] 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. (Cited on pages 3, 14, 17, 30, 33, and 34.)
- [6] Qijie Zhao, Tao Sheng, Yongtao Wang, Zhi Tang, Ying Chen, Ling Cai, and Haibin Ling. M2det: A single-shot object detector based on multi-level feature pyramid network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9259–9266, 2019. (Cited on pages 3, 36, and 37.)
- [7] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. *CoRR*, abs/1805.09501, 2018. (Cited on pages 4, 57, 58, and 59.)
- [8] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical data augmentation with no separate search. *CoRR*, abs/1909.13719, 2019. (Cited on pages 5 and 59.)

- [9] Irving Biederman. Recognition-by-components: a theory of human image understanding. *Psychological review*, 94(2):115, 1987. (Cited on page 12.)
- [10] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *2008 IEEE conference on computer vision and pattern recognition*, pages 1–8. IEEE, 2008. (Cited on pages 12 and 27.)
- [11] Martin A Fischler and Robert A Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on computers*, (1):67–92, 1973. (Cited on page 12.)
- [12] Bastian Leibe, Aleš Leonardis, and Bernt Schiele. Robust object detection with interleaved categorization and segmentation. *International journal of computer vision*, 77(1-3):259–289, 2008. (Cited on page 12.)
- [13] Dariu M Gavrila and Vasanth Philomin. Real-time object detection for "smart" vehicles. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 1, pages 87–93. IEEE, 1999. (Cited on page 12.)
- [14] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (4):509–522, 2002. (Cited on page 12.)
- [15] Bo Wu and Ramakant Nevatia. Detection of multiple, partially occluded humans in a single image by bayesian combination of edgelet part detectors. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 1, pages 90–97. IEEE, 2005. (Cited on page 12.)
- [16] Matthew Turk and Alex Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991. (Cited on page 12.)
- [17] Alex Pentland, Baback Moghaddam, Thad Starner, et al. View-based and modular eigenspaces for face recognition. 1994. (Cited on page 12.)
- [18] Régis Vaillant, Christophe Monrocq, and Yann Le Cun. Original approach for the localisation of objects in images. *IEE Proceedings-*

Vision, Image and Signal Processing, 141(4):245–250, 1994. (Cited on page 13.)

- [19] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. (Cited on page 13.)
- [20] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. (Cited on page 13.)
- [21] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014. (Cited on page 13.)
- [22] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2009. (Cited on pages 14, 17, and 27.)
- [23] Tomasz Malisiewicz, Abhinav Gupta, and Alexei Efros. Ensemble of exemplar-svms for object detection and beyond. 2011. (Cited on page 14.)
- [24] Tomasz Malisiewicz. *Exemplar-based representations for object detection, association and beyond*. Carnegie Mellon University, 2011. (Cited on page 14.)
- [25] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. What is an object? In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 73–80. IEEE, 2010. (Cited on page 14.)
- [26] Jan Hosang, Rodrigo Benenson, Piotr Dollár, and Bernt Schiele. What makes for effective detection proposals? *IEEE transactions on pattern analysis and machine intelligence*, 38(4):814–830, 2015. (Cited on page 14.)
- [27] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. (Cited on pages 14, 16, 30, and 32.)

- [28] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. (Cited on pages 14, 15, 30, and 35.)
- [29] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017. (Cited on pages 14 and 34.)
- [30] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. (Cited on pages 15, 30, 33, and 40.)
- [31] Shifeng Zhang, Longyin Wen, Xiao Bian, Zhen Lei, and Stan Z Li. Single-shot refinement neural network for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4203–4212, 2018. (Cited on page 15.)
- [32] Zhaowei Cai, Quanfu Fan, Rogerio S Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *european conference on computer vision*, pages 354–370. Springer, 2016. (Cited on page 15.)
- [33] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, Sep. 2010. (Cited on page 15.)
- [34] Antonio Torralba and Pawan Sinha. Detecting faces in impoverished images. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB, 2001. (Cited on page 16.)
- [35] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. 2005. (Cited on pages 16, 20, and 27.)
- [36] Santosh K Divvala, Derek Hoiem, James H Hays, Alexei A Efros, and Martial Hebert. An empirical study of context in object detection. In *2009 IEEE Conference on computer vision and Pattern Recognition*, pages 1271–1278. IEEE, 2009. (Cited on page 17.)

- [37] Zeming Li, Yilun Chen, Gang Yu, and Yangdong Deng. R-fcn++: Towards accurate region-based fully convolutional networks for object detection. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. (Cited on page 17.)
- [38] Sean Bell, C Lawrence Zitnick, Kavita Bala, and Ross Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2874–2883, 2016. (Cited on page 17.)
- [39] Jianan Li, Yunchao Wei, Xiaodan Liang, Jian Dong, Tingfa Xu, Jia-shi Feng, and Shuicheng Yan. Attentive contexts for object detection. *IEEE Transactions on Multimedia*, 19(5):944–954, 2016. (Cited on page 17.)
- [40] Chaitanya Desai, Deva Ramanan, and Charless C Fowlkes. Discriminative models for multi-class object layout. *International journal of computer vision*, 95(1):1–12, 2011. (Cited on pages 17 and 18.)
- [41] Zheng Song, Qiang Chen, Zhongyang Huang, Yang Hua, and Shui-cheng Yan. Contextualizing object detection and classification. In *CVPR 2011*, pages 1585–1592. IEEE, 2011. (Cited on page 17.)
- [42] Xinlei Chen and Abhinav Gupta. Spatial memory for context reasoning in object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4086–4096, 2017. (Cited on page 17.)
- [43] Han Hu, Jiayuan Gu, Zheng Zhang, Jifeng Dai, and Yichen Wei. Relation networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3588–3597, 2018. (Cited on page 17.)
- [44] Saurabh Gupta, Bharath Hariharan, and Jitendra Malik. Exploring person context and local scene context for object detection. *arXiv preprint arXiv:1511.08177*, 2015. (Cited on page 17.)
- [45] Yong Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen. Structure inference net: Object detection using scene-level context and instance-level relationships. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6985–6994, 2018. (Cited on page 17.)

- [46] Paul Viola, Michael Jones, et al. Rapid object detection using a boosted cascade of simple features. *CVPR (1)*, 1(511-518):3, 2001. (Cited on pages 18, 19, and 26.)
- [47] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013. (Cited on page 18.)
- [48] Rasmus Rothe, Matthieu Guillaumin, and Luc Van Gool. Non-maximum suppression for object detection by passing messages between windows. In *Asian conference on computer vision*, pages 290–306. Springer, 2014. (Cited on page 18.)
- [49] Damian Mrowca, Marcus Rohrbach, Judy Hoffman, Ronghang Hu, Kate Saenko, and Trevor Darrell. Spatial semantic regularisation for large scale object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2003–2011, 2015. (Cited on page 18.)
- [50] Li Wan, David Eigen, and Rob Fergus. End-to-end integration of a convolution network, deformable parts model and non-maximum suppression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 851–859, 2015. (Cited on page 18.)
- [51] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. Learning non-maximum suppression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4507–4515, 2017. (Cited on page 18.)
- [52] Paul Henderson and Vittorio Ferrari. End-to-end training of object class detectors for mean average precision. In *Asian Conference on Computer Vision*, pages 198–213. Springer, 2016. (Cited on page 18.)
- [53] Constantine P Papageorgiou, Michael Oren, and Tomaso Poggio. A general framework for object detection. In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pages 555–562. IEEE, 1998. (Cited on page 19.)
- [54] Henry A Rowley, Shumeet Baluja, and Takeo Kanade. Human face detection in visual scenes. In *Advances in Neural Information Processing Systems*, pages 875–881, 1996. (Cited on page 19.)

- [55] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017. (Cited on pages 20 and 37.)
- [56] Constantine Papageorgiou and Tomaso Poggio. A trainable system for object detection. *International journal of computer vision*, 38(1):15–33, 2000. (Cited on page 20.)
- [57] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. (Cited on page 21.)
- [58] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111(1):98–136, 2015. (Cited on page 21.)
- [59] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: A benchmark. 2009. (Cited on page 22.)
- [60] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012. (Cited on page 22.)
- [61] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015. (Cited on page 23.)
- [62] Shanshan Zhang, Rodrigo Benenson, and Bernt Schiele. Citypersons: A diverse dataset for pedestrian detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3213–3221, 2017. (Cited on page 23.)
- [63] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016. (Cited on page 23.)

- [64] Ivan Krasin, Tom Duerig, Neil Alldrin, Vittorio Ferrari, Sami Abu-El-Haija, Alina Kuznetsova, Hassan Rom, Jasper Uijlings, Stefan Popov, Andreas Veit, et al. Openimages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://github.com/openimages*, 2:3, 2017. (Cited on page 25.)
- [65] Markus Braun, Sebastian Krebs, Fabian Flohr, and Dariu M Gavrila. The eurocity persons dataset: A novel benchmark for object detection. *arXiv preprint arXiv:1805.07193*, 2018. (Cited on page 26.)
- [66] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004. (Cited on pages 26 and 30.)
- [67] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999. (Cited on page 27.)
- [68] Pedro F Felzenszwalb, Ross B Girshick, and David McAllester. Cascade object detection with deformable part models. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2241–2248. IEEE, 2010. (Cited on page 27.)
- [69] Ross B Girshick, Pedro F Felzenszwalb, and David A Mcallester. Object detection with grammar models. In *Advances in Neural Information Processing Systems*, pages 442–450, 2011. (Cited on page 27.)
- [70] Ross Brook Girshick. *From rigid templates to grammars: Object detection with structured models*. Citeseer, 2012. (Cited on page 27.)
- [71] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. (Cited on pages 30 and 31.)
- [72] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014. (Cited on page 30.)
- [73] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. (Cited on page 34.)

- [74] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. (Cited on page 35.)
- [75] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017. (Cited on page 36.)
- [76] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. (Cited on page 40.)
- [77] Soonmin Hwang, Jaesik Park, Namil Kim, Yukyung Choi, and In So Kweon. Multispectral pedestrian detection: Benchmark dataset and baseline. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1037–1045. IEEE Computer Society, 2015. (Cited on page 43.)
- [78] Free flir thermal dataset for algorithm training. <https://www.flir.com/oem/adas/adas-dataset-form/>. Accessed: 2019-11-20. (Cited on page 43.)
- [79] L. Bienkowski, C. Homma, K. Eisler, and Christian Boller. Hybrid camera and real-view thermography for nondestructive evaluation. 01 2012. (Cited on page 44.)
- [80] Piotr Dollár. Piotr’s Computer Vision Matlab Toolbox (PMT). <https://github.com/pdollar/toolbox>. (Cited on page 45.)
- [81] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>. (Cited on page 48.)
- [82] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010. (Cited on page 48.)
- [83] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. (Cited on page 58.)
- [84] Barret Zoph, Ekin D. Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V. Le. Learning data augmentation strategies for

- object detection. *CoRR*, abs/1906.11172, 2019. (Cited on pages 58 and 59.)
- [85] Abhishek Dutta and Andrew Zisserman. The VIA annotation software for images, audio and video. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, New York, NY, USA, 2019. ACM. (Cited on page 63.)
- [86] A. Dutta, A. Gupta, and A. Zissermann. VGG image annotator (VIA). <http://www.robots.ox.ac.uk/~vgg/software/via/>, 2016. Version: 2.0.8, Accessed: 2020-01-15. (Cited on page 63.)