PEPPER SOCIALE

Object Detection using Pepper front camera

Group 11: {g.desimone39, g.giaquinto9, c.marino34, s.serritiello3}@studenti.unisa.it

1. INTRODUCTION

Nell'ambito della robotica sociale, l'object detection è un compito necessario per dotare un robot di una maggiore percezione dell'ambiente che lo circonda, tramite il semplice riconoscimento degli oggetti davanti a sé in tempo reale. Per fare ciò è dunque necessario principalmente un buon design dell'architettura dei nodi ROS, garantendo una comunicazione efficiente tra di essi, così come la scelta di una rete neurale che presenti caratteristiche adeguate al real time: si tratta quindi di un compromesso tra precisione di rilevamento degli oggetti da parte della rete e velocità di esecuzione, sulla base dell'hardware di cui il robot in questione dispone. In questo esperimento, viene utilizzato il robot Pepper di Aldebaran, e le immagini processate sono prelevate dalla camera frontale che ha risoluzione 1920x1080 pixel, ad un frame rate pari a 15.

2. ARCHITECTURE

La soluzione proposta prevede l'implementazione di un insieme composto da un totale di quattro nodi ROS scritti in Python. Tali nodi comunicano attraverso meccanismi di Publisher/Subscriber o tramite eventi asincroni basati su Servizi. I nodi in questione sono i seguenti e i loro nomi sono autoesplicativi:

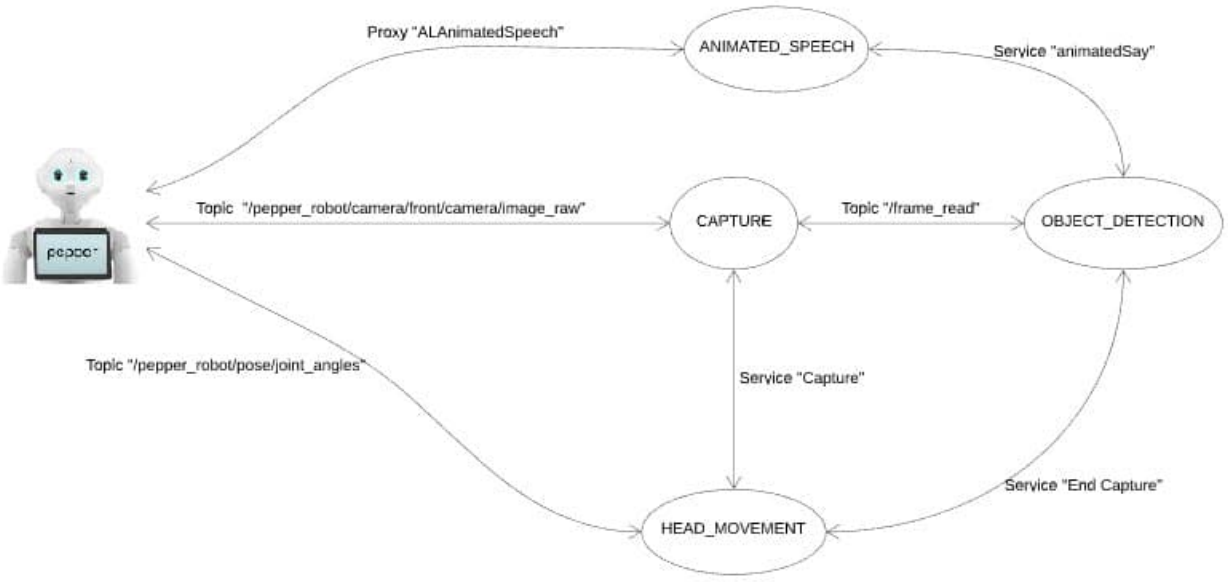


Figura 1: Architettura dei nodi Ros e meccanismi di comunicazione tra loro.

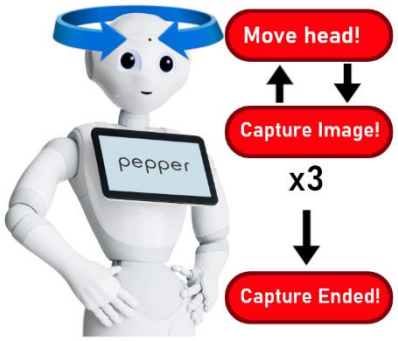
HEAD MOVEMENT

Il nodo “head_movement” è il responsabile della parte di movimento del task richiesto a Pepper. La necessità del movimento della testa di Pepper nasce dal bisogno di dover mettere in condizione Pepper di poter riprendere l’intera scena a sé stante con la sola camera frontale. Data questa correlazione tra movimento della testa del robot e cattura dei frame dalla camera, in questo nodo, infatti, troviamo non solo il controllo del movimento della testa, bensì anche quello relativo al controllo della cattura dei frame per la corretta coordinazione dei movimenti.

Le posizioni desiderate da far assumere a pepper e la velocità di rotazione della testa vengono comunicate da tale nodo tramite la pubblicazione sul topic “*joint_angles*” dell’interfaccia con Pepper. Il nodo “*head_movement*” può muovere la testa di Pepper specificando in input la posizione desiderata da far assumere ai giunti di interesse e un parametro di tipo booleano “capture”, per specificare se va anche catturata l’immagine dopo aver spostato la testa in tale posizione. Il metodo attende che il robot esegua il movimento richiesto e, se capture è asserito, invoca il servizio “Capture” messo a disposizione dal nodo Stream_capture e salva il risultato di questo nel parametro response, che indica l’esito dell’operazione.

Il nodo head_movement gestisce anche la coordinazione di tutte le fasi di movimento e acquisizione immagini, per cui, dopo che i tre frame sono stati catturati, invoca il servizio “capture_ended” per comunicare agli altri nodi di fermare l’acquisizione dei frame e avviare il processo di costruzione del risultato finale della rete a valle delle operazioni di object detection.

I tre metodi descritti sono impiegati nel main del nodo nel seguente modo:



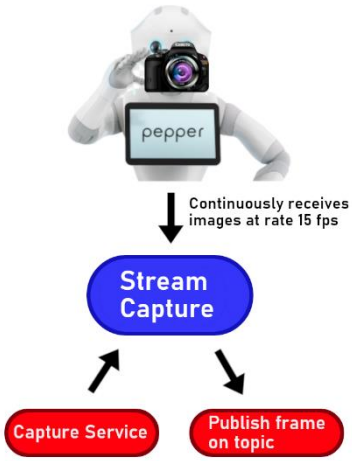
1. Il nodo “head_movement” è inizializzato;
2. Si attende che i servizi *Capture* del nodo *stream_capture* e *capture_ended* del nodo *object_detection* diventino disponibili per poter procedere alla fase di inizializzazione;
3. Si inizializza l’orientamento di Pepper posizionando la testa nella posizione centrale senza acquisire alcuna immagine;
4. Si muove la testa di Pepper nelle tre direzioni definite (sinistra, centro e destra) e si acquisisce un’immagine per ognuna di queste, chiamando il servizio *Capture*;
5. Infine, il servizio *end_capture* viene chiamato per terminare l’operazione di acquisizione delle immagini.

STREAM CAPTURE

Il nodo *stream_capture* viene utilizzato per la cattura di frame immagine ottenuti attraverso la telecamera frontale di Pepper. In seguito alla creazione del servizio *Capture*, il nodo si mette in ascolto dei frame che vengono trasmessi come messaggio sul topic *image_raw* di Pepper. La business logic del nodo è definita attraverso una classe StreamCapture: quando essa viene istanziata, il nodo diventa

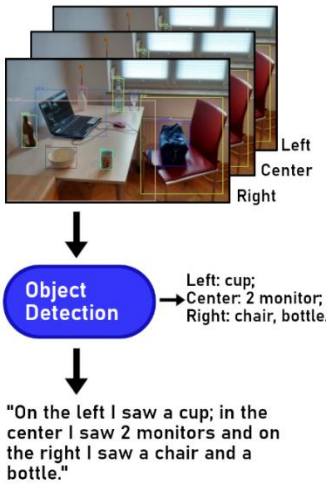
sia publisher del topic *frame_reader* che subscriber del topic *image_raw* di Pepper dove vengono trasmessi i frame della telecamera frontale.

Nel momento in cui il nodo diventa subscriber di *image_raw* esso riceverà le immagini ad una frequenza di 15 fps dalla videocamera di Pepper, ma ai fini del completamento del task assegnato non sono utili tutti i frame provenienti dal topic. Per cui, attraverso un servizio Ros custom, il nodo viene allertato per catturare e processare attivamente i frame provenienti dalla telecamera. Questa operazione viene fatta attraverso una logica Client/Server: il nodo avvia infatti il servizio *Capture* la cui logica di callback è definita da una funzione della classe StreamCapture. Quando il nodo riceve la richiesta di cattura immagine, una flag booleana interna cambia stato e la funzione di callback che gestisce la ricezione di immagini dalla telecamera di Pepper catturerà il primo frame disponibile, lo trametterà sul topic *frame_read* e resetterà il valore della flag interna. L'utilizzo della flag interna permette la temporizzazione dei processi di callback del nodo.



OBJECT DETECTION

Uno dei nodi più importanti presenti in questa architettura è quello responsabile dell'esecuzione del task della object detection. Dopo una fase di inizializzazione e caricamento del modello, il nodo si mette in attesa di ricevere messaggi sul topic *frame_read*: questi rappresentano le immagini acquisite dal nodo s*tream_capture* che devono essere processate. Quando un frame viene ricevuto, il nodo ricerca all'interno di esso gli oggetti presenti, tenendo conto anche della relativa molteplicità, tramite la rete neurale caricata in fase di inizializzazione del nodo.

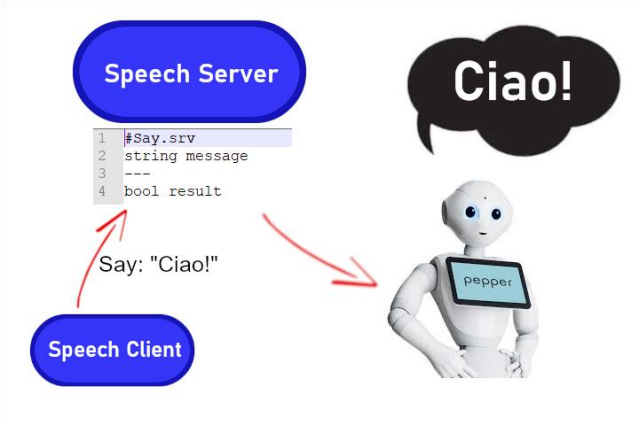


Tale nodo inoltre funge da Service Server esponendo il servizio capture_*ended*: quando il nodo *head_movement* ha terminato le fasi di rotazione della testa e la conseguente acquisizione dei frame, esso chiama questo servizio per segnalare il completamento dell'operazione. Il nodo *object_detection,* a questo punto, prepara una stringa contente tutti gli oggetti rilevati nei tre frame e le relative molteplicità e genera una richiesta al servizio *animatedSay*, descritto successivamente, in modo da passare tale messaggio al nodo *Speech* che effettuerà la sintesi vocale.

SPEECH

Un ulteriore nodo che prende parte al completamento del task in questione è il chiamato "speech", il quale viene anch'esso eseguito tramite il launch file che viene lanciato al momento dell'avvio. Anche in questo caso, il file sorgente è stato strutturato in maniera da descrivere una classe in Python, la quale implementa l'ultimo compito che compone la pipeline di avvenimenti per l'object detection, ovvero esegue il parlato di Pepper. Per fare ciò, il nodo è stato organizzato come Server che mette a disposizione un servizio "*animatedSay*" da cui riceve esattamente la stringa corretta e sintatticamente formattata da far dire a Pepper.

Il servizio è stato predisposto come segue: un generico Client che effettua una richiesta, inserirà la frase nel campo *message* e riceverà un risultato che riporta l'esito dell'operazione di Speech su Pepper. Inoltre, una copia della stringa. verrà stampata su terminale.



Nell'implementazione fornita, l'unico nodo tra quelli descritti che richiede tale servizio è *object_detection*, descritto in precedenza.

Inoltre, all'interno di tale classe viene anche effettuata la connessione all'interfaccia ros per la Speech, offerta dal framework NaoQi. Tra le varie API fornite per la gestione del linguaggio, è stata effettuata una connessione al proxy "ALAnimatedSpeech", il quale permette di far parlare Pepper in maniera espressiva e animata, ovvero può ricevere un testo con annotazioni che specificano quale azione far compiere a Pepper mentre parla e in quale istante avviarla. La connessione al Proxy è stata effettuata specificando l'IP della macchina a cui connettersi in maniera remota, che corrisponde senz'altro al parametro *ip* che viene specificato al momento dell'esecuzione del comando che avvia il launch file. Di default, tale parametro è impostato a "10.0.1.230". Il nodo, infine, resterà attivo in maniera da attendere eventuali richieste del servizio offerto.

3. DESIGN CHOICES

Per effettuare il riconoscimento degli oggetti all'interno delle immagini acquisite dalla camera frontale di Pepper, la scelta di una rete addestrata sulla rilevazione di oggetti di uso comune è di fondamentale importanza. Inoltre, anche la velocità di elaborazione su Pepper e le prestazioni della rete in termini di accuracy sono fattori rilevanti al fine della nostra scelta.

Model	test-dev AP	test-dev AP_{50}	test-dev AP_{75}	val AP	Params	Ratio	FLOPs	Ratio	Latency GPU_{ms}	Latency CPU_s
EfficientDet-D0 (512)	**33.8**	**52.2**	**35.8**	**33.5**	**3.9M**	**1x**	**2.5B**	**1x**	**16**	**0.32**
YOLOv3 [31]	33.0	57.9	34.4	-	-	-	71B	28x	51†	-
EfficientDet-D1 (640)	**39.6**	**58.6**	**42.3**	**39.1**	**6.6M**	**1x**	**6.1B**	**1x**	**20**	**0.74**
RetinaNet-R50 (640) [21]	37.0	-	-	-	34M	6.7x	97B	16x	27	2.8
RetinaNet-R101 (640)[21]	37.9	-	-	-	53M	8.0x	127B	21x	34	3.6
EfficientDet-D2 (768)	**43.0**	**62.3**	**46.2**	**42.5**	**8.1M**	**1x**	**11B**	**1x**	**24**	**1.2**
RetinaNet-R50 (1024) [21]	40.1	-	-	-	34M	4.3x	248B	23x	51	7.5
RetinaNet-R101 (1024) [21]	41.1	-	-	-	53M	6.6x	326B	30x	65	9.7
ResNet-50 + NAS-FPN (640) [8]	39.9	-	-	-	60M	7.5x	141B	13x	41	4.1
EfficientDet-D3 (896)	**45.8**	**65.0**	**49.3**	**45.9**	**12M**	**1x**	**25B**	**1x**	**42**	**2.5**
ResNet-50 + NAS-FPN (1024) [8]	44.2	-	-	-	60M	5.1x	360B	15x	79	11
ResNet-50 + NAS-FPN (1280) [8]	44.8	-	-	-	60M	5.1x	563B	23x	119	17
ResNet-50 + NAS-FPN (1280@384)[8]	45.4	-	-	-	104M	8.7x	1043B	42x	173	27
EfficientDet-D4 (1024)	**49.4**	**69.0**	**53.4**	**49.0**	**21M**	**1x**	**55B**	**1x**	**74**	**4.8**
AmoebaNet+ NAS-FPN +AA(1280)[42]	-	-	-	48.6	185M	8.8x	1317B	24x	259	38
EfficientDet-D5 (1280)	**50.7**	**70.2**	**54.7**	**50.5**	**34M**	**1x**	**135B**	**1x**	**141**	**11**
EfficientDet-D6 (1280)	**51.7**	**71.2**	**56.0**	**51.3**	**52M**	**1x**	**226B**	**1x**	**190**	**16**
AmoebaNet+ NAS-FPN +AA(1536)[42]	-	-	-	50.7	209M	4.0x	3045B	13x	608	83
EfficientDet-D7 (1536)	**52.2**	**71.4**	**56.3**	**51.8**	**52M**	**1x**	**325B**	**1x**	**262**	**24**

We omit ensemble and test-time multi-scale results [27, 10].
†Latency marked with † are from papers, and others are measured on the same machine with Titan V GPU.

Table 2: **EfficientDet performance on COCO** [22] – Results are for single-model single-scale. `test-dev` is the COCO test set and `val` is the validation set. `Params` and `FLOPs` denote the number of parameters and multiply-adds. `Latency` denotes inference latency with batch size 1. `AA` denotes auto-augmentation [42]. We group models together if they have similar accuracy, and compare their model size, FLOPs, and latency in each group.

La tabella riportata in alto è stata estratta da [1] e mostra le performance delle varie reti allo stato dell'arte per il riconoscimento di oggetti, testate su 20.000 e 5.000 campioni del dataset Coco2017 e riporta i valori dell'Average Precision (AP, AP_{50}, AP_{75}), il numero di parametri utilizzati dalla rete, il numero di operazioni complessive effettuate dalla rete per ogni immagine (FLOPS), testate su un hardware con una GPU Titan-V e una CPU single-thread Xeon.

Da questa tabella è quindi possibile notare che le reti basate su EfficientDet riportano risultati largamente migliori: raggruppando le reti secondo le performance, è possibile verificare che, rispetto alle altre reti dello stesso gruppo, le EfficientDet utilizzano immagini più piccole, meno parametri e di conseguenza meno operazioni richieste per il processing di una singola immagine.

Questa osservazione è confermata anche dalla tabella riportata di lato, estratta da [2]: le reti riportate sono pre-addestrate sul dataset Coco 2017, e i dati riportati si riferiscono alla velocità di elaborazione in ms e l'Average Precision (mAP). Le diverse versioni di EfficientDet si dimostrano anche qui migliori rispetto alle altre, sia in termini di velocità e di performance che di dimensioni delle immagini in input.

Model name	Speed (ms)	COCO mAP	Outputs
CenterNet HourGlass104 512x512	70	41.9	Boxes
CenterNet HourGlass104 Keypoints 512x512	76	40.0/61.4	Boxes/Keypoints
CenterNet HourGlass104 1024x1024	197	44.5	Boxes
CenterNet HourGlass104 Keypoints 1024x1024	211	42.8/64.5	Boxes/Keypoints
CenterNet Resnet50 V1 FPN 512x512	27	31.2	Boxes
CenterNet Resnet50 V1 FPN Keypoints 512x512	30	29.3/50.7	Boxes/Keypoints
CenterNet Resnet101 V1 FPN 512x512	34	34.2	Boxes
CenterNet Resnet50 V2 512x512	27	29.5	Boxes
CenterNet Resnet50 V2 Keypoints 512x512	30	27.6/48.2	Boxes/Keypoints
EfficientDet D0 512x512	39	33.6	Boxes
EfficientDet D1 640x640	54	38.4	Boxes
EfficientDet D2 768x768	67	41.8	Boxes
EfficientDet D3 896x896	95	45.4	Boxes
EfficientDet D4 1024x1024	133	48.5	Boxes
EfficientDet D5 1280x1280	222	49.7	Boxes
EfficientDet D6 1280x1280	268	50.5	Boxes
EfficientDet D7 1536x1536	325	51.2	Boxes
SSD MobileNet v2 320x320	19	20.2	Boxes
SSD MobileNet V1 FPN 640x640	48	29.1	Boxes
SSD MobileNet V2 FPNLite 320x320	22	22.2	Boxes
SSD MobileNet V2 FPNLite 640x640	39	28.2	Boxes
SSD ResNet50 V1 FPN 640x640 (RetinaNet50)	46	34.3	Boxes
SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)	87	38.3	Boxes
SSD ResNet101 V1 FPN 640x640 (RetinaNet101)	57	35.6	Boxes
SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)	104	39.5	Boxes
SSD ResNet152 V1 FPN 640x640 (RetinaNet152)	80	35.4	Boxes
SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)	111	39.6	Boxes
Faster R-CNN ResNet50 V1 640x640	53	29.3	Boxes
Faster R-CNN ResNet50 V1 1024x1024	65	31.0	Boxes
Faster R-CNN ResNet50 V1 800x1333	65	31.6	Boxes
Faster R-CNN ResNet101 V1 640x640	55	31.8	Boxes
Faster R-CNN ResNet101 V1 1024x1024	72	37.1	Boxes
Faster R-CNN ResNet101 V1 800x1333	77	36.6	Boxes
Faster R-CNN ResNet152 V1 640x640	64	32.4	Boxes
Faster R-CNN ResNet152 V1 1024x1024	85	37.6	Boxes
Faster R-CNN ResNet152 V1 800x1333	101	37.4	Boxes
Faster R-CNN Inception ResNet V2 640x640	206	37.7	Boxes
Faster R-CNN Inception ResNet V2 1024x1024	236	38.7	Boxes
Mask R-CNN Inception ResNet V2 1024x1024	301	39.0/34.6	Boxes/Masks
ExtremeNet	–	–	Boxes

Sulla base di quanto detto, abbiamo scelto di testare a nostra volta solo quattro di queste reti pre-addestrate, ovvero:

1. EfficientDet D1 con input size 640x640
2. EfficientDet D2 con input size 768x768
3. EfficientDet D3 con input size 896x896
4. CenterNet ResNet101 V1 FPN 512x512

Il nostro test riguarda i tempi di caricamento della rete su Pepper e di elaborazione delle singole immagini in sequenza.

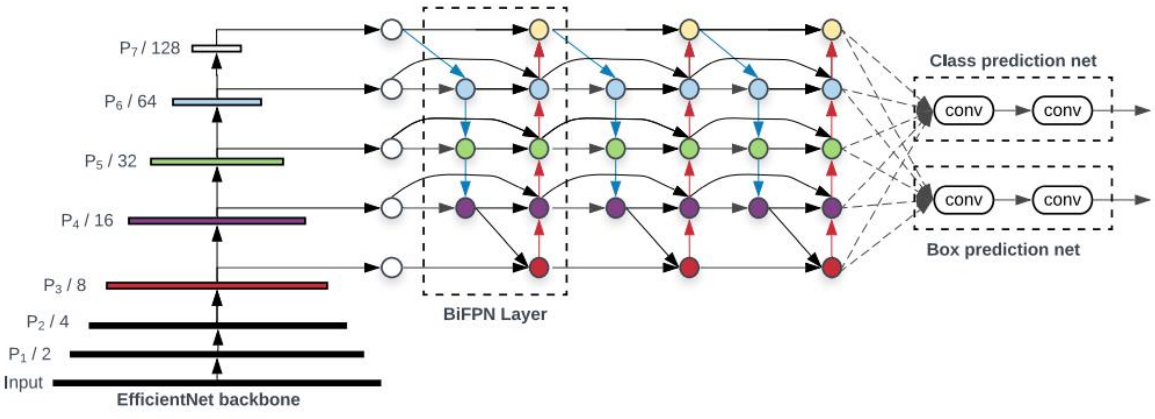
I risultati sono stati estratti durante una singola giornata in modo da avere risultati coerenti per tutti i modelli e il test è stato ripetuto 3 volte per modello, per verificare la coerenza. I dati sono i seguenti:

RETE	*T. Caricamento*	*T. Processing*	*Oggetti Sinistra*	*Oggetti Centro*	*Oggetti Destra*
1	162.77s	3.85/4.18/4.22s	person	2 keyboards, potted plant, 2 tv, chair, mouse, vase	3 chairs
2	187.38s	6.23/5.72/5.37s	nothing	2 keyboards, potted plant, 3 tv, chair, mouse	1 chair
3	194.54s	11.6/10.64/9.42s	dining table	3 keyboards, potted plant, 2 tv, 2 chairs, mouse, bottle	2 chairs, laptop
4	121.95s	8.18/7.37/7.49s	nothing	2 keyboards, potted plant, 3 tv, 2 chairs	1 chair, tv

T.Caricamento: Tempo di caricamento del modello in fase di inizializzazione del nodo Object_detector; T.Processing: Tempi relativi all'elaborazioni delle singole immagini (sinistra, centro e destra); Oggetti Sinistra, Centro e Destra riportano la lista degli oggetti rilevati dalle reti.

Questi tre fattori sono la base della scelta finale e la rete che abbiamo ritenuto adatta sia sotto il punto di vista del tempo di elaborazione delle immagini, sia della differenza riguardo gli oggetti rilevati all'interno dei singoli test effettuati direttamente su Pepper, è la EfficientDet D3.

Infatti, questa rete presenta delle performance migliori sul nostro test effettuato su Pepper, ed è comunque una delle reti migliori per l'object detection allo stato dell'arte attuale. Mentre, tenendo in considerazione tutti gli 8 modelli della EfficientDet, questa risulta un giusto compromesso considerando sia tempi di elaborazione, mAP e numero di parametri.



La figura precedente mostra l'architettura della generica rete EfficientDet, che segue largamente il paradigma del detector one-stage. La rete utilizzata come backbone è la EfficientNet, addestrata sul dataset ImageNet. Da essa, sono stati estratti e fusi insieme le feauture output dei livelli intermedi P3, P4, P5, P6 e l'ultimo P7. Queste feature ottenute dalla fusione sono state date in pasto alla rete finale adibita alla prediction di bounding box contenenti gli oggetti e le rispettive classi.

	Input size R_{input}	Backbone Network	BiFPN #channels W_{bifpn}	BiFPN #layers D_{bifpn}	Box/class #layers D_{class}
D0 ($\phi = 0$)	512	B0	64	3	3
D1 ($\phi = 1$)	640	B1	88	4	3
D2 ($\phi = 2$)	768	B2	112	5	3
D3 ($\phi = 3$)	896	B3	160	6	4
D4 ($\phi = 4$)	1024	B4	224	7	4
D5 ($\phi = 5$)	1280	B5	288	7	4
D6 ($\phi = 6$)	1280	B6	384	8	5
D6 ($\phi = 7$)	1536	B6	384	8	5

Table 1: **Scaling configs for EfficientDet D0-D6 –** ϕ is the compound coefficient that controls all other scaling dimensions; *BiFPN, box/class net, and input size* are scaled up using equation 1, 2, 3 respectively.

In questa tabella è possibile invece vedere le differenze che caratterizzano i 7 diversi modelli di EfficientDet, che riguardano l'input size, il numero di canali e di livelli della rete finale.

Riferimenti

[1] M. T. R. P. Q. V. Le, «EfficientDet: Scalable and Efficient Object Detection,» *IEEE,* 2020.

[2] TensorFlow, 2020. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md.