Fine-tuning Detection

Contents

[1. Fine-tuning Strategy (Supervised) 2](#_Toc10320196)

[a) Introduction to Detection-Models 2](#_Toc10320197)

[b) Details about Detection-Models 3](#_Toc10320198)

[Mask RCNN (CVPR 2017) 3](#_Toc10320199)

[RetinaNet (ICCV 2017) 3](#_Toc10320200)

[Yolov3 (arXiv 2018) 4](#_Toc10320201)

[M2Det (AAAI 2019, the latest SSD-net) 5](#_Toc10320202)

[c) Loss visualization and analysis 5](#_Toc10320203)

[d) Problems in Fine-tuning 6](#_Toc10320204)

[Class imbalance 6](#_Toc10320205)

[e) Fine-tuning Strategy (yolov3) 7](#_Toc10320206)

[2. Fine-tuning Strategy (Unsupervised) 7](#_Toc10320207)

[a) Definition: 7](#_Toc10320208)

[b) Assumptions: 8](#_Toc10320209)

[c) Pipeline: 8](#_Toc10320210)

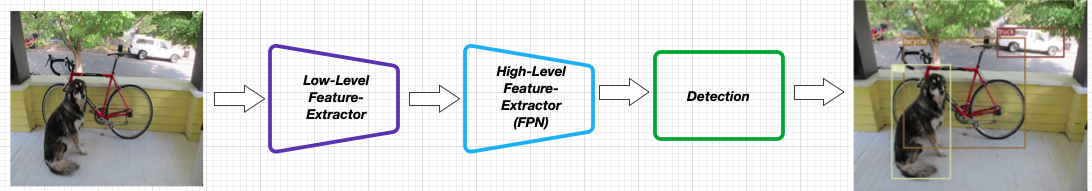
[d) Demo (use yolov3’s detections on low threshold to fine-tune itself): 9](#_Toc10320211)

[3. Code repositories: Detection-Fine-tuning-API 9](#_Toc10320212)

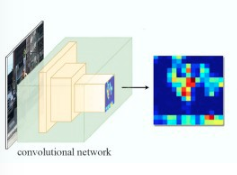
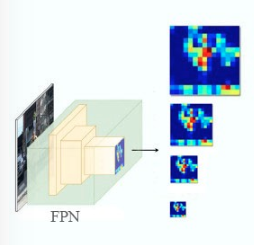
# Fine-tuning Strategy (Supervised)

## Introduction to Detection-Models

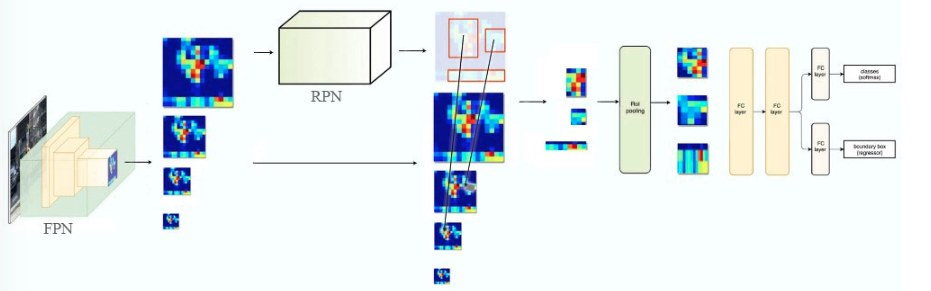
All detection-models consist three components: low-level feature-extractor, high-level feature extractor and detectors.



1. Low-level feature-extractor: it targets to extract the low-level features from images as same as classification and is often pre-trained on ImageNet (We often use backend to denote this in detection-models).
2. High-level (or Multi-scale) feature-extractor: it focuses on extracting features that are relevant to detection (i.e, Fine-grained features of ROIs/BBs). The common structure is Feature Pyramid Network (FPN) and this architecture is more capable to capture the object’s information, both low-level and high-level.

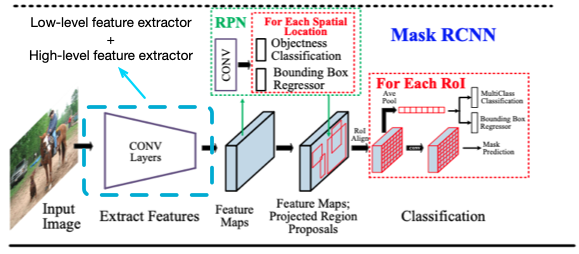
1. Detectors: it targets to generate ROIs’ locations and class-scores. There are two main architectures in Detectors:
2. Two-stage Detectors: they use RPN (region proposal network) to extract all bounding boxes first and then classify each bounding box that has a large score (Faster RCNN, Mask RCNN, etc). In other words, RPN filters bounding boxes that are background.



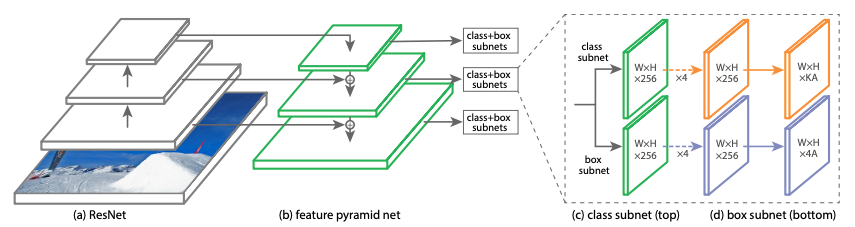
1. One-stage Detectors: they extract bounding boxes and classify them simultaneously (SSD, Yolo, etc). Because they don’t use RPN to generate candidates first, detectors need to process all bounding boxes in the previous feature maps.

## Details about Detection-Models

### Mask RCNN (CVPR 2017)

1. Structure:
2. Pipeline:
   1. It uses CNN layers (Low-level and high-level feature-extractors) to extract features from images (images -> CNN -> feature maps).
   2. It uses RPN (Region Proposal Network) to extract all bounding-box candidates. Each candidate has the corresponding coordinates (xywh) and the class-score (background or foreground).
   3. It uses FC (fully connected layer) to classify each candidate whose class-score is foreground.

### RetinaNet (ICCV 2017)

1. Structure:
2. Pipeline:
   1. It uses ResNet to extract low-level features from images.
   2. It uses FPN to extract high-level features from the low-level feature maps.
   3. It uses two FC layers to extract bounding boxes and class-scores simultaneously.
3. Contribution: Use Focal Loss to solve the imbalance problems in one-stage detection.

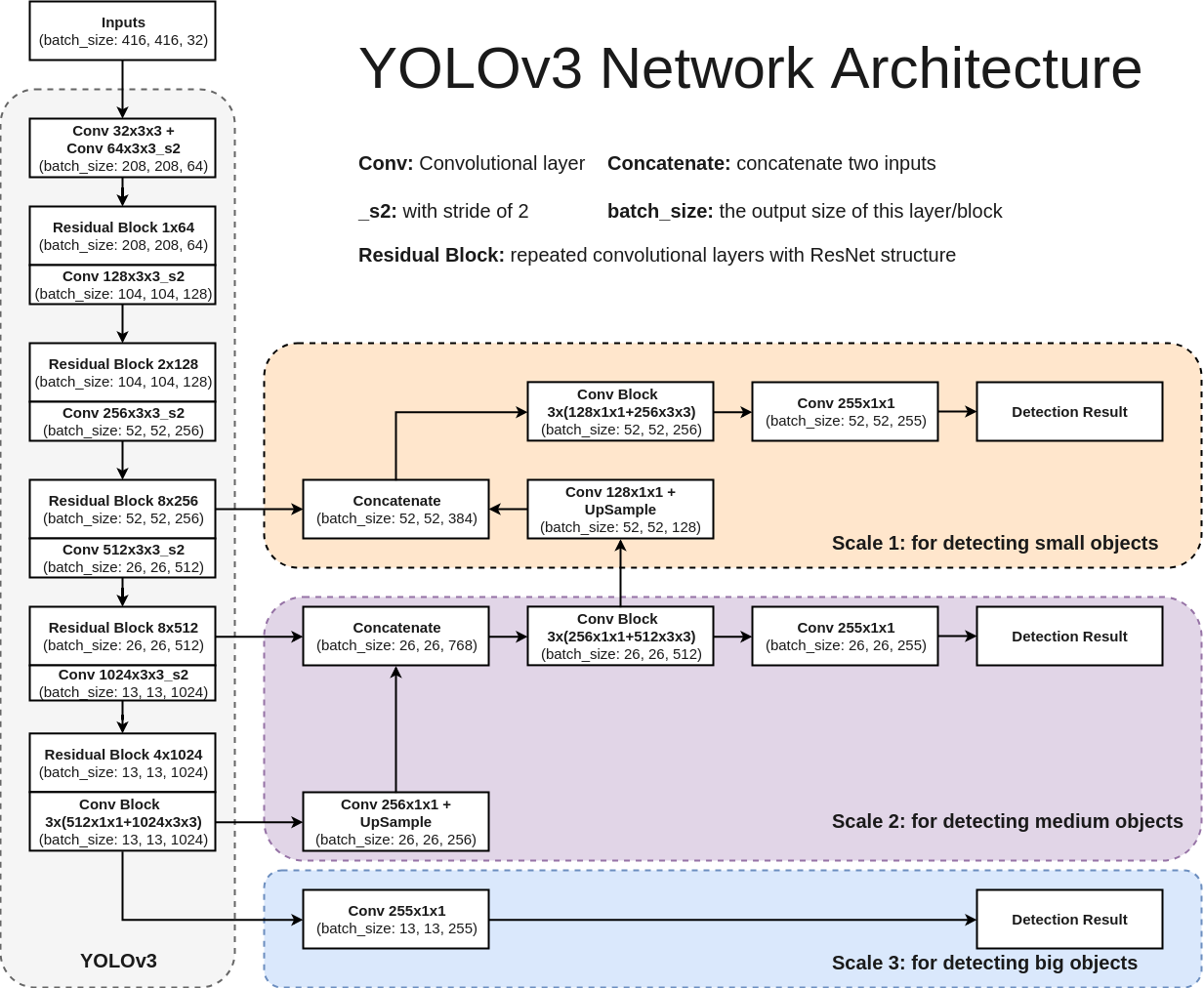


pt denotes the probability of the candidate is target class.

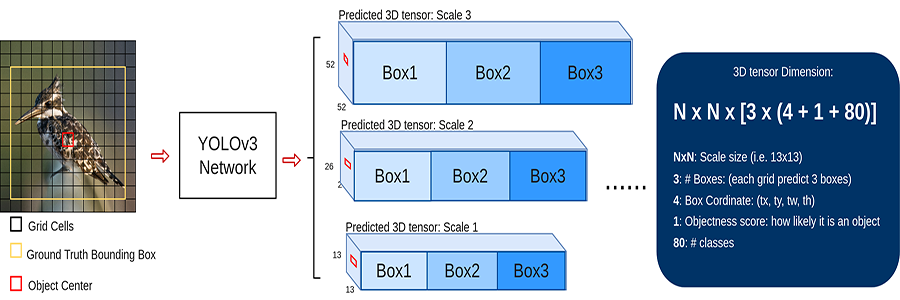
* Hard Examples: pt is small and (1-pt) -> 1, the loss is unaffected.
* Easy Examples: pt is large and (1-pt) -> 0, the loss for well-classified examples is down-weighted.

### Yolov3 (arXiv 2018)

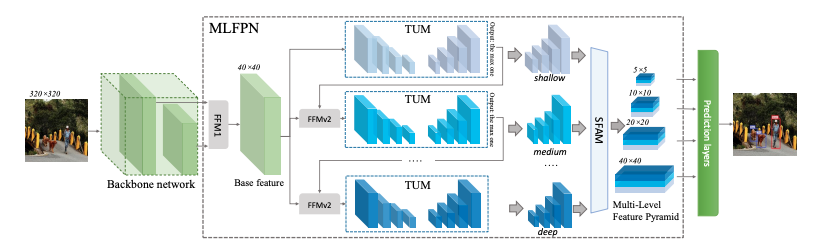
1. Structure:



1. Process Flow:

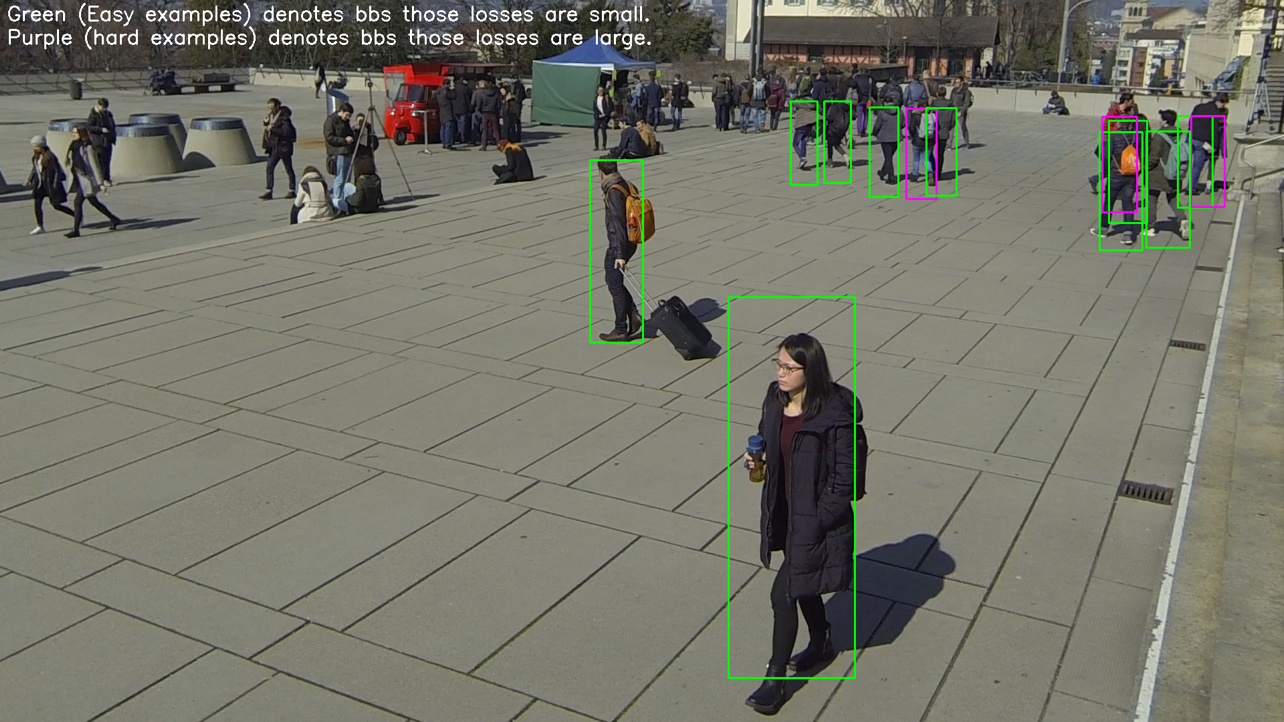


### M2Det (AAAI 2019, the latest SSD-net)

1. Structure:

## c) Loss visualization and analysis

To analysis the relationship between loss and bounding boxes, we plot the bbs that have small loss with green rectangle and bbs that have large loss with purple rectangle in pedestrian detection (dataset from WildTrack). We find that the hard bounding boxes are often having occlusion and hard to detect by models.

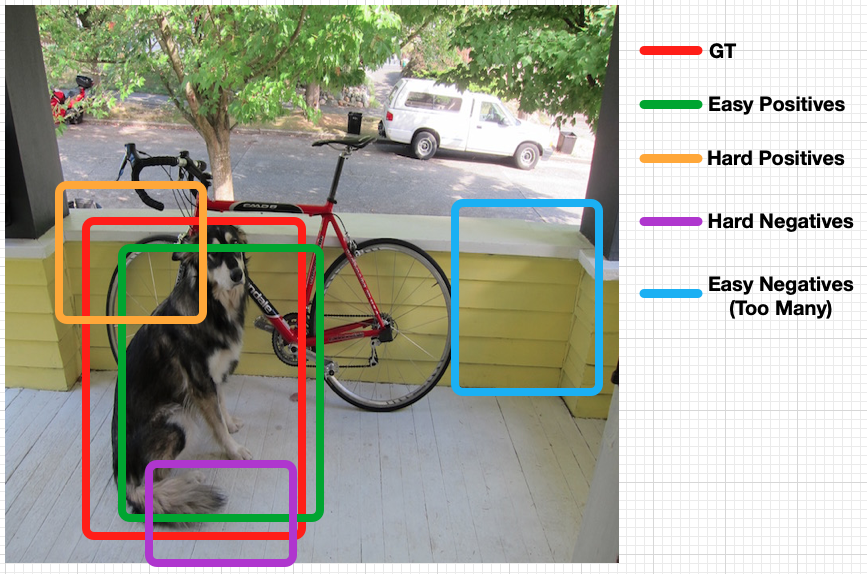


1. Insight-1: Easy examples are much more than hard examples.
2. Insight-2: Hard examples contributes more to training because their losses are large.

## d) Problems in Fine-tuning

### Class imbalance

1. Problem Definition: in training, we must decide how many detections for back-progation because detection-models usually generate too many bounding boxes (many nosie) and the RoIs are overlapping with the small number of detections. If we use all detections as training sets., too many simple negatives or simple positives will influence the performance of training. Therefore, we need to design strategies to resolve the imbalance problem between negative and positive samples.



1. Solutions: there are three main methods: Online hard example mining (OHEM), Focal Loss and Using positives only.
   1. OHEM (Mask RCNN and Faster RCNN):

To get more efficient backpropagation on bounding boxes, we adopt the online hard example mining (OHEM) to training. In other words, we will sort all bounding boxes (positives and negatives) by loss in mini-batch and select B% bounding boxes that have the highest loss. Backprogation is performed based on the selected bounding boxes. Details can be referred to [paper](https://arxiv.org/pdf/1604.03540.pdf) and this method is often used to training two-stage detection-models.

* 1. Focal Loss (RetinaNet):



Details can be referred to [paper](https://arxiv.org/pdf/1708.02002.pdf).

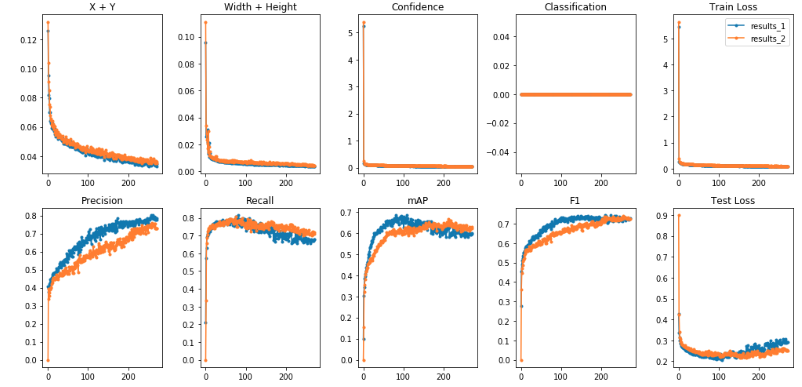
* 1. Using positives only (Yolov3 and M2Det):

In Yolov3, it only chooses the most suitable positive bounding boxes for backpagation. When they get the ground-truth bb1, they will search the image and find the detection bb2 that is the most possible candidate for bb1. Finally, they compute the loss between bb1 and bb2. This means each ground-truth only has one detection as candidate. Thus, there are not existing too many negatives.

## e) Fine-tuning Strategy (yolov3)

Inspired by the transfer learning and gradient vanishing problem, I only fine-tuned the detector when the loss is small. I compare the performance between two strategies.

1. Fine-tuning the whole structure.
2. Setting a dynamic loss-threshold in each epoch. In each mini-batch,
   1. If the loss of the mini-batch is small than the loss-threshold, we only update the detection.
   2. If the loss of the mini-batch is larger than or equal to the loss-threshold, we will update the whole structure.



Other hyper-parameters can be referred to the [code](https://github.com/jacksonly/Detection-Fine-tuning-API/blob/master/yolov3/train.py).

In conclusion, updating the whole structure can get better performance and fast convergence.

# 2. Fine-tuning Strategy (Unsupervised)

## Definition:

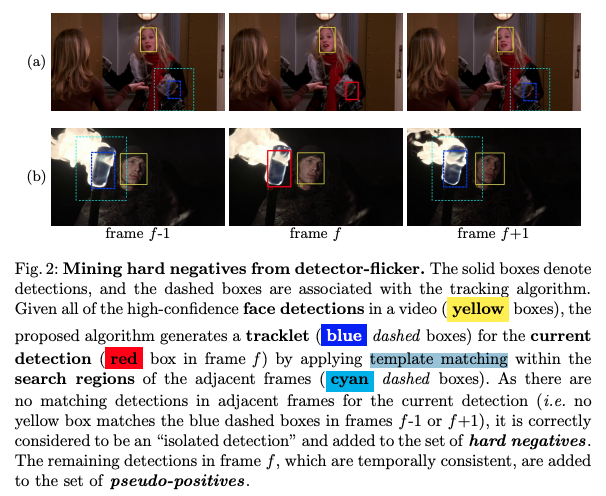
Use the pre-trained model to adapt the new domains of unlabeled data.

## Assumptions:

Unsupervised fine-tuning detection-models needs a large number of unlabeled data available (i.e, unannotated videos).

## Pipeline:

1. Self-Labeling: use the pre-trained model to inference the unlabeled data and get the output (bbs). Sometimes, detection with low threshold may be more valuable. [](https://www.dropbox.com/s/rt66oku7k9kwlyw/threshold.avi?dl=0)
2. Using tracking to refine labels:
   1. Use tracking to find the hard positives: when the car is detected in frame-[i] but isn’t detected in frame-[i+1], we may label this car by tracking.
   2. Use tracking to filter the hard negatives: we can use tracking to get the location of bbs (detected in the frame-[i]) in the frame-[i-1] and frame-[i+1].



1. Fine-tuning on pseudo-labels.

## d) Demo (use yolov3’s detections on low threshold to fine-tune itself):

[](https://drive.google.com/drive/folders/1uxgUBemJVw9wZsdpboYbzUN4bcRhsuAI)

# 3. Code repositories: [Detection-Fine-tuning-API](https://github.com/jacksonly/Detection-Fine-tuning-API#Details-about-Detection-models)