GPU TECHNOLOGY CONFERENCE

S9551 | Mar 20, 2019 | 14:00 pm, RM 231

Turbo-boosting Neural Networks for Object Detection

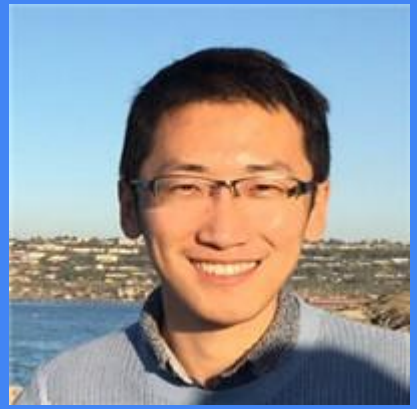
Hongyang Li

The Chinese University of Hong Kong /

Microsoft Research Asia

Hongyang

CUHK Ph.D. candidate /
Microsoft Intern



Research Timeline

- **2015** Ph.D. student start
- **2015** **ImageNet Challenge** (PAMI), **Object Attributes** (ICCV)
- **2016** Multi-bias Activation (ICML)
- **2017** **Recurrent Design for Detection** (ICCV), COCO Loss (NIPS)
- **2018** **Zoom-out-and-in Network** (IJCV), Capsule Nets (ECCV)
- **2019** **Feature Intertwiner** (ICLR), Few-shot Learning (CVPR)

First-author Papers

Outline

1. **Introduction to Object Detection**
   a. Pipeline overview
   b. Dataset and evaluation
   c. Popular methods
   d. Existing problems
2. **Solution: A Feature Intertwiner Module**
3. **Detection in Reality**
   a. Implementation on GPUs
   b. Efficiency and accuracy tradeoff
4. **Future of Object Detection**

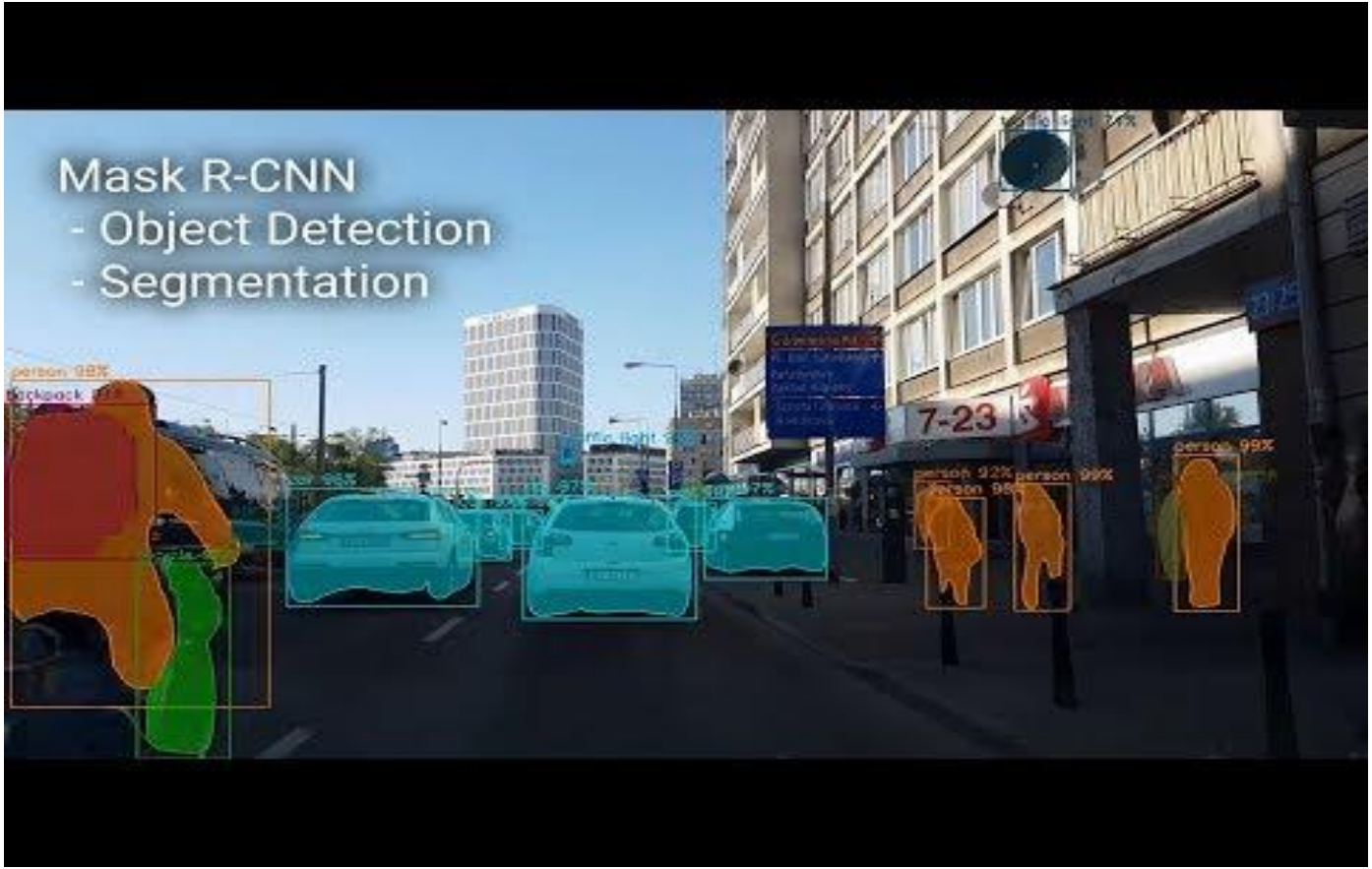
1. Introduction to Object Detection

Object Detection: core and fundamental task in computer vision

He et al.

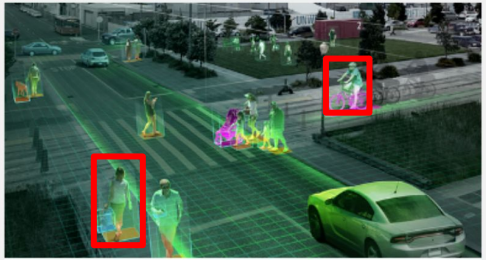
Mask-RCNN

ICCV 2017
Best paper

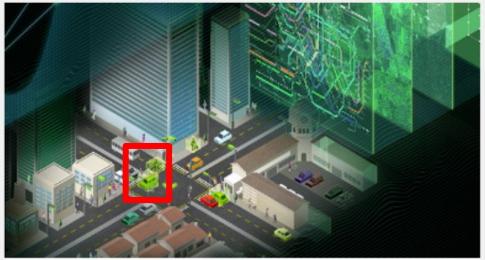


Object Detection is everywhere

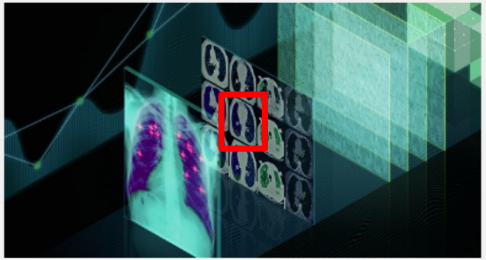
OBJECT DETECTION IS FUELING ALL AREAS OF BUSINESS



AI Cities

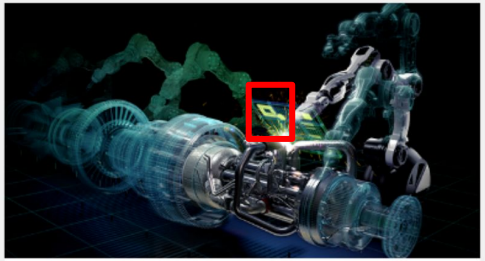


AI for Public Good



Healthcare

Retail

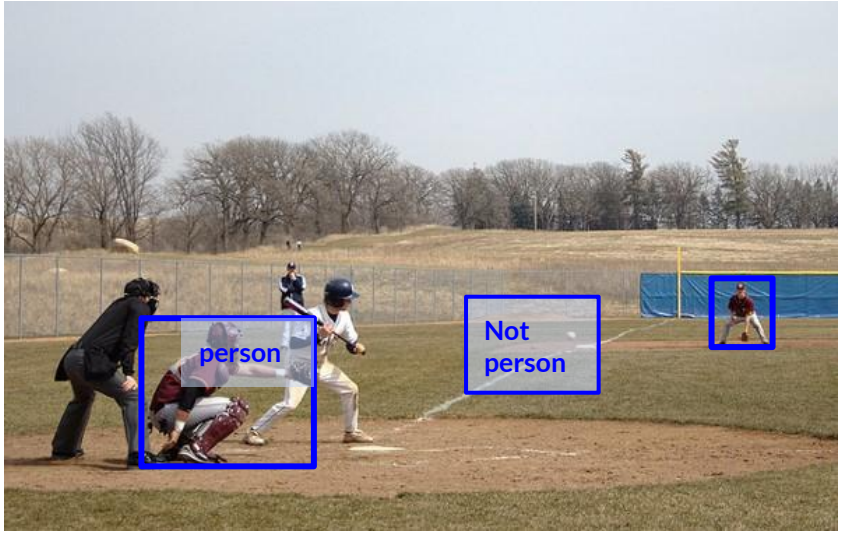


Robotics

Self-Driving Cars

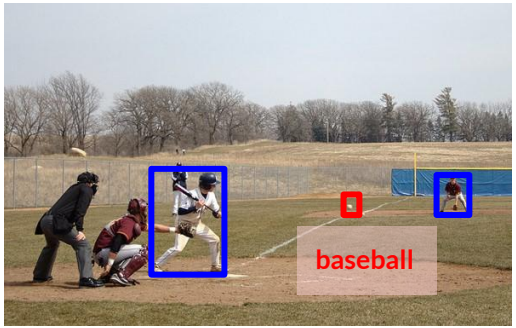
How to solve it?

A naive solution: **place many boxes** on top of image/feature maps and **classify them**!



How to solve it?

And yet challenges are:



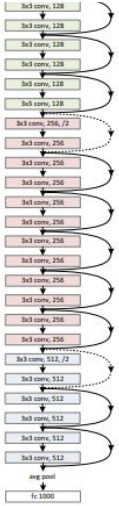
1. **Variations** in shape/appearance/size

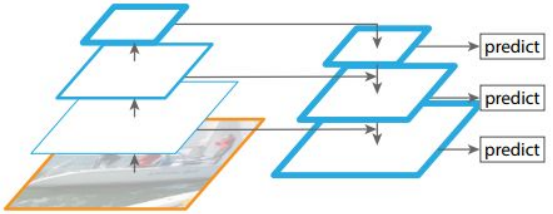
2. Ambiguity in cluttered scenarios

How to solve it?

(a) Place anchors as many as possible and (b) have layers deeper and deeper.

(a) place anchors

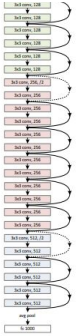


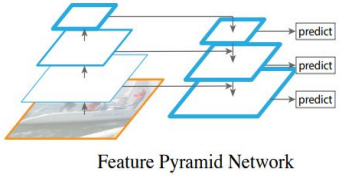


Feature Pyramid Network

(b) network design

Popular methods at a glance





Feature Pyramid Network

Pipeline/system design

One-stage:

YOLO and variants
SSD and variants

Two-stage:

R-CNN family
(Fast RCNN, Faster RCNN, etc)

Component/structure/loss design

Feature Pyramid Network

Focal loss (RetinaNet)

Online hard negative mining (OHEM)

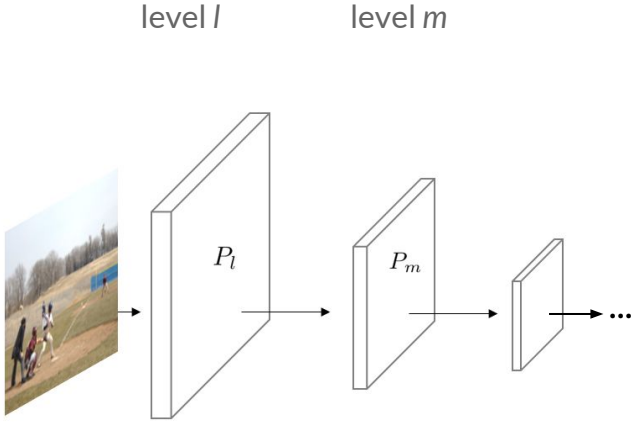
Zoom-out-and-in Network (ours)

Recurrent Scale Approximation (ours)

Feature Intertwiner (ours)

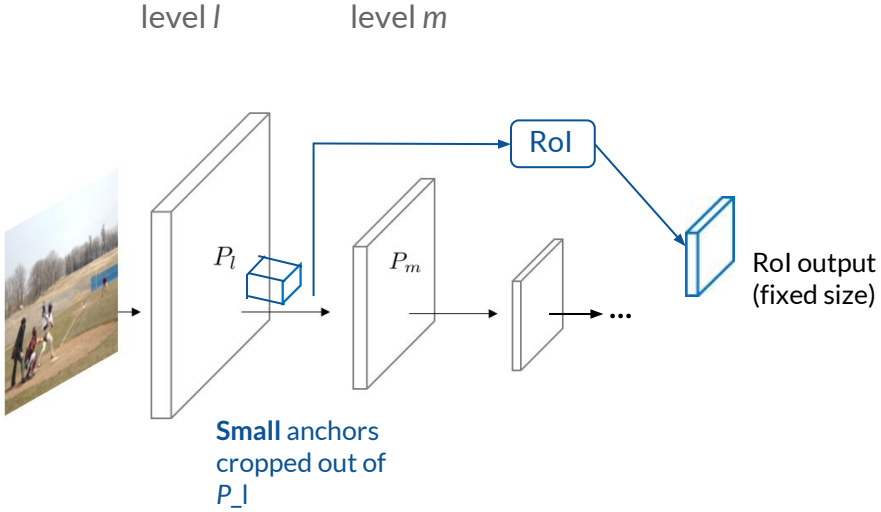
Pipeline: a roadmap of R-CNN family (two-stage detector)

*P*_l is the feature map output at **level *l***;
*P*_m is from a higher level *m*.



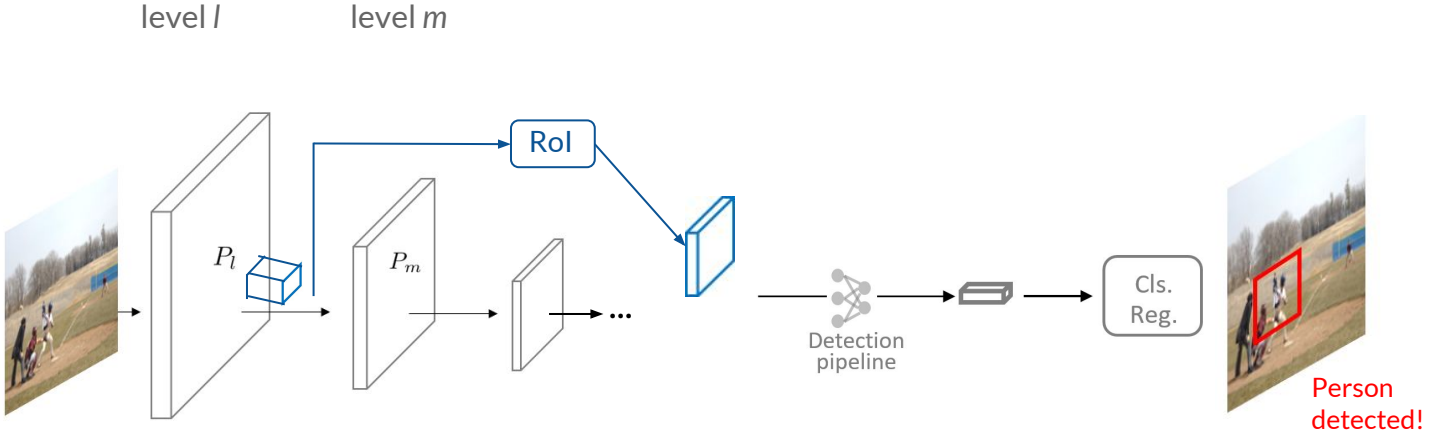
Pipeline: a roadmap of R-CNN family (two-stage detector)

*P*_l is the feature map output at **level *l***;
*P*_m is from a higher level *m*.



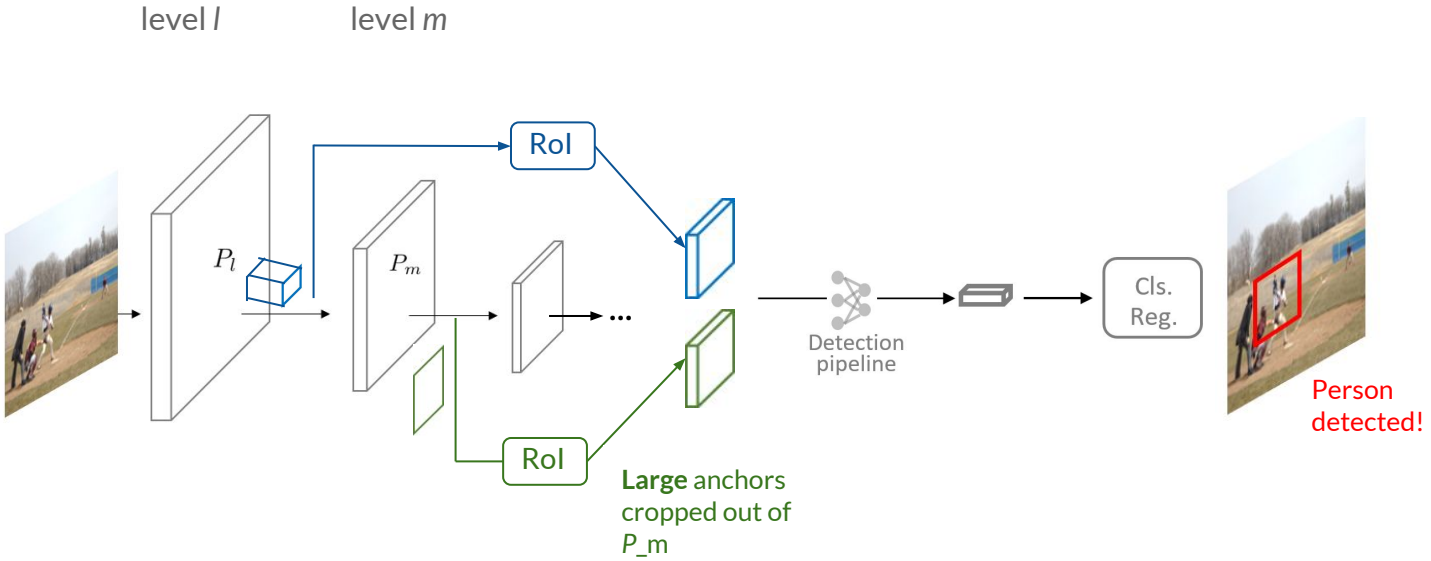
Pipeline: a roadmap of R-CNN family (two-stage detector)

*P*_l is the feature map output at **level *l***;
*P*_m is from a higher level *m*.



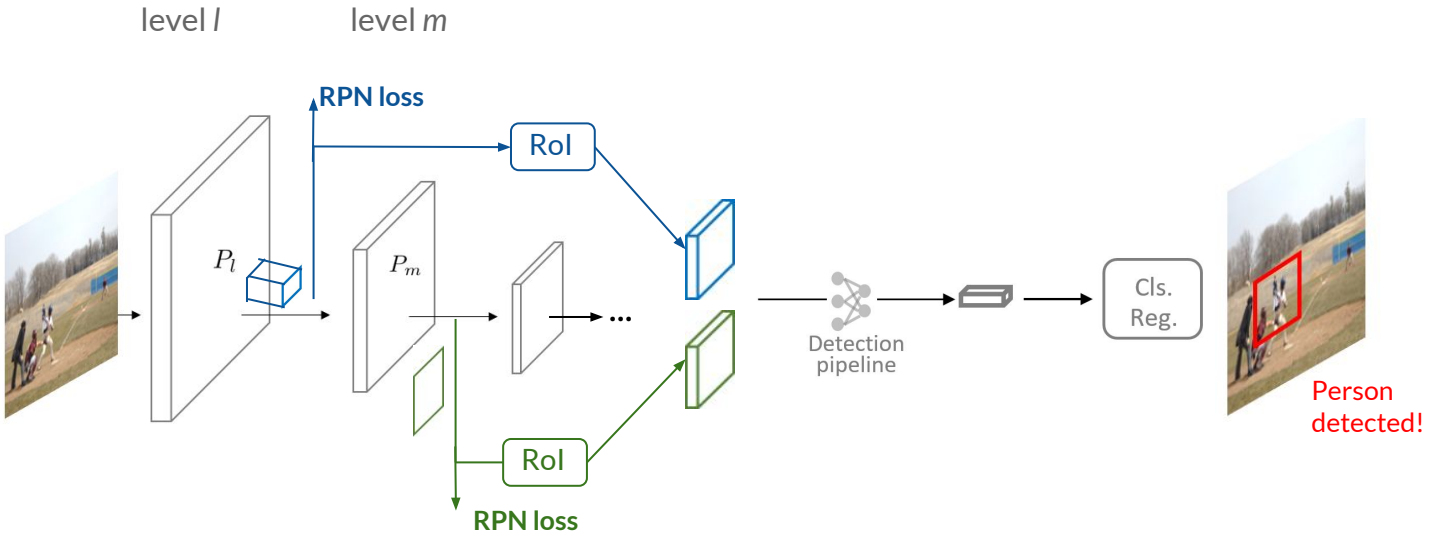
Pipeline: a roadmap of R-CNN family (two-stage detector)

*P*_l is the feature map output at **level *l***;
*P*_m is from a higher level *m*.

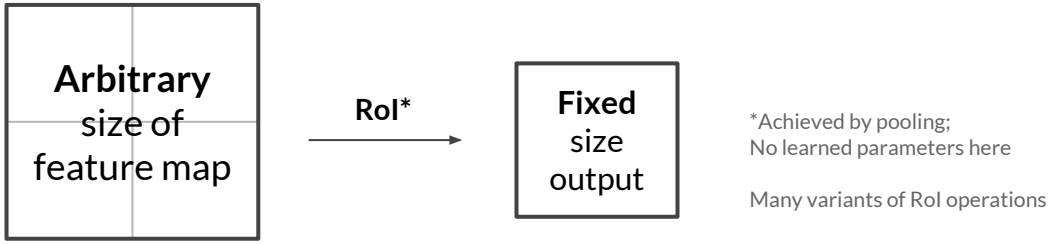


Pipeline: a roadmap of R-CNN family (two-stage detector)

*P*_l is the feature map output at **level *l***;
*P*_m is from a higher level *m*.

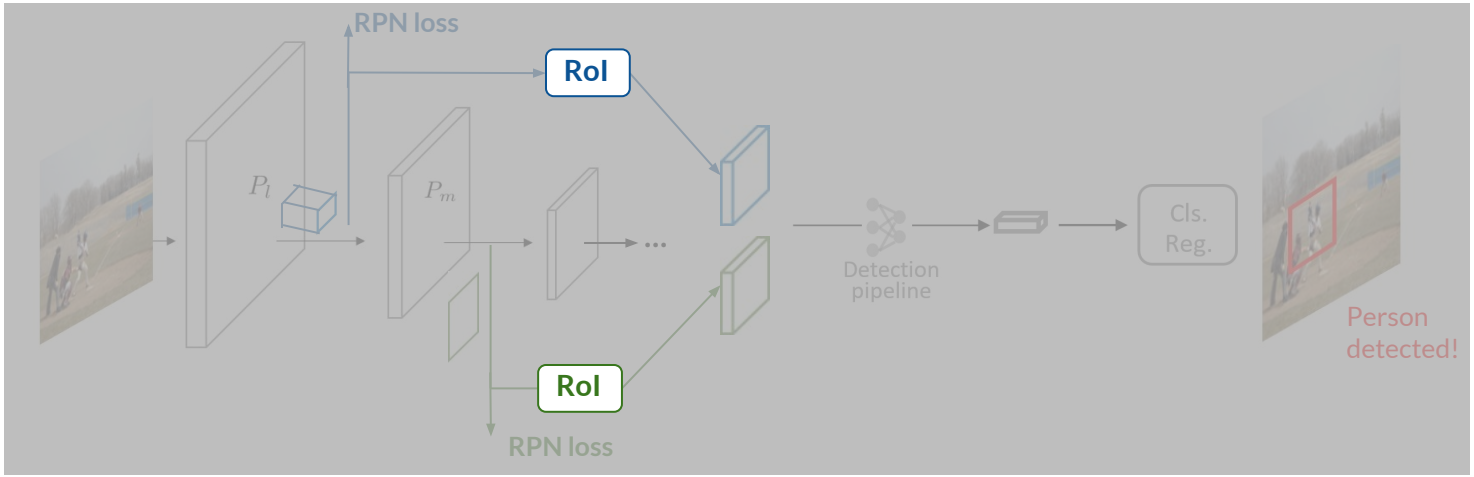


Side: what is RoI (region of interest) operation?



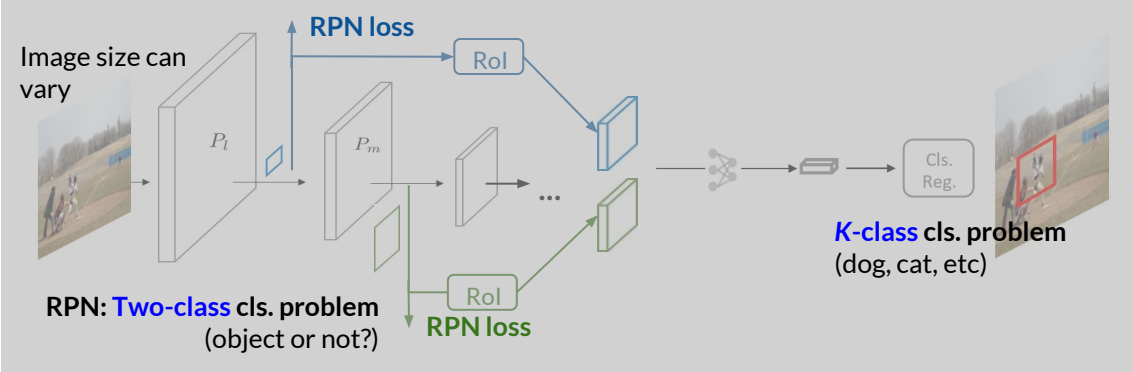
*Achieved by pooling;
No learned parameters here

Many variants of RoI operations



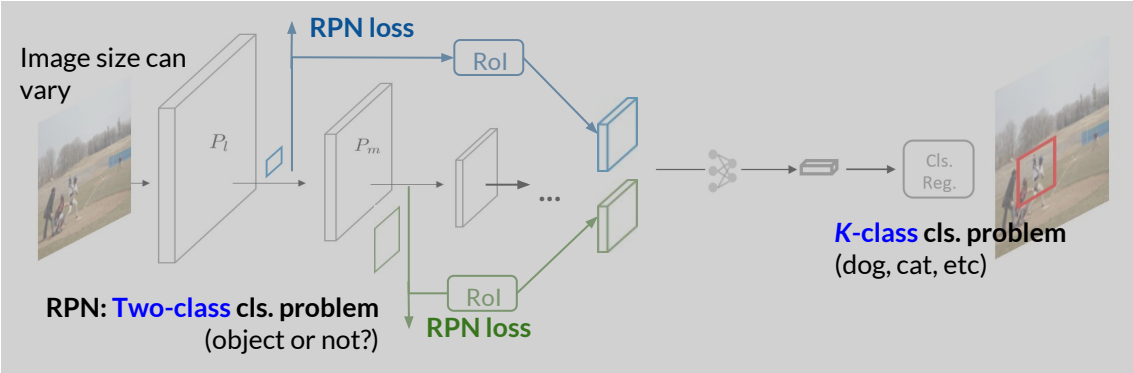
R-CNN family (two-stage detector) vs. YOLO (one -stage detector)

Two stage:
R-CNN family



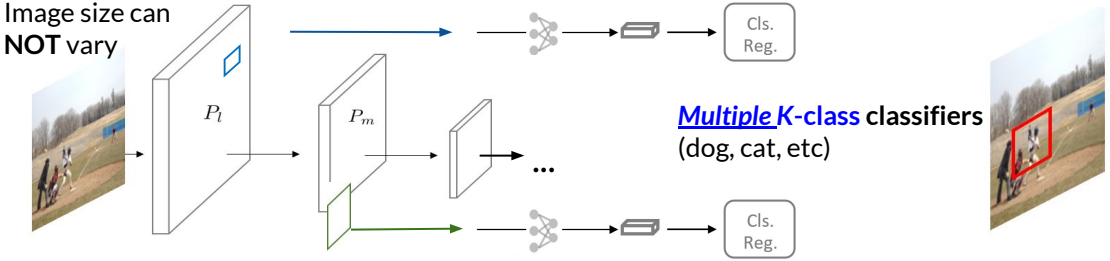
R-CNN family (two-stage detector) vs. YOLO (one -stage detector)

Two stage:
R-CNN family



More accurate

One stage:
YOLO/SSD



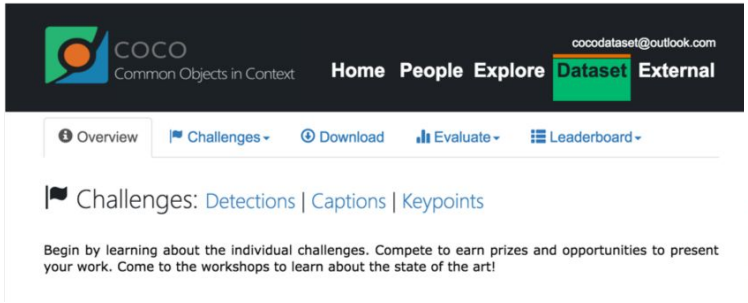
Faster

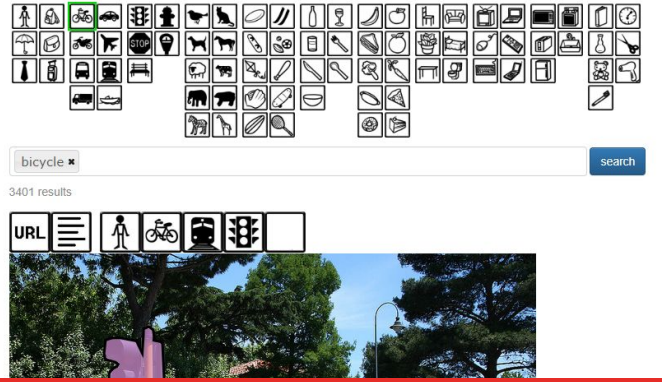
Both R-CNN and SSD models have been tremendously adopted in academia/industry.

In this talk, we focus on the ***two-stage detector*** with RoI operation.

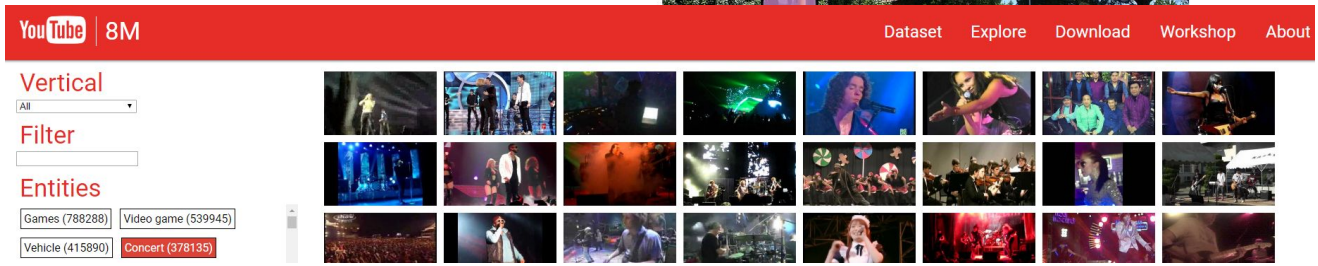
Datasets

COCO dataset
http://mscoco.org/





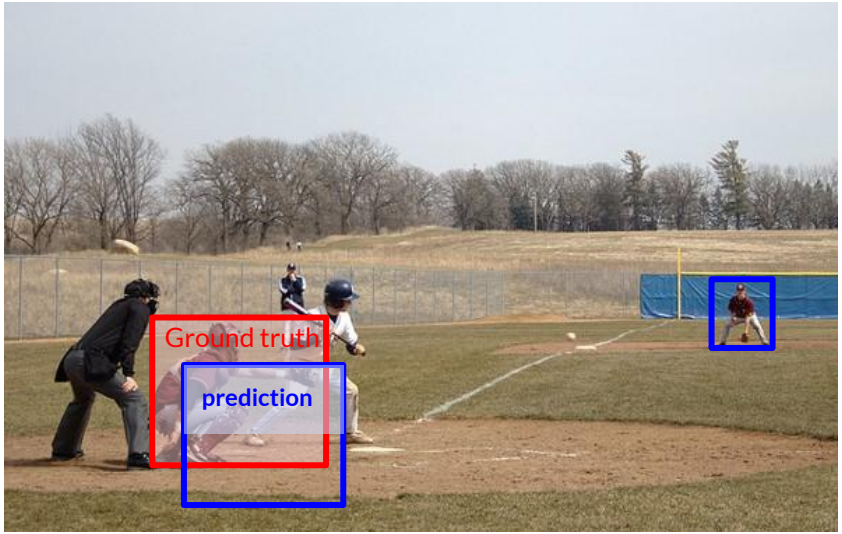
YouTube-8M dataset
https://research.google.com/youtube8m/



And many others
ImageNet, VisualGenome, Pascal VOC, KITTI, etc.

Evaluation - mean AP

For category `person`,



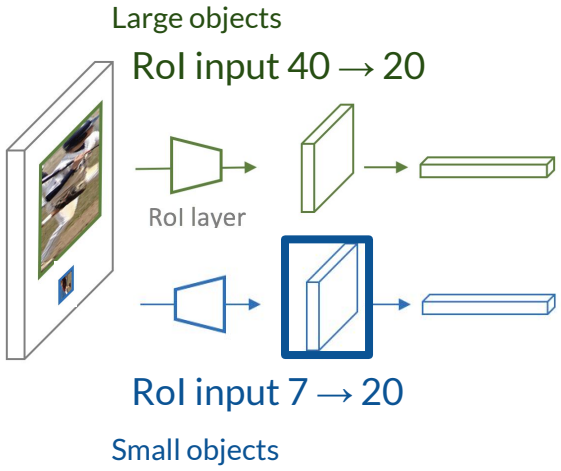
If IoU (intersection / union)
= 0.65 > *`threshold`*,
Then current prediction is counted as **Correct**

Get a set of Correct/incorrect predictions, compute the precision/recall.

Get the average precision (AP) from the precision/recall figure. Done.

Get all categories,
that's **mAP** (under *`threshold`*).

What is uncomfortable in current pipelines?



Accurate features in **down-sampling**!

Inaccurate features due to **up-sampling**!

Assume RoI's output is 20

What percentage of objects suffer from this?

level l	2	3	4	5
proposal # (perc.)	302 (75%)	36 (9%)	54 (14%)	8 (2%)
threshold	0.012	0.0479	0.1914	0.7657
below # / above #	263 / 39	25 / 11	34 / 20	8 / 0

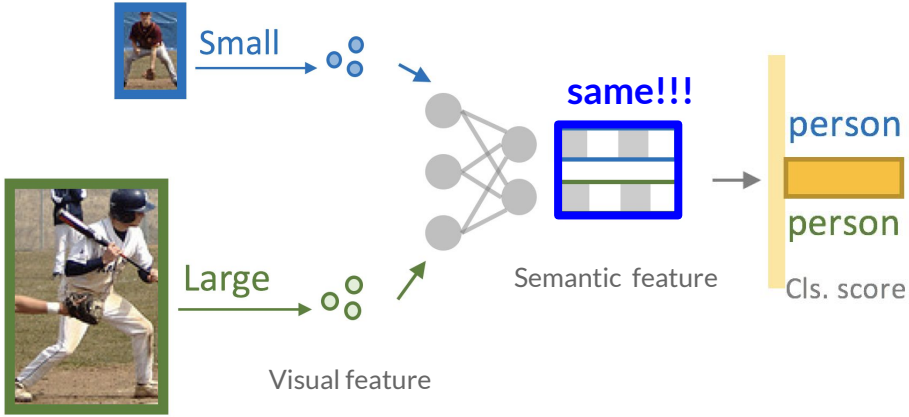
Table 3 in our paper.

Proposal assignment on each level before RoI operation. 'below #' indicates how many proposals are there whose size is **below** the size of RoI output.

We define **small set** to be the anchors on current level and **large set** to be all anchors *above* current level.

2. Solution: A Feature Intertwiner Module

Our assumption

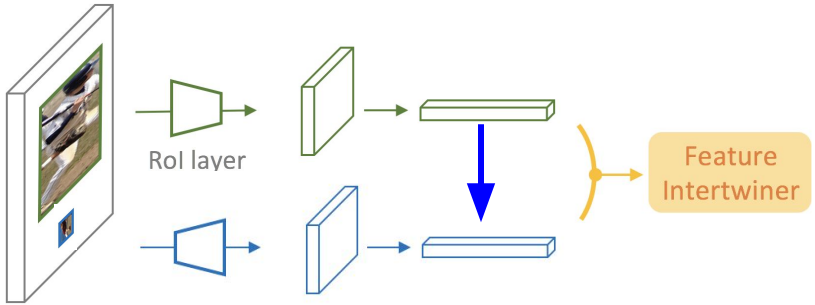


The semantic features among instances (large or small) within the same class should be **the same**.

Our motivation

Suppose we have two sets of features already -
one is from large objects and the other is from small ones.

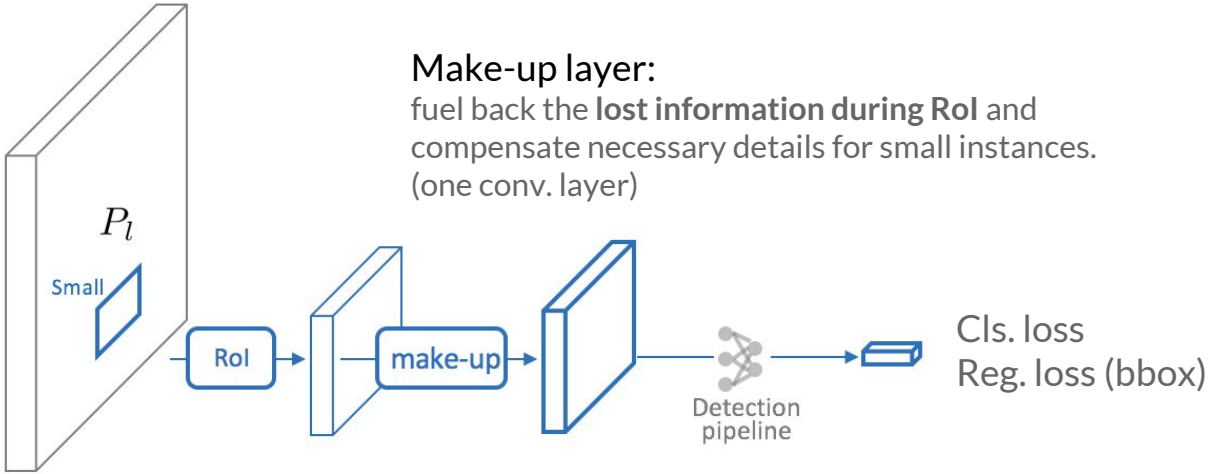
Naive feature intertwiner concept:



Intuition: let **reliable** features **supervise/guide** the learning of the **less reliable** ones.

The Feature Intertwiner

For **small** objects



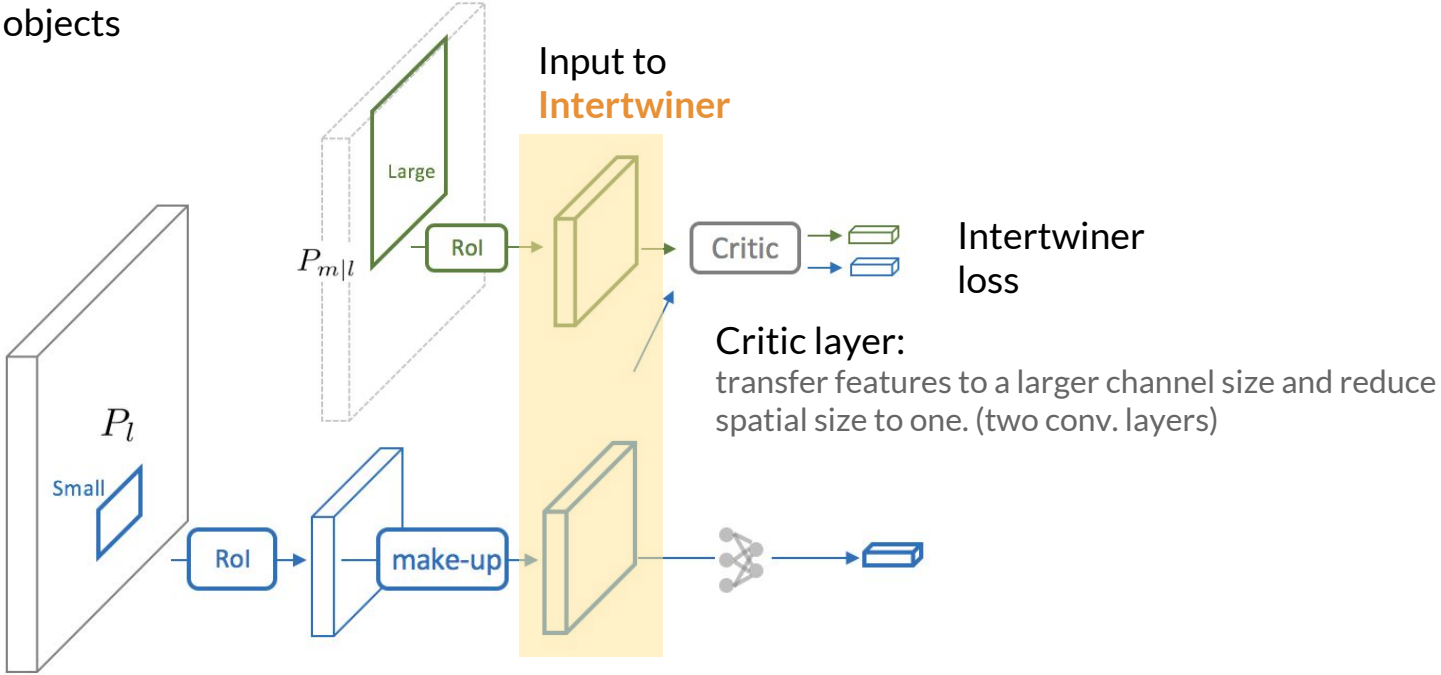
Make-up layer:
fuel back the **lost information during RoI** and compensate necessary details for small instances. (one conv. layer)

Cls. loss
Reg. loss (bbox)

For current level *l*

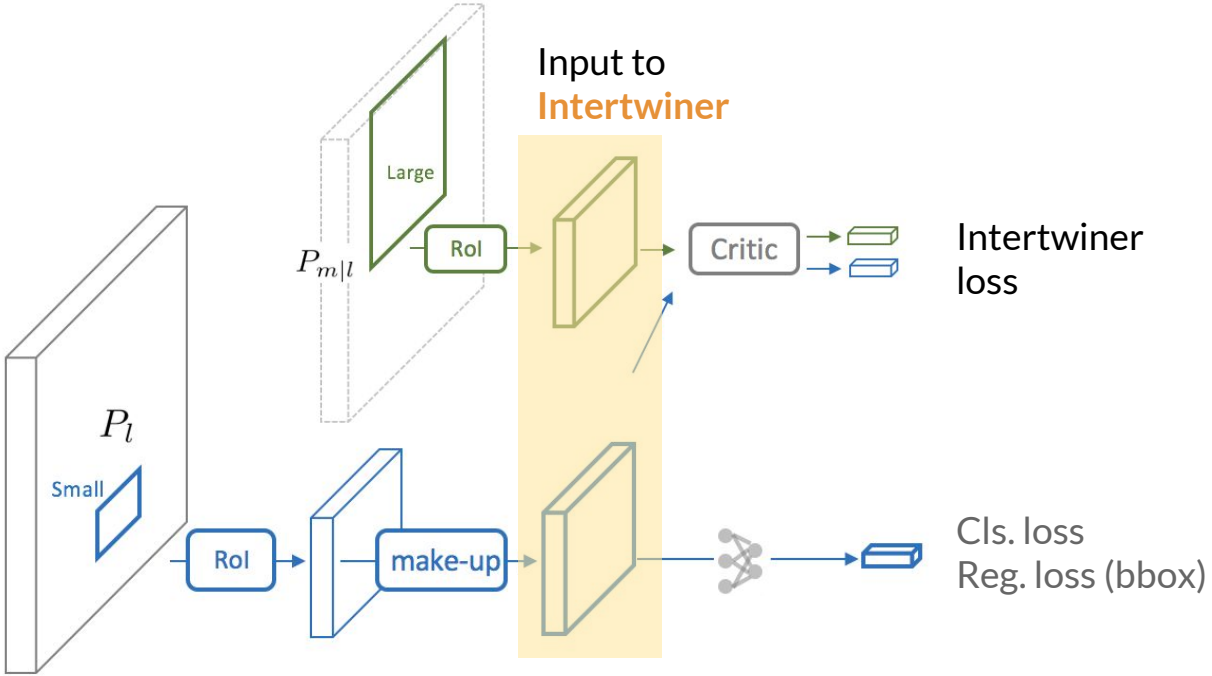
The Feature Intertwiner

For **large** objects



For current level *l*

The Feature Intertwiner

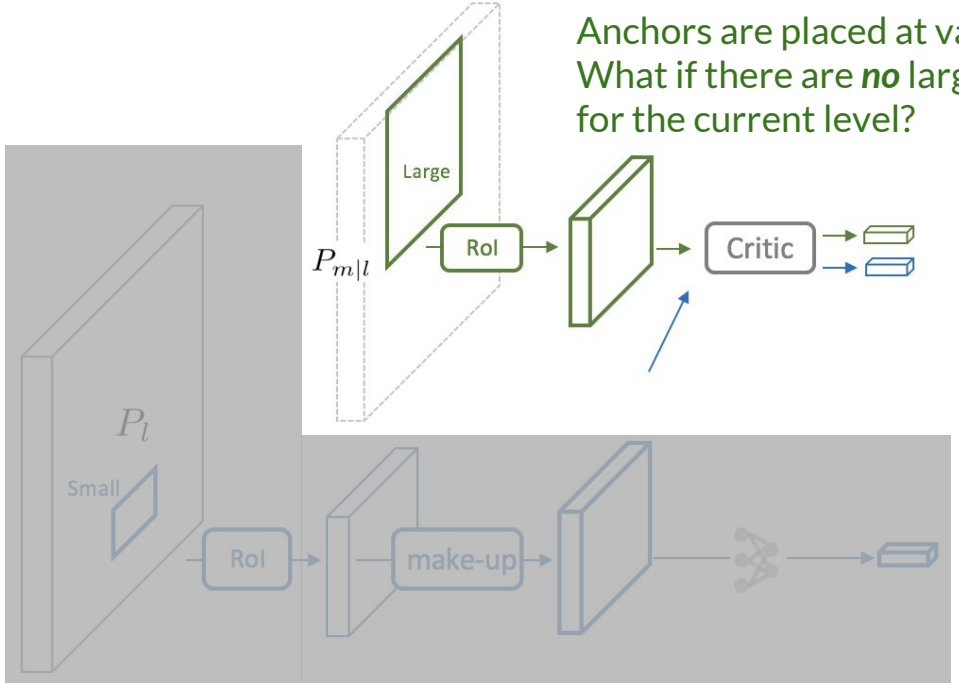


For current level *l*

Total loss = (Intertwiner+cls.+reg.) for *all* levels

The Feature Intertwiner

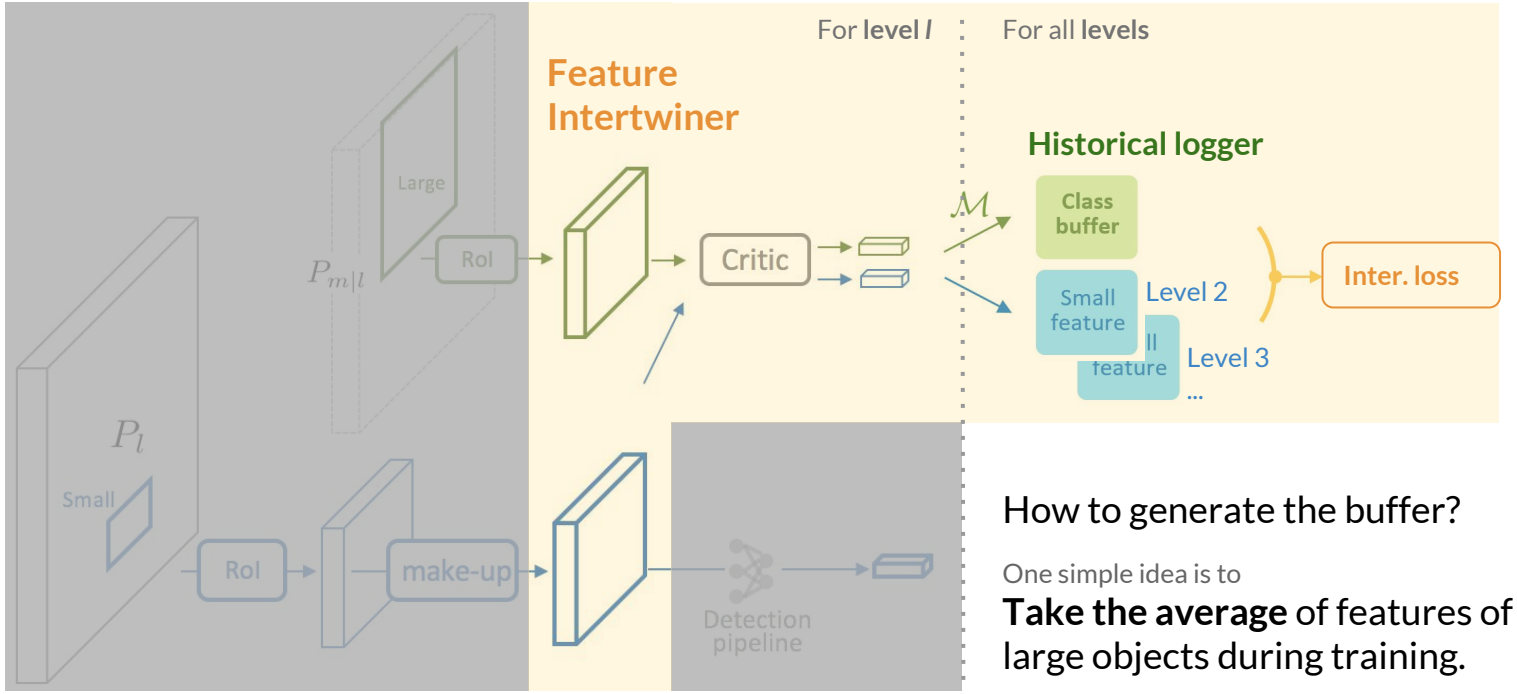
Anchors are placed at various levels.
What if there are ***no*** large instances in this mini-batch, for the current level?



We define **small set** to be the anchors on current level and **large set** to be all anchors *above* current level.

The Feature Intertwiner - class buffer

We use a class buffer to store the accurate feature set from large instances.

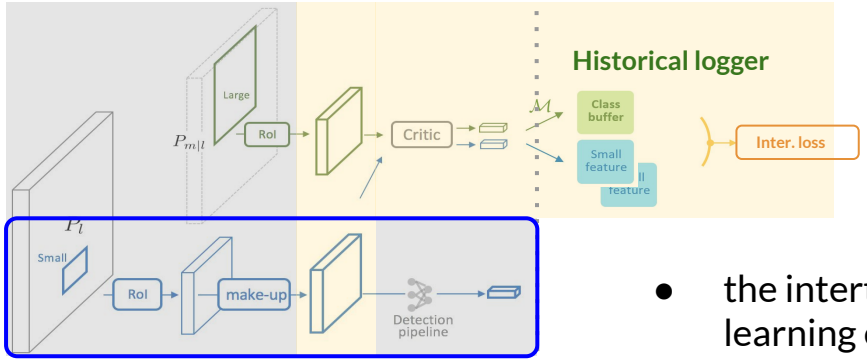


How to generate the buffer?

One simple idea is to

Take the average of features of all large objects during training.

Discussions on Feature Intertwiner

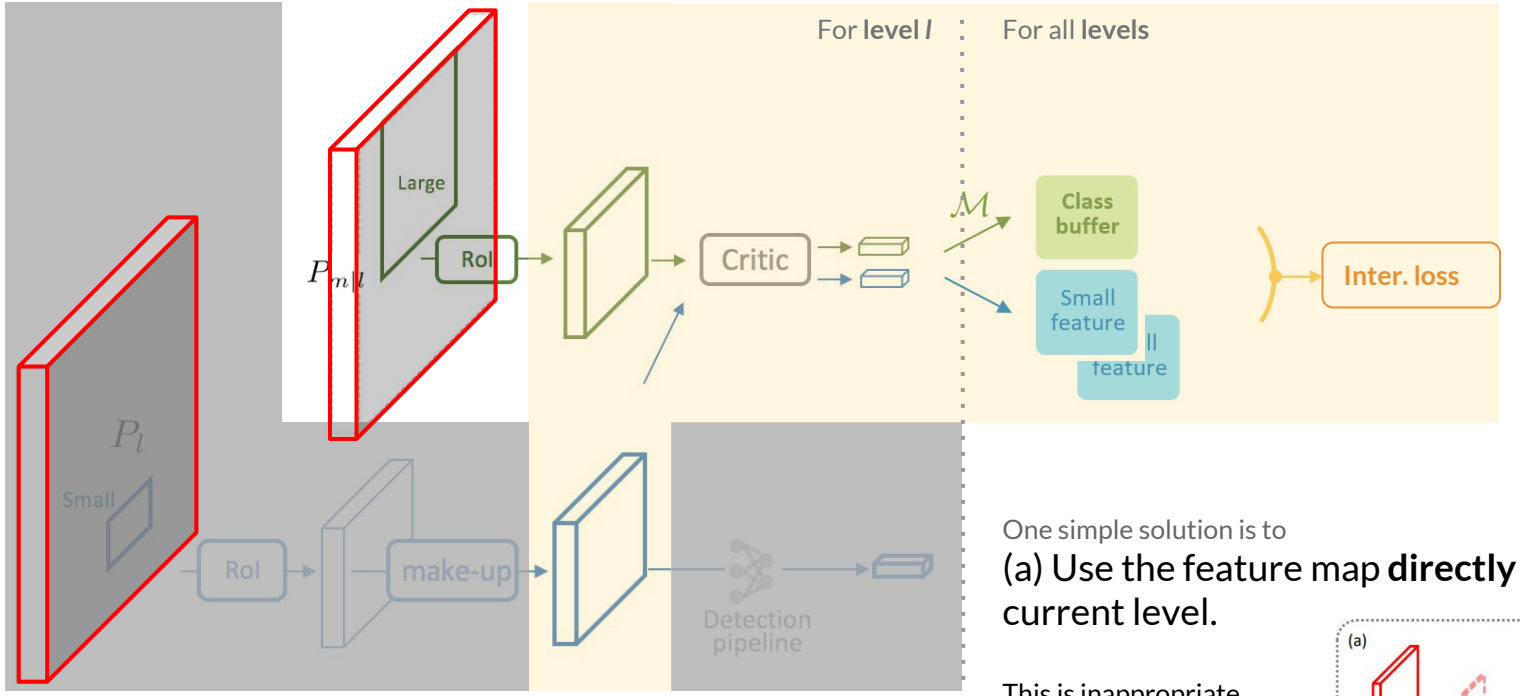


For inference

- the intertwiner is proposed to optimize feature learning of **the less reliable set**. During test, the **green part** will be removed.
- can be seen as a **teacher-student guidance** in the self-supervised domain.
- **detach** the gradient update in buffer will obtain better results. “Soft targets”, similarly as in RL (replay memory).
- The buffer is **level-agnostic**. Improvements over all levels/sizes of objects are observed.

The Feature Intertwiner - choosing optimal feature maps

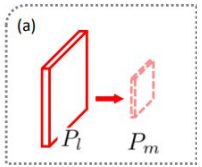
How to choose the **appropriate** maps for large objects? as input to intertwiner



One simple solution is to

(a) Use the feature map **directly** on current level.

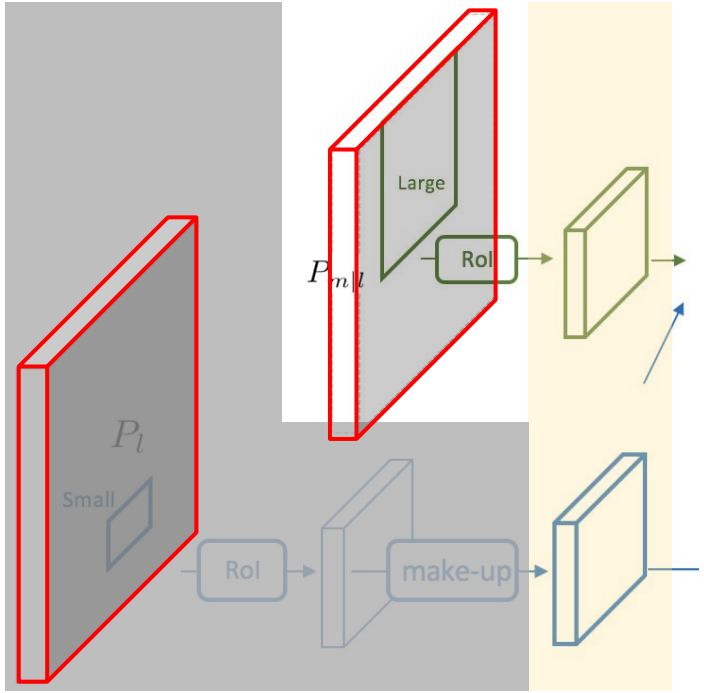
This is inappropriate.
why?



We define **small set** to be the anchors on current level and **large set** to be all anchors *above* current level.

The Feature Intertwiner - choosing optimal feature maps

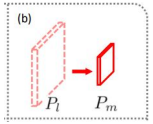
How to choose the **appropriate** maps for large objects? as input to intertwiner



Other options are

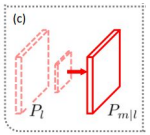
(b) use the feature maps on higher level.

$$P^{(\mathbf{large},l)} = P_m$$



(c) upsample higher-level maps to current level, with learnable parameters (or not).

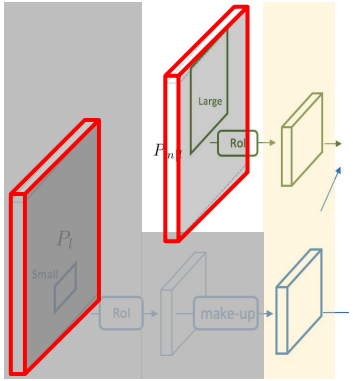
$$P^{(\mathbf{large},l)} = P_{m|l} \triangleq \mathcal{F}(P_m)$$



We will empirically analyze these later.

The Feature Intertwiner - choosing optimal feature maps

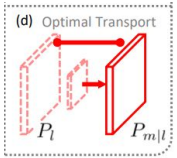
How to choose the **appropriate** maps for large objects? as input to intertwiner



Our final option is based on (c)

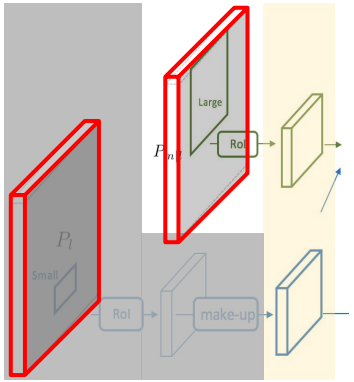
(d), **build a better alignment** between the upsampled feature map with current map.

$$P^{(\mathbf{large},l)} = \mathbf{OT}(P_l, P_{m|l})$$



The Feature Intertwiner - choosing optimal feature maps

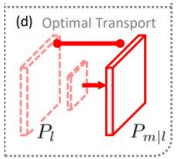
How to choose the **appropriate** maps for large objects? as input to intertwiner



Our final option is based on (c)

(d), **build a better alignment** between the upsampled feature map with current map.

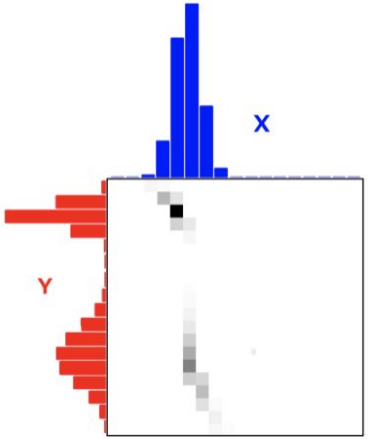
$$P^{(\texttt{large},l)} = \texttt{OT}(P_l, P_{m|l})$$



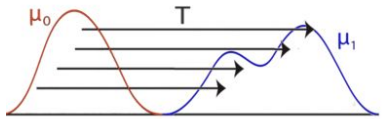
The approach is

Optimal transport (OT).

In a nutshell, OT is to **optimally** move one distribution (*P_m*|l) to the other (*P_*l).



Optimal transportation matrix

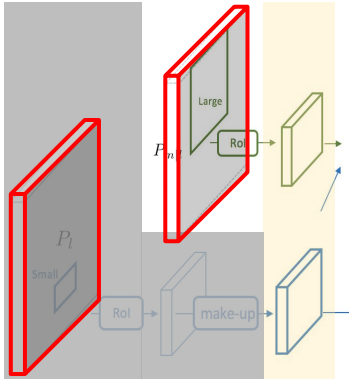


Continuous optimal transport

$$\mathcal{W}_Q(\mathbb{P}_\psi, \mathbb{P}_r) = \inf_{\gamma \in \Gamma(\mathbb{P}_\psi, \mathbb{P}_r)} \mathbb{E}\left[\int_{\mathcal{U}\times\mathcal{U}} Q(u', u) d\gamma(u', u)\right],$$

The Feature Intertwiner - choosing optimal feature maps

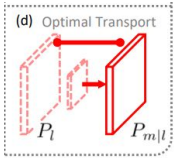
How to choose the **appropriate** maps for large objects? as input to intertwiner



Our final option is based on (c)

(d), **build a better alignment** between the upsampled feature map with current map.

$$P^{(\texttt{large},l)} = \texttt{OT}(P_l, P_{m|l})$$



The approach is

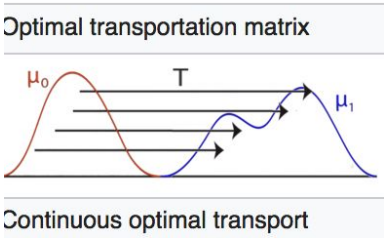
Optimal transport (OT).

In a nutshell, OT is to **optimally** move one distribution (*P_m*|l) to the other (*P_*l).

$$\texttt{OT}(P_l, P_{m|l}) \triangleq \mathcal{W}_Q(\mathbb{P}_\psi, \mathbb{P}_r) \xleftarrow{\text{discrete}} \min_{P \in \mathbb{R}_+^{C_2 \times C_1}} \langle Q, P \rangle,$$

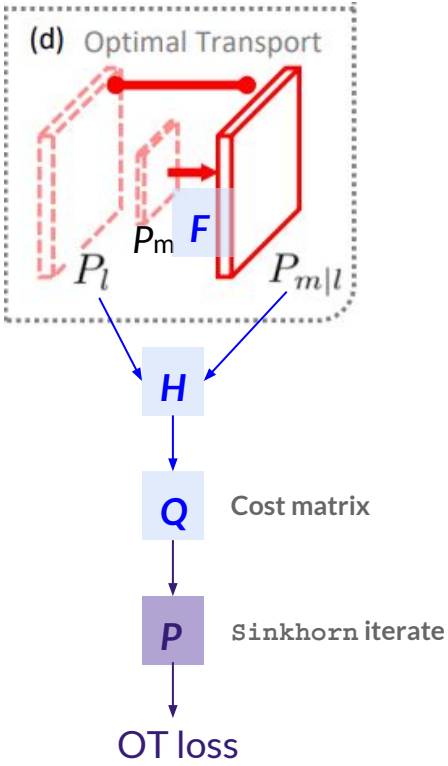
Q is a cost matrix (distance)

P is a proxy matrix satisfying some constraint.



The Feature Intertwiner - choosing optimal feature maps

How to choose the **appropriate** maps for large objects? as input to intertwiner

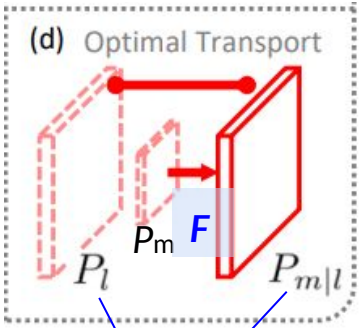


How to compute

Optimal transport (OT). $P^{(\mathtt{large},l)} = \mathtt{OT}(P_l, P_{m|l}) = \min_{P \in \mathbb{R}_+^{C_2 \times C_1}} \langle Q, P \rangle,$

The Feature Intertwiner - choosing optimal feature maps

How to choose the **appropriate** maps for large objects? as input to intertwiner



H

Q Cost matrix

P `Sinkhorn` iterate

OT loss

How to compute

Optimal transport (OT). $P^{(\texttt{large},l)} = \texttt{OT}(P_l, P_{m|l}) = \min_{P \in \mathbb{R}_+^{C_2 \times C_1}} \langle Q, P \rangle,$

Algorithm 1 Sinkhorn divergence $\mathcal{W}_Q$ adapted for object detection (red rectangle in Fig.2)

Input: Feature maps on current and higher levels, P_l, P_m
The generator network $\mathcal{F}$ and the critic unit in OT module $\mathcal{H}$ — Components

Output: Sinkhorn loss $\mathcal{W}_Q^l(P_l, P_m) = \texttt{OT}(P_l, P_{m|l})$

Upsample via generator $P_{m|l} = \mathcal{F}(P_m)$

Feed both inputs into critic $\boldsymbol{p}_l = \mathcal{H}(P_l), \boldsymbol{p}_{m|l} = \mathcal{H}(P_{m|l})$
$\forall (x, y)$ in $\boldsymbol{p}_l, \boldsymbol{p}_{m|l}$, define the ground cost $Q_{x,y} = \texttt{cosine\_dist}(\boldsymbol{p}_l, \boldsymbol{p}_{m|l})$ — H->Q

Initialize coefficients $b^{(0)} = \mathbb{1}_C$
Compute Gibbs kernel $K_{x,y} = \exp(-Q_{x,y}/\varepsilon)$
for $l = 0$ to L **do**
$a^{(l+1)} := \frac{\mathbb{1}_C}{K^\intercal b^{(l)}}, b^{(l+1)} := \frac{\mathbb{1}_C}{K a^{(l)}},$ — P
end for
Compute the proxy matrix $P^{(L)} = \text{diag}(b^{(L)}) \cdot K \cdot \text{diag}(a^{(L)})$
Compute $\mathcal{W}_Q$ based on the dot-product in Eqn. (4): $\langle Q, P \rangle$.

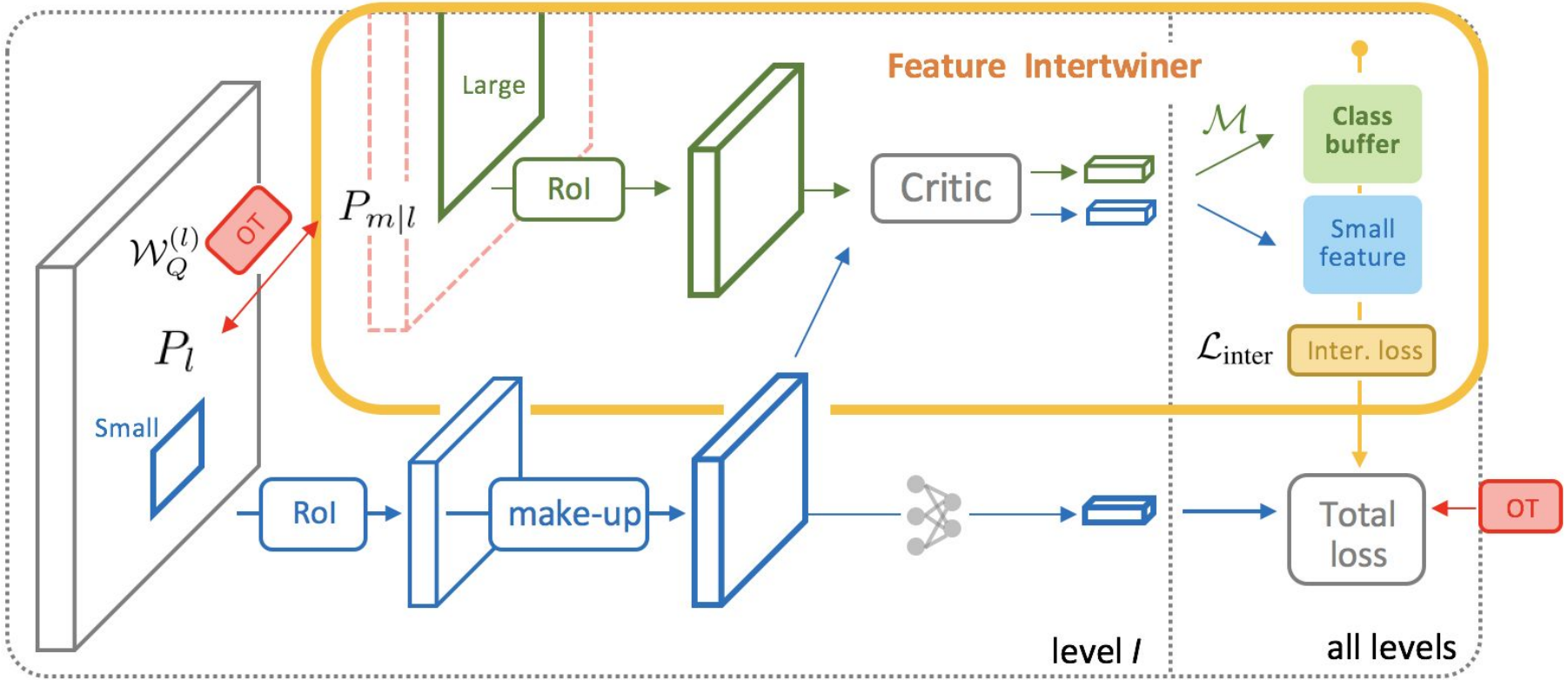
The Feature Intertwiner - choosing optimal feature maps

Hence, the final loss: $\mathcal{L} = \mathcal{L}_{\text{inter}} + \sum_l \mathcal{W}_Q^{(l)}(P_l, P_m) + \mathcal{L}_{\text{standard}},$

Why Optimal transport (OT) is better than others?

- OT metric converges while other variants (KL or JS) don't
- Provides sensible cost functions when learning distributions supported by low-dim manifolds (*p*_l and *p*_m|l)

Summary of our method



Experiments

Setup

- Evaluate our algorithm on COCO dataset
- Train set: `trainval-35k`, test set: `minival`
- Network structure: resNet-50 or resNet-101 with FPN
- Based on Mask-RCNN framework without seg. Branch
- Evaluation metric: **meanAP** under different thresholds and sizes

The rest of details are stated in Sec. 6.5 in the paper.

Ablation on module design

Table 2 in the paper
gray background is the chosen default

	proposal split	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
baseline	by RoI size	30.9	53.7	35.1	10.8	34.7	46.6
	more on higher	31.3	54.0	35.8	11.4	35.1	47.5
	default *	32.8	55.3	37.2	12.7	36.8	49.3
intertwiner	by RoI size	33.7	56.1	37.6	13.5	37.4	50.8
	more on higher	32.3	55.7	37.1	12.9	36.2	49.5
	default **	**35.2**	**57.6**	**38.0**	**15.3**	**38.7**	**51.1**
	*increase from * to***	*+2.4*	*+2.1*	*+0.8*	*+2.6*	*+1.9*	*+0.8*

(a) **Baseline and proposal assignment strategy**: intertwiner increases detection of both small and large objects compared to baseline. Putting more proposals on lower level brings more gain.

Different anchor placements

~2% mAP increase
Large objects also improve.
Why?

Does the intertwiner module work better?

Observation #1:
Feature Intertwiner Module is better than baseline.

Ablation on module design

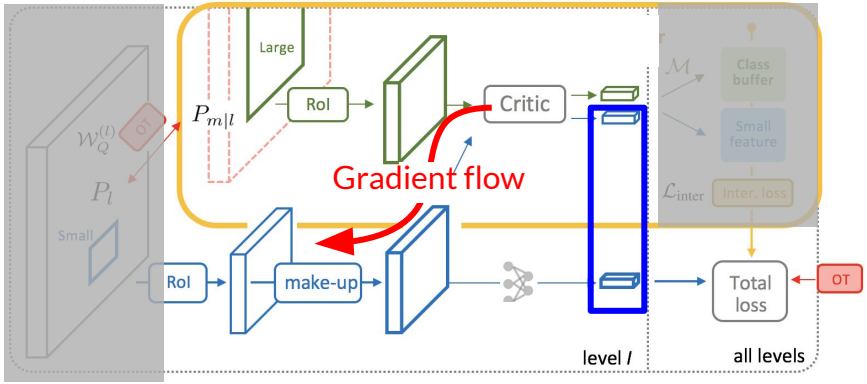
Table 2 in the paper
gray background is the chosen default

How does the intertwiner module affect feature learning?

Observation #2:
By optimizing the make-up layer; the linearly combined features would further boost performance.

	AP	AP_{50}	AP_{75}
separate	34.0	57.1	37.3
naive add		— fail —	
linear	**35.2**	**57.6**	**38.0**

(c) **Boosted detection feature source**: merging f_{critic} into the detection folllow-up pipeline increases result.



Ablation on module design

Table 2 in the paper
gray background is the chosen default

Does buffer size matter? Unified or level-based buffer?

Observation #3:
Recording all history of the large/reliable set would achieve better results (and save mem); one unified buffer is enough.

How to design the buffer?

	size/weight	AP	AP_{50}	AP_{75}
partial	2000	37.3	58.5	44.7
	15k (epoch)	38.8	59.9	46.1
all history	decay weight	39.2	60.6	45.4
	equal weight	**40.5**	**62.8**	**47.6**

(d) **Buffer choice design** (101-layer): buffer taking in all history with equal weight ensures best accuracy. Longer size in 'partial' block enhances result and yet possesses more parameters.

	yes?	AP	AP_{50}	AP_{75}
multiple $\mathcal{B}$	✓	40.58	62.83	47.62
	✗	40.54	62.81	47.61
detach b_i	✓	40.5	62.8	47.6
	✗	40.1	62.4	47.3

(e) **Workflow design** (101-layer): applying different buffers on each level barely matters; detaching b_i during back-propagation is better.

Ablation on OT unit

option	variant	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
(a) use P_l		35.1	54.9	40.7	20.2	38.3	48.5
(b) use P_m (baseline)		40.5	62.8	47.6	23.7	45.2	53.1
(c) $P_{m\|l}$	$\mathcal{F}$ bilinear	40.6	62.9	47.6	23.9	45.4	53.1
	$\mathcal{F}$ neural net*	41.3	63.5	48.5	24.6	46.3	53.8
	increase from (b) *to* (c)*	*+0.8*	*+0.7*	*+0.9*	*+0.9*	*+1.1*	*+0.7*
(d) $P_{m\|l}$	KL, $\mathcal{F}$ neural net	41.0	63.1	48.2	24.5	45.7	53.4
	l_2, $\mathcal{F}$ neural net	41.8	64.2	48.9	24.7	46.0	53.8
	optimal transport (OT)**	**42.5**	**65.1**	**49.4**	**25.4**	**46.6**	**54.3**
	biased version of OT	42.5	65.3	48.6	25.3	46.8	54.3
	increase from (b) *to* (d)**	*+2.0*	*+2.3*	*+1.8*	*+1.7*	*+1.4*	*+1.2*

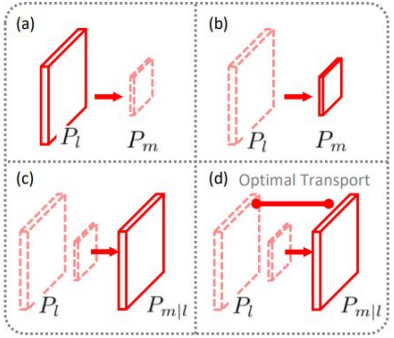
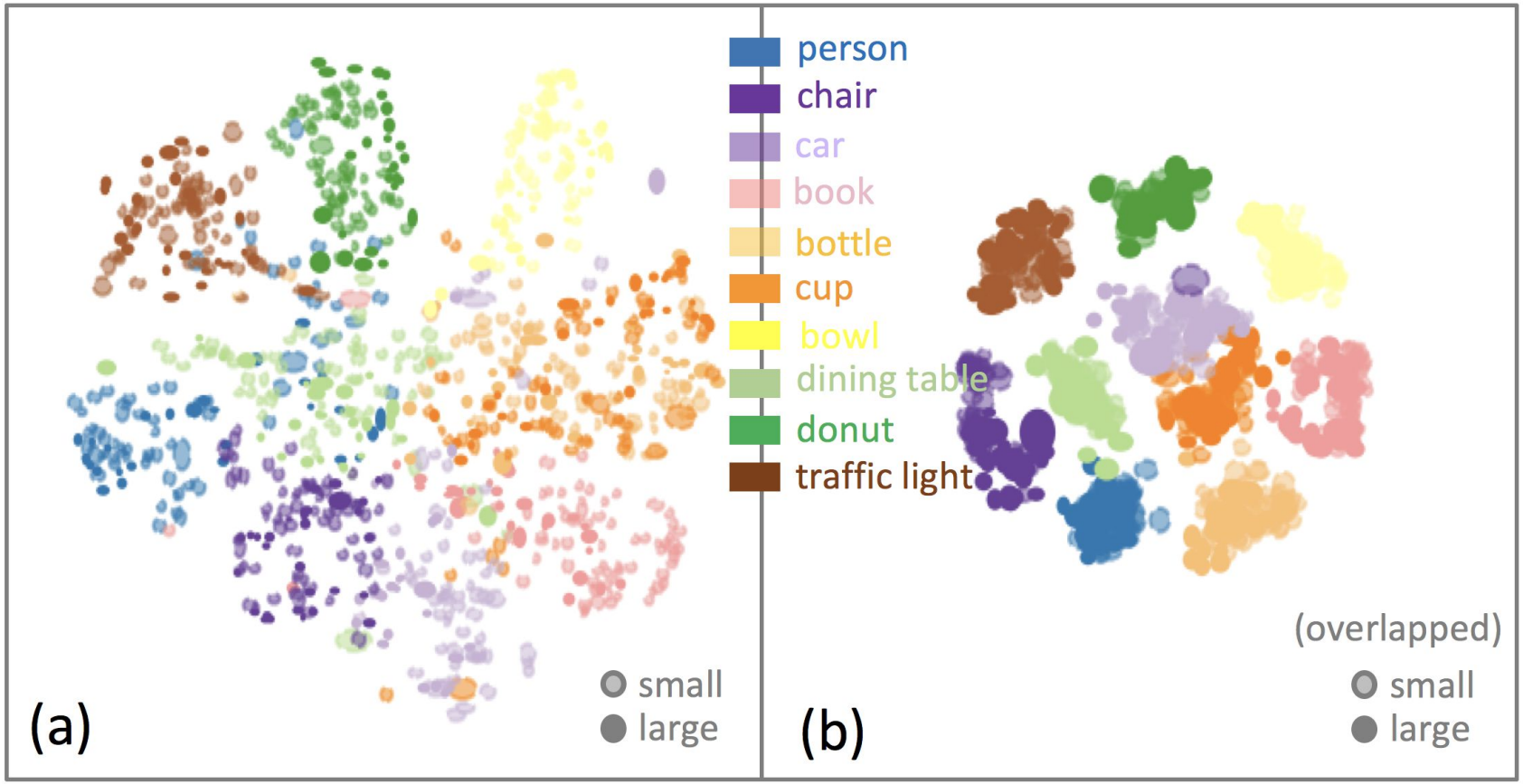


Table 1 in the paper

Different input sources for the reliable set

Visualization on samples within class



Comparison with state-of-the-arts (I)

32.8 (baseline) vs 35.2 (ours)

Most small-sized objects get improved!

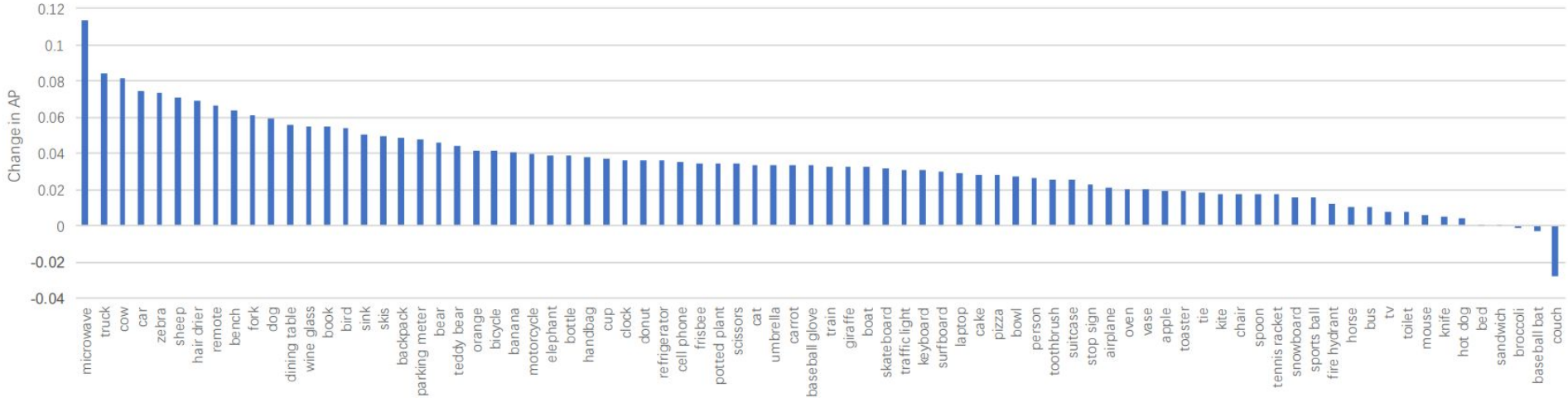
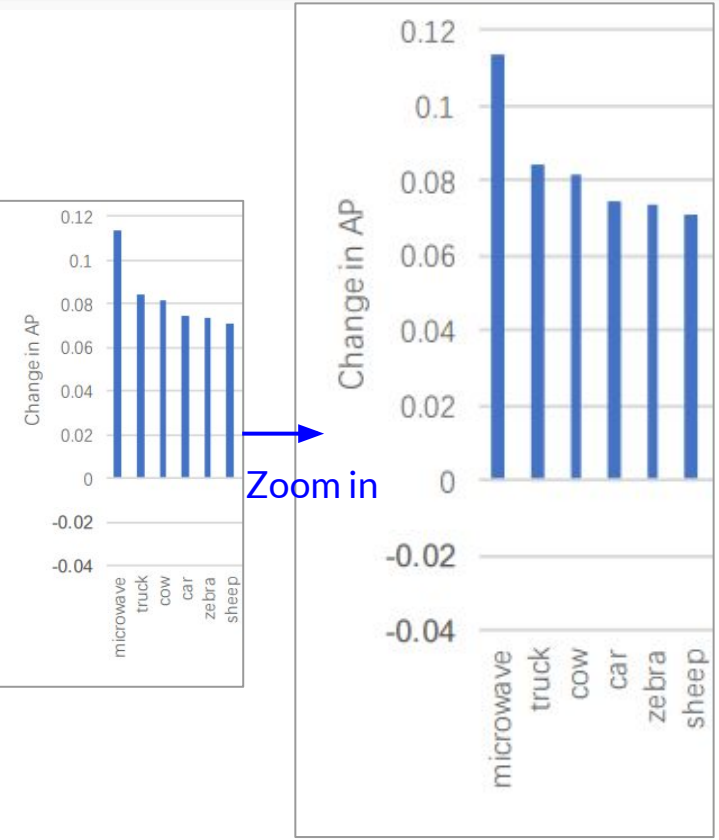


Figure 4 in the paper

Improvement *per category* after embedding the feature intertwiner

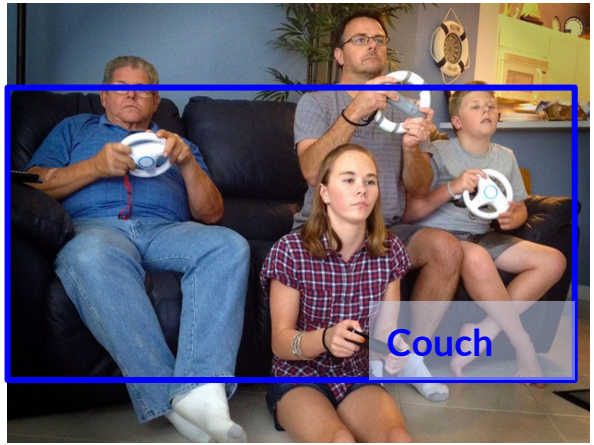
Comparison with state-of-the-arts (I)



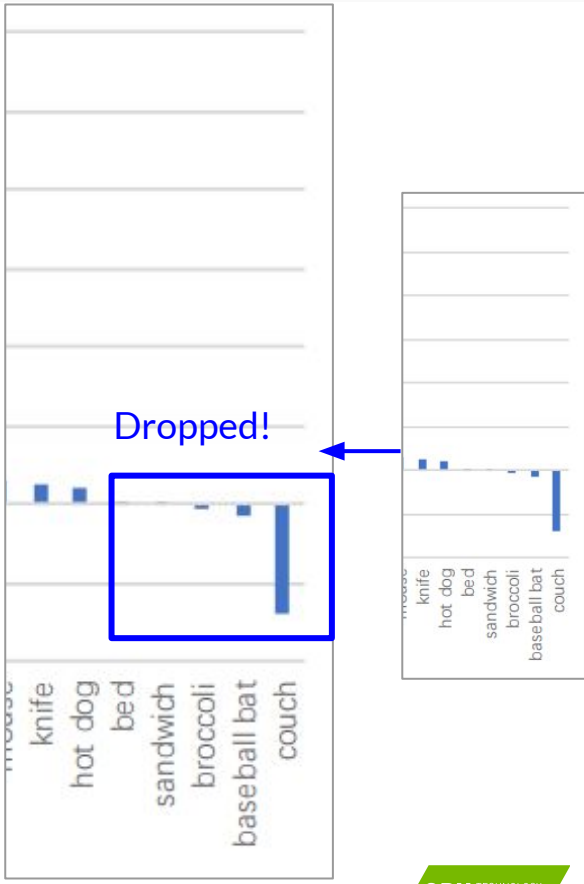
The most distinctive improvements are
`Microwave, truck, cow, car, zebra`

Comparison with state-of-the-arts (I)

Some categories witness a drop of performance
Couch, baseball bat, broccoli



The feature set of large couch is **less** accurate due to noises (of other classes).



Comparison with state-of-the-arts (II)

		backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
	One-stage detector							
	YOLOv2 (Redmon & Farhadi, 2016)	DarkNet-19	21.6	44.0	19.2	5.0	22.4	35.5
SSD	SSD513 (Liu et al., 2015)	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
SSD	DSSD513 (Fu et al., 2017)	ResNet-101-DSSD	**33.2**	53.3	35.2	13.0	35.4	51.1
	Two-stage detector							
Fast-RCNN variants	F-R-CNN+++ (He et al., 2016)	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Fast-RCNN variants	F-R-CNN w FPN (Lin et al., 2017a)	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Fast-RCNN variants	F-R-CNN by G-RMI (Huang et al., 2017)	Incept.-ResNet-v2	34.7	55.5	36.7	13.5	38.1	52.0
Fast-RCNN variants	F-R-CNN w TDM (Shrivastava et al., 2016)	Incept.-ResNet-v2-TDM	**36.8**	57.7	39.2	16.2	39.8	52.1
	R-FCN (Dai et al., 2016)	ResNet-101	29.9	51.9	-	10.8	32.8	45.0
Same backbone	Mask RCNN (He et al., 2017)	ResNet-101-FPN	38.2	60.3	41.7	20.1	41.1	50.2
Same backbone	RetinaNet (Lin et al., 2017b)	ResNet-101-FPN	**39.1**	59.1	42.3	21.8	42.7	50.2
	Mask RCNN, updated in (He et al., 2017)	ResNetX-101-FPN	43.5	65.9	47.2	-	-	-
Proposed	**InterNet** (ours)	ResNet-101-FPN	42.5	65.1	49.4	25.4	46.6	54.3
Proposed	**InterNet** (ours) multi-scale	ResNet-101-FPN	**44.2**	67.5	51.1	27.2	50.3	57.7

Table 4 in the paper

Single-model performance (bounding box AP)

This work is published at **ICLR 2019**

Paper:

https://openreview.net/forum?id=SyxZJn05YX

Code:

https://github.com/hli2020/feature_intertwiner

Check out our poster at GTC!

P9108

AI/Deep Learning Research

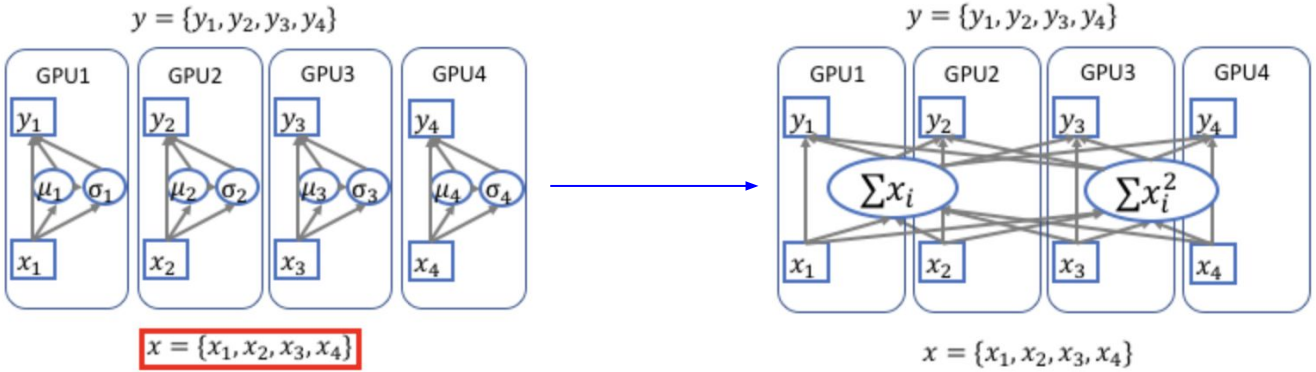
Near the gear store

3. Detection in Reality

Practical issues on multi-GPUs

1. Batch normalization

Standard Implementations of BN in public frameworks (suck as Caffe, MXNet, Torch, TF, **PyTorch**) are **unsynchronized**, which means that the data are normalized **within each GPU**.



Synchronized BN

https://hangzhang.org/PyTorch-Encoding/notes/syncbn.html

Practical issues on multi-GPUs

1. Batch normalization

Does it matter? As long as bs on each GPU is not too few, unsynchronized BN is ok.

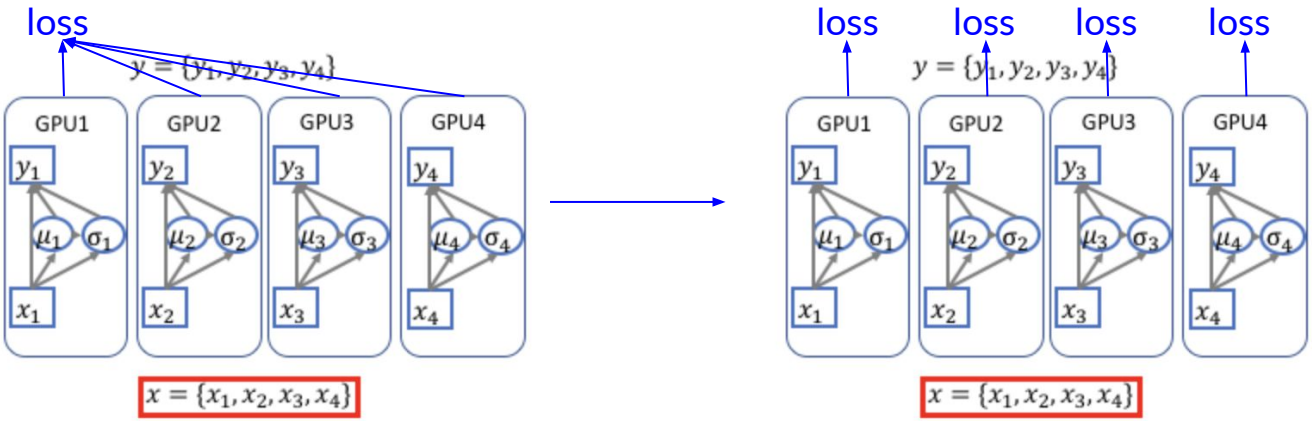
Note that bs in the “deeper” part is the # of RoIs/boxes on each card;
Batch size in the backbone is the # of image!

Another rule of thumb: fixed BN in the backbone when finetune the network on your task.

Practical issues on multi-GPUs

2. Wrap up the loss computation into `forward()` on each card

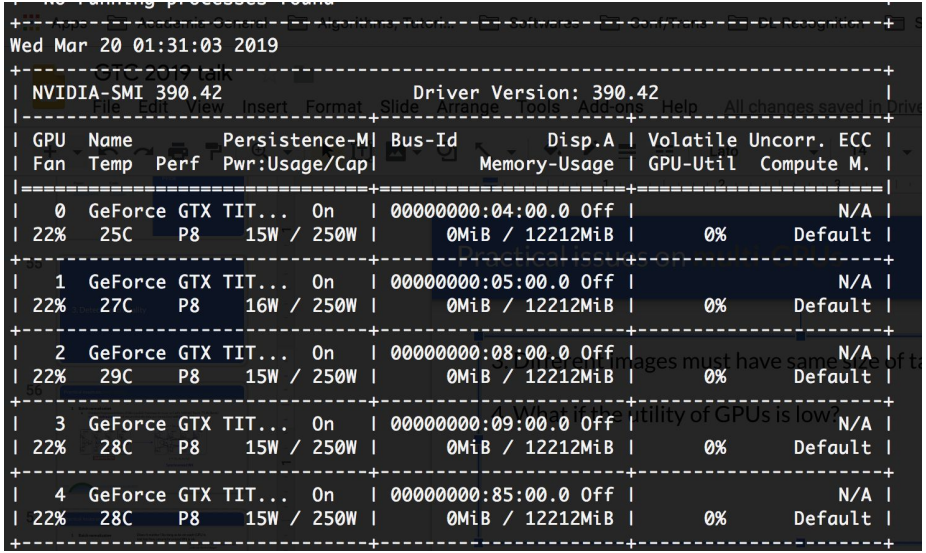
Otherwise GPU 0 would take too much memory in some cases, causing mem imbalance and decrease utility of other GPUs.



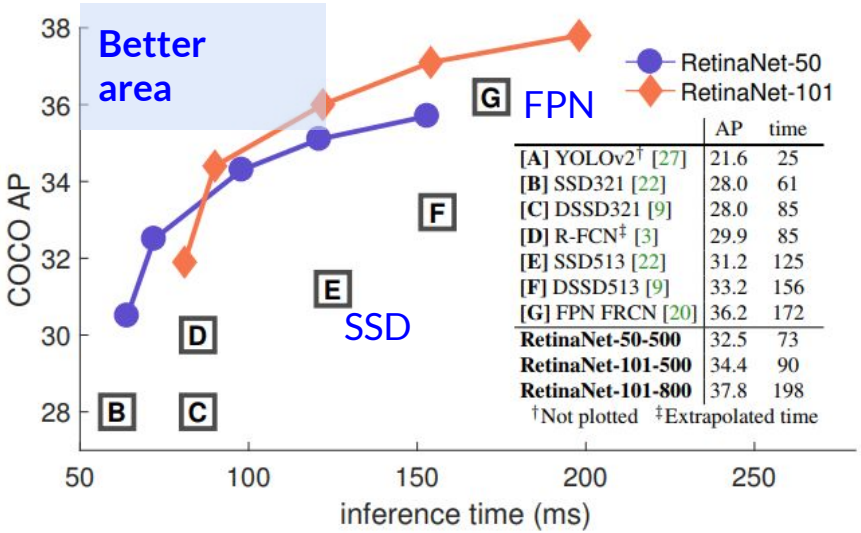
Practical issues on multi-GPUs

3. Different images must have same size of targets as input

4. What if the utility of GPUs is low?
   - `Dataloader` is slow
   - Move op. to `Tensor`
   - …
   - Or change to another workstation
     - (often during inference, utility is low)



Trade-off between accuracy and efficiency



	AP	time
[A] YOLOv2† [27]	21.6	25
[B] SSD321 [22]	28.0	61
[C] DSSD321 [9]	28.0	85
[D] R-FCN‡ [3]	29.9	85
[E] SSD513 [22]	31.2	125
[F] DSSD513 [9]	33.2	156
[G] FPN FRCN [20]	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	37.8	198

†Not plotted ‡Extrapolated time

Additional model capacity increase in our method:

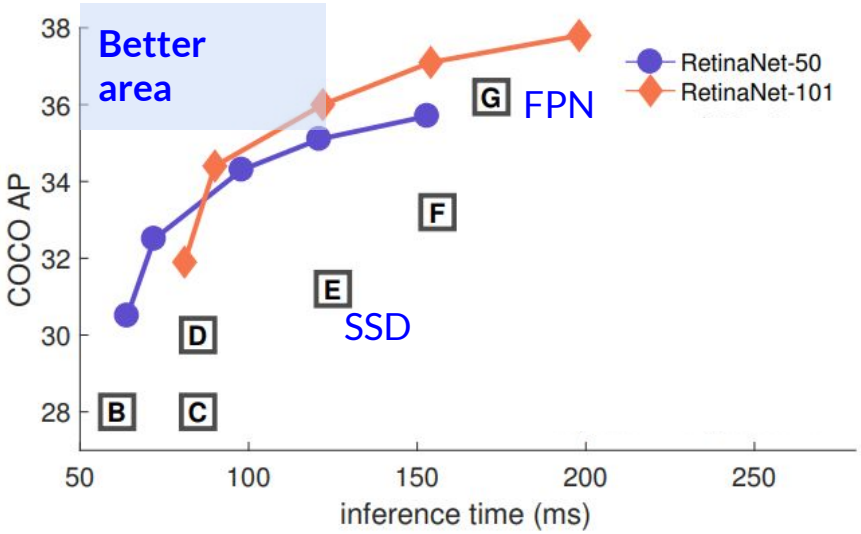
- Critic/make-up layers
- Buffer
- OT module

But these new designs only have light-weight effect.

Trade-off between accuracy and efficiency

InterNet (42.5)

Mask-RCNN (39.2)



More facts:

Training: 8 GPUs, batch size=8, **3.4 days**

Mem cost **9.6G/gpu**, baseline 8.3G

Test (input 800 on Titan X):

325 ms/image, baseline 308 ms/image

4. Future of Object Detection

Any alternatives? to abandon current anchor-based pipeline

Idea:

Current solution are all based on anchors (one-stage or two-stage).
Is bounding box really accurate to detector all objects?

How about detect objects **using bottom-up approaches? Like pixel-wise segmentation?** In this way, we can walkaround the box detection pipeline.

Take-away Messages

1. Object detection is the **basic and core** task of other high-level vision problems.
2. **Feature engine** (backbone) and **detector design** (domain knowledge) are important.
3. ***Beyond*** current pipeline (dense anchors):
   solve detection via **bottom-up approaches** or **3D structure of objects**.
4. ***Beyond*** detection only - one model to learn them *all*:
   detection, segmentation, pose estimation, captioning,
   zero-shot detection, curriculum learning, ...

Thank you! Questions?

Slides at: http://www.ee.cuhk.edu.hk/~yangli/ twitter **@francislee2020**

Email: yangli@ee.cuhk.edu.hk

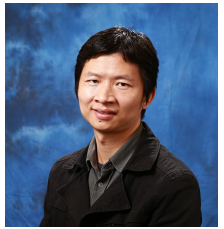
Collaborators:

Yu Liu

Bo Dai

Xiaoyang

Shaoshuai



Wanli

Xiaogang