UNIVERSITÀ
DEGLI STUDI
FIRENZE

Scuola di Scienze Matematiche, Fisiche e Naturali

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

TITOLO IN ITALIANO

TITLE IN ENGLISH

FEDERICO SCHIPANI

Relatore: Prof. *Marco Bertini*

Anno Accademico 2018-2019 (spero)

INDICE

Introduzione 6
1 Object Detection 9
1.1 Tipologie di detector 9
1.1.1 One Stage Detector 9
1.1.2 Two Stage Detector 9
1.2 RetinaNet 9
1.2.1 Focal Loss 9
2 SISTEMA 11
2.1 Dataset 11
2.1.1 KAIST Multispectral Pedestrian Dataset 11
2.1.2 FLIR Thermal Starter Dataset 14
2.1.3 Video di RFI 16
2.2 Addestramento iniziale di RetinaNet 16
2.2.1 Transfer Learning 16
2.2.2 Addestramento sulle immagini RGB 19
2.2.3 Passaggio alle immagini termiche su KAIST 22
2.2.4 ALTRA ROBA DA SCRIVERE 24
2.3 Data Augmentation 24
2.3.1 Auto Augment 24
2.3.2 Rand Augment 26
3 Esperimenti 29
3.1 Organizzazione dei dataset 29
3.2 Esperimenti iniziali 29
3.3 Data Augmentation 29
4 Conclusioni 31
A Implementazione di RandAugment su RetinaNet 33
Acronimi 34

ELENCO DELLE TABELLE

Tabella 1 Schema riassuntivo delle categorie di Transfer Learning 20
Tabella 2 Test complessivo delle performance dopo l'addestramento 21
Tabella 3 Risultati della valutazione separata 21
Tabella 4 Test complessivo delle performance dopo l'addestramento 23
Tabella 5 Risultati della valutazione separata 23

ELENCO DELLE FIGURE

Figura 1 Heatmap riguardante la posizione dei pedoni 14
Figura 2 Differenze tra apprendimento tradizionale e transfer learning 17
Figura 3 Transfer learning da COCO a KAIST 20
Figura 4 Esempio di predizioni, in verde la *ground truth* in rosso le predizioni. 21
Figura 5 Transfer learning da KAIST Visibile a KAIST Termico 22
Figura 6 Schema di funzionamento di AutoAugment 24
Figura 7 Esempio di policy con 5 sub-policy, immagine tratta da [1] 25

INTRODUZIONE

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 Graphics Processing Unit (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.

Sempre grazie allo sviluppo di hardware sempre più potente l'interesse sta sempre più virando verso il mondo del *Deep Learning*. In questo capitolo cercheremo di classificare le varie metodologie con cui si porta a compimento la *Object Detection*. La letteratura sui detector è molto disomogenea e variegata, prenderemo quindi come riferimento i lavori di *Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng and Rong Qu* [2] e *Zhengxia Zou, Zhenwei Shi, Yuhong Guo and Jieping Ye* [3].

Una prima, ma importante distinzione va fatta tra metodi non basati sul *deep learning* e metodi basati sul *deep learning*.

1.1 TIPOLOGIE DI DETECTOR

U

1.1.1 *One Stage Detector*

1.1.2 *Two Stage Detector*

1.2 RETINANET

1.2.1 *Focal Loss*

2

SISTEMA

2.1 DATASET

Inizialmente i dataset usati per gli esperimenti sono due. Il primo è KAIST Multispectral Pedestrian Dataset (KAIST MPD) [4], per cui è disponibile un'ampia documentazione, il secondo è un dataset gratuito realizzato dalla FLIR [5] per cui è disponibile una documentazione molto stringata.

2.1.1 *KAIST Multispectral Pedestrian Dataset*

Il Korea Advanced Institute of Science and Technology (KAIST) propone in [4] 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 costrutito a partire da un wafer di silicio zincato.

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 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.* [6]. 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 la calibrazione stereo. Si può osservare che gli assi ottici delle telecamere al di là della divisione del fascio di luce sono paralleli a causa delle impostazioni tecniche. Di conseguenza, fra i due domini dell'immagine, è presente unicamente una traslazione ed è necessario solamente aggiustare la posizione tramite *camera jig* a tre assi 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 procedura, oltre alla focale, i domini condividono i punti principali e sono privi di baseline. Il dominio dell'immagine, virtualmente allineato, ha 640x512 pixel di risoluzione spaziale e un FOV di 39°, analogamente a quella umano. Visto che un pattern a scacchiera convenzionale non è osservabile con telecamera termica, 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.

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

ACQUISIZIONE DEI DATI E ANNOTAZIONI Tutto il marchingegno 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 le quali

sono state annotate manualmente con un totale di 103128 Bounding Box (BB). Per realizzare le annotazioni è stata usata una versione modificata del Piotr's Computer Vision Toolbox [7]. 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 affermazione. 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 vengono definite *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%.

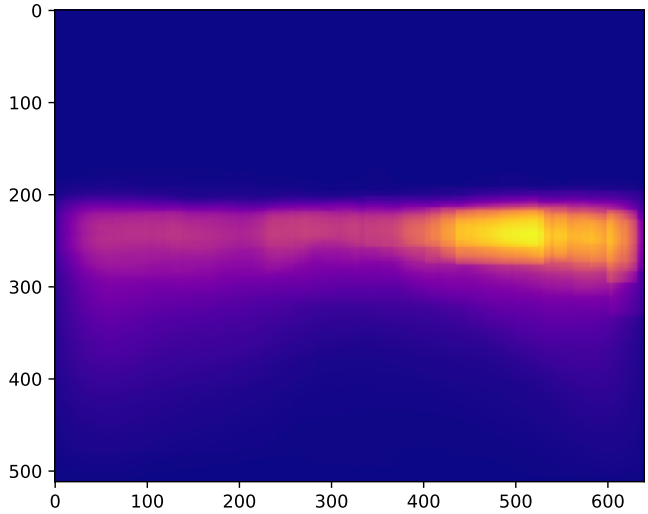

Figura 1.: Heatmap riguardante la posizione dei pedoni

- 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 1 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. In figura **INSERIRE FIGURA** è presente un esempio.

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 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, nello 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 moticicli. 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.

2.1.3 *Video di RFI*

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* [8]

> 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* [9].

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.

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.

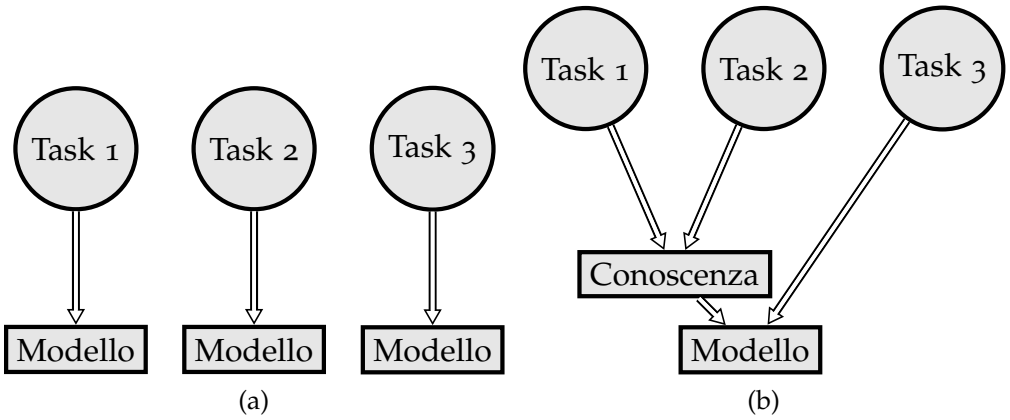

Figura 2.: Differenze tra apprendimento tradizionale e transfer learning

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 2.

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 $X = \{x_1, x_2, \ldots, x_n\} \in \mathcal{X}$ 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 $y_i \in \mathcal{Y}$. 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 $D_S = \{(x_{S_1}, y_{S_1}), \ldots, (x_{S_n}, y_{S_n})\}$ 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 $D_T = \{(x_{T_1}, y_{T_1}), \ldots, (x_{T_n}, y_{T_n})\}$ 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 $\mathcal{D}_T$ 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* $\mathcal{T}_S$ e $\mathcal{T}_T$. 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 $\mathcal{T}_S$, un dominio target $\mathcal{D}_T$ ed un task di apprendimento $\mathcal{T}_T$, il transfer learning tenta di migliorare l'apprendimento di $f(\cdot)_T \in \mathcal{D}_T$ usando la conoscenza in $\mathcal{D}_S$ e $\mathcal{T}_S$, con $\mathcal{D}_S \neq \mathcal{D}_T$ o $\mathcal{T}_S \neq \mathcal{T}_T$.

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 obbiettivo.
  - 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 obbiettivo sono totalmente differenti, e quindi non direttamente utilizzabili.

- *Transfer Learning Trasduttivo (si traduce così "transductive"?)*: in questo caso il task obbiettivo 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 obbiettivo. Possiamo a sua volta dividere questa categoria in ulteriori due sottocategorie.
  - Gli spazi delle feature tra sorgente e obbiettivo sono diffrenti, più formalmente abbiamo $\mathcal{X}_S \neq \mathcal{X}_T$
  - Gli spazi delle feature tra sorgente e obbiettivo 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 obbiettivo 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 obbiettivo, come possono essere il *clustering*, *dimensionality reduction* o *density estimation*. Non abbiamo quindi annotazioni né nel dominio sorgente, né nel dominio obbiettivo.

In Tabella 1 sono riassunte tutte le caratteristiche principali delle varie categorie di *transfer learning*. **SISTEMARE TABELLA 1**

2.2.2 *Addestramento sulle immagini RGB*

Per il primo esperimento è stato deciso di effettuare un training di *RetinaNet* partendo dai pesi della rete precedentemente addestrata sul dataset

gia	Similarità	Annotazioni Sorgente	Annotazioni Target	Campo di applicabilità
ivo	Multitask Learning	SI	SI	Regressione e Classificazione
	Self-taught Learning	NO	SI	Regressione e Classificazione
ttivo	Domain adaptation	SI	NO	Regressione e Classificazione
visionato	46501	NO	NO	Clustering, dimensionality reduc

Tabella 1.: Schema riassuntivo delle categorie di Transfer Learning

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 3 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.

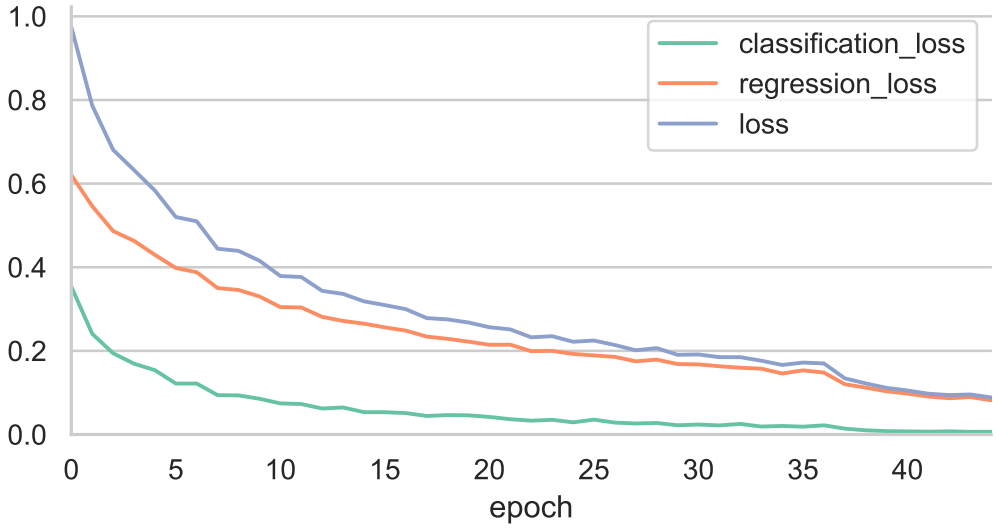

Figura 3.: Transfer learning da COCO a KAIST

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 2. La Mean Average Precision (mAP) mostrata

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

Tabella 2.: Test complessivo delle performance dopo l'addestramento

	# Istanze	mAP
Person	33688	0.4493
Cyclist	818	0.2015
Complessivo	34506	0.4434

(a) Giorno

	# Istanze	mAP
Person	11507	0.3281
Cyclist	578	0.0029
Complessivo	12085	0.3125

(b) Notte

Tabella 3.: Risultati della valutazione separata

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 3a e Tabella 3b. Come lecito aspettarsi, avendo visibilità limitata in notturna si ottengono risultati mediamente peggiori. In Figura 4 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 considerazio-

Figura 4.: Esempio di predizioni, in verde la *ground truth* in rosso le predizioni.

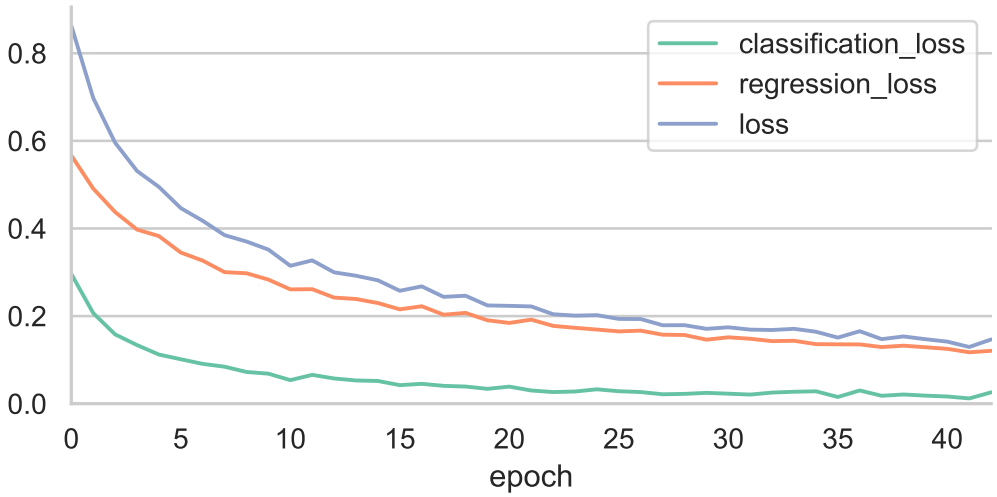

Figura 5.: Transfer learning da KAIST Visibile a KAIST Termico

ne anche la classe *people*, ma rinominandola in *person* in maniera tale da farla digerire **rivedere questo periodo** alla rete già addestrata. **INSERIRE VALUTAZIONE EFFETTUATA CON TUTTE LE CLASSI, APPENA LE GPU SI LIBERANO. I PESI SONO QUELLI DELLA RUN RUBY-YOGURT**

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, come si può vedere in Figura 5 le varie loss scendono in maniera stabile e controllata. In maniera del tutto similare a quanto si può vedere in Figura 3.

I risultati della valutazione sul test set sono mostrati in maniera complessiva in Tabella 4. Il test è stato effettuato usando i label *person*, *cyclist*

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

Tabella 4.: Test complessivo delle performance dopo l'addestramento

	# Istanze	mAP
Person	38802	0.3415
Cyclist	818	0.2015
Complessivo	39620	0.3353

(a) Giorno

	# Istanze	mAP
Person	14641	0.4440
Cyclist	578	0.0029
Complessivo	15219	0.4288

(b) Notte

Tabella 5.: Risultati della valutazione separata

e *people*, quest'ultimi oppurtunamente 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 2). 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 5. La prima cosa che si nota guardando la Tabella 5a è un drastico calo della mAP nella parte diurna, mentre in Tabella 5b 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, 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 luce propria, di giorno invece accade l'esatto contrario. **aggiungere tabella con una colonna in più che rappersenta la differenza di performance tra le valutazioni**

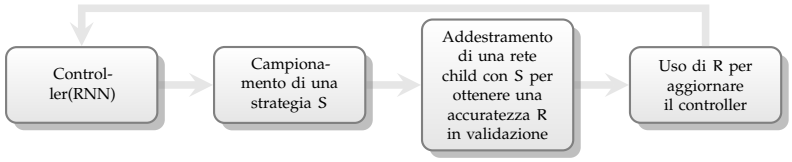

Figura 6.: Schema di funzionamento di AutoAugment

2.2.4 *ALTRA ROBA DA SCRIVERE*

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. Nel caso di **SEZIONE DA SCRIVERE SU ANNOTAZIONI MANUALI** 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. Ad esempio, 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.

2.3.1 *Auto Augment*

Lo scopo di *AutoAugment* [1] è automatizzare il processo che porta ad ottenere delle regole di *Data Augmentation* per un determinato dataset obbiettivo. La ricerca delle migliori policy per il dataset viene trattato alla pari di un problema di ricerca discreta, in Figura 6 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 Recurrent Neural Network (RNN) campiona una policy *S*. Questa policy al suo inter-

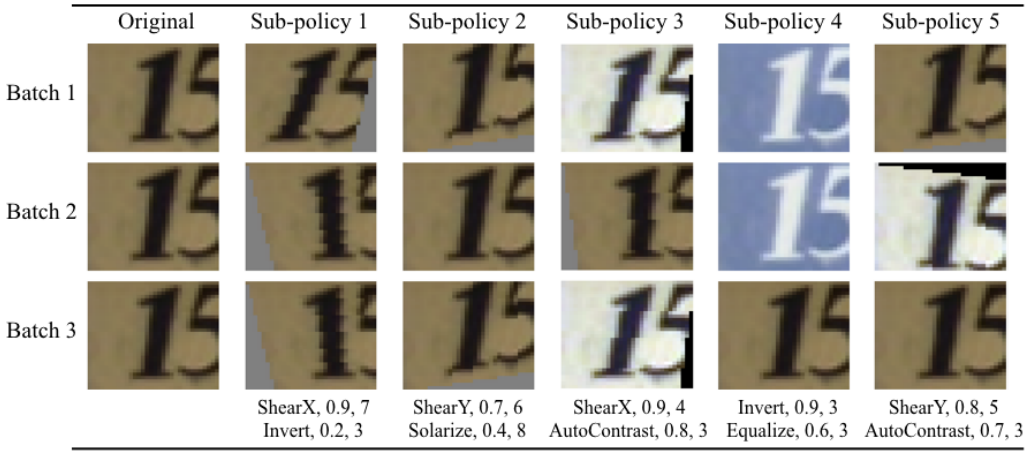

Figura 7.: Esempio di policy con 5 sub-policy, immagine tratta da [1]

no 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 7. La prima subpolicy 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 i valori di probabilità in un intervallo di 11. 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* [10]. 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 6 la ricompensa è R, ovvero il risultato della valutaizone sul validation set. Alla fine di questa fase di ricerca vengono trovate le migliori 5 policy, che vengono concatenate in un'unica grande policy contentente 25 subpolicy.

AUTOAUGMENT PER OBJECT DETECTION Fino ad ora abbiamo parlato di *AutoAugment* in termini di classificazione di una immagini, lo scopo

della tesi è però realizzare *Object Detection*. Gli stessi autori di [1] in [11] hanno evoluto questa tecnica per far si 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 granulare la scelta dei possibili valori discretizati, 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.

Tuttavia sono state rilasciate delle policy apprese da dataset differenti con cui sono stati eseguiti esperimenti che verranno successsivamente 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* [12] è un nuovo modo di realizzare *Data Augmentation* proposto dagli stessi autori di [1] e [11] 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 [1], 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 gli iperparametri 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 [12]

```
transforms = [
'Identity', 'AutoContrast', 'Equalize',
'Rotate', 'Solarize', 'Color', 'Posterize',
'Contrast', 'Brightness', 'Sharpness',
'ShearX', 'ShearY', 'TranslateX', 'TranslateY']

def randaugment(N, M):
"""Generate a set of distortions.

   Args:
     N: Number of augmentation transformations to apply
         sequentially.
     M: Magnitude for all the transformations.
"""

  sampled_ops = np.random.choice(transforms, 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 caretteristiche 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

3.1 ORGANIZZAZIONE DEI DATASET

3.2 ESPERIMENTI INIZIALI

3.3 DATA AUGMENTATION

4

CONCLUSIONI

A

IMPLEMENTAZIONE DI RANDAUGMENT SU RETINANET

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

BIBLIOGRAFIA

[1] 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 5, 24, 25, 26, and 27.)

[2] 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 page 9.)

[3] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *CoRR*, abs/1905.05055, 2019. (Cited on page 9.)

[4] 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 11.)

[5] Free flir thermal dataset for algorithm training. `https://www.flir.com/oem/adas/adas-dataset-form/`. Accessed: 2019-11-20. (Cited on page 11.)

[6] L. Bienkowski, C. Homma, K. Eisler, and Christian Boller. Hybrid camera and real-view thermography for nondestructive evaluation. 01 2012. (Cited on page 12.)

[7] Piotr Dollár. Piotr's Computer Vision Matlab Toolbox (PMT). `https://github.com/pdollar/toolbox`. (Cited on page 13.)

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. (Cited on page 16.)

[9] 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 16.)

[10] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. (Cited on page 25.)

[11] 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 page 26.)

[12] 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 26 and 27.)