UNIVERSITY OF AMSTERDAM

MSc ARTIFICIAL INTELLIGENCE
MASTER THESIS

Betting on Proposals: Multi-Scale Gambler for Object Detection

by
MELIKA AYOUGHI
12224855

August 20, 2020

48 ECTS
November 2019 — August 2020

Supervisor:
Dr. Pascal Mettes
Dr. Yu Wang
Dr. Marco Manfredi

Assessor:
Prof. Dr. Cees Snoek

TOMTOM

Abstract

In the standard one-stage object detection, the loss is calculated per anchor by ignoring high-level structural information and prediction inter-dependencies. In this work, we rethink object detection training to enforce the *bird's-eye perspective* on predictions. We propose a Full-Scale Gambler architecture that is inspired by Gambling networks from semantic segmentation. The Gambler receives the full prediction map from the one-stage detector, plus the corresponding image and outputs a weight per anchor to focus detector training on possibly inaccurate predictions. The detector is involved in an adversarial min-imax game with the Gambler. Our method yields minimal gain in detection performance on benchmarks like COCO and LVIS, however, introducing marginal improvements for long-tailed categories. We perform extensive quantitative and qualitative analysis and identify several potential problems and recommend future directions that could further benefit Gambling-based object detection.

Contents

Introduction

Object detection is one of the most fundamental and challenging problems in computer vision and has been widely studied in the past decade. Object detection aims to precisely localize objects and assign each object instance the corresponding class label in an image. In the standard one-stage object detection, the objective function is computed per anchor, irrespective of anchor inter-dependencies. These objective functions disregard the possibility of incorporating high-level structural consistencies in an image and missing the bird's-eye perspective. Gambling networks [1] have been introduced with this scope for semantic segmentation. They proved to decrease inconsistencies in semantic segmentation predictions by capturing the top-down structure of the scene, which helped generalize beyond the noise and lead to *learned hard negative mining*. In this work, we propose to apply Gambling networks to the object detection setting.

We would like to answer the following research questions:

1. How can we incorporate Gambling-based adversarial training in object detection?

2. What are the effects of the Gambler on object detection?

An adaptation of the original Gambling net to object detection is not trivial for several factors:

1. The most challenging problem in object detection is the foreground-background imbalance, which is identified as the main obstacle in poor object detection performance [2]. The output of the detection network consists mostly of backgrounds and few foregrounds. This highlights another major difference between segmentation and detection tasks, where the output of segmentation is dense, while detection is sparse.

2. Most state-of-the-art one-stage detectors compute predictions at multiple levels of a feature hierarchy, to predict objects at different scales. In other words, for each pixel location, feature maps from different levels are combined to fully capture the semantic of the object centered at that location.

3. Unlike segmentation, where each output prediction can be easily mapped to an input location, the input and output spaces of object detectors differ. Box predictions are encoded as regressed coordinates with respect to the image frame or the nearest anchor and need to be decoded in order to map them on the input image. This poses an additional challenge for the Gambler.

We propose a Full-Scale Gambler architecture that generalizes from gambling-based segmentation to object detection. We perform thorough ablation studies to validate the design choices. The full-scale Gambler receives the complete hierarchical prediction map from a one-stage detector, and the corresponding image and outputs a weight per anchor to focus training on likely incorrect predictions. During training, the detector is involved in an adversarial minimax game with the Gambler.

We present an extensive quantitative and qualitative evaluation on two public benchmarks, COCO [3] and the more challenging long-tailed dataset, LVIS [4]. The improvements in the overall mAP are, however, minimal. For this reason, we perform an in-depth investigation of the results. We show that although the full-scale Gambler for object detection is not flexible to noise or overfitting, it marginally improves the long-tailed classes. This analysis helps us to point to potential problems in the formulation, and present future directions that we believe might help to improve gambling-based object detection.

Background and Related Work

Object Detection

Object detection is a fundamental visual recognition task in computer vision, widely studied in the past decade. Object detection aims to find objects of specific categories and localize them with a bounding box in a given image, assigning each bounding box the corresponding object category label.

Classically the pipeline of object detection is divided into three steps: 1) proposal generation, 2) feature vector extraction, and 3) classification. In proposal generation, the goal is to localize regions of interest (ROI) in the image which are likely to contain objects. Typically this is achieved by scanning the whole image in a sliding window fashion [5]–[9]. In order to capture information on multiple scales and with different aspect ratios of objects, input images are resized to different scales, and multi-scale sliding windows are used to slide through these images. In the feature vector extraction step a fixed-length feature vector is computed from each ROI. They are constructed in a way to capture discriminative semantic information. This feature vector is often one or a combination of low-level visual descriptors such as SIFT [10], Haar [11], HOG [8], or SURF [12]. For the classification step a classification model is trained to predict the correct object category for a given feature vector. One commonly utilized classifier are SVMs [13]. Further, different classification techniques such as bagging [14], cascade learning [9], and AdaBoost [15] were used in the region classification step, which led to further improvements in detection accuracy. However, most of the successful traditional methods for object detection focused on carefully designing better feature descriptors to obtain embedding for a region of interest. From 2008 to 2012, the progress on Pascal VOC [16] with traditional methods had become incremental, with only minor gains, which showed the limitations of traditional detectors.
These limitations include:

1. A vast number of redundant proposals leading to false positives.

2. Hand-crafted feature descriptors based on low-level visual signals [17], [18], [12], making it challenging to capture representative semantic information in complex contexts.

3. Separate design and optimization of detection pipeline, not leading to a globally optimal solution.

In 2012, after the success of applying deep convolutional neural networks for image classification [19], [20], object detection also achieved remarkable progress based on deep learning techniques [21], [22]. The new deep learning based algorithms showed promising results compared to traditional methods. CNNs generate hierarchical deep features from low-level to high-level information. Those features are learned end-to-end with one fixed training objective and no necessity of hand-crafting features. By learning the deep features

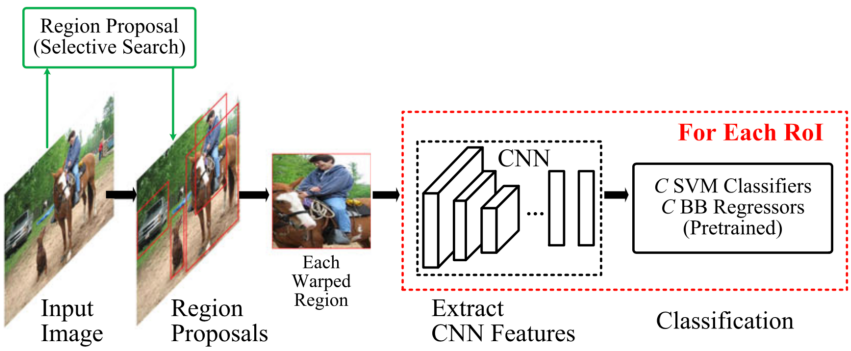


Figure 1: An overview of the R-CNN architecture [29]

directly from data the features are optimized to be discriminative with regards to the used classifier or object detector.

Currently, deep learning based object detection frameworks can be primarily divided into three families: (1) two-stage detectors [23], [24], [25] (2) anchor-free detectors, and (3) one-stage detectors [26], [27], [2].

2-stage (Sparse) Detectors

Two-stage detectors split the detection objective into two stages: In the first stage a proposal generator generates a sparse set of proposals and extracts features from each proposal region. In the second stage a classifier predicts the object category for each proposal region.

R-CNN [22] is a pioneering two-stage object detector. For each image, R-CNN generates a sparse set of proposals (around 2,000 proposals) via Selective Search [28], which is designed to reject regions that can easily be identified as background. Each proposal is cropped and resized into a fixed-size region and encoded into a feature vector, followed by a one-vs-all SVM classifier. Finally, the bounding box regressors are learned using the extracted features as input in order to make the original proposals tightly bound the objects.

However, in R-CNN, the feature proposals were extracted separately (i.e., the computation was not shared), making it extremely time-consuming for training and testing. Besides, the three steps of R-CNN were independent components that could not be optimized in an end-to-end manner. Selective Search relied on low-level visual cues and thus struggled to generate high-quality proposals in complex contexts.

To accelerate R-CNN and learn more discriminative features, instead of cropping region proposals and feeding them into the CNN model separately, Fast R-CNN [24] computes the feature map from the whole image using a deep convolutional network and extracts fixed-length feature vectors on the feature map. It also uses the ROI Pooling layer to extract region features. ROI pooling layer only takes a single scale to partition the proposal into a fixed number of divisions and backpropagates errors through the CNN. After feature extraction, feature vectors are fed into a sequence of fully connected layers before two output layers: classification head (cls) and regression head (reg). The classification head is responsible for generating SoftMax probabilities over C+1 classes (C classes plus one background class), while the regression head encodes 4 real-valued parameters to refine bounding boxes.

Despite the progress in object detectors, the proposal generation step still relies on traditional methods such as Selective Search. To address this problem, Faster R-CNN [25] relies on the fully convolutional Region Proposal Network(RPN). RPN takes an image of

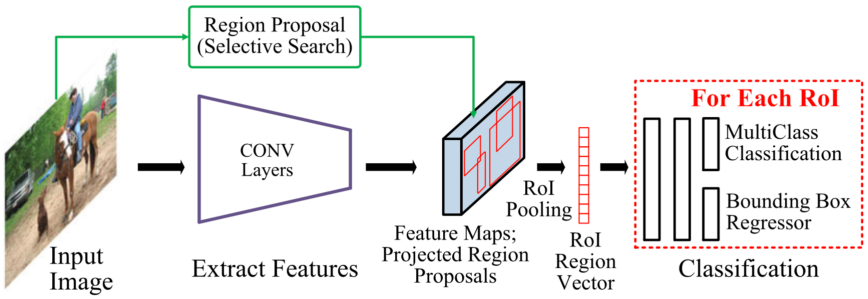


Figure 2: An overview of the Fast R-CNN architecture [29]

arbitrary size and generates a set of object proposals on each position of the feature map in a sliding window manner. The feature vector is then fed into two output branches, the object classification layer classifies a region proposal as an object or background, and the bounding box regression layer. These results are then fed into the final layer for the actual object classification and bounding box localization like in Fast R-CNN.

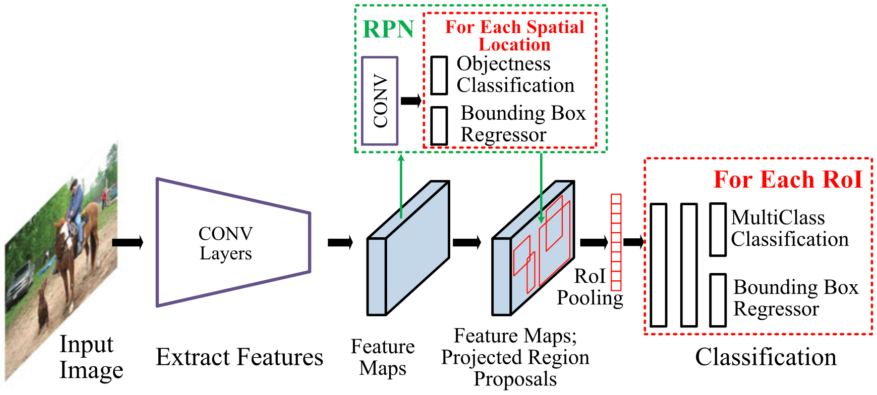


Figure 3: An overview of the Faster R-CNN architecture [29]

Two-stage detectors usually achieve better detection performance, while compensating for memory usage.

1-stage (Dense) Detectors

Unlike two-stage, one-stage detectors make a direct categorical prediction of objects on each location of the feature maps, bypassing the proposal classification step. They consider all positions on the image as the center of bounding boxes with different scales and aspect ratios (these boxes are called anchors) as potential objects and try to classify each anchor as either background or a target category.

One of the first successful one-stage detectors is called YOLO [26] (You Only Look Once). YOLO solves object detection as a regression problem by spatially dividing the whole image into a fixed number of grid cells (e.g., a 7×7 grid). Each cell is considered a proposal. Which based on the original implementation, it could contain the center of up to two objects. For each cell, the following are predicted:

1. Whether that cell has an object
2. The bounding box coordinates, the width, and the height
3. The target class of that object

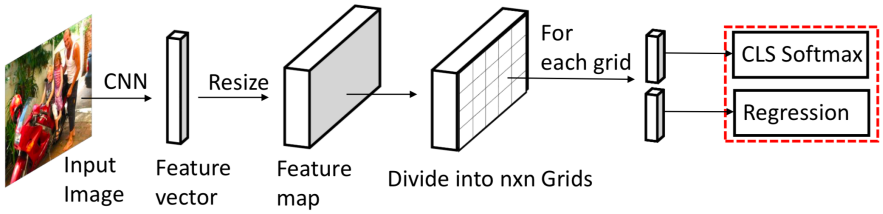


Figure 4: An overview of the YOLO architecture [30]

The following is the objective function in YOLO:

$$
\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
&+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
&+ \lambda \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
\tag{1}
$$

Since YOLO is a single network, it could be optimized in an end-to-end manner. However, YOLO can only detect up to two objects at a given location, making it difficult to detect small objects and objects in clutter. Also, since only the last feature map is used for prediction, predicting objects at multiple scales and aspect ratios becomes more challenging.

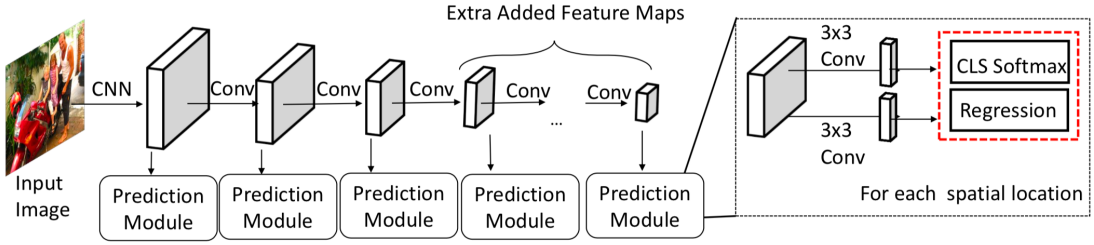


Figure 5: An overview of the SSD architecture [30]

To address the limitations of YOLO, Single-Shot Multibox Detector (SSD)[31] divides images into grid cells, but in this method in each grid cell, a set of anchors with multiple scales and aspect ratios are generated to capture objects with different sizes and shapes. Each anchor is refined by 4-value offsets learned by the regressors and is assigned C+1 (C categories + 1 background) categorical probabilities by the classifiers. In addition, SSD predicts objects on multiple feature maps, and each of these feature maps is responsible for detecting a particular scale of objects based on the receptive field. Extra convolutional

feature maps are added to the original backbone architecture to detect large objects and increase receptive fields.

The whole network is optimized end-to-end with a weighted sum of localization loss and classification loss. The final prediction is made by merging all detection results from different feature maps. After the matching step, most of the default boxes are background proposals that dominate the training gradients. Thus, background proposals are sorted ascending by confidence loss and picked to keep the 3:1 ratio between background and foreground proposals (hard negative mining). The detection performance can also be improved by several data augmentation techniques.

Retinanet and Focal Loss

So far, one-stage detectors have trailed the accuracy of two-stage detectors while being significantly more time-efficient and having greater applicability to real-time object detection. Lin et al. [2] claim that the extreme foreground-background (fg-bg) class imbalance encountered during the training of dense detectors is the cause for such low performance. Since one-stage detectors lack the proposal generation step, several tricks are used to compensate for the foreground-background imbalance, such as hard negative mining. In RetinaNet [2], the foreground-background imbalance problem is addressed in a more flexible manner. RetinaNet uses focal loss, which suppresses the gradients of easy negative samples instead of simply discarding them. Focal loss could be considered "soft" hard-example mining. The proposed focal loss outperforms naive hard negative mining strategy by large margins.

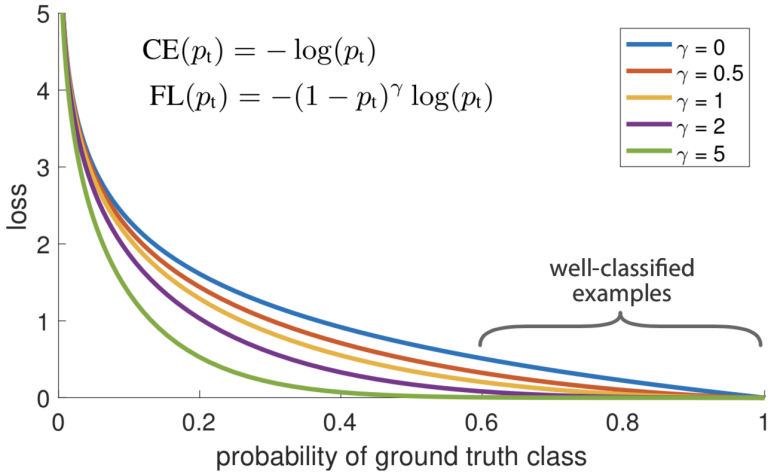


Figure 6: Setting $\gamma > 0$ reduces the relative loss for well-classified examples ($p_t > 0.5$), putting more focus on hard, misclassified examples. [2]

Here, we will go into more detail about Focal loss by starting from the Cross-Entropy (CE) loss for binary classification, see Equation 2.

$$CE(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{otherwise.} \end{cases} \tag{2}$$

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1-p & \text{otherwise.} \end{cases} \tag{3}$$

In this formula y specifies the ground-truth class (that could be either 0 or 1), and $p \in [0, 1]$ is the model's estimated class probability for the class with label $y = 1$. p_t

is defined for notational convenience so the binary cross-entropy formulation would be simplified to Equation 4.

$$CE(p, y) = CE(p_t) = -\log(p_t) \tag{4}$$

The potential problem with cross-entropy loss is that even easily classified anchors would still have a small loss, but when summed over a large number of them, in this case, with the domineering background class, the sum would dominate the gradient and the total loss and degrade performance for the less frequent foreground anchors. To overcome this issue, Focal loss is proposed, which down-weights easy examples and thus focus training on hard negatives, in this case, most probably the foreground anchors, see Equation 5. For misclassified anchors, pt is small; thus, $1 - p_t$ is near 1, and the loss is unaffected. As pt goes to 1 for well-classified anchors, $1 - p_t$ goes to 0, and the loss is down-weighted.

$$FL(P_t) = -\alpha_t \cdot (1 - p_t)^{\gamma} \cdot \log(p_t) \tag{5}$$

$0 \leq \gamma$ is a tunable focusing parameter that smoothly adjusts the rate at which easy examples are down-weighted. Focal loss with $\gamma = 0$ is equivalent to cross-entropy. $\alpha \in [0, 1]$ is a weighting factor introduced as an extension to cross-entropy called the α-balanced cross-entropy. α_t is defined just like the p_t formulation.

Apart from the different classification loss formulation, in RetinaNet, a Feature Pyramid Network (FPN) is proposed as the backbone architecture, which combines deep-layer features with shallow-layer features to enable multi-scale object detection. This backbone's main idea is to strengthen the spatially-powerful shallow layer features with semantically-rich deep layer features. FPN achieves significant progress in detecting multi-scale objects.

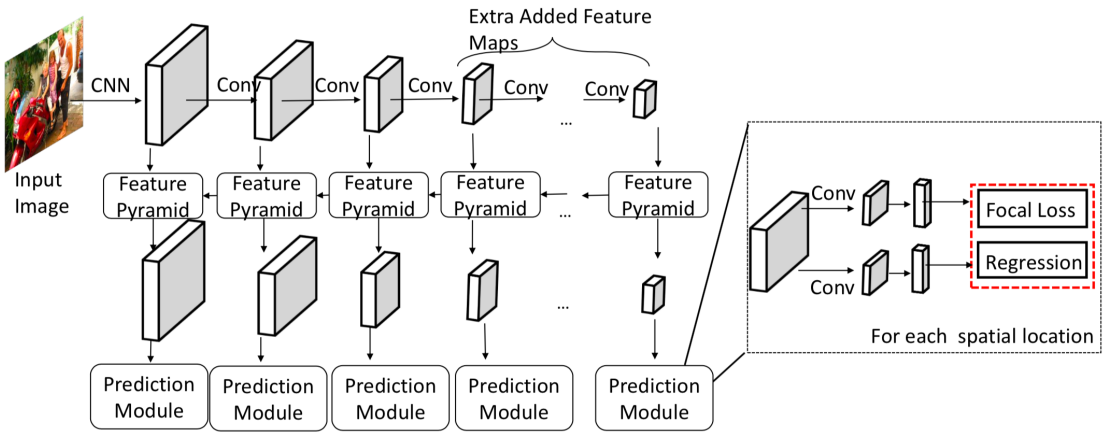


Figure 7: An overview of the RetinaNet architecture [30]

Gambler

Adversarial training has been recently employed for realizing structured semantic segmentation, in which the aim is to preserve high-level scene structural consistencies in dense predictions. In Gambling Adversarial Networks (Gambler) [1] paper, they rethink adversarial training for semantic segmentation and reformulate the fake/real discriminator loss function from the original GAN to the correct/incorrect loss function. The

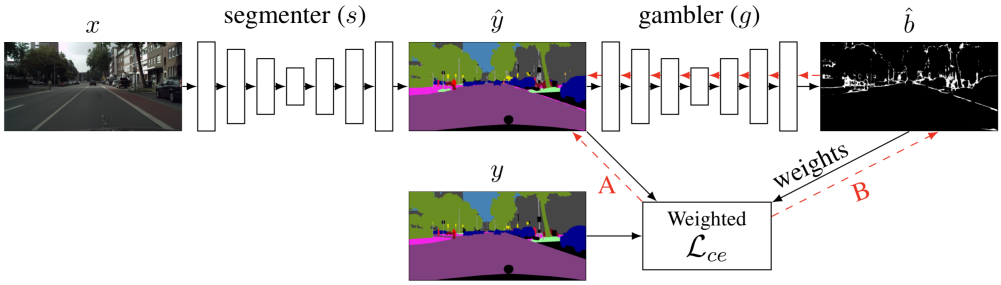


Figure 8: An overview of gambling adversarial networks. The solid black arrows indicate the forward pass. The red dashed arrows represent the two gradient flows of the weighted cross-entropy loss. Gradient flow A provides pixel-level feedback independent of other pixel predictions. Gradient flow B, going through the gambler network, enables feedback reflecting the inter-pixel and structural consistency, [1]

discriminator network is replaced with a "Gambler" network that is involved in a minimax game with the segmentation network. The Gambler learns to distribute its budget on areas where the predictions are incorrect, while the segmenter network learns to leave no (structural) clues for the Gambler to make an easy, profitable bet.

An overview of adversarial gambling networks is provided in Figure 8. The Gambler observes the RGB-image in combination with the predictions of the segmenter. Given a limited investment budget, it predicts an image-sized betting map, where high bets indicate pixels that are possibly incorrectly classified. Since the Gambler receives the entire prediction map, it can model interpixel dependencies. Consequently, structural anomalies in predictions are obvious visual clues for profitable investments. It also leverages the correlation between predictions and ground-truth by incorporating ground-truth as supervision for the Gambler network.

To better understand this method, we will look into the objective functions. Here, x and y are the input image and the corresponding label-map. $S(x)$ is the segmentation map of input image x. $D(x, y)$ represents the discriminator operating on segmentations y, conditioned on input image x, and the binary cross-entropy is defined as in *Retinanet and Focal Loss*. λ is the importance weighting of the adversarial loss. Equation 6 shows the original adversarial training objective for the discriminator, and Equation 7 the segmentation loss. Typically, the loss function for the segmenter is a combination of low-level (pixel-wise) and high-level (adversarial) loss terms.

$$L_D(x, y) = L_{bce}(D(x, S(x)), 0) + L_{bce}(D(x, y), 1) \tag{6}$$

$$L_S(x, y) = \underbrace{L_{ce}(S(x), y)}_{\text{pixel-wise}} + \underbrace{\lambda \cdot L_{bce}(D(x, S(x)), 1)}_{\text{adversarial}} \tag{7}$$

Similar to conventional adversarial training, the Gambler and segmenter play a minimax game. However, the Gambler maximizes the expected weighted pixel-wise cross-entropy, Equation 8. These weights are the outputs of the Gambler, called the betting map B. Intuitively, after perfect convergence, the bets should be maximum where the pixel-wise cross-entropy loss is maximum.

$$L_G(x, y) = -\frac{1}{wh} \sum_{i,j}^{w,h} B_{i,j}(x, S(x)) L_{ce}(S(x)_{i,j}, y_{i,j}) \tag{8}$$

On the other hand, the segmenter optimizes to minimize the gambler loss, which means improving the segmentation map to complete the adversarial loop:

$$L_S(x, y) = L_{ce}(S(x), y) - L_G(x, y) \tag{9}$$

This formulation of gambler loss encourages all bets to be maximized unless there is a budget to spend on certain bets. Therefore, they change the betting map into a smooth probability distribution to give a budget to the Gambler. To avoid focusing bets on a single location, the smoothing parameter β is introduced to regularizes the model to spread the bets.

$$B(x, \hat{y})_{i,j} = \frac{b_\sigma(x, \hat{y})_{i,j} + \beta}{\sum_{k,l}^{W,H} b_\sigma(x, \hat{y})_{k,l} + \beta} \tag{10}$$

Label Noise[1]: As mentioned in *1-stage (Dense) Detectors*, focal loss addresses the generic problem of hard-example mining for object detection without any sampling strategy. However, an issue with focal loss is its inherent limitation in dealing with label noise [1], since it is solely dependent on the ground truth label. The Gambler could be perceived as a *learned focal loss* that learns to up weight difficult/wrong samples. This approach can generalize beyond the label noise in training data by interpreting the structural dependencies and down weighting noisy samples, while focal loss is derived in a pixel-wise manner and, therefore, cannot provide high-level structural feedback.

Gambler in Object Detection: The Gambler can be especially useful in object detection where anchor boxes are usually independently predicted, losing the global perspective over the image. A potential benefit of the Gambler for detection is that it perceives the whole prediction map, consequently enabling it to model inter-class and inter-prediction dependencies (among all anchors).

To generalize from gambling-based segmentation to detection, we need to utilize one-stage object detectors, because they provide dense, per location predictions that is analogous to per-pixel predictions in segmentation. This is why we choose RetinaNet as our detection baseline. Furthermore, the FPN backbone of RetinaNet urges us to adapt the original Gambler architecture to object detection, see *Full-scale Gambler Network Architecture*.

Methodology

Proposed Method

The proposed method consists of two sub-networks, i.e., 1) detection network (denoted as D): Retinanet and 2) gambler network (denoted as G): Full-Scale Gambler. The architecture of our proposed framework is presented in Figure 9. The Gambler's goal is to provide a confidence map that outlines the trustworthy and untrustworthy regions in the prediction map from the detection network by using inter-prediction information, the image, and by incorporating the ground truth labels as a supervision signal.

To ease the description of the proposed algorithm, we first give the formal notation used throughout the thesis. The input to the detection network is image $x \in \mathbb{R}^{H \times W \times 3}$, and the outputs of the classification head are P, class prediction scores, and L the localization score for each bounding box proposal. Gambler receives the Image x and the layered classification head predictions P_l (in our case P_3 to P_7) as the input. The per-anchor output of the Gambler at each layer is denoted as the **Betting Map** B_l.

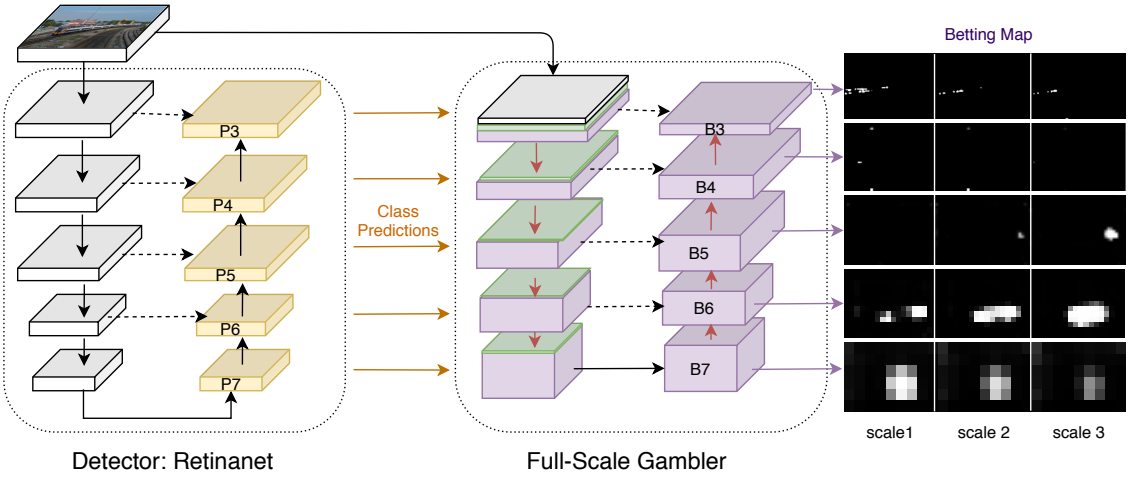


Figure 9: The high-level Architecture

Full-scale Gambler Network Architecture

Unlike the original use-case of the Gambler in segmentation where the output shape of the segmentor is $P \in \mathbb{R}^{H \times W \times C}$, the Retinanet backbone has a layered structure (FPN backbone), where each layer is used to detect objects at a certain scale. In order to use Gambler in our approach, we propose a full-scale UNet [32] architecture that integrates predictions of different layers, see Figure 10. The purple boxes in the heart of the architecture show the UNet-like formation. The UNet helps to preserve the spatial structure of the predictions while the residual blocks help to strengthen the betting map with semantically-rich deep-layer features.

At each layer, the detector predictions are concatenated with UNet features after going through a shared 1×1 convolution, which transforms prediction $p_l \in \mathbb{R}^{W_l \times H_l \times (n_{\text{class}} \times n_{\text{scale}} \times n_{\text{aspect\_ratios}})}$ into $R^{W_l \times H_l \times 32}$. 32 is a design choice hyperparameter called *fixed channels* that we mention later in the ablation study. The detector's output has the same channel dimensionality for each FPN layer because it represents (80) class scores. For this reason, we apply the same 1×1 convolution filter that transforms channels to a fixed-size channel (the *fixed channels*). As mentioned, the output space of the predictions is sparse; thus, the 1×1 convolution helps reduce sparsity by reducing the dimensionality. The same reasoning holds for converting from the UNet outputs to the final betting maps. By having a fixed number of channels per layer instead of having more channels for smaller layers, we enforce an imbalance between the amount of information carried from previous layers and the new information. This module, on the left side, is denoted as the **PreGambler Prediction** module. Using shared convolutions reduces the number of parameters, and the GPU memory usage dramatically, leading to easier training. We experiment with not-shared convolutions for this module as well but see no further improvements.

The **PreGambler Image Module** prepares the image to be merged with the predictions. We experiment with three different architectures. First, downsampling the original RGB image. Second, not using the image. Third, using a few convolution layers to map the image to the feature space. The outcome of the PreGambler Image is then concatenated with the channels from P_3 after going through the PreGambler Predictions.

The **PostGambler Prediction** module transforms the UNet predictions into the per-anchor betting map, via 5 individual 1×1 convolutions and a sigmoid nonlinearity, where the betting map at each layer is $B_l \in \mathbb{R}^{W_l \times H_l \times (1 \times 3 \times 1)}$. Thus, bets will have values in the range of $0 \leq bets \leq 1$. In this module, we cannot have shared convolutions because of different channel sizes per layer. We prefer to have a class-agnostic betting map; since it will reduce class-wise competition among bets on a specific anchor, and we could benefit from less memory usage (hence a bigger batch size), especially when the number of classes is high, like in LVIS Dataset. In this case, the Gambler will still see the correlation between class predictions and would be able to capture global structural information.

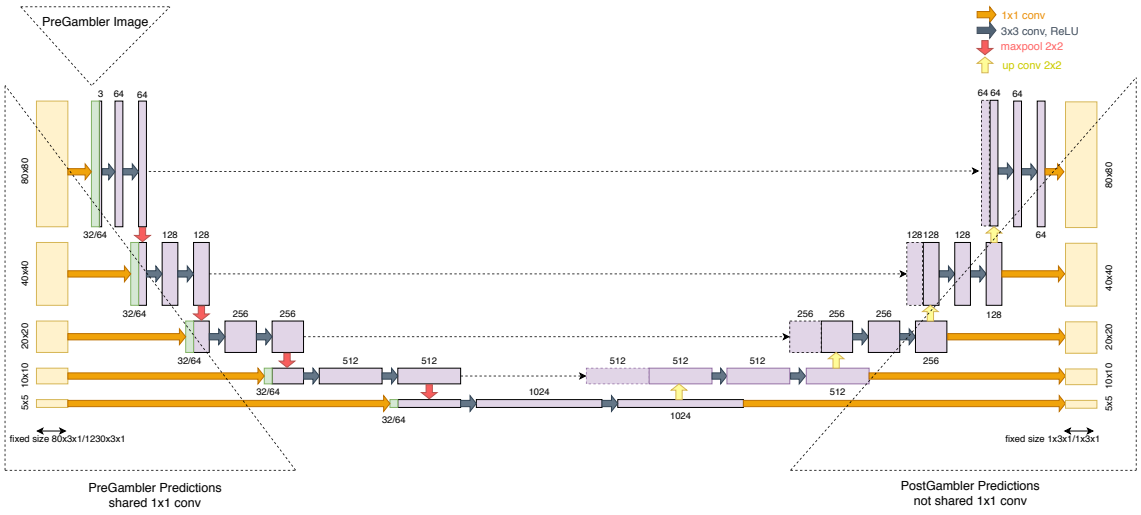


Figure 10: The full-scale gambler network architecture, including PreGambler Image, PreGambler Predictions, Post Gambler Predictions, and a UNet.

Losses

Gambler Loss

$D(x)$ is the Detector and $G(x, p)$ is the Gambler module. x is the input image, p the predictions from the detector, and y is the corresponding ground-truth label map that is obtained from the matching strategy. The typical detector loss is the sum of the classification and the regression losses, Equation 11.

$$L_{Detector}(x, y) = \underbrace{L_{reg}(x, y)}_{\text{regression loss}} + \underbrace{L_{cls}(x, y)}_{\text{classification loss}} \tag{11}$$

Similar to conventional adversarial training, the Gambler and detector play a minimax game, the Gambler is analogous to the discriminator and the detector the generator. However, the Gambler maximizes the expected weighted classification loss, Equation 12. These weights are the outputs of the Gambler, called the betting map B. Ideally, after convergence, the bets should be maximum where the loss is maximum.

It is possible to use any classification loss in Equation 12. However, we use the focal loss, which is the detector loss for better alignment of both training modules. In the ablation study, we experiment with the original binary cross-entropy loss as well.

$$L_G(x, y) = -\sum_{l}^{L} \sum_{i,j,s,a}^{W,H,S,A} B_l(x, D(x))_{i,j,s,a} \cdot L_{cls}(D(x)_{i,j,s,a}, y_{i,j,s,a}) \tag{12}$$

On the other hand, the detector optimizes a combination of loss terms: the regression loss, the conventional focal loss for classification and the reverse of the newly *adversarially weighted focal loss*. Intuitively, the Gambler will point to faulty predictions, potentially where the loss is high. This means the detector tries to improve prediction quality in places where the Gambler identified as faulty, hence completing the adversarial loop:

$$L_D(x, y) = \underbrace{L_{reg}(x, y)}_{\text{regression loss}} + \underbrace{L_{FL}(x, y)}_{\text{classification loss}} \underbrace{- \lambda \cdot L_G(x, D(x))}_{\text{adversarial}} \tag{13}$$

λ controls the influence of the Gambler loss on the total detector loss.

Normalization

This formulation of gambler loss encourages all bets to be maximized unless there is a budget to spend on certain bets. Therefore, we use the same formulation as in [1] in order to distribute the bets. The normalization forces the bets to sum up to one. To avoid a focused spiky bet on a single location (all bets 0 except for one), the smoothing parameter β is used to spread the bets. We show later in the ablation study and analysis how crucial the β is and how it influences the betting maps.

$$B_l(x, p)_{i,j,s,a} = \frac{b_l(x, p)_{i,j,s,a} + \beta}{\sum_{l}^{L} \sum_{k,m,n,o}^{W,H,S,A} (b_l(x, p)_{k,m,n,o} + \beta)} \tag{14}$$

It is important to note that the original Focal loss is normalized by the number of matched foreground anchors since the other anchors are down-weighted and contribute the least. Here, the normalization of weights that sum to 1 and multiplying it by the loss directly normalizes the loss. Hence, we multiply the Gambler loss by the number of matched foregrounds to avoid double normalization.

Training and Inference

Training

Our method's training follows an adversarial training scheme, as explained in more detail in *Training Details*. The detector and the Gambler parameters are optimized interchangeably. For each minibatch, the Retinanet outputs the corresponding prediction logit maps and the loss map. The Gambler loss is calculated accordingly: The sigmoid of the prediction logits is summed with the smoothing parameter β, after which the per-anchor normalization is done. The classification loss is summed per-anchor over all the classes. Finally, the negative sum of the Hadamard product (element-wise) of the betting map with the classification loss is calculated as the Gambler loss. The Gambler and detector are then optimized with losses 12 and 13, respectively.

Algorithm 1 Training Loop

Pre-train Detector with IMAGENET
Initialize Gambler last layer with $-\log \frac{1-\pi}{\pi}$
gambler_optimizer only optimizes the Gambler parameter
detector_optimizer only optimizes the Detector parameter
while *iter* $<$ MAX_ITER **do**
    data.img, data.gt $\leftarrow$ sample a minibatch from the train loader
    prediction_logits, L_D = Retinanet(data)
    if Gambler_iter **then**
        gambler_output = Gambler(prediction_logits, data.img)
        $L_G \leftarrow$ CALCULATE_GAMBLER_LOSS(gambler_output, $L_D.cls$)
        L_G.backward()
        gambler_optimizer.step()
    if Detector_iter **then**
        gambler_output = Gambler(prediction_logits, data.img)
        $L_G \leftarrow$ CALCULATE_GAMBLER_LOSS(gambler_output, $L_D.cls$)
        $L_D.cls \leftarrow L_D.cls - \lambda \cdot L_G$
        L_D.backward()
        detector_optimizer.step()
function CALCULATE_GAMBLER_LOSS(gambler_output, L_{cls})
    betting_map $\leftarrow \sigma(gambler\_output)$
    betting_map $\leftarrow$ betting_map $+ \beta$
    $b_{w,h,s,a} \leftarrow \frac{b_{w,h,s,a}}{\text{betting\_map.sum()}} \; \forall w,h,s,a \in W,H,S,A, \; b \in betting\_map$
    $L_G \leftarrow -\sum$betting_map $\cdot L_{cls}$
return L_G

Inference

The Gambler is only used during training. During inference, the Gambler is ignored, and only the detector is used, so no extra inference time is needed. RetinaNet is comprised of the ResNet-50-FPN backbone, a classification subnetwork, and a box regression subnetwork. Thus, inference involves forwarding an image batch through the network. After setting detector confidence threshold at 0.05, to improve speed, top 1k box predictions per FPN level are decoded. The top predictions from all FPN levels are merged, and non-maximum suppression with a threshold of 0.5 is applied to yield the final detections.

Experimental Setup

Datasets

COCO

MS COCO [3] is a response to the criticism of ImageNet, where objects tend to be mostly large and well-centered, making the ImageNet dataset far from reality. Thus, MS COCO was introduced that contained complex everyday scenes with common objects in their natural context, closer to real-life scenarios. COCO is a large scale dataset with 80 categories. There are three image splits in COCO (version 2017): training, validation, and test with 118287, 5000, and 40670 images. The COCO test set only contains box and mask annotations. We train our models on the training set and report all results on the validation set.

LVIS

Currently, COCO is the most commonly used detection benchmark. However, COCO has only 80 categories, which is still too small to understand more complicated scenes in the real world. Recently, a newly collected benchmark dataset, LVIS [4], has been proposed to present a more challenging object detection benchmark, where annotations are closer to real-world data distributions. LVIS contains the same COCO images; nonetheless, annotations comprise of 1230 categories in version v0.5. LVIS simulates the real-world long-tail scenario where a large number of categories are rare, and a small number of categories are frequent. Figure 11 compares COCO and LVIS datasets, where LVIS contains more diverse categories per image and has a long-tail distribution of instances per class.

In LVIS, categories are divided into three groups based on the number of images containing those categories: rare (1-10 images), common (11-100), and frequent (>100). We train our models on the 57k training set and evaluate them on the 5k validation set.

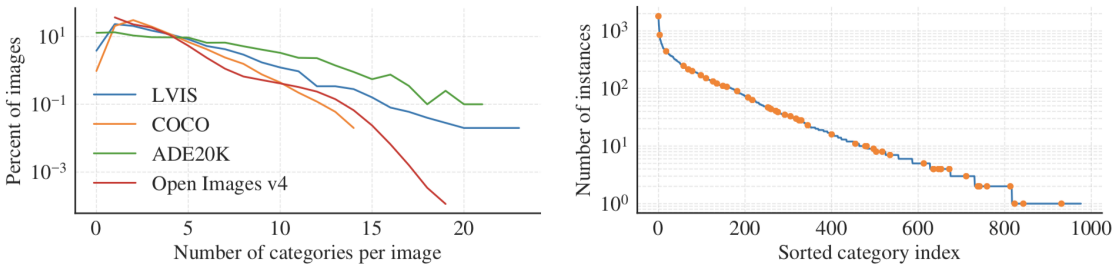


Figure 11: **Left:** compares the LVIS dataset statistics with COCO. LVIS contains more diverse categories per image and more few-shot classes. **Right:** The long-tail number of instances per category. The orange dots show categories that are in common with COCO. Image courtesy of [4]

Implementation details

Baseline Detector

We implement standard Retinanet equipped with an FPN backbone [2] as our baseline model. We use translation-invariant anchor boxes. The anchors have areas of 32^2 to 512^2 on pyramid levels P_3 to P_7, respectively. At each pyramid level, there are anchors at three scales of sizes 2^0, $2^{1/3}$, $2^{2/3}$, and one aspect ratio 1:1. Altogether, there are 3 anchors per level, and across levels, they cover the scale range of 32 - 813 pixels in relation to the network's input image.

Gambler

We implement the Gambler as specified in more detail in *Full-scale Gambler Network Architecture*. The PreGambler Image module downsamples the RGB image with bilinear interpolation and a stride of 8, keeping the color channels intact. The Gambler is trained with the focal loss in Equation 12 with the smoothing parameter β of 0.1. The default $\lambda = 1$ unless specified otherwise.

The source code is available at:
https://github.com/Melika-Ayoughi/Full-Scale-Gambler-for-Object-Detection/.

Training Details

Training Schedule

For training the models, similar to the conventional adversarial training, the detector and Gambler are trained in an alternating fashion. First, the Gambler is trained for 20 iterations while the detector is frozen, and then the detector is trained for 40 iterations while the Gambler is frozen. In order to keep the performance of the detector trained with and without Gambler comparable, we fix training the detector to 90k iterations, and train the Gambler for 45k iterations, making a total of 135k iterations. For LVIS, batch-size is halved, so the detector is trained for 180k iterations and the Gambler for 90k iterations. Training time is approximately 33 hours for COCO and 50 hours for LVIS.

Optimization

We use an initial learning rate of 0.01 for the detector and 0.0001 for the Gambler, which is then divided by 10 at 105k and again at 120k iterations. A gradual warmup learning rate scheduling from [33] is adopted that linearly ramps up the learning rate from a small to a large value. This gradual increase avoids a sudden jump in the learning rate, allowing for better convergence at the start of the training. After the warmup period of 1000 iterations, we start to the original learning rate scheduling. The Gambler is trained with the Adam optimizer [34] unless otherwise specified. RetinaNet is trained with stochastic gradient descent (SGD) [35] with a weight decay of 0.0001 and momentum of 0.9. We use two 32-Gigabyte GPUs with a total of 16 images per minibatch (8 images per GPU). The training loss of the detector is the sum of the focal loss and the standard smooth L1 loss used for box regression minus the Gambler loss, see *Losses* for more details.

Preprocessing

Training images are randomly resized such that their shorter edge is 800 pixels while the longer edge is no more than 1333. We use horizontal image flipping as another form of data augmentation unless otherwise noted.

Initialization

We experiment with the ResNet-50-FPN backbone for the detector. It is pretrained on ImageNet1k; we use the models released by [36]. New layers added for FPN are initialized as in [2]. All new convolution layers except the final one in the RetinaNet subnets are initialized with zero bias, and a Gaussian weight with $\sigma = 0.01$. For the final convolution layer of the classification subnet, we initialize the bias to $b = -\log \frac{1-\pi}{\pi}$, see Equation 15. This initialization setting is the same as in the original Retinanet.

$$\begin{aligned} \sigma(x) &= \pi \\ \frac{1}{1+e^{-x}} &= \pi \\ 1+e^{-x} &= \frac{1}{\pi} \\ \log \frac{1-\pi}{\pi} &= -x \end{aligned} \tag{15}$$

This ensures that at the start of the training, every anchor is foreground with a low probability of almost π. Based on [focal loss], we use $\pi = 0.01$ in all experiments. This initialization prevents the large number of background anchors from generating a large, destabilizing loss value in the first iterations of training.

On the other hand, the Gambler is initialized from scratch. All layers except the last layer are initialized with the default initialization of Pytorch. For the final convolution layer, we initialize the bias to $b = -\log \frac{1-\pi}{\pi}$ again with the same $\pi = 0.01$. This initialization scheme helps the Gambler to start with a stronger prior that most anchors should have low bets since they are most probably backgrounds. In our experiments, we observed that this helps faster convergence.

Evaluation Metrics

IoU (Intersection over Union)

For any predicted box proposal, we use IoU or Jaccard Index to decide whether it matches a ground-truth object. IOU is defined as the intersection area between the predicted bounding box and the ground-truth bounding box divided by their union. A prediction is considered True Positive if $IoU > threshold$ and False Positive otherwise.

Precision and Recall

Recall is the True Positive rate, i.e., of all the actual positives, how many are True Positive predictions. Precision is the ratio of True Positives to all the positive predictions, i.e., of all the positive predictions, how many are correct True Positives.

$$Recall := \frac{TP}{TP+FN} = \frac{TP}{\#ground\ truths} \tag{16}$$

$$Precision := \frac{TP}{TP+FP} = \frac{TP}{\#predictions} \tag{17}$$

Given the list of proposal boxes, corresponding confidence scores, and the IoUs, we calculate AP in the following manner. First, we sort the predictions based on the confidence scores per class. In case there is more than one proposal for a single ground-truth

object, the proposal with the highest IoU is considered as TP, the rest as FP. Then, we calculate the precision and recall based on the formula 16. The precision and recall values are plotted to get the precision-recall (PR) curve. The precision at each recall level R is interpolated by taking the maximum precision measured for any recall bigger than R. Average Precision (AP) is the area under the PR curve. Now, we have AP per class, Mean Average Precision (mAP) is the averaged AP over all the object categories. Unless stated otherwise, AP and mAP can be used interchangeably.

COCO mAP

Usually, a prediction with $IoU > 0.5$ is considered as True Positive. This means that two predictions of IoU 0.5 and 0.9 would have equal weight in the mAP. Thus, a certain threshold introduces a bias in the evaluation metric. To solve this, they [3] use a range of IoU threshold values, calculate mAP for each threshold, and average to get the final mAP. In COCO, the IoU threshold ranges from 0.5 to 0.95, with a step size of 0.05 represented as AP@[.5:.05:.95].

$$mAP_{COCO} = \frac{mAP_{0.50} + mAP_{0.55} + ... + mAP_{0.95}}{10} \tag{18}$$

Thus, in COCO evaluation: First, for each class, AP is calculated at different IoU thresholds and averaged to get the AP per class. Second, the final AP is the average of per-class APs.

$$AP[class] = \frac{1}{n_{\text{threshold}}} \cdot \sum_{iou \in thresholds} AP[class, iou] \tag{19}$$

$$AP = \frac{1}{n_{\text{class}}} \cdot \sum_{class \in thresholds} AP[class] \tag{20}$$

In COCO evaluation, six metrics are provided, three of which are thresholding the BB at different IoUs:

1. AP: AP at IoU=0.50:0.05:0.95 (primary challenge metric)
2. AP@IoU=0.5 (PASCAL VOC metric)
3. AP@IoU=0.75 (strict metric)

The remaining three are calculating AP across scales, in order to capture performance across objects of different sizes:

1. AP_s: AP for small objects: $area < 32^2$ px
2. AP_m: AP for medium objects: $32^2 < area < 96^2$ px
3. AP_l: AP for large objects: $area > 96^2$ px

LVIS mAP

Unlike COCO evaluation process, since LVIS is a sparse annotated dataset, detection results of categories that are not listed in the image level labels will not be evaluated. In addition to the six AP values in COCO, in LVIS, categories are divided into three groups based on the number of images containing those categories and the AP is evaluated for rare ($1 - 10$ images) , common ($11 - 100$ images), and frequent (> 100 images) classes; AP_r, AP_c, AP_f respectively.

Results

In this chapter, we present our quantitative findings and comprehensive analysis to find the angle where our method outperformed the baseline. We perform ablation study on the most significant parameters, compare our method to the baseline on COCO and LVIS, analyze per-class performance and conclude by qualitative analysis to observe what is learned by the Gambler.

Ablation Study

In this section, we discuss the effects of the most influential hyperparameters to justify specific design choices throughout the thesis.

Classification Loss Mode

First, we investigate the influence of the classification loss used in the Gambler loss formulation 12. The Gambler loss can be the weighted sum of the Focal loss or the Cross-Entropy loss. In these experiments, we train the detector with the original loss (sum of the Focal loss and the regression loss) and only change the Gambler loss formulation. By looking at Table 1, we observe that for any β, reweighting the Focal loss results in better outcomes, although limited. One of the reasons for that is the cross-entropy was used in its original formulation, unlike the $\alpha-$balanced cross-entropy loss in [2] that could help with foreground background imbalance. Another reason why Focal outperforms CE loss is that Focal is also used in training the detector.

Smoothing (β)

The smoothing parameter β is the most critical hyperparameter. β ranges between 0 to 1. $\beta = 0$ would result in a focused betting map of all zeros except for one point because the Gambler loss is maximized when the bet corresponding to the maximum loss is 1. Too big of a β would further result in all bets being equally small. This would prevent the gambler from focusing on any specific prediction, limiting its effect on training. Therefore finding the perfect $0 < \beta \leq 1$ is crucial to spread the bets to uncertain predictions; not too big that the bets would be wasted and not too small that the effect would be insignificant. The smaller the β, the higher the maximum bet and the lower the minimum bet, thus a bigger gap between the maximum and the minimum bets, see Figure 21 in the Appendix A. In that section, we also prove the higher gap between bets for different β values. Table 1 suggests that $\beta = 0.1$ achieves the highest AP, although the difference between results with the β in the correct range leads to somewhat similar outcomes.

Table 1: Varying β and formulations for gambler loss, on COCO validation set

	Focal Mode			CE Mode		
β	AP	AP50	AP75	AP	AP50	AP75
1	34.553	55.037	36.305	34.407	54.993	36.038
0.1	35.029	55.523	36.977	34.360	55.229	35.949
0.01	34.700	55.425	36.513	34.213	55.198	35.961

Lambda (λ)

λ controls the influence of the Gambler loss on the total detector loss, Equation 13. Too high of λ degrades performance to a great extent. This is probably due to the Gambler loss dominating the total loss, with the box regression term being almost disregarded, leading to incorrect localization. However, the experiments are not sensitive to the range of $1 < \lambda < 10$, as shown in Table 2.

Table 2: Varying λ values for the detector loss, on COCO validation set

λ	AP	AP50	AP75	APs	APm	APl
1	35.029	55.523	36.977	20.493	**39.268**	45.615
2	**35.245**	**55.845**	37.204	**21.631**	39.021	44.887
10	35.155	55.732	**37.275**	21.508	39.058	**45.848**
100	28.949	47.409	30.145	14.549	31.230	39.035
200	29.199	47.974	30.607	15.057	31.750	39.154

PreGambler Image Module

The design choice of the PreGambler Image Module is essential because it determines how the Gambler network perceives the image integrated with the prediction features. Thus, we experiment with three different architectures. In `downsample_image`, we downsample the original RGB image with a stride of 8. In `no_image`, we do not use the image as an input to the Gambler. In `img_conv_64`, a Double convolution module (2 sequences of 3×3 convolution, batch normalization and ReLu) followed by three Downsampling modules (2-dimensional Maxpooling, and a Double convolution) are used to map the image to a feature space with 64 channels. 64 is a hyperparameter called the *fixed channels*. Table 3 shows how each design choice influences the final AP result. The good results in `no_image` suggest that the Gambler network only uses the detector's predictions to reason about uncertainties, and that can improve AP for small and medium objects while slightly degrading performance for larger objects. `img_conv_64` achieves the lowest result probably because training extra convolutions for the image is expensive and difficult to do with the current Gambler loss.

Comparison with Baseline

We evaluate the baseline RetinaNet, explained in *Baseline Detector*, on COCO and LVIS datasets, and compare `coco_2017_val` and `lvis_v0.5_val` results to our approach. The results suggest minimal influence of the Gambler on the training of the detector. We will discuss our main takeaways and suggestions from these experiments in the *Conclusion*.

Table 3: Varying architectures for PreGambler Image

PreGambler Image Mode	AP	AP50	AP75	APs	APm	APl
downsample_image	**35.113**	**55.705**	**37.229**	20.536	38.960	**45.166**
no_image	35.052	55.240	37.105	**22.268**	**39.013**	44.881
img_conv_64	34.529	55.620	36.488	20.730	38.554	44.724

Table 4: Comparison of our approach with the baseline on COCO dataset.

Experiment	AP	AP50	AP75	APs	APm	APl
Baseline	35.113	55.705	**37.229**	20.536	38.960	**45.166**
Ours	**35.245**	**55.845**	37.204	**21.631**	**39.021**	44.887

Table 5: Comparison of our approach with the baseline on LVIS dataset.

Experiment	AP	AP50	AP75	APs	APm	APl	APr	APc	APf
Baseline	**17.921**	**28.902**	18.340	**13.904**	**22.889**	30.588	**4.702**	16.397	**25.108**
Ours	17.839	28.656	**18.363**	13.271	22.241	**31.331**	4.475	**17.021**	24.402

Analysis

Correlation Analysis

The minimum effects of our method compared to the baseline model motivated supplementary per category analyses. In this analysis, the correlation of class frequency, size of objects, class-entropy, and the baseline AP on per-class improvements are investigated on the COCO dataset.

In **class frequency**, the categories are sorted based on the number of instances in descending order, with frequent classes on the left. Figure 17a shows the per-class AP difference between our method and the baseline model. For several of the rarest categories, our method shows a consistent improvement over the baseline, even up to 8 AP difference, however, there is no clear correlation between class frequency and AP improvements.

In **object sizes**, the classes are sorted based on the average area of the annotations relative to the image size, from large to small. Figure 17b depicts no correlation between object sizes and their performance, even though some of the largest classes show better performance for the baseline.

A big challenge in object detection is clusters of similar objects. For instance, detecting all the instances of books in an image of a bookshelf poses a problem for many detection methods. One reason for this difficulty is, that any anchor-based approach would need a very fine grid of anchors, to make sure that around one anchor only one instance is captured. Secondly, clusters of similar objections will often also exhibit occlusion (like a flock of sheep) in which boundaries are not clear anymore. As a proxy analysis of this problem, we plot the class-entropy in Figure 17c, which shows the average number of instances of a class in an image with at least one object of that class.

In the last correlation analysis, we analyze whether our method improves categories that initially score a low **AP** in the baseline, namely the hard categories. In Figure 17d, the categories are sorted in descending order based on their AP performance in the

baseline, with hairdryer the hardest category. It is observed that, indeed, most improved categories have low APs in the baseline. The Gambler helps in focusing on the low performing classes, by focusing the loss on them. Nevertheless, our approach performs worse on initially high-AP classes.

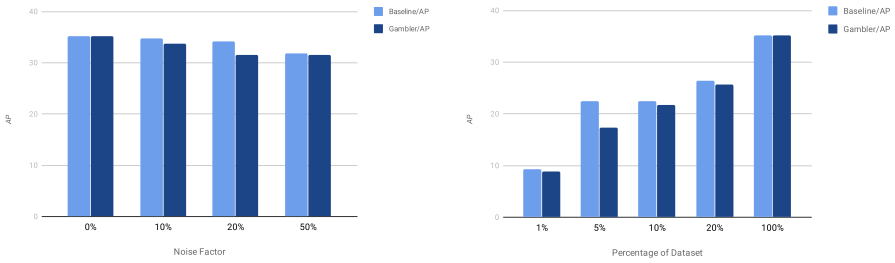
For all the above analyses, the per-class AP values with the same sorting scheme are depicted in Figure 22 in the Appendix section.

Noise Analysis

As mentioned in *Gambler*, Samson et al. [1] perceive the Gambler as a *learned focal loss* that learns to generalize beyond the label noise by interpreting the structural inter-dependencies and down-weighting noisy samples, while Focal loss is calculated pixel-wise, ignoring high-level structural information. In order to observe the flexibility of our approach to label noise, we create three noisy datasets, with $p_1 = 10\%, 20\%, 50\%$ noise factors. The label of each object annotation will switch (maybe also to itself) with a probability of p_1. The new label is chosen with a probability of p_2 that is proportional to the frequency of categories, to preserve the class label distribution. Figure 12 compares the AP of the baseline and our approach trained on these three datasets. The AP naturally decreases with higher noise factors; however, either the baseline is the winner, or it performs almost equally well. This experiment suggests that the Gambler does not improve robustness on noisy labels, at least in the detection setting.

Generalization Analysis

Training with the Gambler improves the AP for several rare classes. One possible hypothesis is that it can help generalize to categories with fewer training samples. This analysis aims to observe the generalization ability of our approach and whether the Gambler loss acts as a regularizer for the detection network during training. To test this hypothesis, we build four sampled datasets from the *COCO 2017 Train* set containing 20%, 10%, 5%, and 1% of the images. We train the baseline detector with and without the Gambler with these four training sets and evaluate the results on the *COCO 2017 Validation* set. Figure 12 compares the AP of baseline and our approach trained on each dataset. Although the AP values are close for the entire dataset, the baseline generalizes better sometimes even with a great margin (the 5% dataset).



(a) Noise Analysis (b) Generalization Analysis

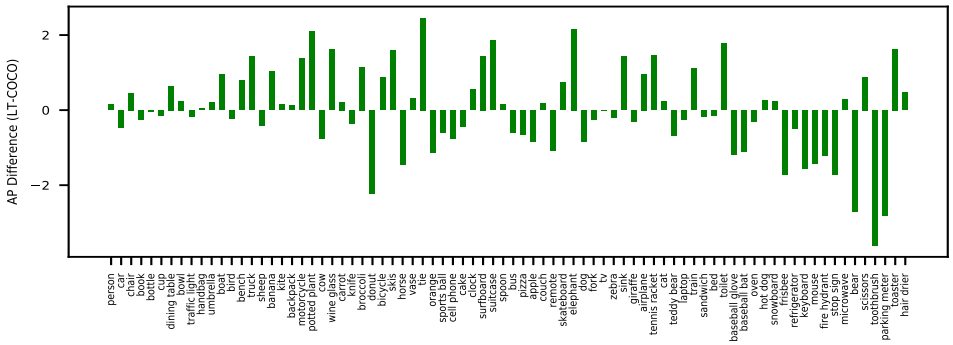
Figure 12: **Left:** Comparison of the AP performance of our method with the baseline under 4 different label noise factors. **Right:** Comparison of the AP of our method versus baseline at different sizes of dataset to understand the generalization ability.

Table 6: Comparison of our approach with the baseline trained on COCO-LT dataset.

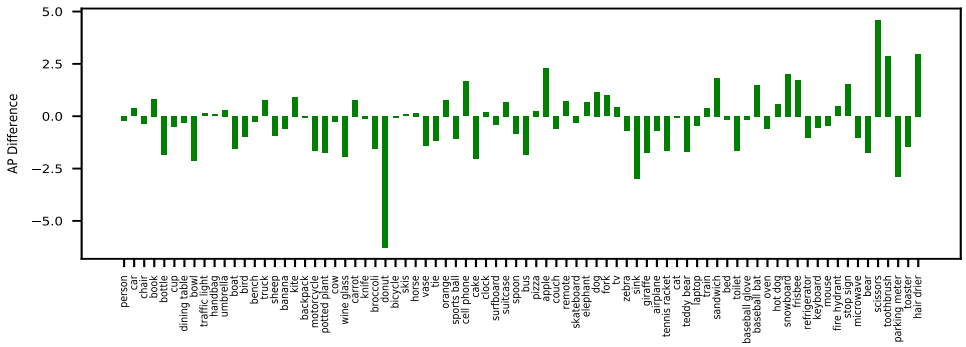
Experiment	AP	AP50	AP75	APr
Baseline	**35.117**	**55.263**	**37.290**	37.46
Ours	34.868	55.053	36.808	**37.88**

Long-tailed Analysis

In most experiments, it is observed that our approach outperforms the baseline on the two most rare categories: *toaster* and *hairdrier*. To test our approach on rare classes, we build a long-tailed COCO dataset where we aim to keep the frequency of all categories intact, except for the top 10 most rare categories. This is done by eliminating images with the most instances of rare categories until the number of instances is halved while having the least effect on other categories. The diagrams of #instances per category of the original COCO and the COCO-LT can be viewed in Figures 18, respectively. Figure 13a depicts the drop in those 10 categories for the baseline detector trained with COCO-LT versus COCO. Figure 13b shows the AP differences between our approach and the baseline detector trained with COCO-LT. The figure shows our approach outperformed the baseline for 5/10 categories, and the average AP for those 10 categories marginally improved from 37.46 to 37.88, although the total AP dropped from 35.11 to 34.86, Table 6.



(a) The AP difference of the baseline model trained with COCO-LT compared to COCO. The AP for the last 10 categories has decreased for the baseline detector.



(b) The AP difference of our method compared to baseline on COCO-LT. AP of half of the rare categories improved, leading to marginal 0.5 AP improvement for rare classes.

Qualitative Analysis

Thus far, we have observed various analyses that confirm our method has a minimal effect on training an object detection system. This necessitates visualizing betting maps corresponding to each image to prove the multi-scale Gambler is learning meaningful bets. For each image, the multi-scale Gambler predicts 15 per-location bets, corresponding to 5 FPN layers and 3 different anchor scales. For better analyzing the role of the Gambler, we visualize the loss and anchors matched with foreground/background/ignored, alongside the betting map at each individual layer. Figure 14 shows one of the layers from Figure 15. In that layer, the top row shows the corresponding matched ground-truth, middle row the loss, and the bottom row the corresponding betting map. Each column visualizes the same three maps for a given scale. Scale 1 denotes the anchor scaled $2^{\frac{2}{3}}$ times, for detecting bigger objects and scale 3 is the original anchor scale, for smaller objects. Since we have a **full-scale** Gambler, the same 9 maps are visualized per layer in Figure 15. All the samples in Figure 19 follow the same structure.

At each layer and scale, per-location anchors are matched with either one of the foreground classes or a background class. In this visualization, if the anchor is matched with a foreground ($0.5 < IoU$) is grey, if in the ignored region ($0.4 < IoU < 0.5$) is white and if background ($IoU < 0.4$) is black; this is how matching between anchors and ground-truth objects is done in Retinanet. The loss and the betting map are normalized between 0 and 1 per layer for visualization purposes. Also 10×10 and 5×5 layers are upsampled with bilinear interpolation for better visualization.

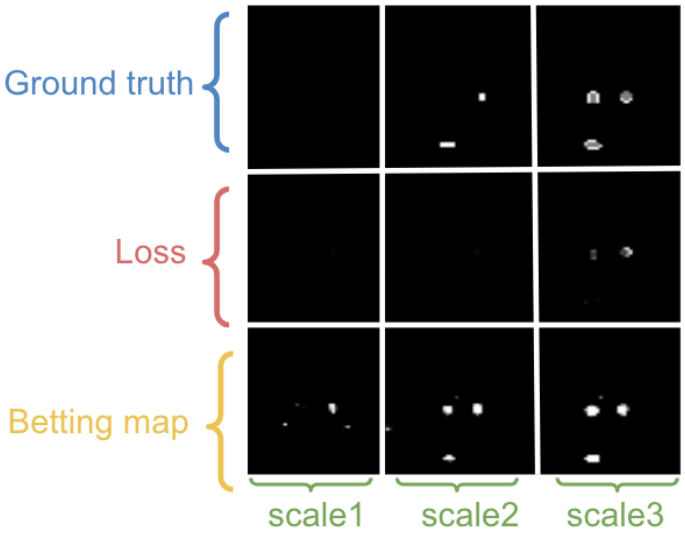


Figure 14: the top row shows the corresponding matched ground-truth, middle row the loss, the bottom row the corresponding betting map. Each column visualizes the same three maps for a given scale. Scale 1 denotes the anchor scaled $2^{\frac{2}{3}}$ times, for detecting bigger objects and scale 3 is the original anchor scale, for smaller objects. **Groundtruth:** if the anchor is matched with a foreground ($0.5 < IoU$) is grey, if in the ignored region ($0.4 < IoU < 0.5$) is white and if background ($IoU < 0.4$) is black

The betting map agrees mostly with the ground-truth map at each layer rather than correlating with the loss. In other words, gambling bets are high wherever there is an object. Depending on the smoothing parameter β, the blobbiness of the bets change,

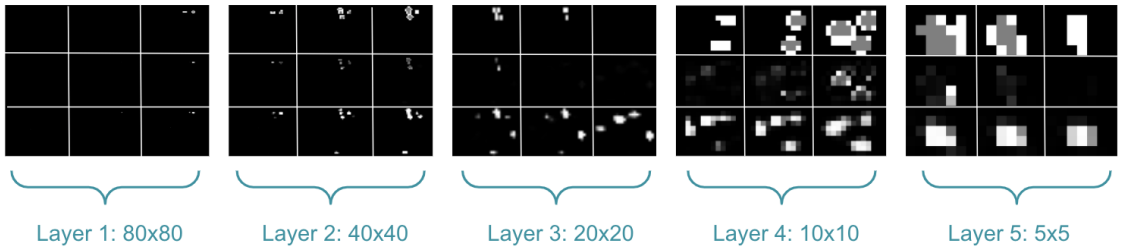


Figure 15: Visualization of the 14 for all 5 layers.

but they always correlate with where the ground truth objects are. Figure 16 shows the betting maps at different β parameters for the same image. Besides, usually, the betting maps at each layer are almost the same across all scales. This is expected because the full-scale Gambler has only access to an approximate location of predictions among all scales (the center of the anchors), and the anchor location is not explicitly given to the Gambler.

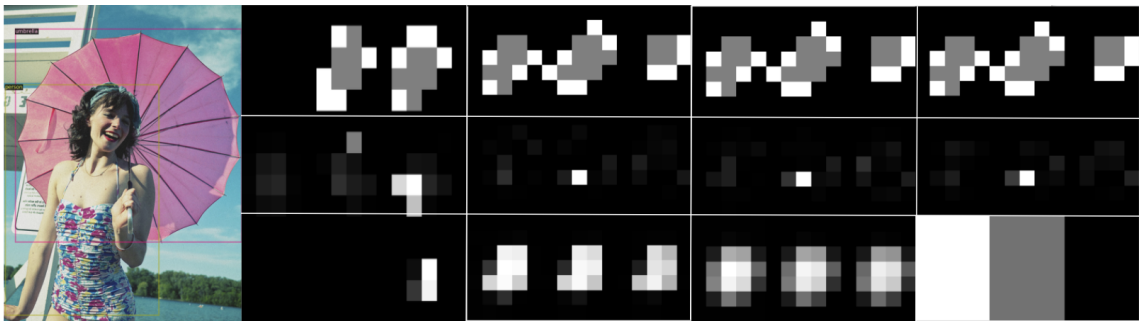
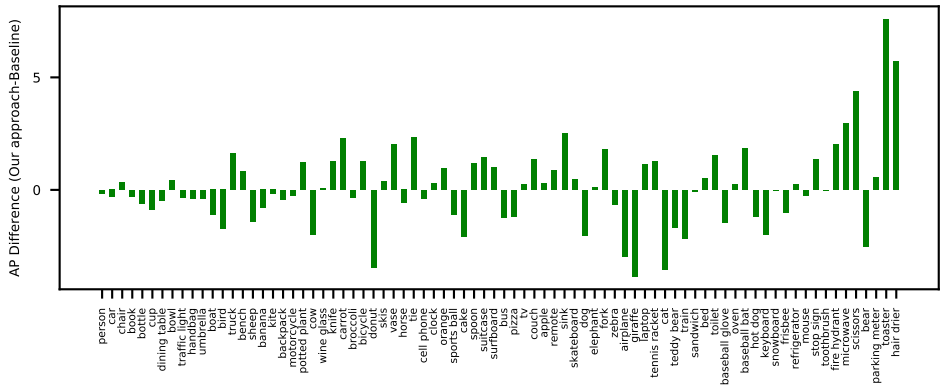


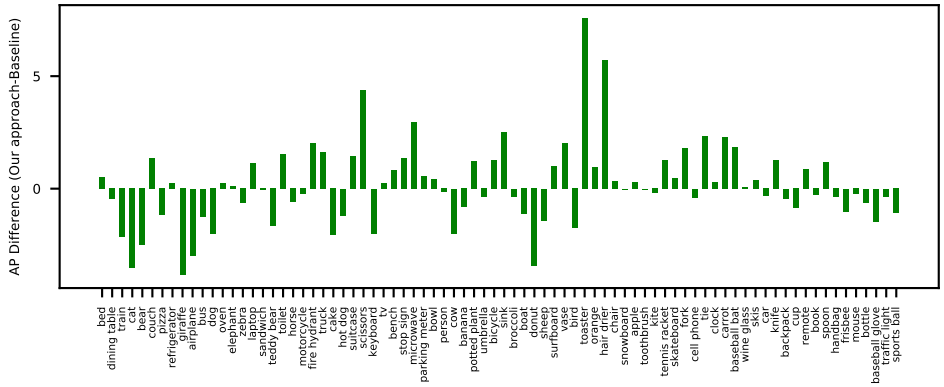
Figure 16: For the given image, each column shows the corresponding (ground-truth, loss and betting map) of the layer 5×5 with $\beta = 0$, $\beta = 0.01$, $\beta = 0.1$, $\beta = 1$, respectively. Higher β leads to more uniform bets while $\beta = 0$ generates the most spiky bets and the middle-ranged β is more blobby.

By looking at the betting maps and how they correlate with the ground-truth objects, our method could be perceived as a *reversed* two-stage detector, that unlike conventional two-stage detectors where a few region proposals are filtered for the second stage, in this method, all anchors are used during the first stage and the second stage determines the essential focus points or the foreground regions.

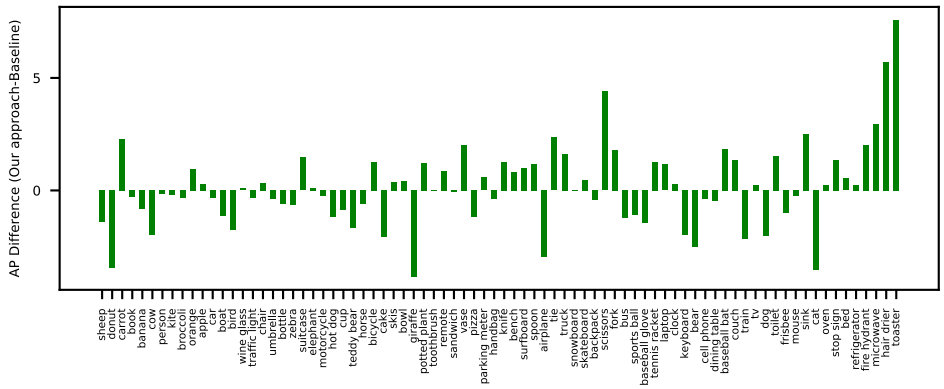
In Figure 20, the predictions of our method (left), the baseline (middle), and the ground-truth annotations (right) are compared for a confidence threshold of 0.5. The default confidence threshold during inference is 0.05; however, we choose to see higher-confidence proposals for visualization purposes. We observe that the average confidence score is higher in our method for proposals with confidence scores higher than 0.5 although this is not true for low confidence thresholds. These observations suggest that training an object detection model with a Gambler focuses on certain proposals that make the detector more confident about those proposals, however compensating more uncertain proposals.



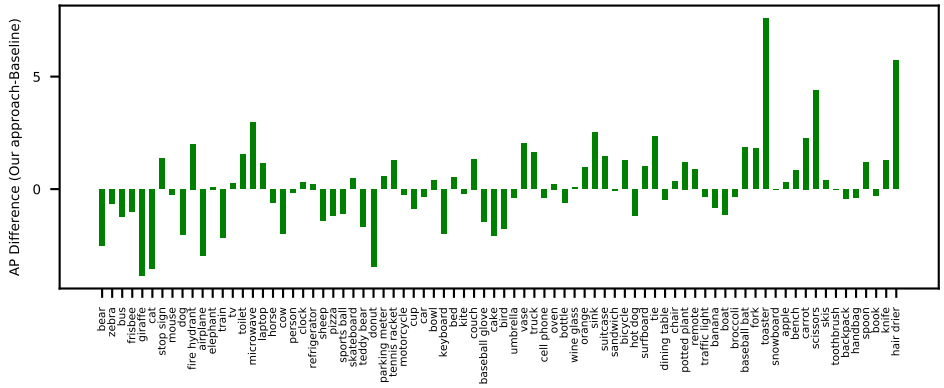
(a) Frequency



(b) Size

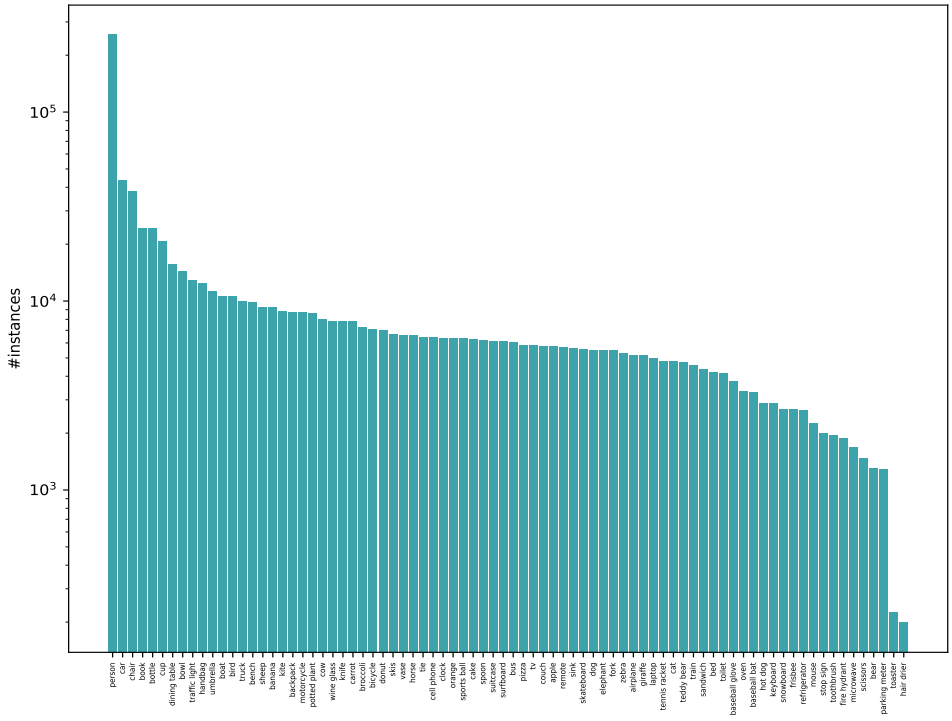


(c) The class-entropy. The class sheep has the highest average number of instances per image while images with a toaster are less likely to contain multiple toasters.

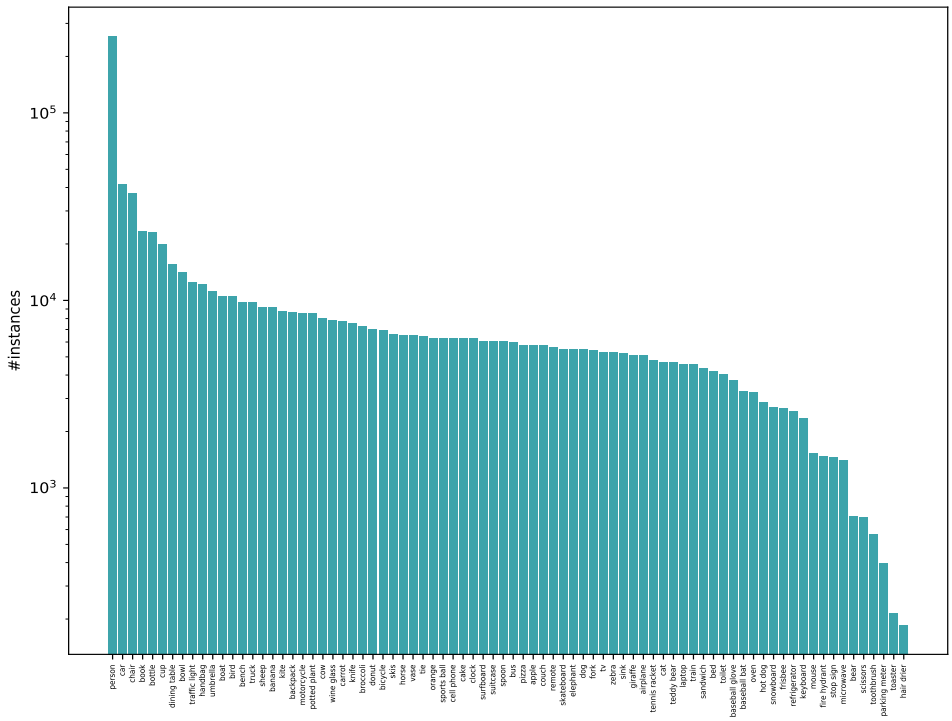


(d) Baseline AP

Figure 17: Difference of AP between our method and the baseline with different sorting schemes. Values higher than $y = 0$ mean that our method outperformed baseline. This evaluation is done on the final results reported in Table 4 with ($\lambda = 2$)



(a) COCO



(b) COCO-LT

Figure 18: #instances per category in log-scale sorted by category frequency in COCO and COCO-LT datasets.

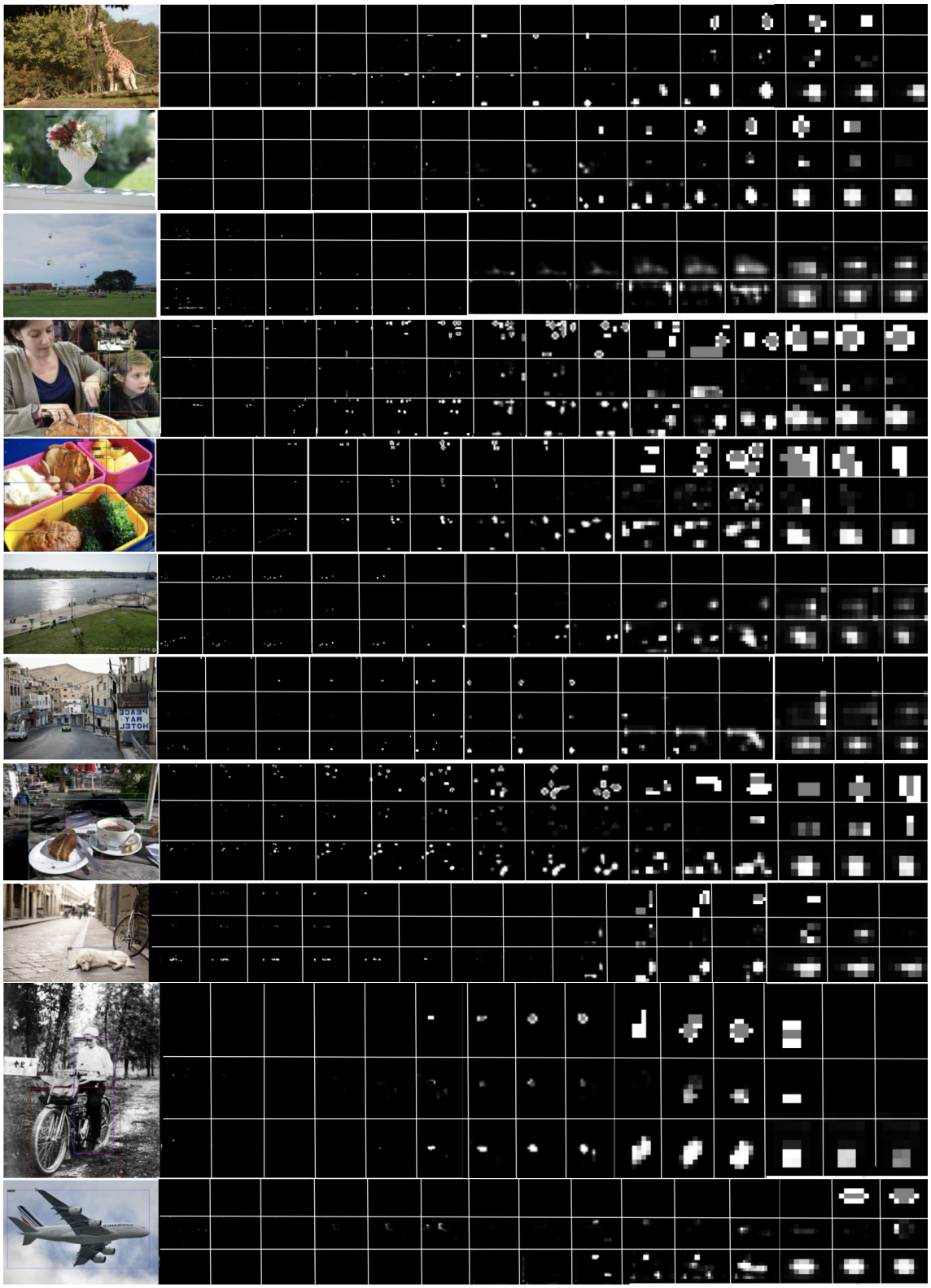


Figure 19: Samples of images and the corresponding ground-truth, loss and the betting maps at 5 different layers and across 3 scales.

Figure 20: Left: our method prediction, middle: baseline predictions, right: ground-truth annotations, for visualization purposes only proposals with confidence threshold $Conf > 0.5$ are shown.

Conclusion

In this work, we introduced a new adversarial training scheme for object detection that incorporates high-level structural information in predictions. We proposed a Full-Scale Gambler architecture that adapts Gambling networks in segmentation to object detection. We performed thorough ablation studies to validate our architectural design choices and performed an extensive analysis to determine why our method yielded minimal gain compared to the baseline.

We determine the following potential sore points as the cause:

1. One of the fundamental differences between segmentation and detection is: In segmentation, the entire output is evaluated both during training and inference. Nevertheless, in detection, during training, a lot of low-quality anchors are predicted and matched with a particular object; however, during inference, a lot of those anchors are removed by non-maximum suppression (NMS). Giving all the low-quality predictions to the Gambler could make betting on the incorrect predictions difficult because it sees a lot of predictions that will never "survive" NMS.

2. As mentioned in the introduction, the most challenging problem in one-stage object detection is the foreground-background imbalance, which leads to poor performance [2]. The output of the detection network consists mostly of backgrounds and a few foregrounds. In segmentation, the outputs are dense, while in detection, the outputs are sparse across predictions. The Gambler potentially suffers from the same foreground-background imbalance, considering the imbalance in predictions given as input.

3. Unlike binary cross-entropy loss where correct classifications are rewarded, and incorrect classifications are penalized, in the original Gambler loss formulation, wrong bets are not penalized, but only correct bets are rewarded. This formulation leads to unstable training and slows down convergence.

4. One of the biggest challenges for the Gambler is that it only has access to the approximate spatial location of predictions given by the (x, y) location of the anchor center on the prediction feature map, which is later sustained in the UNet. Although the anchor locations can be implicitly derived from the loss function, the lack of explicit anchor location and coverage could limit the Gambler's reasoning about the spatial extent of objects and their relationships.

5. Based on our ablation study on different architectures for the*PreGambler Image*, we observed that the Gambler does not exploit the input image, making it even harmful for performance in certain settings. This suggests that the Gambler is only using the predictions. By leveraging the information from the image alongside the predictions, the Gambler task would be much more straightforward.

6. The UNet structure is a clean way of aggregating predictions from several layers, but it has a disadvantage: the predictions coming from the highest resolution layer pass through the entire UNet, while the predictions coming from the lowest resolution only go through a few layers. This might suggest that not all predictions are considered and trained in the same way, potentially impacting the results.

Future Work To overcome some of the challenges, we have some insights and ideas that can be investigated further as future directions. In response to the first challenge, where low-quality anchors confuse the Gambler, a masked loss could be proposed where only high-quality anchors or anchors within a particular range of IoU are accepted.

As discussed before, the foreground-background imbalance is not only challenging in generic one-stage object detection but also in our method. Especially in our method, the minimum bet per location is quite small, but the sum of many bets could dominate the gradient and the loss function, which could be twice as harmful to the detector. These small values could be omitted with a thresholding function in the last layer of the Gambler.

To encourage faster convergence and stable training, we could propose the Gambler loss in the form of cross-entropy loss with masked classification loss as pseudo labels. Also, adapting the traditional GAN loss could be beneficial.

Incorporating box proposal coordinates in Gambler is extra challenging because they are relative to the image, and simply concatenating them with the predictions would probably not work, and besides, learning the spatial interpretation of coordinates is not trivial. One idea would be masking the inputs to the Gambler with a binary mask. Another is to use the regression loss as an additional supervision signal for the Gambler loss.

To conclude, we propose to make the task for the Gambler easier by limiting the inputs or outputs or proposing a more powerful proven loss function like the standard GAN loss that could help point to uncertainties better.

Bibliography

[1] L. Samson, N. van Noord, O. Booij, M. Hofmann, E. Gavves, and M. Ghafoorian, "I Bet You Are Wrong: Gambling Adversarial Networks for Structured Semantic Segmentation," Aug. 7, 2019. arXiv: 1908.02711 [cs] (cit. on pp. 1, 8–10, 13, 22).

[2] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," Feb. 7, 2018. arXiv: 1708.02002 [cs] (cit. on pp. 1, 4, 7, 16, 17, 19, 30).

[3] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft COCO: Common Objects in Context," Feb. 20, 2015. arXiv: 1405.0312 [cs] (cit. on pp. 2, 15, 18).

[4] A. Gupta, P. Dollár, and R. Girshick, "LVIS: A Dataset for Large Vocabulary Instance Segmentation," Sep. 15, 2019. arXiv: 1908.03195 [cs] (cit. on pp. 2, 15).

[5] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman, "Multiple kernels for object detection," in *2009 IEEE 12th international conference on computer vision*, IEEE, 2009, pp. 606–613 (cit. on p. 3).

[6] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, IEEE, vol. 1, 2001, pp. I–I (cit. on p. 3).

[7] H. Harzallah, F. Jurie, and C. Schmid, "Combining efficient object localization and image classification," in *2009 IEEE 12th international conference on computer vision*, IEEE, 2009, pp. 237–244 (cit. on p. 3).

[8] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, IEEE, vol. 1, 2005, pp. 886–893 (cit. on p. 3).

[9] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, no. 2, pp. 137–154, 2004 (cit. on p. 3).

[10] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*, Ieee, vol. 2, 1999, pp. 1150–1157 (cit. on p. 3).

[11] R. Lienhart and J. Maydt, "An extended set of haar-like features for rapid object detection," in *Proceedings. international conference on image processing*, IEEE, vol. 1, 2002, pp. I–I (cit. on p. 3).

[12] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*, Springer, 2006, pp. 404–417 (cit. on p. 3).

[13] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intelligent Systems and their applications*, vol. 13, no. 4, pp. 18–28, 1998 (cit. on p. 3).

[14] D. Opitz and R. Maclin, "Popular ensemble methods: An empirical study," *Journal of artificial intelligence research*, vol. 11, pp. 169–198, 1999 (cit. on p. 3).

[15] Y. Freund, R. E. Schapire, *et al.*, "Experiments with a new boosting algorithm," in *icml*, Citeseer, vol. 96, 1996, pp. 148–156 (cit. on p. 3).

[16] D. Hoiem, S. K. Divvala, and J. H. Hays, "Pascal voc 2008 challenge," in *PASCAL challenge workshop in ECCV*, Citeseer, 2009 (cit. on p. 3).

[17] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004 (cit. on p. 3).

[18] T. Ojala, M. Pietikainen, and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 7, pp. 971–987, 2002 (cit. on p. 3).

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105 (cit. on p. 3).

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778 (cit. on p. 3).

[21] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99 (cit. on p. 3).

[22] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587 (cit. on pp. 3, 4).

[23] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," Jan. 24, 2018. arXiv: 1703.06870 [cs] (cit. on p. 4).

[24] R. Girshick, "Fast R-CNN," Sep. 27, 2015. arXiv: 1504.08083 [cs] (cit. on p. 4).

[25] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," Jan. 6, 2016. arXiv: 1506.01497 [cs] (cit. on p. 4).

[26] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," May 9, 2016. arXiv: 1506.02640 [cs] (cit. on pp. 4, 5).

[27] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," vol. 9905, pp. 21–37, 2016. arXiv: 1512.02325 [cs] (cit. on p. 4).

[28] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013 (cit. on p. 4).

[29] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, *Deep learning for generic object detection: A survey*, 2018. arXiv: 1809.02165 [cs.CV] (cit. on pp. 4, 5).

[30] X. Wu, D. Sahoo, and S. C. H. Hoi, *Recent advances in deep learning for object detection*, 2019. arXiv: 1908.03673 [cs.CV] (cit. on pp. 6, 8).

[31] (). [1512.02325] SSD: Single Shot MultiBox Detector, [Online]. Available: https://arxiv.org/abs/1512.02325 (visited on 11/14/2019) (cit. on p. 6).

[32] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, 2015. arXiv: 1505.04597 [cs.CV] (cit. on p. 11).

[33] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, *Accurate, large minibatch sgd: Training imagenet in 1 hour*, 2017. arXiv: 1706.02677 [cs.CV] (cit. on p. 16).

[34] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. arXiv: 1412.6980 [cs.LG] (cit. on p. 16).

[35] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, no. 3, pp. 400–407, Sep. 1951 (cit. on p. 16).

[36] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, *Detectron2*, https://github.com/facebookresearch/detectron2, 2019 (cit. on p. 17).

A

In order to calculate maximum and minimum betting map, we can use Equation 21. There are approximately $25k$ anchors, thus the minimum and maximum bet can be calculated as in Equation 22. The difference between the bets becomes larger as β goes to 0.

$$\begin{aligned} B_{max} &= \frac{1+\beta}{\#\text{anchors} \times \beta + 1} \\ B_{min} &= \frac{\beta}{\#\text{anchors} \times \beta + 1} \\ \#\text{anchors} &= (80 \times 80 \times 3 + 40 \times 40 \times 3 + 20 \times 20 \times 3 + 10 \times 10 \times 3 + 5 \times 5 \times 3) \\ &= 25575 \end{aligned} \tag{21}$$

Figure 21 shows the maximum bet during training across different β values.

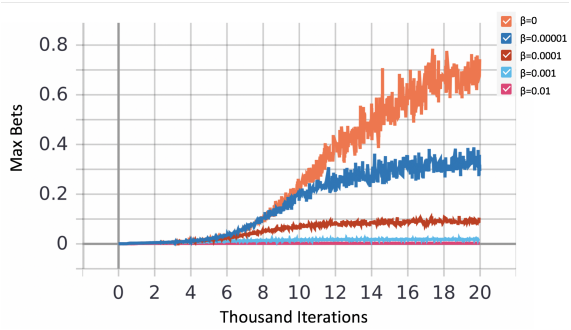
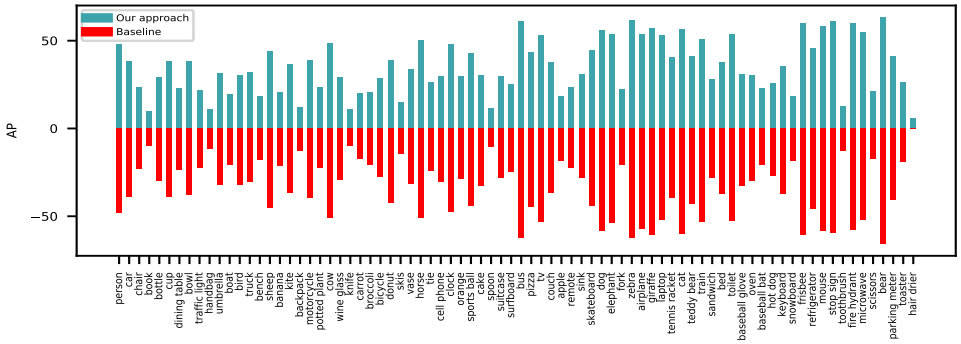
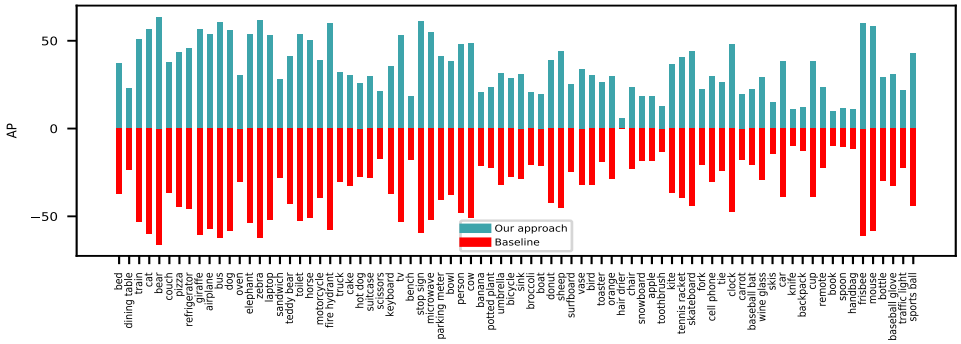


Figure 21: shows the maximum bet during only training the gambler and fixing the detector. After $20k$ iterations, $\beta = 0$ achieves the highest bet.

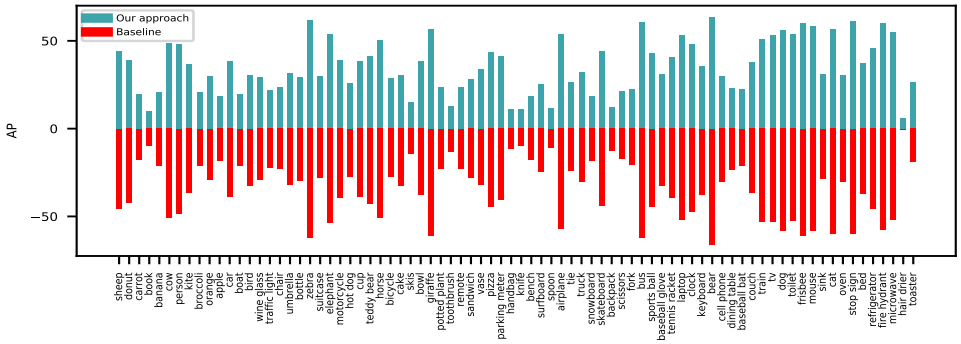
$$\begin{cases} B_{max} = 1 & \beta = 0 \\ B_{min} = 0 & \beta = 0 \end{cases} \begin{cases} B_{max} = 0.00393 & \beta = 0.01 \\ B_{min} = 0.00003 & \beta = 0.01 \end{cases} \begin{cases} B_{max} = 0.00042 & \beta = 0.1 \\ B_{min} = 0.00003 & \beta = 0.1 \end{cases} \begin{cases} B_{max} = 0.00007 & \beta = 1 \\ B_{min} = 0.00003 & \beta = 1 \end{cases} \tag{22}$$



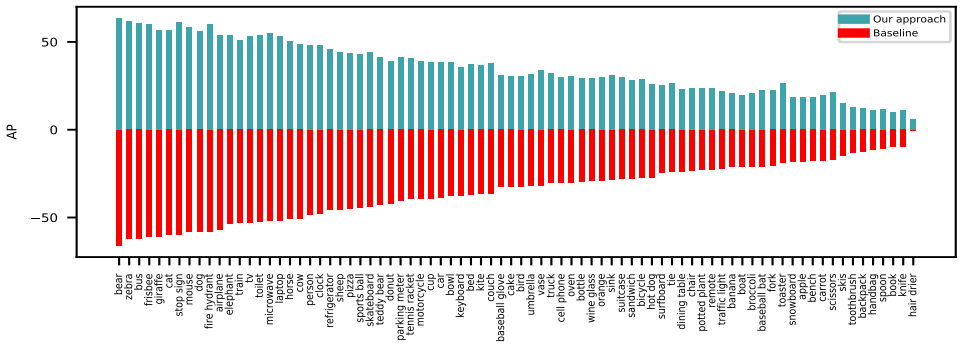
(a) Frequency



(b) Size



(c) Class-Entropy



(d) Baseline AP

Figure 22: The per-class AP values comparing our approach (blue) to the baseline (red), categories are sorted by frequency, object size, average class entropy, and baseline AP from top to bottom.

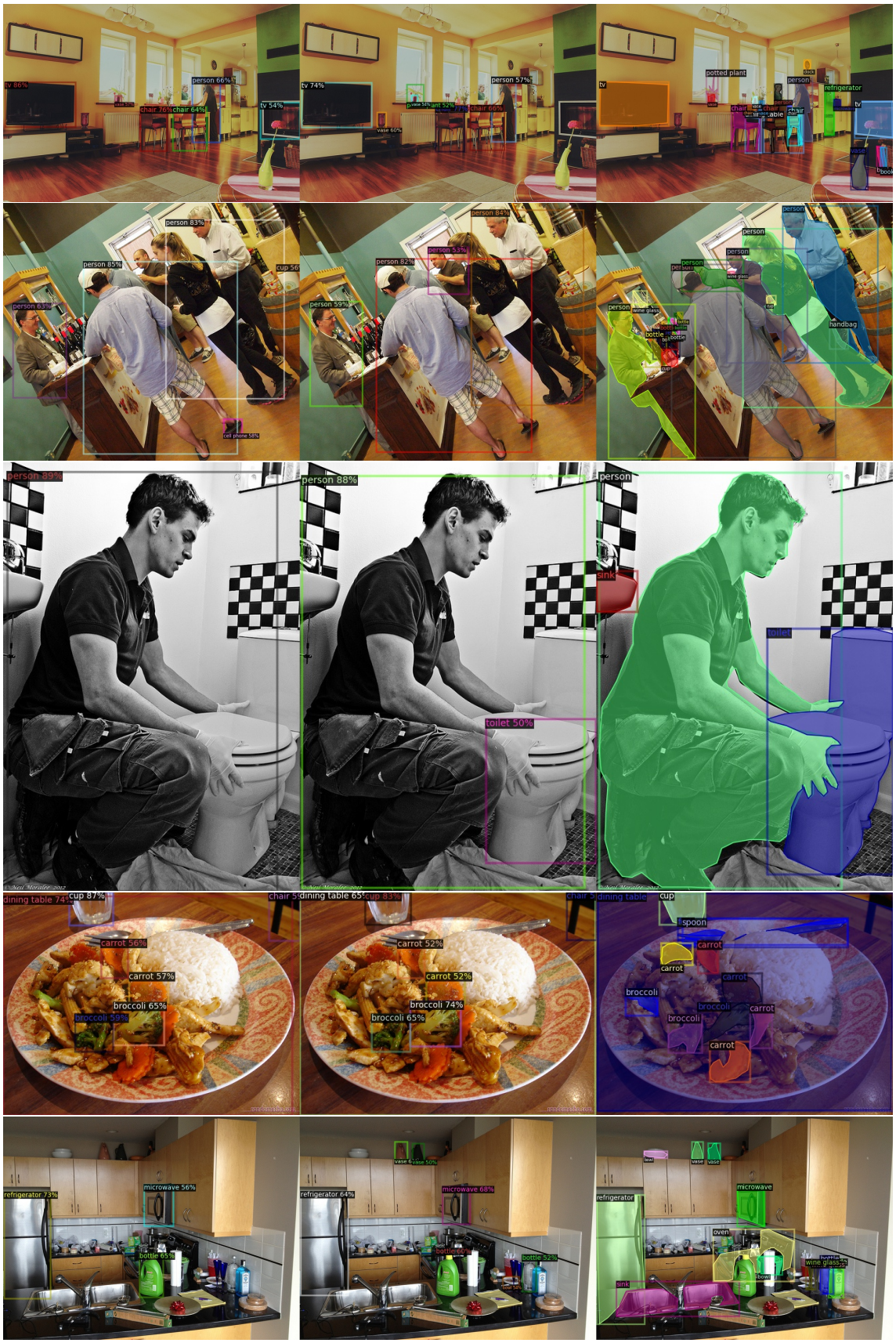


Figure 23: A few more examples of the comparison of our method and the baseline proposals.