End-to-end training of object class detectors for mean average precision

Paul Henderson & Vittorio Ferrari

University of Edinburgh

Abstract. We present a method for training CNN-based object class detectors directly using mean average precision (mAP) as the training loss, in a truly end-to-end fashion that includes non-maximum suppression (NMS) at training time. This contrasts with the traditional approach of training a CNN for a window classification loss, then applying NMS only at test time, when mAP is used as the evaluation metric in place of classification accuracy. However, mAP following NMS forms a piecewise-constant structured loss over thousands of windows, with gradients that do not convey useful information for gradient descent. Hence, we define new, general gradient-like quantities for piecewise constant functions, which have wide applicability. We describe how to calculate these efficiently for mAP following NMS, enabling to train a detector based on Fast R-CNN [1] directly for mAP. This model achieves equivalent performance to the standard Fast R-CNN on the PASCAL VOC 2007 and 2012 datasets, while being conceptually more appealing as the very same model and loss are used at both training and test time.

1 Introduction

Object class detection is the task of localising all instances of a given set of object classes in an image. Modern techniques for object detection [1–4] use a convolutional neural network (CNN) classifier [5, 6], operating on object proposal windows [7–9]. Given an image, they first generate a set of windows likely to include all objects, then apply a CNN classifier to each window independently. The CNN is trained to output one score for each possible object class on each window, and an additional one for 'background' or 'no object'. Such models are trained for window classification accuracy: the loss attempts to maximise the number of training windows for which the CNN gives the highest score to the correct class. At test time, the CNN is applied to every window in a test image, followed by a non-maximum suppression processing stage (NMS). This eliminates windows that are not locally the highest-scored for a class, yielding the output set of detections. Typically, the performance of the detector is evaluated using mean average precision (mAP) over classes, which is based on the ranking of detection scores for each class [10].

Thus, the traditional approach is to train object detectors with one measure, classification accuracy over all windows, but test with another, mAP over locally highest-scoring windows. While the training loss correlates somewhat with the test-time evaluation metric, they are not really the same, and furthermore, training ignores the effects

of NMS. As such, the traditional approach is not true end-to-end training for the final *detection* task, but for the surrogate task of window *classification*.

In this work, we present a method for training object detectors directly using mAP computed after NMS as the loss. This is in accordance with the machine learning dictum that the loss we minimise at training time should correspond as closely as possible to the evaluation metric used at test time. It also fits with the recent trend towards training models end-to-end for their ultimate task, in vision [11–13] and other areas [14, 15], rather than training individual components for engineered sub-tasks, and combining them by hand.

Directly optimising for mAP following NMS is very challenging for two main reasons: (i) mAP depends on the global ordering of class scores for all windows across all images, and as such is piecewise constant with respect to the scores; and, (ii) NMS has highly non-local effects within an image, as changing one window score can have a cascading effect on the retention of many other windows. In short, we have a structured loss over many thousands of windows, that is non-convex, discontinuous, and piecewise constant with respect to its inputs. Our main contribution is to overcome these difficulties by proposing new gradient-like quantities for piecewise constant functions, and showing how these can be computed efficiently for mAP following NMS. This allows us to train a detector based on Fast R-CNN [1] in a truly end-to-end fashion using stochastic gradient descent, but with NMS included at training time, and mAP as the loss.

Experiments on the PASCAL VOC 2007 and 2012 detection datasets [16] show that end-to-end training directly for mAP with NMS reaches equivalent performance to the traditional way of training for window classification accuracy and without NMS. It achieves this while being conceptually simpler and more appealing from a machine learning perspective, as exactly the same model and loss are used at both training and test time. Furthermore, our method is widely applicable on two levels: firstly, our loss is a simple drop-in layer that can be directly used in existing frameworks and models; secondly, our approach to defining gradient-like quantities of piecewise-constant functions is general and can be applied to other piecewise-constant losses and even internal layers. For example, using our method can enable training directly for other rank-based metrics used in information retrieval, such as discounted cumulative gain [17]. Moreover, we do not require a potentially expensive max-oracle to find the most-violating inputs with respect to the model and loss, as required by [18, 19, 2].

2 Background

We recap here how NMS is performed (Sec. 2.1) and mAP calculated (Sec. 2.2). Then, we describe Fast R-CNN [1] in more detail (Sec. 2.3), as it forms the basis for our proposed method.

2.1 Non-maximum suppression (NMS)

Given a set of windows in an image, with scores for some object class, NMS removes those windows which are not locally the highest-scored, to yield a final set of detec-

tions [20]. Specifically, all the windows are marked as retained or suppressed by the following procedure: first, the highest-scored window is marked as retained, and all those overlapping with it by more than some threshold (*e.g.* 30% in [1, 4]) intersection-over-union (IoU) are marked as suppressed; then, the highest-scored window neither retained nor suppressed is marked as retained, and again all others sufficiently-overlapping are marked as suppressed. This process is repeated until all windows are marked as either retained or suppressed. The retained windows then constitute the final set of detections.

2.2 Mean Average Precision (mAP)

The mAP [16, 21, 10] for a set of detections is the mean over classes, of the interpolated AP [22] for each class. This per-class AP is given by the area under the precision/recall (PR) curve for the detections (Fig. 1).

The PR curve is constructed by first mapping each detection to its most-overlapping ground-truth object instance, if any overlaps sufficiently—for PASCAL VOC, this is defined as overlapping with $> 50\%$ IoU [16]. Then, the highest-scored detection mapped to each ground-truth instance is counted as a true-positive, and all other detections as false-positives. Next, we compute recall and precision values for increasingly large subsets of detections, starting with the highest-scored detection and adding the remainder in decreasing order of their score. Recall is defined as the ratio of true-positive detections to ground-truth instances, and precision as the ratio of true-positive detections to all detections. The PR curve is then given by plotting these recall-and-precision pairs as progressively lower-scored detections are included. Finally, dips in the curve are filled in (interpolated) by replacing each precision with the maximum of itself and all precisions occurring at higher recall levels (pink shading in Fig. 1) [10, 22].

The area under the interpolated PR curve is the AP value for the class. For the PASCAL VOC 2007 dataset, this area is calculated by a rough quadrature approximation sampling at 11 uniformly spaced values of recall [10]; for the VOC 2012 dataset it is the true area under the curve [16].

2.3 Fast R-CNN

Model. Our model is based on Fast R-CNN [1] (Figs. 2a, 2b), without bounding-box regression. This model operates by classifying proposal windows of an image, as belonging to one of a set of object classes, or as 'background'. Whole images are processed by a sequence of convolutional layers; then, for each window, convolutional features with spatial support corresponding to that window are extracted and resampled to fixed dimension, before being passed through three fully-connected layers, the last of which yields a score for each object class and 'background'. The class scores for each window are then passed through a softmax function, to yield a distribution over classes.

Training. This network is trained with a window classification loss. If a window overlaps a ground-truth object with IoU > 0.5, its true class is defined as being that object class; otherwise, its true class is 'background'. For each window, the network outputs softmax probabilities for each class, and the negative log likelihood (NLL) of the true class is used as the loss for that window; the total loss over a minibatch is simply a

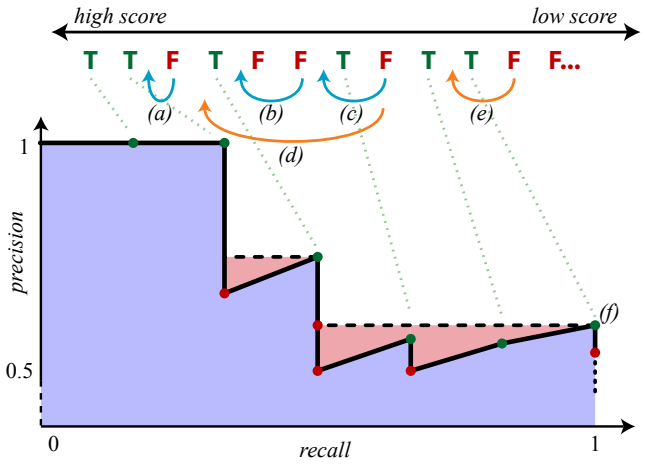


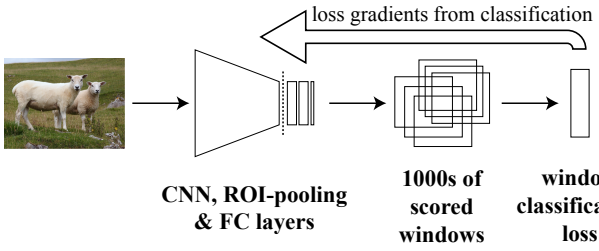
Fig. 1: Precision/recall curve (bottom) for a sequence of true-positive (TP) and false-positive (FP) detections ordered by score (top) for some object class with six ground-truth instances. Plotting the sequence of precision and recall values yields the black curve. The pink area shows the result of replacing each precision with the maximum at same or higher recall. AP is the total area of the pink and blue regions. The arrows (a-e) show the effect of positive perturbations to scores of FP detections. Blue arrows (a-c) show perturbations with no effect on AP: (a) the order of detections does not change; (b) the detection swaps places with another FP; (c) the detection swaps places with a TP, but a higher-recall TP (f) has higher precision so there is no change to area under the filled-in curve (pink shading). Orange arrows (d-e) show perturbations that do affect AP: (d) the same FP as (c) is moved beyond a TP that does appear on (hence affect) the filled in curve; (e) the FP moves past a single TP, altering the filled-in curve as far away as 0.5 recall.

sum of the losses over all windows in it. The network is trained by stochastic gradient descent (SGD) with momentum, operating on minibatches of two images at a time.

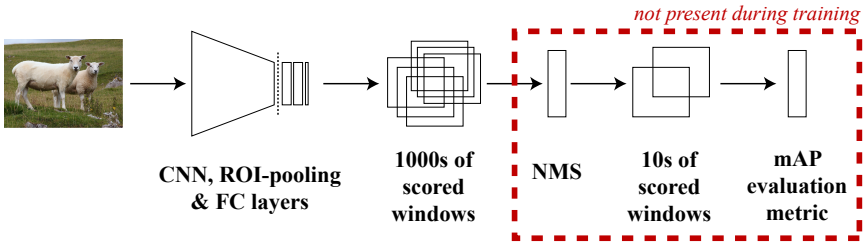
Testing. At test time, windows are scored by passing them forwards through the network, and recording the final softmax probabilities for each class. Then, for each class and image, NMS is applied to the scored windows (Sec. 2.1). Note that this NMS stage is not present at training time. Finally, the detections are evaluated using mAP over the full test set.

3 Related Work

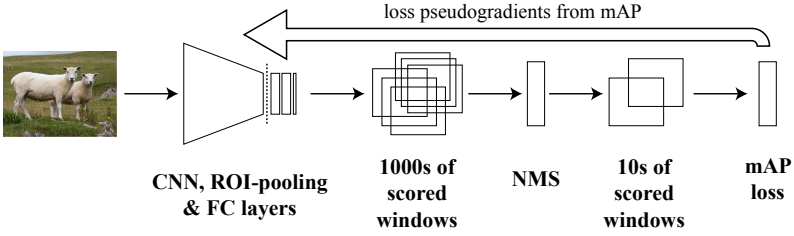
Nearly all works on object class detection train a window classifier, and ignore NMS and mAP at training time. Earlier approaches [20, 23–25] apply the classifier to all windows in a dense regular grid, while more recently, object proposal methods [7, 9] have been used to greatly reduce the number of windows [8, 4]. Below we review the



(a) Fast R-CNN, training: scores passed directly to window classification loss



(b) Fast R-CNN, testing: NMS applied, and detections evaluated with mAP



(c) Our method, both training and testing: exactly the same operations occur at train and test time , with identical model structure and the training loss matching the test-time evaluation metric

Fig. 2: Fast R-CNN [1] architecture during training (a) and testing (b) phases, and our architecture (c), which is the same in both phases.

few works that try to either train for AP or other structured losses, or include NMS at training time.

Blaschko et al. [26] formulate object detection as a structured prediction problem, outputing a binary indicator for object presence and a set of bounding-box coordinates. This is trained using a structured SVM, with a task loss that aims for correct classification and maximal IoU of predicted and ground-truth boxes in images containing the target class. Like our method, this is a structured loss involving IoU of detections and ground-truth objects; however, it does not correspond to maximising AP, and only a single detection is returned in each image, so there is no NMS. More recently, [2] uses the same structured SVM loss, but with a CNN in place of a kernelised linear model over SURF features [26]. This work directly optimises the structured SVM loss via gradient descent, allowing backpropagation to update the nonlinear CNN layers.

There exist works that train specifically for AP, but for classification problems, rather than for object detection with NMS. Yue et al. [18] optimizes AP in the structured SVM framework—with a linear model, trained using a hinge loss weighted according to AP. This requires solving a loss-augmented inference problem, *i.e.* finding the scores that maximise the sum of AP and the output of the current model. They present a dynamic programming algorithm to solve this, which has quadratic complexity in the number of training points. Extending this work, [19] presents a more general technique for training nonlinear structured models directly for non-differentiable losses, again assuming that loss-augmented inference can be performed efficiently. Using the same dynamic-programming approach as [18], they apply it to the case of single-class AP with a model based on R-CNN [4], without NMS at training time. While their method requires changes to the optimiser itself, ours does not. Instead, we simply define a new loss layer that can be easily dropped into existing frameworks, and do not require solving a loss-augmented inference problem. Furthermore, our approach can incorporate NMS and train simultaneously for multiple classes. Thus, while [19] trains for AP over binary window classification scores, ours trains directly for mAP over object detections.

Taylor et al. [27] discuss a different formulation for gradient-descent optimisation of certain losses based on ranking of scores (though not AP specifically). They define a smooth proxy loss for a non-differentiable, piecewise constant ranking loss. They treat the predicted score of each training point as a Gaussian random variable centered on the actual value, and hence compute the distribution of ranks for each score, by pairwise comparisons to all other scores. This distribution is used in place of the usual hard ranks when evaluating the loss, and the resulting quantity is differentiable with respect to the original scores. This method has cubic complexity in the number of training samples, making it intractable when there are tens of classes and thousands of windows (e.g. in PASCAL VOC).

Unlike most other approaches to object detection, [28] includes NMS at training time as well as test time. They use a deformable parts model over CNN features, that outputs scored windows derived from a continuous response map (in contrast to feeding fixed proposal windows through a CNN [1]). The windows are passed through a non-standard variant of NMS. Instead of training for mAP or window classification accuracy, the authors then introduce a new structured loss. This includes terms for detections retained by NMS, but also for suppressed windows, in a fashion requiring knowledge of which detection suppressed them. As such, it is deeply tied to the NMS implementation at training time, rather than being a generally-applicable loss such as mAP.

4 Proposed Method

We now describe our proposed method (Fig. 2c). We discuss how our model differs from Fast R-CNN (Sec. 4.1) and why it is challenging to train (Sec. 4.2). Then we introduce our general method for defining gradients of piecewise-constant functions (Sec. 4.3) and how we apply it to train our model (Sec. 4.4).

4.1 Detection Framework

Model. Our model is identical to Fast R-CNN as described above, up to the softmax layer: windows are still scored by passing through a sequence of convolutional and fully-connected layers. As in [1], we can use different convolutional network architectures pretrained for ILSVRC 2012 [21] classification, such as AlexNet [5] or VGG16 [6]. We omit the softmax layer, using the activations of the last fully-connected layer directly as window scores. In our experiment we found that the softmax has little effect on the final performance, but its tendency to saturate causes problems with propagating the loss gradients back through it. In contrast to Fast R-CNN, our model also includes an NMS layer immediately after the last fully-connected layer, which performs the same operation as used at test time for Fast R-CNN. We regard the NMS layer as part of the model itself, present at both training and test time.

Training. During training, we add a loss layer that computes mAP over the minibatch, after NMS. Thus, at training time, minibatches undergo exactly the same sequence of operations as at test time, and the training loss matches the test-time evaluation metric. The network is still trained using SGD with momentum. Section 4.2 describes how to define derivatives of the mAP and NMS layers, while Sec. 4.5 discusses some additional techniques used during training.

Testing. During testing, our method is identical to Fast R-CNN, except that the softmax layer is omitted.

4.2 Gradients of mAP and NMS Layers

In order to minimise our loss by gradient descent, we need to propagate derivatives back to the fully-convolutional layers of the CNN and beyond. However, mAP is a piecewise constant function of the detection scores, as it depends only on their ordering—each score can be perturbed slightly without changing the loss. The partial derivatives of such a loss function do not convey useful information for gradient descent (Fig. 3a) as they are almost everywhere zero (in the constant regions), and otherwise undefined (at the steps). The subgradient is also undefined, as the function is non-convex.

Furthermore, even if we could compute the derivatives of mAP with respect to the class scores, they still need to be propagated back through the NMS layer. This requires a definition of the Jacobian of NMS, which is again non-trivial. Note that max-pooling layers are similarly non-differentiable, but good results are achieved by simply propagating the gradient back to the maximal input only. We could do similar for NMS: allow only the locally-maximal windows propagate gradients back; however, this loses valuable information. For example, if all detections overlapping some ground-truth object are suppressed, then there should be a gradient signal favouring increasing the score of those windows (or decreasing that of their suppressors). This does not occur if we naïvely copy gradients back through to maximal windows. In contrast, we require a Jacobian-like quantity for NMS that does capture this information.

We therefore develop general definitions for gradient-like quantities of piecewise-constant functions in Sec. 4.3, and then describe how to apply them efficiently to NMS and mAP in Sec. 4.4.

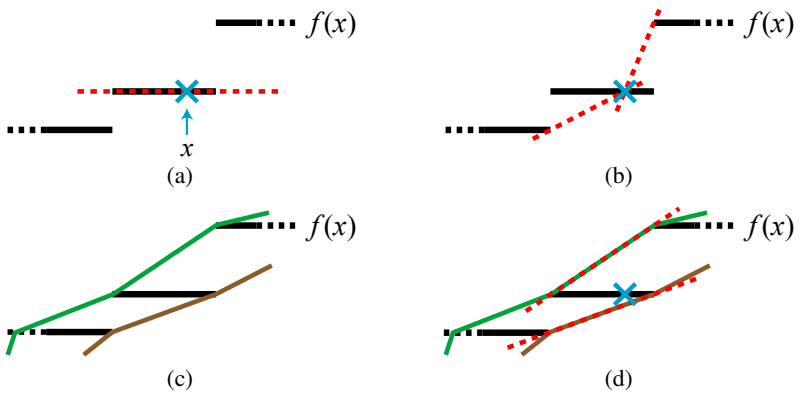


Fig. 3: A piecewise constant function $f(x)$ with steps at two points, and various definitions for gradients. (a) Conventional partial derivative (red dashed) at x, equal to zero, does not convey useful information for gradient descent. (b) Gradients at x given by positive-perturbing and negative-perturbing finite difference estimators. (c) Piecewise-linear upper (green) and lower (brown) envelopes of $f(x)$. (d) Gradients at x given by slope of upper/lower envelopes. When applied to our model, $f(x)$ is mAP, and the horizontal axis corresponds to the score of a single window with respect to which the partial derivative is being computed.

4.3 Pseudogradients of General Piecewise-Constant Functions

We consider how to define a general *pseudo partial derivative* (PPD) operation for piecewise-constant functions, that can be used to define quantities analogous to the gradient and the Jacobian. For any piecewise-constant function $f(\mathbf{x})$ with countably many discontinuities (steps), we denote the PPD with respect to x_i by $\widetilde{\partial}_{x_i} f$. When the PPD is non-zero we need to move some non-infinitesimal distance before any change in the function occurs (unlike a conventional partial derivative). However when there is a change, it will be in the direction indicated by the PPD, and in magnitude corresponding to the PPD (this is made more precise below). We then use our PPD to define an analogue to the gradient by $\widetilde{\nabla} f = (\widetilde{\partial}_{x_1} f, \ldots, \widetilde{\partial}_{x_N} f)$. Intuitively, this tells us locally what direction to move so that the function will decrease, if we move some non-infinitesimal distance in this direction. Similarly, for the Jacobian of vector-valued $\mathbf{f}$, we have $\widetilde{J}_{ij} = \widetilde{\partial}_{x_j} f_i$.

We now discuss two possible definitions for the PPD; these and the regular partial derivative are illustrated in Fig. 3 for a one-dimensional function, at a point lying in a constant region between two steps.

Finite difference estimators. Most simply, we can apply a traditional single-sided finite difference estimator, as used for computing numerical gradients of a differentiable function. Here, a small, fixed perturbation δx is added to x, the function evaluated at this point, and the resulting slope used to approximate the gradient, by $\widetilde{\partial}_x f = \frac{f(x+\delta x) - f(x)}{\delta x}$. The piecewise-constant functions we are interested in have finitely many steps, and so

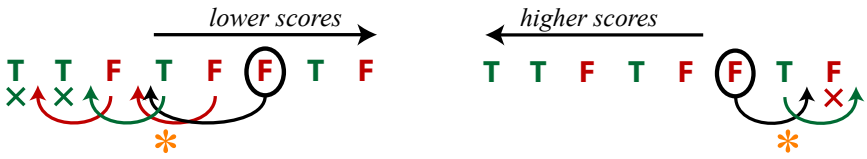
the probability of f being undefined at the perturbed point is zero. However, the constant regions of our function vary in size by several orders of magnitude, and so it is impossible to pre-select a suitable value for δx. Instead, we use an adaptive approach: given x, set δx to the smallest value such that $f(x + \delta x) \neq f(x)$, then compute $\widetilde{\partial}_x f$ as above (Fig. 3b). Note that this method is single-sided: it only takes account of the change due to perturbing x in one direction or the other. This is undesirable, as in general, it delivers different results for each direction, perhaps yielding complementary information. We address this issue by performing the same calculation independently with positive then negative perturbations δx^+ and δx^-, and taking a mean of the resulting pseudogradients. We refer to this mean pseudogradient as SDE, for symmetric difference estimator. This approach has the disadvantage that the magnitude of the gradient is sensitive to the exact location of x: if it is nearer to a step, the gradient will be larger, yet a correspondingly larger change to the network parameters may be undesirable.

Linear envelope estimators. An alternative approach to defining the PPD is to fit a piecewise-linear upper or lower envelope to the steps of the piecewise-constant function (Fig. 3c). The PPD $\widetilde{\partial}_x f$ is then given by the slope of the envelope segment at the point x (Fig. 3d). In practice, we take the average of the gradients of the upper and lower envelopes. Unlike SDE, this estimator does not become arbitrarily large as x approaches a step. If f has finitely many steps, then for all points before the first step and after the last, both linear envelopes have zero gradient; we find however that better results are achieved by using SDE in these regions, but with an empirically-tuned lower-bound on δx. We refer to this pseudogradient as MEE, for mean envelope estimator.

4.4 Application to mAP and NMS

To apply the above methods to mAP, we must compute the PPD of each class' AP with respect to each window score independently, holding the other scores constant. This raises two questions: (i) how to efficiently find the locations of the nearest step before and after a point, and (ii) how to efficiently evaluate the loss around those locations. We solve these problems by noting that changes to AP only occur when two scores change their relative ordering, and even then, only in certain cases. Specifically, AP changes value only when a window counted as a true-positive changes place with one counted as a false-positive. Also, the effective precision at a given recall is the maximum precision at that or any higher recall (Sec. 2.2 and Fig. 1). So we have further conditions, *e.g.* decreasing the score of a false-positive only affects AP when it drops below that of a true-positive at which precision is higher than any with even lower score. This effect and other perturbations are illustrated in Fig. 1 (blue and orange arrows).

Thus, for each class, we can find the nearest step before and after each point by making two linear passes over the detections, in descending then ascending order of score (Fig. 4). Assuming we have computed AP as described in Sec. 2.2, we know whether each detection is a true- or false-positive, and can keep track of the last-seen detection of each kind. In the descending pass, for each detection, we find the smallest increase to its score that would result in a change to AP, thus giving the location of the nearest step on the positive side. This score increase is that which moves it an infinitesimal

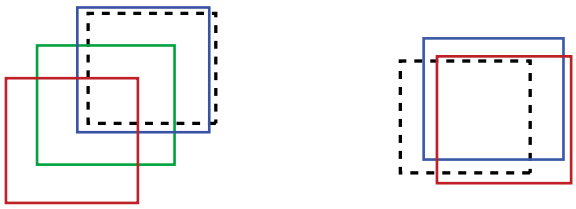


(a) (b)

Fig. 4: Efficient calculation of smallest perturbations to detection scores to cause a step in AP. In each case the circled FP is currently being considered. (a) Iterating detections in decreasing order of score, finding the smallest increase to each score that causes a change in AP (higher for TPs, lower for FPs). Detections already considered have an arrow showing where they are perturbed to; a cross indicates no increase to that score affects AP. When considering the circled FP, the last-seen TP is shown by the orange asterisk; perturbing the score of the circled detection just beyond (left) of this is the minimal change to affect AP. (b) Similar but iterating in increasing order of score, and hence calculating minimal decreases in score to affect AP.

amount higher than the score of the last-seen window of the other kind (true-positive vs. false-positive), subject to the additional conditions mentioned above. Similarly, in the ascending pass, we can find the required decreases in scores that would cause a change in AP. Once the step locations have been found, the new AP values resulting from perturbing the scores accordingly can be calculated by updating the relevant part of the PR curve, and then computing its area as normal. Given the step locations and AP values, it is then straightforward to use the methods of Sec. 4.3 to compute the SDE or MEE.

Incorporating NMS. We must also account for NMS when propagating gradients back. The PPDs of NMS can be used to define a Jacobian as described in Sec. 4.3, which may then be composed with the pseudogradient of mAP to define the gradient of mAP with respect to the pre-NMS scores. However, subject to a small approximation, it is both easier and more efficient to consider NMS simultaneously with AP when determining step locations and the resultant changes to the loss. Specifically, we introduce two transitivity approximations (Fig. 5): (i) we do not attempt to model cascaded long-distance interactions between windows through multiple steps of NMS; (ii) we assume in certain cases that windows suppressed by some detection overlap exactly the same ground-truth instances as the detection itself. Under these approximations, it is possible to compute the PPDs with respect to pre-NMS scores in linear time in the number of windows. This is achieved by: (i) adding gradient contributions due to windows suppressed by a true-positive or false-positive detection at the same time as that detection, as these suppressed windows need to have their scores perturbed to the same point as their suppressor did to cause a change in AP; (ii) including a third pass that adds gradient contributions from suppressed windows overlapping ground-truth instances that were missed entirely (*i.e.* no detection covers them); (iii) also adding gradient contributions from the detections that caused the suppressed-but-overlapping windows of (ii) to be suppressed.



(a) (b)

Fig. 5: Transitivity approximations for NMS. Dashed black box is a ground-truth object; coloured boxes are scored windows, red $>$ green $>$ blue. (a) Red overlaps green sufficiently for NMS inhibition, and green overlaps blue similarly, but red does not overlap enough with blue. However, whether red is retained indirectly affects whether blue is retained, as if red suppresses green, then green does not suppress blue. In our approximation, this long-distance interaction between red and blue is ignored; however the two local interactions (red-green and green-blue) are included. (b) Red and blue overlap each other sufficiently for NMS inhibition; given that red suppresses blue, our approximation assumes that blue overlaps the same ground-truth instance as red (if any).

4.5 Training Protocol

In order to train our model successfully, we make various changes to the training protocol used for Fast R-CNN in [1]. The impact of each of these changes is given in Sec. 5.

Minibatch composition. We use larger minibatches than [1], as (i) object detection mAP has a much higher batch-to-batch variance than simple window classification accuracy, and (ii) including more windows increases the density of the gradient signal, as there are likely to be more false positives which score higher than some true positive (and vice versa). We also find that performance is improved by using proportionally fewer foreground windows (those overlapping a ground-truth instance as opposed to background) in each training minibatch. While Fast R-CNN uses 25% foreground windows, we use 5%, which roughly corresponds to the distribution of windows seen at test time, when 5% of all selective search proposals overlap a ground-truth instance.

Regularisation. Using our method, we found empirically that scores are prone to grow very large after several hundred iterations of training. This is effectively mitigated by introducing a regulariser on the window scores. We find that an L4 regulariser with very small weight performs best, as it gives greater freedom to smaller-magnitude scores while imposing a relatively hard constraint on magnitude, compared to the more common L1/L2 regularisation.

Log-space. We find it is beneficial to follow gradients of $\log(\text{mAP}+\epsilon)$ instead of mAP itself, for some small, fixed constant ϵ. Early in training when mAP is low, scores of true-positive windows are uniformly distributed amongst those of false-positive windows, and so an increase in the score of a true-positive often yields only a very small gain in mAP. Using $\log(\text{mAP}+\epsilon)$ instead amplifies the effect of these changes, so training quickly escapes from the initial very low mAP.

Table 1: Performance of our method measured by mAP on VOC 2007 test set, with different pseudogradients (MEE vs SDE), network architectures (AlexNet vs VGG16), and training sets (VOC 2007 trainval vs union of VOC 2007 trainval and VOC 2012 trainval). We also give results for Fast R-CNN trained using a traditional softmax loss, without bounding box regression.

trained on...	*2007 only*		*2007 + 2012*	
	AlexNet	VGG16	AlexNet	VGG16
Ours, MEE	51.6	58.9	54.9	62.5
Ours, SDE	51.3	60.7	54.8	62.3
Fast R-CNN	52.0	62.4	53.8	63.5

Gradient clipping. We find that numerical behaviour is improved (particularly at high learning rates) by clipping elements of the gradient to a fixed threshold.

5 Experiments

We now evaluate the performance of our approach on two datasets: PASCAL VOC 2007 and 2012 [16]. Both datasets have 20 object classes; for VOC 2007, we train on the trainval subset (5011 images) and test on the test subset (4952 images); for VOC 2012, we train on the train subset (5717 images) and test on the validation subset (5823 images). We also give results training on the union of VOC 2007 trainval and VOC 2012 trainval (total 16551 images), and testing on VOC 2007 test.

We compare our method to two others: (i) Fast R-CNN trained with the standard NLL loss for window classification, as described in [1] (bounding box regression is disabled, to give a fair comparison with our method); and (ii) [19], which also trains an R-CNN-like model for AP, but with a separate model for each class, no NMS at training time, and with a different way to compute parameter gradients. This is the closest work in spirit to ours.

Settings. We use Fast R-CNN as described in [1], built upon AlexNet [5] or VGG16 [6], with weights initialised on ILSVRC 2012 classification [21]. We then remove the softmax layers at both training and test time, as described in Sec. 4.1, and replace the training loss layer with our NMS layer and mAP loss.

Incorporating the techniques described in Sec. 4.5, the overall loss we minimise by SGD is $L = -\log\{\sum_c \text{AP}(\text{NMS}(\mathbf{s}_c))/K\} + \lambda \sum_{c,b} |s_c^b|^4$, where $\mathbf{s}_c$ are the window scores for class c, K is the total number of classes, and b indexes over windows.

The AP calculation during training is always matched to that used for evaluation. When testing on VOC 2007, we train using the VOC 2007 approximation to AP (Sec. 2.2); when testing on VOC 2012, we train using the true AP. In order to compute pseudogradients for training, we try both SDE and MEE and compare their performance (Sec. 4.3). As our method works best with large minibatches, for the VGG16 experiments, we clamp the maximum image dimension to 600 pixels, to conserve GPU memory (this does not have a significant impact on the baseline performance).

Main results on VOC 2007. Table 1 shows how our methods compare with Fast R-CNN, testing on the PASCAL VOC 2007 dataset. Overall, our method achieves comparable performance to Fast R-CNN. The results also show that using a larger training set (union of VOC 2007 and 2012 trainval subsets) increases performance by up to 3.6% mAP, compared to training from VOC 2007 trainval alone. This effect is significantly stronger for our method than for Fast R-CNN: for AlexNet, we gain 3.3% mAP compared with 1.8% for Fast R-CNN; for VGG16, we gain 3.6% compared with 1.1% for Fast R-CNN. This indicates that our approach particularly benefits from more training data, possibly because optimising for mAP implies many comparisons between windows. Of our two pseudo-gradient estimators, MEE slightly outperforms SDE, in all cases apart from VGG16 training on VOC 2007 trainval only. This is likely because MEE is insensitive to the distances from points to nearest steps, in contrast to SDE (Sec. 4.3); hence, MEE is a more robust estimator of the impact of a score change, whereas SDE may introduce very large derivatives for a particular window. In all cases, VGG16 significantly outperforms AlexNet, confirming previous studies [6, 1].

Ablation study. In Sec. 4.5, we noted that certain modifications to the original training procedure of Fast R-CNN were necessary to achieve these results. Ablating away these modifications reduces our mAP, as follows (all using AlexNet on VOC 2007 with the MEE gradient estimator): (i) minibatch composition: increasing foreground fraction to 25% (as used in Fast-RCNN): -6.1 mAP (ii) minibatch size: halving batch size but doubling iteration count (so the same amount of data is seen): -0.8 mAP (iii) score regularisation: with L2 regularisation instead of L4 and the constant adjusted appropriately: -1.0 mAP. With no regularisation, training fails after < 100 iterations as the magnitude of the classification scores explode. (iv) gradient clipping: with this disabled, training fails after < 100 iterations due to numerical issues caused by large gradients.

Comparison to [19] on VOC 2012. The only previous work that attempts to train a CNN-based object detector directly for AP is [19]. Table 2 compares this method to ours; we use the PASCAL VOC 2012 dataset (testing on the validation subset) as this is what [19] reports results on. Our method achieves comparable performance to [19], with the MEE estimator again being slightly better than SDE.

Unlike our method, [19] trains a separate model for each class; their dynamic-programming solution to the loss-augmented inference problem is for single-class AP only (not mAP over all classes). Moreover, their training procedure does not take into account NMS.

Discussion. We hypothesise that our methods do not significantly outperform Fast R-CNN overall for three reasons. (i) Our gradients are sparser than those of a softmax loss: not every window propagates information back for every class, as changing scores of certain windows has no effect on mAP (*e.g.* low-scored background windows suppressed by NMS). For example, for VOC 2007, around 20% of scores have a non-zero gradient — compared with 100% when using a softmax loss. (ii) mAP is a more rapidly changing function than the softmax loss: an estimate over a minibatch is a much higher-variance estimator of loss over the full set. (iii) It can be shown numerically that mAP over a minibatch of images is a biased estimator of mAP over the population of images from which that minibatch was drawn.

Table 2: Performance of our method compared with [19] (which trains for single-class AP, with a technique very different from ours). All models were trained on VOC 2012 train subset, tested on VOC 2012 validation subset, and use AlexNet. Bounding box regression was not used in any of the models.

Ours, MEE	**Ours, SDE**	**Song et al. [19]**
48.2	48.0	48.5

The real advantage of our method over the standard training procedure of Fast R-CNN is being more principled by respecting the theoretical need for having the same evaluation during training and test.

6 Conclusions

We have presented two definitions of pseudo partial derivatives of piecewise-constant functions. Using these, we have trained a Fast R-CNN detector directly using mAP as the loss, with identical model structure at training and test time, including NMS during training. This ensures that training is truly end-to-end for the final detection task, as opposed to window classification. Our method achieves equivalent performance to Fast R-CNN. It is easily integrated with standard frameworks for SGD, such as Caffe [29], as our NMS and mAP loss layers can be dropped in without affecting the minimisation algorithm or other elements of the model. Our definitions of pseudogradients open up the possibility of training for other piecewise-constant losses. In particular, ranking-based metrics are common in information retrieval, including simple AP on document scores, and discounted cumulative gain [17]. Our method is very general as it does not require definition of an efficient max-oracle, in contrast to [19] and structured SVM methods. Indeed, our approach can also be applied to piecewise-constant internal layers of a network, allowing back-propagation of gradients through such layers.

References

1. Girshick, R.: Fast R-CNN. In: ICCV. (2015)
2. Zhang, Y., Sohn, K., Villegas, R., Pan, G., Lee, H.: Improving object detection with deep convolutional networks via Bayesian optimization and structured prediction. In: CVPR. (2015)
3. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y.: Overfeat: Integrated recognition, localization and detection using convolutional networks. In: ICLR. (2014)
4. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: CVPR. (2014)
5. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: NIPS. (2012)
6. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR. (2015)
7. Alexe, B., Deselaers, T., Ferrari, V.: What is an object? In: CVPR. (2010)

8. Uijlings, J.R.R., van de Sande, K.E.A., Gevers, T., Smeulders, A.W.M.: Selective search for object recognition. IJCV (2013)
9. Zitnick, C.L., Dollár, P.: Edge boxes: Locating object proposals from edges. In: ECCV. (2014)
10. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes (VOC) Challenge. IJCV (2010)
11. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: CVPR. (2015)
12. Vinyals, O., Toshev, A., Bengio, S., Erhan, D.: Show and tell: A neural image caption generator. In: CVPR. (2015)
13. Pfister, T., Charles, J., Zisserman, A.: Flowing convnets for human pose estimation in videos. In: ICCV. (2015)
14. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: NIPS. (2014)
15. Levine, S., Finn, C., Darrell, T., Abbeel, P.: End-to-end training of deep visuomotor policies. JMLR **17** (2016) 1–40
16. Everingham, M., Eslami, S., van Gool, L., Williams, C., Winn, J., Zisserman, A.: The PASCAL visual object classes challenge: A retrospective. IJCV (2015)
17. Järvelin, K., Kekäläinen, J.: IR evaluation methods for retrieving highly relevant documents. In: SIGIR. (2000)
18. Yue, Y., Finley, T., Radlinski, F., Joachims, T.: A support vector method for optimizing average precision. In: SIGIR. (2007)
19. Song, Y., Schwing, A.G., Zemel, R.S., Urtasun, R.: Training deep neural networks via direct loss minimization. In: ICML. (2016) 2169–2177
20. Felzenszwalb, P., Girshick, R., McAllester, D., Ramanan, D.: Object detection with discriminatively trained part based models. IEEE Trans. on PAMI **32** (2010)
21. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A., Fei-Fei, L.: ImageNet large scale visual recognition challenge. IJCV (2015)
22. Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill (1986)
23. Harzallah, H., Jurie, F., Schmid, C.: Combining efficient object localization and image classification. In: ICCV. (2009)
24. Dalal, N., Triggs, B.: Histogram of Oriented Gradients for human detection. In: CVPR. (2005)
25. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: CVPR. (2001) 511–518
26. Blaschko, M.B., Lampert, C.H.: Learning to localize objects with structured output regression. In: ECCV. (2008)
27. Taylor, M., Guiver, J., Robertson, S., Minka, T.: SoftRank: Optimising non-smooth rank metrics. In: WSDM. (2008)
28. Wan, L., Eigen, D., Fergus, R.: End-to-end integration of a convolution network, deformable parts model and non-maximum suppression. In: CVPR. (2015)
29. Jia, Y.: Caffe: An open source convolutional architecture for fast feature embedding. http://caffe.berkeleyvision.org/ (2013)