ABSTRACT

Title of dissertation:	Efficient Detection of Objects and Faces with Deep Learning
	Mahyar Najibi Doctor of Philosophy, 2020
Dissertation directed by:	Professor Larry S. Davis Computer Science Department

Object detection is a fundamental problem in computer vision and is an essential building block for many applications such as autonomous driving, visual search, and object tracking. Given its large-scale and real-time applications, scalable training and fast inference are critical. Deep neural networks, although powerful in visual recognition, can be computationally expensive. Besides, they introduce shortcomings such as lack of scale-invariance and inaccurate predictions in crowded scenes that can affect detection. This dissertation studies the intrinsic problems which emerge when deep convolutional neural networks are used for object and face detection. We introduce methods to overcome these issues which are not only accurate but also efficient.

First, we focus on the problem of lack of scale-invariance. Performing inference on a multi-scale image pyramid, although effective, increases computation noticeably. Moreover, multi-scale inference really blooms when the model is also trained using expensive multi-scale approaches. As a result, we start by introducing an efficient multi-scale training algorithm called "SNIPER" (Scale Normalization for Image Pyramids with Efficient

Re-sampling). Based on the ground-truth annotations, SNIPER sparsely samples high-resolution image regions wherever needed. In contrast to training, at inference, there is no ground-truth information to guide region sampling. Thus, we propose “AutoFocus”. AutoFocus predicts regions to be zoomed-in from low resolutions at inference time, making it possible to skip a large portion of the input pyramid. While being as efficient as single-scale detectors, these methods boost performance noticeably.

Second, we study the problem of efficient face detection. Compared to generic objects, faces are rigid and crowded scenes containing hundreds of faces with extreme scales are more common. In this dissertation, we present “SSH” (Single Stage Headless Face Detector). A method that unlike two-stage localization/classification detectors, performs both tasks in a single stage, efficiently models scale variation by design, and removes most of the parameters from its underlying network, but still achieves state-of-the-art results on challenging benchmarks. Furthermore, for the two-stage detection paradigm, we introduce “FA-RPN” (Floating Anchor Region Proposal Network). FA-RPN takes the spatial structure of faces into account and allows modification of the prediction density during inference to efficiently deal with crowded scenes.

Finally, we turn our attention to the first step in two-stage localization/classification detectors. While neural networks were deployed for classification, localization was previously solved using classic algorithms which became the bottleneck. To remedy, we propose “G-CNN” which models localization as a search in the space of all possible bounding boxes and deploys the same neural network used for classification. Furthermore, for tasks such as saliency detection, where the number of predictions is typically small, we develop an alternative approach that runs at speeds close to 120 frames/second.

Efficient Detection of Objects and Faces with Deep Learning

by

Mahyar Najibi

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2020

Dedication

To my lovely parents, Shideh and Hassan.

Acknowledgments

First and foremost, I am greatly indebted to my advisor, Prof. Larry Davis who has always guided me throughout my research with his invaluable knowledge. He patiently allowed me to follow my interests, to freely fail time to time, and then helped me to learn from my failures. He has been always there to advise me even during the chaotic moments just before the deadlines. I always felt to have his full support and have learned a lot from him.

I had the privilege to work with many bright researchers throughout my Ph.D. I would like to express my strong gratitude to all my collaborators who influenced my works with their diverse backgrounds and knowledge: Ali Shafahi, Mohammad Rastegari, Bharat Singh, Pouya Samangouei, Amin Ghiasi, Zheng Xu, Bahadir Ozdemir, Fan Yang, Peng Zhou, Zhe Wu, Navaneeth Bodla, Robinson Piramithu, and Qiaosong Wang.

I would also like to extend my thanks to my committee members Prof. Rama Chellappa, Prof. Tom Goldstein, Prof. John Dickerson, and Prof. David Jacobs for all the insights they shared with me in our research collaborations and their comments and suggestions, without which this dissertation would not be in its current shape.

Part of our research was supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via IARPA R&D Contract No. 2019-022600002. However, the views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government.

I also greatly thank all my old and new friends for their support. I would like to thank Ali Shafahi, for being a close friend besides a research collaborator. Mohsen Abbasi, Sina Salehi, Hessam Mohammadian, and Mohammad Javad Amiri who I know for a long time. All my friends in Maryland: Kiana Roshan Zamir, Milad Gholami, Saeed Seddighin, Sarvenaz Memarzadeh, Pouya Samangouei, Sina Dehghani, Hadi Yami, Amin Ghiasi, and Saba Ahmadi who helped my quick adaptation to life in Maryland.

Last but most importantly, I dedicate this dissertation to my family. To my lovely parents Shideh and Hassan, without whom this journey was not even possible. My brothers, Mehran and Mehrdad, who have always backed me up unconditionally. I was greatly blessed to have them by my side through all these years and cannot find words adequate enough to truly thank them.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Detection, the problem of localizing and classifying objects in a visual scene, is fundamental in computer vision. Object detection is an essential building block for many applications such as autonomous driving, visual search, and object tracking. Given its large-scale and real-time applications, an object detection system is truly valuable if it is not only accurate but also efficient.

Deep convolutional neural networks (CNNs) are among the most effective approaches for describing visual data. In their simplest forms, these networks process images through several blocks, each of which usually consists of convolution operations, a down-sampling operation, and a non-linearity. Although powerful in visual recognition and successfully applied to the task of image classification [1], they introduce important shortcomings when it comes to object detection.

An important limitation that affects detection significantly is the lack of scale-invariance. To put it in another way, CNNs in their simple form, perform poorly in describing objects that appear in extreme sizes and scales, and deploying them for localization is more challenging than classification. Especially, in face detection scenarios, where crowded scenes with hundreds of small faces are common, care should be taken when designing the localization stage of the detection systems.

The goal of this dissertation is to study the intrinsic shortcomings that emerge when deep convolutional neural networks are deployed for the tasks of object and face detection and to propose solutions that are not only accurate but also efficient. The following subsections give an overview of the topics covered in this dissertation.

1.1 Lack of Scale-Invariance: Efficient Multi-Scale Object Detection

Objects appear in different scales. This is while CNNs have a difficult time describing objects in extreme scales. To make it more clear, consider a classification CNN. Due to the existence of the down-sampling (pooling) operations, the spatial resolution of the feature maps extracted from the image keeps decreasing. After some layers, it is very hard to extract features from objects which are relatively small in the original image. It should be noted that these operations cannot be simply removed. Partly due to memory constraints, and partly because the pooling operations are necessary to aggregate the contextual information for describing visual content.

A possible solution is to apply the CNN to an image pyramid consisting of the resampled versions of the input at different scales. Although effective, such an approach would increase the computation noticeably during inference. Moreover, multi-scale inference blooms when the model is also trained in a similar multi-scale fashion. However, this significantly increases the training time, making training data-extensive CNNs intractable on large-scale datasets.

We introduce SNIPER (Scale Normalization for Image Pyramids with Efficient Re-Sampling). SNIPER is an efficient multi-scale training algorithm with "scale normalization". Not all objects are reliably detectable in all pyramid levels. As a result, enforcing the model to correctly classify all objects in all scales leads to noisy parameter updates. Consequently, SNIPER assigns objects to their correct pyramid level based on their scales. Moreover, SNIPER performs efficient input re-sampling. That is, instead of processing the whole image pyramid, it adaptively re-samples a sparse number of regions, referred to as "chips", from the high-resolution pyramid levels. These low-resolution chips are sampled based on the ground-truth information as well as the estimated hard negative regions. While being in contrast to the current trend in using high-resolution inputs for instance-level recognition tasks, SNIPER achieves state-of-the-art results in object detection and instance segmentation. Besides, by reducing the input resolution, increasing the training batch size, and skipping a large portion of the image pyramid, it makes the computation cost of multi-scale training comparable to single-stage training.

Unlike training, there is no ground-truth information available at inference time to guide chip re-sampling. To speed up multi-scale inference, we propose AutoFocus. AutoFocus starts from processing the lowest scale in the image pyramid and besides detection, it also predicts regions in the image that need to be further zoomed-in for more reliable localization. Only those regions are then processed in higher resolutions. With this strategy, it is possible to extend the idea of low-resolution chip processing from training to inference. AutoFocus can be as efficient as single-scale processing but noticeably improves the performance on challenging benchmarks.

1.2 Faces: Efficient Detection in the Wild

Compared to generic objects, faces are more rigid, and spatial relations between the parts are more persistent. Besides, crowded scenes containing many faces are more common. When it comes to face detection, the detector is expected to perform well, even when there are hundreds of tiny faces in a single scene.

We introduce the "SSH" (Single Stage Headless) face detector. Most state-of-the-art detectors consist of two stages, namely localization (proposal generation), and classification. In contrast, SSH is a single-stage detector performing both localization and classification concurrently. For efficient scale-invariance, SSH places multiple light-weight detection modules on different internal feature maps of a CNN with different resolutions. Each of these modules is specialized to detect faces from a specific range of scales (*i.e.* small, medium, large), allowing SSH to detect faces of various scales in a single network forward pass. As another step towards efficiency, SSH replaces most of the parameters of its underlying classification network with a light-weight detection head. All of which enable SSH to achieve state-of-the-art results while being comparably faster than previous approaches.

Besides, we also study the popular two-stage detection paradigm in which localization is addressed by a proposal generation stage. We propose "FA-RPN" (Floating Anchor Region Proposal Network). A proposal generation algorithm that ties parameters to spatial locations inside the full extent of the objects which is especially beneficial for rigid objects like faces. Moreover, it allows iterative refinement of proposals and modifying the density of proposals at inference time efficiently, without requiring further training.

These features make it an effective approach, especially in crowded scenes.

1.3 Deep Learning and Efficient Localization

Most state-of-the-art object detectors consist of two stages: localization and classification. In the localization phase, hundreds of bounding boxes (proposals) are generated which with a high probability contain objects. In the second stage, these boxes are classified into one of the object categories or background.

Classification networks are naturally applied to the second stage. However, classic proposal generation approaches [2] were previously deployed for the localization stage, making it the bottleneck in object detection systems. To overcome this, we propose "G-CNN" where a single CNN is used for both localization and classification. To do so, we model localization as a search in the space of all possible bounding boxes. Starting from a regular grid, G-CNN iteratively moves and scale boxes towards objects in the scene and classifies them. By removing the external proposal generation algorithm, as well as solving both sub-problems with a unified network, G-CNN performs detection more efficiently compared to its predecessors.

When the number of predictions is expected to be small, it is possible to reduce the computation of the localization stage more. Consider the task of saliency detection where the goal is to localize only a few most visually salient objects in the scene. For this application, we propose a real-time detection algorithm by reformulating the localization problem. Instead of proposing hundreds of candidate boxes and then classifying them, we model localization by directly predicting a saliency map over the image. Once this saliency map is predicted, it is possible to infer the same number of bounding boxes as the existing salient objects present in the image. This strategy significantly speeds up the process, leading to a system that meets real-time requirements.

1.4 Outline

The rest of this dissertation is organized as follows. Chapters 2 and 3 focus on the problem of lack of scale-invariance in CNNs and propose efficient methods for training and performing inference on multi-scale image pyramids. Chapters 4 and 5 discuss the problem of face detection and efficient ways to detect faces from a wide range of scales and in crowded scenes. Chapters 6 and 7 study the problem of efficient localization with deep learning for tasks of generic and salient object detection. Finally, chapter 8 concludes the dissertation.

Chapter 2: SNIPER: Efficient Multi-Scale Training

2.1 Introduction

Humans have a foveal visual system which attends to objects at a fixed distance and size. For example, when we focus on nearby objects, far away objects get blurred [3]. Naturally, it is difficult for us to focus on objects of different scales simultaneously [4]. We only process a small field of view at any given point of time and adaptively ignore the remaining visual content in the image. However, computer algorithms which are designed for instance level visual recognition tasks like object detection depart from this natural way of processing visual information. For obtaining a representation robust to scale, popular detection algorithms like Faster-RCNN/Mask-RCNN [5, 6] are trained on a multi-scale image pyramid [7, 8]. Since every pixel is processed at each scale, this approach to processing visual information increases the training time significantly. For example, constructing a 3 scale image pyramid (*e.g.* scales=1x,2x,3x) requires processing 14 times the number of pixels present in the original image. For this reason, it is impractical to use multi-scale training in many scenarios.

Recently, it is shown that ignoring gradients of objects which are of extreme resolutions is beneficial while using multiple scales during training [8]. For example, when constructing an image pyramid of 3 scales, the gradients of large and small objects should be ignored at large and small resolutions respectively. If this is the case, an intuitive question which arises is, do we need to process the entire image at a 3x resolution? Wouldn't it suffice to sample a much smaller RoI (chip) around small objects at this resolution? On the other hand, if the image is already high resolution, and objects in it are also large in size, is there any benefit in upsampling that image?

While ignoring significant portions of the image would save computation, a smaller chip would also lack context required for recognition. A significant portion of background would also be ignored at a higher resolution. So, there is a trade-off between computation, context and negative mining while accelerating multi-scale training. To this end, we present a novel training algorithm called *Scale Normalization for Image Pyramids with Efficient Resampling (SNIPER)*, which adaptively samples chips from multiple scales of an image pyramid, conditioned on the image content. We sample positive chips conditioned on the ground-truth instances and negative chips based on proposals generated by a region proposal network. Under the same conditions (fixed batch normalization), we show that SNIPER performs as well as the multi-scale strategy proposed in SNIP [8] while reducing the number of pixels processed by a factor of 3 during training on the COCO dataset. Since SNIPER is trained on 512x512 size chips, it can reap the benefits of a large batch size and training with batch-normalization on a single GPU node. In particular, we can use a batch size of 20 per GPU (leading to a total batch size of 160), even with a ResNet-101 based Faster-RCNN detector. While being efficient, SNIPER obtains competitive performance on the COCO detection dataset even with simple detection architectures like Faster-RCNN.

2.2 Background

Deep learning based object detection algorithms have primarily evolved from the R-CNN detector [9], which started with object proposals generated with an unsupervised algorithm [2], resized these proposals to a canonical 224x224 size image and classified them using a convolutional neural network [10]. This model is scale invariant, but the computational cost for training and inference for R-CNN scales linearly with the number of proposals. To alleviate this computational bottleneck, Fast-RCNN [11] proposed to project region proposals to a high level convolutional feature map and use the pooled features as a semantic representation for region proposals. In this process, the computation is shared for the convolutional layers and only lightweight fully connected layers are applied on each proposal. However, convolution for objects of different sizes is performed at a single scale, which destroys the scale invariance properties of R-CNN. Hence, inference at multiple scales is performed and detections from multiple scales are combined by selecting features from a pair of adjacent scales closer to the resolution of the pre-trained network [12, 11]. The Fast-RCNN model has since become the *de-facto* approach for classifying region proposals as it is fast and also captures more context in its features, which is lacking in RCNN.

It is worth noting that in multi-scale training, Fast-RCNN upsamples *and* downsamples every proposal (whether small or big) in the image. This is unlike R-CNN, where each proposal is resized to a canonical size of 224x224 pixels. Large objects are not upsampled and small objects are not downsampled in R-CNN. In this regard, R-CNN more appropriately does not up/downsample every pixel in the image but only in those regions which are likely to contain objects to an appropriate resolution. However, R-CNN does not share the convolutional features for nearby proposals like Fast-RCNN, which makes it slow. To this end, we propose SNIPER, which retains the benefits of both these approaches by generating scale specific context-regions (chips) that cover maximum proposals at a particular scale. SNIPER classifies all the proposals inside a chip like Fast-RCNN which enables us to perform efficient classification of multiple proposals within a chip. As SNIPER does not upsample the image where there are large objects and also does not process easy background regions, it is significantly faster compared to a Fast-RCNN detector trained on an image pyramid.

SNIP [8] is also trained on almost all the pixels of the image pyramid (like Fast-RCNN), although gradients arising from objects of extreme resolutions are ignored. In particular, 2 resolutions of the image pyramid (480 and 800 pixels) always engage in training and multiple 1000 pixel crops are sampled out of the 1400 pixel resolution of the image in the finest scale. SNIPER takes this cropping procedure to an extreme level by sampling 512 pixels crops from 3 scales of an image pyramid. At extreme scales (like 3x), SNIPER observes less than one tenth of the original content present in the image! Unfortunately, as SNIPER chips generated only using ground-truth instances are very small compared to the resolution of the original image, a significant portion of the background does not participate in training. This causes the false positive rate to increase. Therefore, it is important to generate chips for background regions as well. In SNIPER, this is achieved by randomly sampling a fixed number of chips (maximum of 2 in the experiments) from regions which are likely to cover false positives. To find such regions, we train a lightweight RPN network with a short schedule. The proposals of this network are used to generate chips for regions which are likely to contain false positives (this could potentially be replaced with unsupervised proposals like EdgeBoxes [13] as well). After adding negative chip sampling, the performance of SNIPER matches SNIP, but it is 3 times faster! Since we are able to obtain similar performance by observing less than one tenth of the image, it implies that very large context during training is *not* important for training high-performance detectors but sampling regions containing hard negatives is.

2.3 SNIPER

We describe the major components of SNIPER in this section. One is positive/negative chip mining and the other is label assignment after chips are generated. Finally, we will discuss the benefits of training with SNIPER.

2.3.1 Chip Generation

SNIPER generates chips $\mathcal{C}^i$ at multiple scales $\{s_1, s_2, .., s_i, ..s_n\}$ in the image. For each scale, the image is first re-sized to width (W_i) and height (H_i). On this canvas, $K \times K$ pixel chips are placed at equal intervals of d pixels (we set d to 32). This leads to a two-dimensional array of chips at each scale.

2.3.2 Positive Chip Selection

For each scale, there is a desired area range $\mathcal{R}^i = [r^i_{min}, r^i_{max}]$, $i \in [1, n]$ which determines which ground-truth boxes/proposals participate in training for each scale i. The valid list of ground-truth bounding boxes which lie in $\mathcal{R}^i$ are referred to as $\mathcal{G}^i$. Then, chips are greedily selected so that maximum number of valid ground-truth boxes ($\mathcal{G}^i$) are covered. A ground-truth box is said to be covered if it is completely enclosed inside a chip. All the positive chips from a scale are combined per image and are referred to as $\mathcal{C}^i_{pos}$. For each ground-truth bounding box, there always exists a chip which covers it. Since consecutive $\mathcal{R}^i$ contain overlapping intervals, a ground-truth bounding box may be assigned to multiple chips at different scales. It is also possible that the same ground-truth bounding box may be in multiple chips from the same scale. Ground-truth instances which have a partial overlap (IoU ¿ 0) with a chip are cropped. All the cropped ground-truth boxes (valid or invalid) are retained in the chip and are used in label assignment.

In this way, every ground-truth box is covered at the appropriate scale. Since the crop-size is much smaller than the resolution of the image (*i.e.* more than 10x smaller for high-resolution images), SNIPER does not process most of the background at high-resolutions. This leads to significant savings in computation and memory requirement while processing high-resolution images. We illustrate this with an example shown in Figure 2.1. The left side of the figure shows the image with the ground-truth boxes represented by green bounding boxes. Other colored rectangles on the left side of the figure show the chips generated by SNIPER in the original image resolution which cover all objects. These chips are illustrated on the right side of the figure with the same border color. Green and red bounding boxes represent the valid and invalid ground-truth objects corresponding to the scale of the chip. As can be seen, in this example, SNIPER efficiently processes all ground-truth objects in an appropriate scale by forming 4 low-resolution chips.

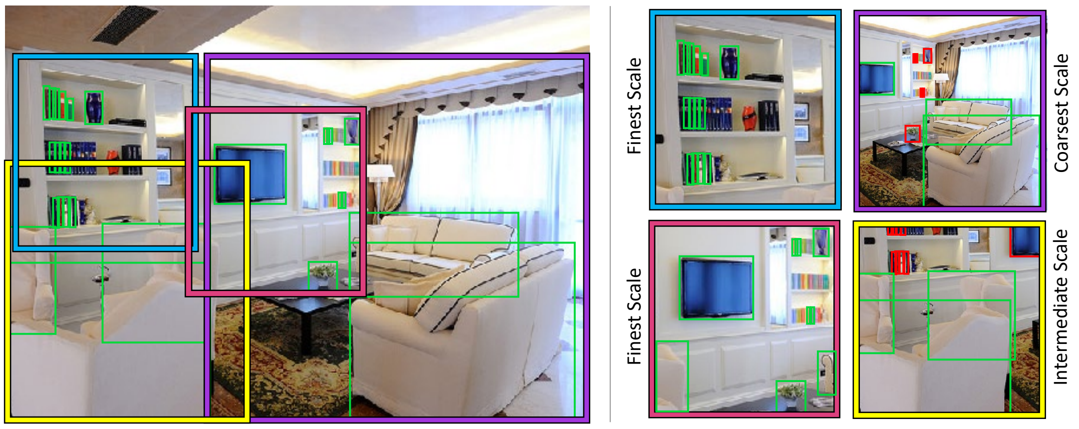


Figure 2.1: SNIPER Positive chip selection. SNIPER adaptively samples context regions (aka chips) based on the presence of objects inside the image. Left side: The image, ground-truth boxes (represented by green lines), and the chips in the original image scale (represented by the blue, yellow, pink, and purple rectangles). Right side: Down/up-sampling is performed considering the size of the objects. Covered objects are shown in green and invalid objects in the corresponding scale are shown as red rectangles.

2.3.3 Negative Chip Selection

Although positive chips cover all the positive instances, a significant portion of the background is not covered by them. Incorrectly classifying background increases the false positive rate. In current object detection algorithms, when multi-scale training is performed, every pixel in the image is processed at all scales. Although training on all scales reduces the false positive rate, it also increases computation. We posit that a significant amount of the background is easy to classify and hence, we can avoid performing any computation in those regions. So, how do we eliminate regions which are easy to classify? A simple approach is to employ object proposals to identify regions where objects are likely to be present. After all, our classifier operates on region proposals and if there are no region proposals in a part of the image, it implies that it is very easy to classify as background. Hence, we can ignore those parts of the image during training.

To this end, for negative chip mining, we first train RPN for a couple of epochs. No negative chips are used for training this network. The task of this network is to roughly guide us in selecting regions which are likely to contain false positives, so it is not necessary for it to be very accurate. This RPN is used to generate proposals over the entire training set. We assume that if no proposals are generated in a major portion of the image by RPN, then it is unlikely to contain an object instance. For negative chip selection, for each scale i, we first remove all the proposals which have been covered in $\mathcal{C}^i_{pos}$. Then, for each scale i, we greedily select all the chips which cover at least M proposals in $\mathcal{R}^i$. This generates a set of negative chips for each scale per image, $\mathcal{C}^i_{neg}$. During training, we randomly sample a fixed number of negative chips per epoch (per image) from this pool of negative chips which are generated from all scales, i.e. $\bigcup_{i=1}^{n} \mathcal{C}^i_{neg}$. Figure 2.2 shows examples of the generated negative chips by SNIPER. The first row shows the image and the ground-truth boxes. In the bottom row, we show the proposals not covered by $\mathcal{C}^i_{pos}$ and the corresponding negative chips generated (the orange boxes). However, for clarity, we represent each proposal by a red circle in its center. As illustrated, SNIPER only processes regions which likely contain false positives, leading to faster processing time.

2.3.4 Label Assignment

Our network is trained end to end on these chips like Faster-RCNN, i.e. it learns to generate proposals as well as classify them with a single network. While training, proposals generated by RPN are assigned labels and bounding box targets (for regression) based on *all* the ground-truth boxes which are present inside the chip. We do not filter

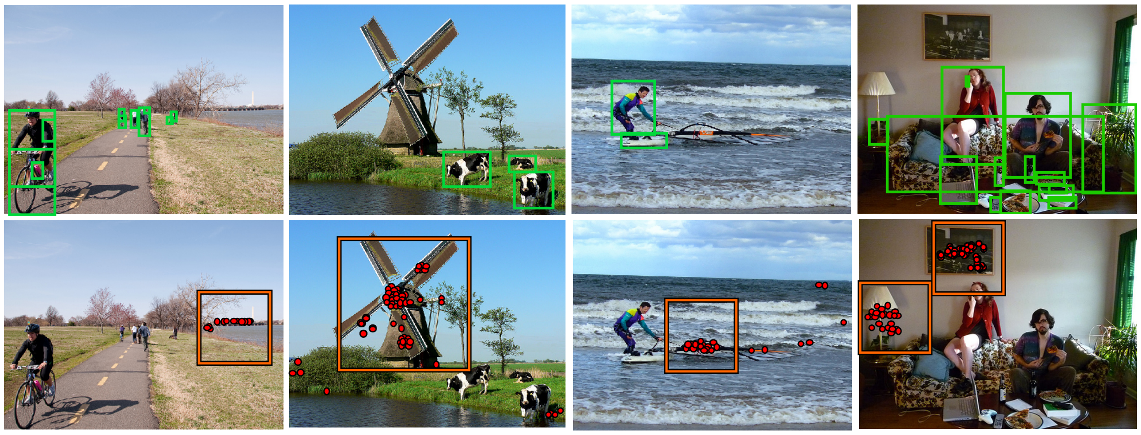


Figure 2.2: SNIPER negative chip selection. First row: the image and the ground-truth boxes. Bottom row: negative proposals not covered in positive chips (represented by red circles located at the center of each proposal for the clarity) and the generated negative chips based on the proposals (represented by orange rectangles).

ground-truth boxes based on $\mathcal{R}^i$. Instead, proposals which do not fall in $\mathcal{R}^i$ are ignored during training. So, a large ground-truth box which is cropped, could generate a valid proposal which is small. Like Fast-RCNN, we mark any proposal which has an overlap greater than 0.5 with a ground-truth box as positive and assign bounding-box targets for the proposal. Our network is trained end to end and we generate 300 proposals per chip. We do not apply any constraint that a fraction of these proposals should be re-sampled as positives [5], as in Fast-RCNN. We did not use OHEM [14] for classification and use a simple softmax cross-entropy loss for classification. For assigning RPN labels, we use valid ground-truth boxes to assign labels and invalid ground-truth boxes to invalidate anchors, as done in SNIP [8].

2.3.5 Benefits

For training, we randomly sample chips from the whole dataset for generating a batch. On average, we generate $\sim$ 5 chips of size 512x512 per image on the COCO dataset (including negative chips) when training on three scales (512/ms [1], 1.667, 3). This is only 30% more than the number of pixels processed per image when single scale training is performed with an image resolution of 800x1333. Since all our images are of the same size, data is much better packed leading to better GPU utilization which easily overcomes the extra 30% overhead. But more importantly, we reap the benefits of multi-scale training on 3 scales, large batch size and training with batch-normalization without any slowdown in performance on a single 8 GPU node.

It is commonly believed that high resolution images (e.g. 800x1333) are necessary for instance level recognition tasks. Therefore, for instance level recognition tasks, it was not possible to train with batch-normalization statistics computed on a single GPU. Methods like synchronized batch-normalization [7, 15] or training on 128 GPUs [16] have been proposed to alleviate this problem. Synchronized batch-normalization slows down training significantly and training on 128 GPUs is also impractical for most people. Therefore, group normalization [17] has been recently proposed so that instance level recognition tasks can benefit from another form of normalization in a low batch setting during training. With SNIPER, we show that the image resolution bottleneck can be alleviated for instance level recognition tasks. As long as we can cover negatives and use appropriate scale normalization methods, we can train with a large batch size of resampled low resolution chips, even on challenging datasets like COCO. Our results suggest that context beyond a certain field of view may not be beneficial during training. It is also possible that the effective receptive field of deep neural networks is not large enough to leverage far away pixels in the image, as suggested in [18].

[1] max($width_{im}$,$height_{im}$)

In very large datasets like *OpenImagesV4* [19] containing 1.7 million images, most objects are large and images provided are high resolution (1024x768), so it is less important to upsample images by $3\times$. In this case, with SNIPER, we generate 3.5 million chips of size 512x512 using scales of (512/ms, 1). Note that SNIPER also performs adaptive downsampling. Since the scales are smaller, chips would cover more background, due to which the impact of negative sampling is diminished. In this case (of positive chip selection), SNIPER processes only half the number of pixels compared to naïve multi-scale training on the above mentioned scales in OpenImagesV4. Due to this, we were able to train Faster-RCNN with a ResNet-101 backbone on 1.7 million images in just 3 days on a single 8 GPU node!

2.4 Experimental Details

We evaluate SNIPER on the COCO dataset for object detection. COCO contains 123,000 images in the training and validation set and 20,288 images in the test-dev set. We train on the combined training and validation set and report results on the test-dev set. Since recall for proposals is not provided by the evaluation server, we train on 118,000 images and report recall on the remaining 5,000 images (commonly referred to as the *minival* set).

On COCO, we train SNIPER with a batch-size of 128 and with a learning rate of 0.015. We use a chip size of 512×512 pixels. Training scales are set to (512/ms, 1.667, 3) where ms is the maximum value width and height of the image[2]. The desired area ranges (*i.e.* $\mathcal{R}^i$) are set to (0,80^2), (32^2, 150^2), and (120^2, inf) for each of the scales respectively.

[2]For the first scale, zero-padding is used if the smaller side of the image becomes less than 512 pixels.

Training is performed for a total of 6 epochs with step-down at the end of epoch 5. Image flipping is used as a data-augmentation technique. Every epoch requires 11,000 iterations. For training RPN without negatives, each epoch requires 7000 iterations. We use RPN for generating negative chips and train it for 2 epochs with a fixed learning rate of 0.015 without any step-down. Therefore, training RPN for 2 epochs requires less than 20% of the total training time. RPN proposals are extracted from all scales. Note that inference takes $1/3$ the time for a full forward-backward pass and we do not perform any flipping for extracting proposals. Hence, this process is also efficient. We use mixed precision training as described in [20]. To this end, we re-scale weight-decay by 100, drop the learning rate by 100 and rescale gradients by 100. This ensures that we can train with activations of half precision (and hence $\sim$ 2x larger batch size) without any loss in accuracy. We use fp32 weights for the first convolution layer, last convolution layer in RPN (classification and regression) and the fully connected layers in Faster-RCNN.

We evaluate SNIPER using a popular detector, Faster-RCNN with ResNets [21, 22] and MobileNetV2 [23]. Proposals are generated using RPN on top of conv4 features and classification is performed after concatenating conv4 and conv5 features. In the conv5 branch, we use deformable convolutions and a stride of 1. We use a 512 dimensional feature map in RPN. For the classification branch, we first project the concatenated feature map to 256 dimensions and then add 2 fully connected layers with 1024 hidden units. For lightweight networks like MobileNetv2 [23], to preserve the computational processing power of the network, we did not make any architectural changes to the network like changing the stride of the network or added deformable convolutions. We reduced the RPN dimension to 256 and size of fc layers to 512 from 1024. RPN and classification

Method	AR	AR^{50}	AR^{75}	0-25	25-50	50-100	100-200	200-300
ResNet-101 With Neg	65.4	93.2	76.9	41.3	65.8	74.5	76.9	78.7
ResNet-101 W/o Neg	65.4	93.2	77.6	40.8	65.7	74.7	77.4	79.3

Table 2.1: We plot the recall for SNIPER with and without negatives. Surprisingly, recall is not effected by negative chip sampling

branch are both applied on the layer with stride 32 for MobileNetv2.

SNIPER generates 1.2 million chips for the COCO dataset after the images are flipped. This results in around 5 chips per image. In some images which contain many object instances, SNIPER can generate as many as 10 chips and others where there is a single large salient object, it would only generate a single chip. In a sense, it reduces the imbalance in gradients propagated to an instance level which is present in detectors which are trained on full resolution images. At least in theory, training on full resolution images is biased towards large object instances.

2.4.1 Recall Analysis

We observe that recall (averaged over multiple overlap thresholds 0.5:0.05:0.95) for RPN does not decrease if we do not perform negative sampling. This is because recall does not account for false positives. As shown in Section 2.4.2, this is in contrast to mAP for detection in which negative sampling plays an important role. Moreover, in positive chip sampling, we do cover every ground truth sample. Therefore, for generating proposals, it is sufficient to train on just positive samples. This result further bolsters SNIPER's strategy of finding negatives based on an RPN in which the training is performed just on positive samples.

2.4.2 Negative Chip Mining and Scale

SNIPER uses negative chip mining to reduce the false positive rate while speeding up the training by skipping the *easy* regions inside the image. As proposed in Section 2.3.3, we use a region proposal network trained with a short learning schedule to find such regions. To evaluate the effectiveness of our negative mining approach, we compare SNIPER's mean average precision with a slight variant which only uses positive chips during training (denoted as *SNIPER w/o neg*). All other parameters remain the same. Table 2.2 compares the performance of these models. The proposed negative chip mining approach noticeably improves AP for all localization thresholds and object sizes. Noticeably, negative chip mining improves the average precision from 43.4 to 46.1. This is in contrast to the last section where we were evaluating proposals. This is because mAP is affected by false positives. If we do not include regions in the image containing negatives which are similar in appearance to positive instances, it would increase our false positive rate and adversely affect detection performance.

SNIPER is an efficient multi-scale training algorithm. In all experiments, we use the aforementioned three scales (See Section 2.4 for the details). To show that SNIPER effectively benefits from multi-scale training, we reduce the number of scales from 3 to 2 by dropping the high resolution scale. Table 2.2 shows the mean average precision for SNIPER under these two settings. As can be seen, by reducing the number of scales, the performance consistently drops by a large margin on all evaluation metrics.

Method	Backbone	AP	AP^{50}	AP^{75}	AP^{S}	AP^{M}	AP^{L}
SNIPER	ResNet-101	46.1	67.0	51.6	29.6	48.9	58.1
SNIPER 2 scale	ResNet-101	43.3	63.7	48.6	27.1	44.7	56.1
SNIPER w/o negatives	ResNet-101	43.4	62.8	48.8	27.4	45.2	56.2

Table 2.2: The effect training on 2 scales (1.667 and max size of 512). We also show the impact in performance when no negative mining is performed.

2.4.3 Timing

It takes 14 hours to train SNIPER end to end on a 8 GPU V100 node with a Faster-RCNN detector which has a ResNet-101 backbone. It is worth noting that we train on 3 scales of an image pyramid (max size of 512, 1.667 and 3). Training RPN is much more efficient and it only takes 2 hours for pre-training. Not only is SNIPER efficient in training, it can also process around 5 images per second on a single V100 GPU. For better utilization of resources, we run multiple processes in parallel during inference and compute the average time it takes to process a batch of 100 images.

2.4.4 Inference

We perform inference on an image pyramid and scale the original image to the following resolutions (480, 512), (800, 1280) and (1400, 2000). The first element is the minimum size with the condition that the maximum size does not exceed the second element. The valid ranges for training and inference are similar to SNIP [8]. For combining the detections, we use Soft-NMS [24]. We do not perform flipping [25], iterative bounding box regression [26] or mask tightening [7].

2.4.5 Comparison with State-of-the-art

It is difficult to fairly compare different detectors as they differ in backbone architectures (like ResNet [21], ResNext [27], Xception [28]), pre-training data (*e.g.* ImageNet-5k, JFT [29], OpenImages [19]), different structures in the underlying network (*e.g* multi-scale features [30, 31], deformable convolutions [32], heavier heads [16], anchor sizes, path aggregation [7]), test time augmentations like flipping, mask tightening, iterative bounding box regression etc.

Therefore, we compare our results with SNIP [8], which is a recent method for training object detectors on an image pyramid. The results are presented in Table 2.3. Without using batch normalization [33], SNIPER achieves comparable results. While SNIP [8] processes almost all the image pyramid, SNIPER on the other hand, reduces the computational cost by skipping easy regions. Moreover, since SNIPER operates on a lower resolution input, it reduces the memory footprint. This allows us to increase the batch size and unlike SNIP [8], we can benefit from batch normalization during training. With batch normalization, SNIPER significantly outperforms SNIP in all metrics. It should be noted that not only the proposed method is more accurate, it is also $3\times$ faster during training. To the best of our knowledge, for a Faster-RCNN architecture with a ResNet-101 backbone (with deformable convolutions), our reported result of 46.1% is state-of-the-art. This result improves to 46.8% if we pre-train the detector on the OpenImagesV4 dataset. Adding an instance segmentation head and training the detection network along with it improves the performance to 47.6%.

With our efficient batch inference pipeline, we can process 5 images per second on

Method	Backbone	AP	AP^{50}	AP^{75}	AP^{S}	AP^{M}	AP^{L}
SSD	MobileNet-v2	22.1	-	-	-	-	-
SNIP	ResNet-50 (fixed BN)	43.6	65.2	48.8	26.4	46.5	55.8
	ResNet-101 (fixed BN)	44.4	66.2	49.9	27.3	47.4	56.9
SNIPER	MobileNet-V2	34.1	54.4	37.7	18.2	36.9	46.2
	ResNet-50 (fixed BN)	43.5	65.0	48.6	26.1	46.3	56.0
	ResNet-101	46.1	67.0	51.6	29.6	48.9	58.1
	ResNet-101 + OpenImages	46.8	67.4	52.5	30.5	49.4	59.6
	ResNet-101 + OpenImages + Seg Binary	47.1	67.8	52.8	30.2	49.9	60.2
	ResNet-101 + OpenImages + Seg Softmax	47.6	68.5	53.4	30.9	50.6	60.7
SNIPER	ResNet-101 + OpenImages + Seg Softmax	38.9	62.9	41.8	19.6	41.2	55.0
SNIPER	ResNet-101 + OpenImages + Seg Binary	41.3	65.4	44.9	21.4	43.5	58.7

Table 2.3: Ablation analysis and comparison with full resolution training. Last two rows show instance segmentation results when the mask head is trained with N+1 way softmax loss and binary softmax loss for N classes.

a single V100 GPU and still obtain an mAP of 47.6%. This implies that on modern GPUs, it is practical to perform inference on an image pyramid which includes high resolutions like 1400x2000. We also show results for Faster-RCNN trained with MobileNetV2. It obtains an mAP of 34.1% compared to the SSDLite [23] version which obtained 22.1%. This again highlights the importance of image pyramids (and SNIPER training) as we can improve the performance of the detector by 12%.

We also show results for instance segmentation. The network architecture is same as Mask-RCNN [6], just that we do not use FPN [30] and use the same detection architecture which was described for object detection. For multi-tasking, we tried two variants of loss functions for training the mask branch. One was a foreground-background softmax function for N classes and another was a N+1 way softmax function. For instance segmentation, the network which is trained with 2-way Softmax loss for each class clearly performs better. But, for object detection, the N+1 way Softmax loss leads to slightly better results. We only use 3 scales during inference and do not perform flipping, mask tightening, iterative bounding-box regression, padding masks before resizing etc. Our instance segmentation results are preliminary and we have only trained 2 models so far.

2.5 Related Work

SNIPER benefits from multiple techniques which were developed over the last year. Notably, it was shown that it is important to train with batch normalization statistics [16, 7, 15] for tasks like object detection and semantic segmentation. This is one important reason for SNIPER's better performance. SNIPER also benefits from a large batch size which was shown to be effective for object detection [16]. Like SNIP [8], SNIPER ignores gradients of objects at extreme scales in the image pyramid to improve multi-scale training.

In the past, many different methods have been proposed to understand the role of context [34, 35, 36], scale [37, 38, 30, 31] and sampling [39, 14, 40, 41]. Considerable importance has been given to leveraging features of different layers of the network and designing architectures for explicitly encoding context/multi-scale information [31, 42, 25, 43] for classification. Our results highlight that context may not be very important for training high performance object detectors.

2.6 Conclusion

We presented an algorithm for efficient multi-scale training which sampled low resolution chips from a multi-scale image pyramid to accelerate multi-scale training by a factor of 3 times. In doing so, SNIPER did not compromise on the performance of the detector due to effective sampling techniques for positive and negative chips. As SNIPER operates on re-sampled low resolution chips, it can be trained with a large batch size on a single GPU which brings it closer to the protocol for training image classification. This is in contrast with the common practice of training on high resolution images for instance-level recognition tasks.

Chapter 3: AutoFocus: Efficient Multi-Scale Inference

3.1 Introduction

Human vision is foveal and active [44, 45]. The fovea, which observes the world at high-resolution, only corresponds to 5 degrees of the total visual field [46]. Our lower resolution peripheral vision has a field of view of 110 degrees [47]. To find objects, our eyes perform saccadic movements which rely on peripheral vision [48]. When moving between different fixation points, the region in between is simply ignored, a phenomenon known as saccadic masking [49, 50, 51]. Hence, finding objects is an active process and the search time depends on the complexity of the scene. For example, locating a face in a portrait photograph would take much less time than finding every face in a crowded market.

Adaptive processing, which is quite natural, brings several benefits. Many applications do not have real-time requirements and detectors are applied offline on billions of images/videos. Therefore, computational savings in a batch mode provide substantial monetary benefits. Examples include large-scale indexing of images and videos for visual search, APIs provided by cloud services, smart retail stores *etc*. While there is work on image classification which performs conditional computation [52, 53, 54], modern object detection algorithms perform static inference and process every pixel of a multi-scale

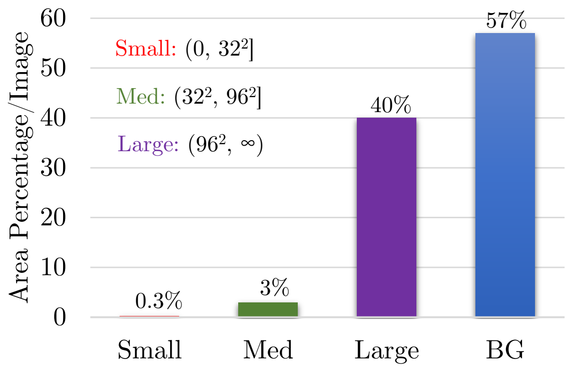


Figure 3.1: Area of objects of different sizes and the background in the COCO validation set. Objects are divided based on their area (in pixels) into small, medium, and large.

image pyramid to detect objects of different sizes [8, 55, 16]. This is a very inefficient process as the algorithm spends equal energy at every pixel at different scales.

To provide some perspective, we show the percentage of pixels occupied per image for different size objects in the COCO dataset in Fig 3.1. Even though 40% of the object instances are small, they only occupy 0.3% of the area. If the image pyramid includes a scale of 3, then just to detect such a small fraction of the dataset, we end up performing 9 times more computation at finer-scales. If we add some padding around small objects to provide spatial context and only upsample these regions, their area would still be small compared to the resolution of the original image. So, when performing multi-scale inference, can we predict regions containing small objects from coarser scales?

If deep convolutional neural networks are an approximation of biological vision, it should be possible to localize object-like regions at lower resolution and recognize them by zooming on them at higher resolution - similar to the way our peripheral vision is coupled with foveal vision. To this end, we propose an object detection framework called *AutoFocus*, which adopts a coarse to fine approach and learns where to look in the next (larger) scale in the image pyramid. Thus, it saves computation while processing finer scales. This is achieved by predicting category agnostic binary segmentation maps for small objects, which we refer to as *FocusPixels*. A simple algorithm which operates on FocusPixels is designed to generate chips for the next image scale. AutoFocus only processes 20% of the image at the largest scale in the pyramid on the COCO dataset, without any drop in performance. This can be improved to as little as 5% with a 1% drop in performance.

3.2 Related Work

Image pyramids [56] and convolutional neural networks [10] are fundamental building blocks in the computer vision pipeline. Unfortunately, convolutional neural networks are not scale invariant. Therefore, for instance-level visual recognition problems, to recognize objects of different sizes, it is beneficial to rely on image pyramids [8]. While efficient training solutions have been proposed for multi-scale training [55], inference on image pyramids remains a computational bottleneck which prohibits their use in practice. Recently, a few methods have been proposed to accelerate multi-scale inference, but they have only been evaluated under constrained settings like pedestrian/face detection or object detection in videos [57, 58, 59, 60, 61, 62]. In this work, we propose a simple and pragmatic framework to accelerate multi-scale inference for generic object detection which is evaluated on benchmark datasets.

Accelerating object detection has a long history in computer vision. The Viola-Jones detector [63] is a classic example. It rejects easy regions with simple filters and spends more energy on promising object-like regions to accelerate the process. Several methods since then have been proposed to improve it [64, 65, 66]. Prior to deep-learning based object detectors, it was common to employ a multi-scale approach for object detection [67, 68, 69, 70, 71, 72] and several effective solutions were proposed to accelerate detection on image pyramids. Common techniques involved approximation of features to reduce the number of scales [71, 72], cascades [64, 73] or feature pyramids [74]. Recently, feature-pyramids have been extensively studied and employed in deep learning based object detectors as the representation provides a boost in accuracy without compromising speed [75, 42, 38, 37, 30, 76, 31, 6, 39, 7]. Although the use of image pyramids is common in challenge winning entries which primarily focus on performance [21, 32, 16, 7], efficient detectors which operate on a single low-resolution image (*e.g.* YOLO [77], SSD [42], RetinaNet [39]) are commonly deployed in practice. This is because multi-scale inference on pyramids of high-resolution images is prohibitively expensive.

AutoFocus alleviates this problem to a large extent and is designed to provide a smooth trade-off between speed and accuracy. It shows that it is possible to predict the presence of a small object at a coarser scale (referred to as FocusPixels) which enables avoiding computation in large regions of the image at finer scales. These are different from object proposals [78, 2, 5] where region candidates need to have a tight overlap with objects. Learning to predict FocusPixels is an easier task and does not require instance-level reasoning. AutoFocus shares the motivation with saliency and reinforcement learning based methods which perform a guided search while processing images [79, 80, 81, 82, 83, 84, 85, 86, 87], but it is designed to predict small objects in coarser scales and they need not be salient.

3.3 Background

We provide a brief overview of SNIP, which is the multi-scale training and inference method described in [8]. The core idea is to restrict the training samples to be in a pre-defined scale range which is appropriate for the input scale. For example, the detector is only trained on small objects at high resolution (larger scale) and large objects at low resolution (smaller scale). Because it is not trained on large objects at high resolution images, it is unlikely to detect them during inference as well. Rules are also defined to ignore large detections in high-resolution images during inference and vice-versa. Therefore, while merging detections from multiple scales, SNIP simply ignores large detections in high resolution images which contain most of the pixels.

Since the size of objects is known during training, it is possible to ignore large regions of the image pyramid by only processing appropriate context regions around objects. SNIPER [55] showed that training on such low resolution *chips* with appropriate scaling does not lead to any drop in performance when compared to training on full-resolution images. If we can automatically predict these chips for small objects at a coarser scale, we may not need to process the entire high-resolution image during inference as well. But when these chips are generated during training, many object instances get cropped and their size changes. This did not hurt performance during training and can also be regarded as a data augmentation strategy. Unfortunately, if chips are generated during inference and an object is cropped into multiple parts, it would increase the error rate. So, apart from predicting where to look at the next scale, we also need to design an algorithm which correctly merges detections from chips at multiple scales.

3.4 The AutoFocus Framework

Classic features like SIFT [88] / SURF [89], combine two major components - the detector and the descriptor. The detector typically involved lightweight operators like Difference of Gaussians (DoG) [90], Harris Affine [91], Laplacian of Gaussians (LoG) [92] *etc*. The detector was applied on the entire image to find *interesting* regions. Therefore, the descriptor, which was computationally expensive, only needed to be computed for these interesting regions. This cascaded model of processing the image made the entire pipeline efficient.

Likewise, the AutoFocus framework is designed to predict interesting regions in the image and discards regions which are unlikely to contain objects at the next scale. It zooms and crops only such interesting regions when applying the detector at successive scales. AutoFocus is comprised of three main components: the first learns to predict *FocusPixels*, the second generates *FocusChips* for efficient inference and the third merges detections from multiple scales, which we refer to as *focus stacking* for object detection.

3.4.1 FocusPixels

FocusPixels are defined at the granularity of the convolutional feature map (like conv5). A pixel in the feature map is labelled as a FocusPixel if it has any overlap with a small object. An object is considered to be small if it falls in an area range (between 5 $\times$ 5 and 64 $\times$ 64 pixels in our implementation) in the resized chip (Sec. 3.4.2) which is input to the network . To train our network, we mark FocusPixels as positives. We also define some pixels in the feature map as invalid. Those pixels which overlap objects that have an area smaller or slightly larger than those defined as small are considered invalid (smaller than 5×5 or between 64×64 and 90×90). All other pixels are considered as negatives. AutoFocus is trained to generate high activations on regions which contain FocusPixels.

Formally, given an image of size $X \times Y$, and a fully convolutional neural network whose stride is s, then the labels L will be of size $X' \times Y'$, where $X' = \lceil \frac{X}{s} \rceil$ and $Y' = \lceil \frac{Y}{s} \rceil$. Since the stride is s, each label $l \in L$ corresponds to $s \times s$ pixels in the image. The label l is defined as follows,

$$l = \begin{cases} 1, & IoU(GT, l) > 0, a < \sqrt{GTArea} < b \\ -1, & IoU(GT, l) > 0, \sqrt{GTArea} < a \\ -1, & IoU(GT, l) > 0, b < \sqrt{GTArea} < c \\ 0, & \text{otherwise} \end{cases}$$

where IoU is intersection over union of the $s \times s$ label block with the ground truth bounding box. $GTArea$ is the area of the ground truth bounding box after scaling. a is typically 5, b is 64 and c is 90. If multiple ground-truth bounding boxes overlap with a pixel, FocusPixels ($l = 1$) are given precedence. Since our network is trained on 512×512 pixel chips, the ratio between positive and negative pixels is around 10, so we do not perform any re-weighting for the loss. Note that during multi-scale training, the same ground-truth could generate a label of 1, 0 or -1 depending on how much it has been scaled. The reason we regard pixels for medium objects as invalid ($l = -1$) is that the transition from small to large objects is not visually obvious. Extremely small objects in each scale are also

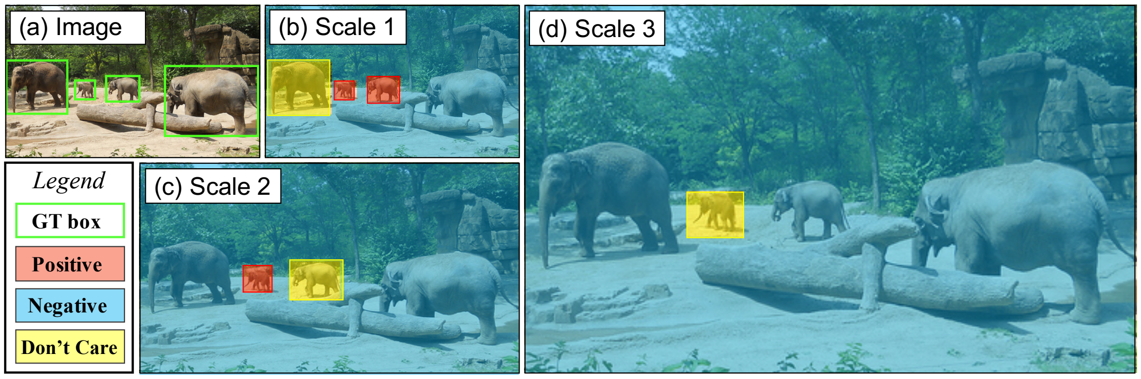


Figure 3.2: The figure illustrates how FocusPixels are assigned at multiple scales of an image. At scale 1 (b), the smallest two elephants generate FocusPixels, the largest one is marked as background and the one on the left is ignored during training to avoid penalizing the network for borderline cases (see Sec. 3.4.1 for assignment details). The labelling changes at scales 2 and 3 as the objects occupy more pixels. For example, only the smallest elephant would generate FocusPixels at scale 2 and the largest two elephants would generate negative labels.

marked as invalid because after the early down-sampling operations, the network does not have sufficient information to make a correct prediction about them at that particular scale. The labelling scheme is visually depicted in Fig 3.2. For training the network, we add two convolutional layers (3×3 and 1×1) with a ReLU non-linearity on top of the conv5 feature-map. Finally, we have a binary softmax classifier to predict FocusPixels, shown in Fig 3.3.

3.4.2 FocusChip Generation

During inference, we mark those pixels $\mathcal{P}$ in the output as FocusPixels, where the probability of foreground is greater than a threshold t, which is a parameter controlling the speed-up and can be set with respect to the desired speed accuracy trade-off. This generates a number of connected components $\mathcal{S}$. We dilate each component with a filter of size $d \times d$ to increase contextual information needed for recognition. After dilation,

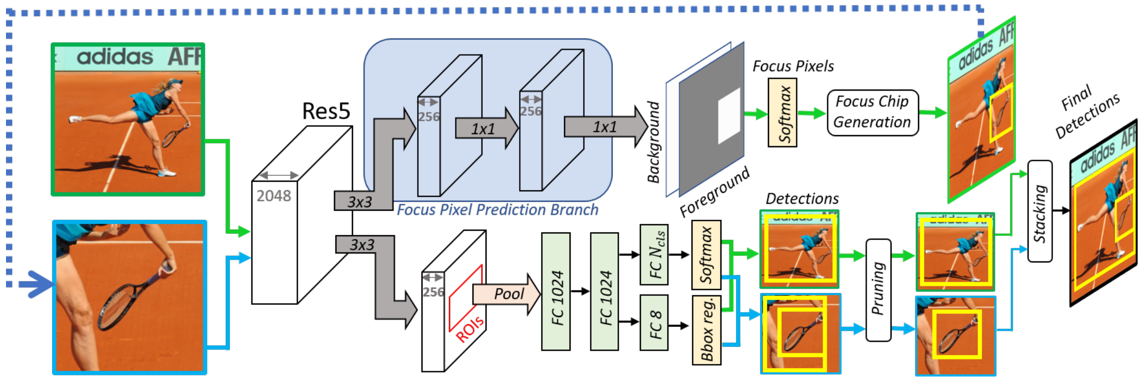


Figure 3.3: The figure illustrates how AutoFocus detects a person and a racket in an image. The green borders and arrows are for inference at the original resolution. The blue borders and arrows are shown when inference is performed inside FocusChips. In the first iteration, the network detects the person and also generates a heat-map to mark regions containing small objects. This is depicted in the white/grey map - it is used to generate FocusChips. In the next iteration, the detector is then applied inside FocusChips only. Inside FocusChips, there could be detections for the cropped object present at the larger resolution. Such detections are pruned and finally valid detections are stacked across multiple scales.

components which become connected are merged. Then, we generate chips $\mathcal{C}$ which enclose these connected components. Note that chips of two connected components could overlap. As a result, these chips are merged and overlapping chips are replaced with their enclosing bounding-boxes. Some connected components could be very small, and may lack the contextual information needed to perform recognition. Many small chips also increase fragmentation which results in a wide range of chip sizes. This makes batch-inference inefficient. To avoid these problems, we ensure that the height and width of a chip is greater than a minimum size k. This process is described in Algorithm 1. Finally, we perform multi-scale inference on an image pyramid but successively prune regions which are unlikely to contain objects.

```
Algorithm 1: FocusChip Generator
    Input  : Predictions for feature map $\mathcal{P}$, threshold $t$, dilation constant $d$,
             minimum size of chip $k$
    Output: Chips $\mathcal{C}$
1   Transform $\mathcal{P}$ into a binary map using the threshold $t$
2   Dilate $\mathcal{P}$ with a $d \times d$ filter
3   Obtain a set of connected components $\mathcal{S}$ from $\mathcal{P}$
4   Generate enclosing chips $\mathcal{C}$ of size $> k$ for each component in $\mathcal{S}$
5   Merge chips $\mathcal{C}$ if they overlap
6   return Chips $\mathcal{C}$
```

3.4.3 Focus Stacking for Object Detection

One issue with such cascaded multi-scale inference is that some detections at the boundary of the chips can be generated for cropped objects which were originally large. At the next scale, due to cropping, they could become small and generate false positives, such as the detections for the horse and the horse rider on the right, shown in Fig 3.4c. To alleviate this effect, Step 2 in Algorithm 1 is very important. Note that when we dilate the map $\mathcal{P}$ and generate chips, this ensures that no *interesting* object at the next scale would be observed at the boundaries of the chip (unless the chip shares a border with the image boundary). Otherwise, it would be enclosed by the chip, as these are generated around the dilated maps. Therefore, if a detection in the zoomed-in chip is observed at the boundary, we discard it, even if it is within valid SNIP ranges, such as the horse rider eliminated in Fig 3.4d.

There are some corner cases when the detection is at the boundary (or boundaries x, y) of the image. If the chip shares one boundary with the image, we still check if the other side of the detection is completely enclosed inside or not. If it is not, we discard it, else we keep it. In another case, if the chip shares both the sides with the image boundary

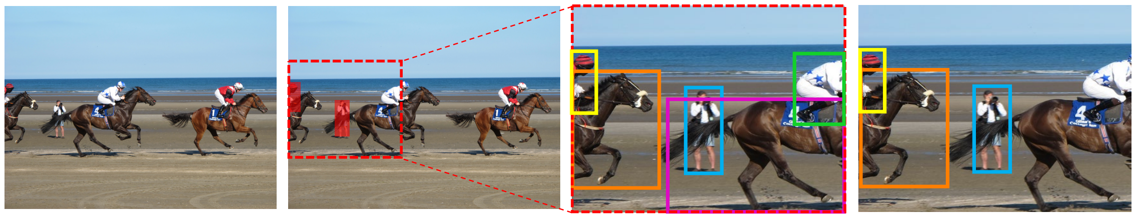

(a) (b) (c) (d)

Figure 3.4: Pruning detections while FocusStacking. (a) Original Image (b) The predicted FocusPixels and the generated FocusChip (c) Detection output by the network (d) Final detections for the FocusChip after pruning.

and so does the detection, then we keep the detection.

Once valid detections from each scale are obtained using the above rules, we merge detections from all the scales by projecting them to the image co-ordinates after applying appropriate scaling and translation. Finally, Non-Maximum Suppression is applied to aggregate the detections. The network architecture and an example of multi-scale inference and focus stacking is shown in Fig 3.3.

3.5 Datasets and Experiments

We evaluate AutoFocus on the COCO [93] and the PASCAL VOC [94] datasets. As our baseline, we use the SNIPER detector[1] [55] which obtains an mAP of 47.9% (68.3% at 50% overlap) on the COCO test-dev set and 47.5% (67.9% at 50% overlap) on the COCO validation set. We add the fully convolutional layers for AutoFocus which predict the FocusPixels. No other changes are made to the architecture or the training schedule. We use Soft-NMS [24] at test-time for Focus Stacking with $\sigma = 0.55$. Following SNIPER [55], the resolutions used for the 3 scales at inference are S1=$(480, 512)$, S2=$(800, 1280)$, and S3=$(1400, 2000)$. The first resolution is the minimum size of a side and the second

[1]`http://www.github.com/mahyarnajibi/SNIPER`

one is the maximum in pixels. The scales corresponding to these resolutions are referred to as scales 1, 2 and 3 respectively in the following sections.

Since FocusChips of different size are generated, we group chips which are of similar size and aspect ratio to achieve a high batch inference throughput. In some cases, we need to perform padding when performing batch inference, which can slightly change the number of pixels processed per image. For large datasets, this overhead is negligible as the number of groups (for size and aspect ratio) can be increased without reducing the batch size.

3.5.1 Stats for FocusPixels and FocusChips

In high resolution images (scale 3), the percentage of FocusPixels is very low (*i.e.* $\sim 4\%$). So, ideally a very small part of the image needs to be processed at high resolution. Since the image is upsampled, the FocusPixels projected on the image occupy an area of 63^2 pixels on average (the highest resolution images have an area of 1602^2 pixels on average). At lower scales (like scale 2), although the percentage of FocusPixels increases to $\sim 11\%$, their projections only occupy an area of 102^2 pixels on average (each image at this scale has an average area of 940^2 pixels). After dilating FocusPixels with a kernel of size 3×3, their percentages at scale 3 and scale 2 change to 7% and 18% respectively.

Using the chip generation algorithm, for a given minimum chip size (like $k = 512$), we also compute the upper bound on the speedup which can be obtained. This is under the assumption that FocusPixels can be predicted without any error (*i.e.* based on GTs). The bound for the speedup can change as we change the minimum chip size in the algorithm.

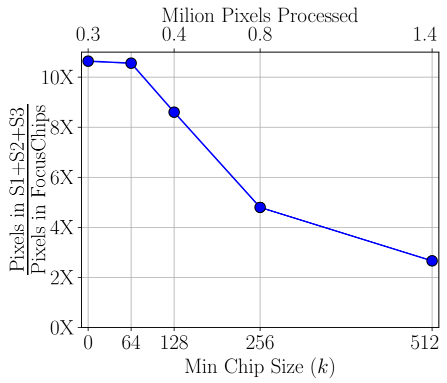
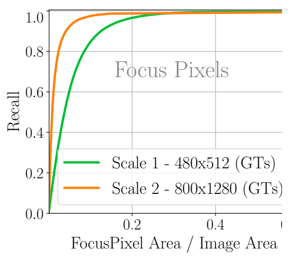


Figure 3.5: Upper-bound on the speed-up using FocusChips generated from optimal FocusPixels.

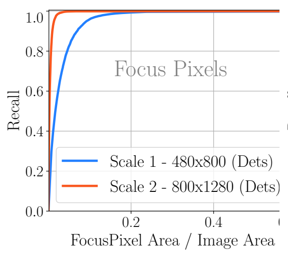
Fig 3.5 shows the effect of the minimum chip size parameter k for FocusChip generation in algorithm 1. The same value is used at each scale. For example, reducing the minimum chip size from 512 to 64 can lead to a theoretical speedup of ~ 10 times over the baseline which performs inference on 3 scales. However, a significant reduction in minimum chip size can also affect detection performance - a reasonable amount of context is necessary for retaining high detection accuracy.

3.5.2 Quality of FocusPixel prediction

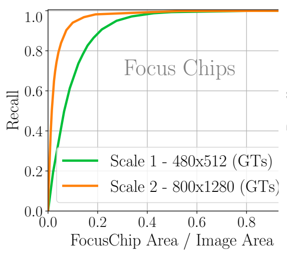
We evaluate how well our network predicts FocusPixels at different scales. To measure the performance, we use two criteria. First, we measure recall for predicting FocusPixels at two different resolutions. This is shown in Fig 3.6a. This gives us an upper bound on how accurately we localize small objects using low resolution images. However, not all ground-truth objects which are annotated might be correctly detected. Note that our eventual goal is to accelerate the detector. Therefore, if we crop a region in the image which contains a ground-truth instance but the detector is not able to detect it, cropping that region would not be useful. The final effectiveness of FocusChips is



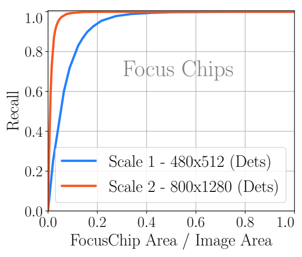
(a)



(b)



(c)



(d)

Figure 3.6: Quality of the FocusPixels and FocusChips. The x-axis represents the ratio of the area of FocusPixels or FocusChips to that of the image. The y-axis changes as follows, (a) FocusPixel recall is computed based on the GT boxes (b) FocusPixel recall is computed using the confident detections (c) FocusChip recall is computed based on the GT boxes (d) FocusChip recall is computed based on the confident detections.

coupled with the detector, hence we also evaluate the accuracy of FocusPixel prediction on regions which are confidently detected as shown in Fig 3.6b. To this end, we only consider FocusPixels corresponding to those GT boxes which are covered (IoU > 0.5) by a detection with a score greater than 0.5. At a threshold of 0.5, the detector still obtains an mAP of 47% which is within 1% of the final mAP and does not have a high false positive rate.

As expected, we obtain better recall at higher resolutions with both metrics. We can cover all confident detections at the higher resolution (scale 2) when the predicted FocusPixels cover just 5% of total image area. At a lower resolution (scale 1), when the FocusPixels cover 25% of the total image area, we cover all confident detections, see Fig 3.6b.

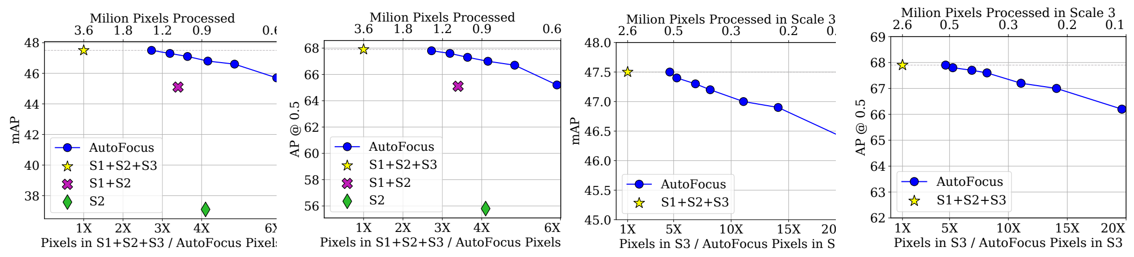
3.5.3 Quality of FocusChips

While FocusPixels are sufficient to generate enclosing regions which need to be processed, current software implementations require the input image to be a rectangle for efficient processing. To this end, we evaluate the performance of the enclosing chips generated using the FocusPixels. Similar to Section 3.5.2, we use two metrics - one is recall of all GT boxes which are enclosed by FocusChips, the other one is recall for GT boxes enclosed by FocusChips which have a confident overlapping detection. To achieve perfect recall for confident detections at scale 2, FocusChips cover 5% more area than FocusPixels. At scale 1, they cover 10% more area. This is because objects are often not rectangular in shape. These results are shown in Fig 3.6d.

3.5.4 Speed Accuracy Trade-off

We perform grid-search on different parameters, which are dilation, min-chip size and the threshold to generate FocusChips on a subset of 100 images in the validation set. For a given average number of pixels, we check which configuration of parameters obtains the best mAP on this subset. Since there are two scales at which we predict FocusPixels, we first find the parameters of AutoFocus when it is only applied to the highest resolution scale. Then we fix these parameters for the highest scale, and find parameters for applying AutoFocus at scale 2.

In Fig 3.7 we show that the multi-scale inference baseline which uses 3 scales obtains an mAP of 47.5% (and 68% at 50% overlap) on the val-2017 set. Using only the lower two scales obtains an mAP of 45.4%. The middle scale alone obtains an mAP of 37%. This is partly because the detector is trained with the scale normalization scheme proposed in [8]. As a result, the performance on a single scale alone is not very good, although multi-scale performance is high. The maximum savings in pixels which we



(a) (b) (c) (d)

Figure 3.7: Results are on the val-2017 set. (a,c) show the mAP averaged for IoU from 0.5 to 0.95 with an interval of 0.05 (COCO metric). (b,d) show mAP at 50% overlap (PASCAL metric). We can reduce the number of pixels processed by a factor of 2.8 times without any loss of performance. A 5 times reduction in pixels is obtained with a drop of 1% in mAP.

can obtain while retaining performance is 2.8 times. We lose approximately 1% mAP to obtain a 5 times reduction over our baseline in the val-2017 set.

We also perform an ablation experiment for the FocusPixels predicted using scale 2. Note that the performance of just using scales 1 and 2 is 45%. We can retain the original performance of 47.5% on the val-2017 set by processing just one fifth of scale 3. With a 0.5% drop we can reduce the pixels processed by 11 times in the highest resolution image. This can be improved to 20 times with a 1% drop in mAP, which is still 1.5% better than the performance of the lower two scales.

Results on the COCO test-dev set are provided in Table 3.1. While matching SNIPER's performance of 47.9% (68.3% at 0.5 IoU), AutoFocus processes 6.4 images per second on the test-dev set with a Titan X Pascal GPU. SNIPER processes 2.5 images per second. RetinaNet with a ResNet-101 backbone and a FPN architecture processes 6.3 images per second on a P100 GPU (which is like Titan X), but obtains 37.8% mAP [2]. We also report the number of pixels processed with a few efficient recent detectors. Detectors

[2] https://github.com/facebookresearch/Detectron/blob/master/MODEL_ZOO.md

Method	**Pixels**	**AP**	**AP^{50}**	**S**	**M**	**L**
Retina [39]	950^2	37.8	57.5	20.2	41.1	49.2
LightH [95]	940^2	41.5	-	25.2	45.3	53.1
Refine+ [96]	3100^2	41.8	62.9	25.6	45.1	54.1
Corner+ [97]	1240^2	42.1	57.8	20.8	44.8	56.7
SNIPER [55]	1910^2	**47.9**	**68.3**	**31.5**	**50.5**	**60.3**
AutoFocus	1175^2	**47.9**	**68.3**	**31.5**	**50.5**	**60.3**
	930^2	47.2	67.5	30.9	49.0	60.0
	860^2	46.9	67.0	30.1	48.9	60.0

Table 3.1: Comparison with SNIPER on the COCO test-dev. This is our multi-scale baseline. Results for others are taken from the papers/GitHub of the authors. Note that average pixels processed over the dataset are reported (instead of the shorter side). All methods use a ResNet-101 backbone. '+' denotes the multi-scale version provided by the authors.

Method	**Pixels**	**AP^{50}**	**AP^{70}**
Deformable ConvNet [32]	705^2	82.3	67.8
Deformable ConvNet v2 [98]	705^2	84.9	73.5
SNIPER [55]	1915^2	86.6	80.5
AutoFocus*	860^2	85.8	79.5
AutoFocus	700^2	85.3	78.1
	1250^2	86.5	80.2

Table 3.2: Comparison on PASCAL VOC 2007 test-set. All methods use ResNet-101 and trained on VOC2012 trainval+VOC2007 trainval. The average pixels processed over the dataset are also reported. To show the robustness of AutoFocus to hyper-parameter choices, in '*' we use the same parameters as COCO and run the algorithm on PASCAL.

which perform better than SNIPER like MegDet [16] or PANet [7] are slower because they use complex architectures like ResNext-152 [27] etc. To the best of our knowledge, AutoFocus is the fastest detector which obtains an mAP of 47.9% (or 68.3% at 0.5 IoU) on the COCO dataset. We show the inference process for AutoFocus on a few images in the COCO val-2017 set in Fig 3.8. We also report results on the PASCAL VOC dataset in Table 3.2. To show the robustness of AutoFocus to its hyper-parameters, we use exactly the same hyper-parameters tuned for COCO (shown as AutoFocus*). While processing the same area as DeformableV2 [98], AutoFocus achieves 4.6% better AP at 0.7 IoU. It also matches the performance of SNIPER while being considerably more efficient. Its mAP (on the COCO metric) can be further improved by using refinement techniques like cascade-RCNN [99].

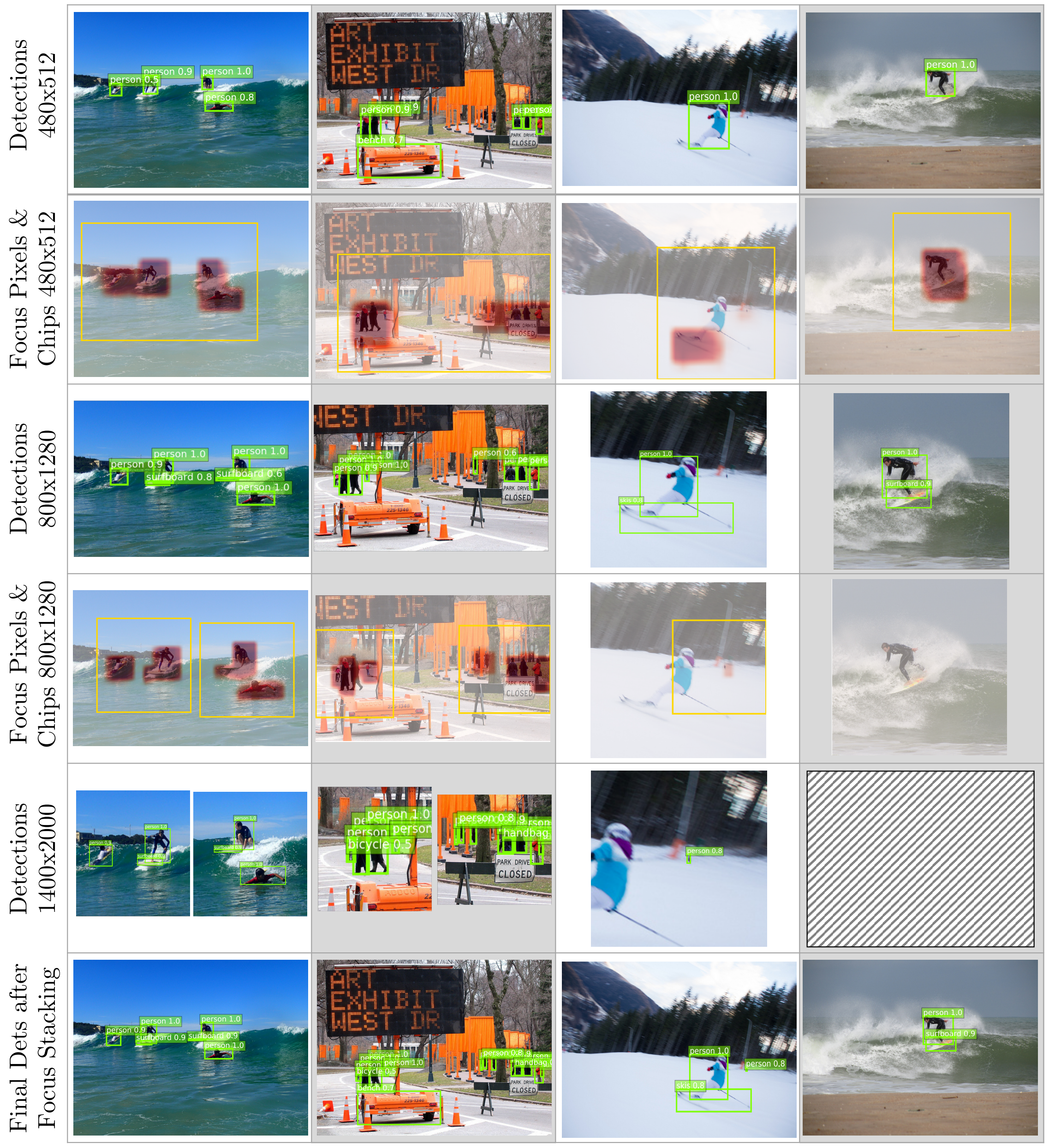

Detections 480x512
Focus Pixels & Chips 480x512
Detections 800x1280
Focus Pixels & Chips 800x1280
Detections 1400x2000
Final Dets after Focus Stacking

Figure 3.8: Each column shows the inference pipeline in AutoFocus. The confidence for FocusPixels and FocusChips are shown in red, and yellow respectively in the second and fourth rows. Detections are shown in green. As can be seen, complex images containing many small objects like the two leftmost columns can generate multiple FocusChips in high resolutions like 1400×2000. Images which do not contain small objects are not processed at all in high resolution, like the one in the rightmost column.

Chapter 4: SSH: Single Stage Headless Face Detector

4.1 Introduction

Face detection is a crucial step in various problems involving verification, identification, expression analysis, *etc*. From the Viola-Jones [63] detector to recent work by Hu *et al*. [100], the performance of face detectors has been improved dramatically. However, detecting small faces is still considered a challenging task. The recent introduction of the *WIDER* face dataset [101], containing a large number of small faces, exposed the performance gap between humans and current face detectors. The problem becomes more challenging when the speed and memory efficiency of the detectors are taken into account. The best performing face detectors are usually slow and have high memory foot-prints (*e.g*. [100] takes more than 1 second to process an image, see Section 4.4.5) partly due to the huge number of parameters as well as the way robustness to scale or incorporation of context are addressed.

State-of-the-art CNN-based detectors convert image classification networks into two-stage detection systems [11, 5]. In the first stage, early convolutional feature maps are used to propose a set of candidate object boxes. In the second stage, the remaining layers of the classification networks (*e.g*. *fc6~8* in *VGG-16* [102]), which we refer to as the network "*head*", are deployed to extract local features for these candidates and clas-

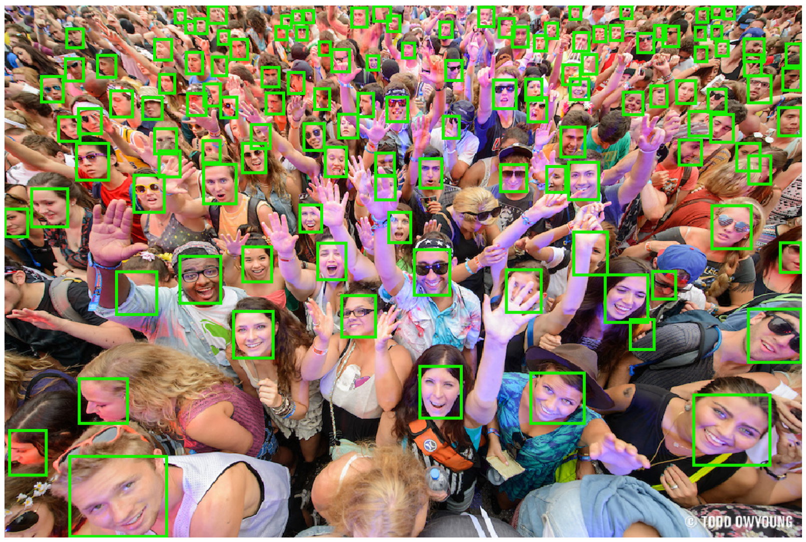


Figure 4.1: *SSH* detects various face sizes in a single CNN forward pass and without employing an image pyramid in ~ 0.1 second for an image with size 800×1200 on a GPU.

sify them. The head in the classification networks can be computationally expensive (*e.g.* the network head contains $\sim$ 120M parameters in *VGG-16* and $\sim$ 12M parameters in *ResNet-101*). Moreover, in the two stage detectors, the computation must be performed for all proposed candidate boxes.

Very recently, Hu *et al.* [100] showed state-of-the-art results on the *WIDER* face detection benchmark by using a similar approach to the Region Proposal Networks (*RPN*) [5] to directly detect faces. Robustness to input scale is achieved by introducing an image pyramid as an integral part of the method. However, it involves processing an input pyramid with an up-sampling scale up to 5000 pixels per side and passing each level to a very deep network which increased inference time.

In this section, we introduce the Single Stage Headless (*SSH*) face detector. *SSH* performs detection in a single stage. Like *RPN* [5], the early feature maps in a classification network are used to regress a set of predefined anchors towards faces. However, unlike two-stage detectors, the final classification takes place together with regressing the anchors. *SSH* is headless. It is able to achieve state-of-the-art results while removing the head of its underlying network (*i.e.* all fully connected layers in *VGG-16*), leading to a light-weight detector. Finally, *SSH* is scale-invariant by design. Instead of relying on an external multi-scale pyramid as input, inspired by [30], *SSH* detects faces from various depths of the underlying network. This is achieved by placing an efficient convolutional detection module on top of the layers with different strides, each of which is trained for an appropriate range of face scales. Surprisingly, *SSH* based on a headless *VGG-16*, not only outperforms the best-reported *VGG-16* by a large margin but also beats the current *ResNet-101*-based state-of-the-art method on the *WIDER* face detection dataset. Unlike the current state-of-the-art, *SSH* does not deploy an input pyramid and is 5 times faster. If an input pyramid is used with *SSH* as well, our light-weight *VGG-16*-based detector outperforms the best reported *ResNet-101* [100] on all three subsets of the *WIDER* dataset and improves the mean average precision by 4% and 2.5% on the validation and the test set respectively. *SSH* also achieves state-of-the-art results on the *FDDB* and *Pascal-Faces* datasets with a relatively small input size, leading to a runtime of 50 ms/image.

The rest of the section is organized as follows. Section 4.2 provides an overview of the related works. Section 4.3 introduces the proposed method. Section 4.4 presents the experiments and finally the conclusion is presented in Section 4.5.

4.2 Related Works

4.2.1 Face Detection

Prior to the re-emergence of convolutional neural networks (*CNN*), different machine learning algorithms were developed to improve face detection performance [63, 103, 104, 105, 106, 107, 108]. However, following the success of these networks in classification tasks [1], they were applied to detection as well [9]. Face detectors based on *CNN*s significantly closed the performance gap between human and artificial detectors [109, 110, 111, 112, 100]. However, the introduction of the challenging *WIDER* dataset [101], containing a large number of small faces, re-highlighted this gap. To improve performance, *CMS-RCNN* [112] changed the *Faster R-CNN* object detector [5] to incorporate context information. Very recently, Hu *et al*. proposed a face detection method based on proposal networks which achieves state-of-the-art results on this dataset [100]. However, in addition to skip connections, an input pyramid is processed by re-scaling the image to different sizes, leading to slow detection speeds. In contrast, *SSH* is able to process multiple face scales simultaneously in a single forward pass of the network, which reduces inference time noticeably.

4.2.2 Single Stage Detectors and Proposal Networks

The idea of detecting and localizing objects in a single stage has been previously studied for general object detection. *SSD* [42] and *YOLO* [77] perform detection and classification simultaneously by classifying a fixed grid of boxes and regressing them towards objects. *G-CNN* [113] models detection as a piece-wise regression problem and iteratively pushes an initial multi-scale grid of boxes towards objects while classifying them. However, current state-of-the-art methods on the challenging *MS-COCO* object detection benchmark are based on two-stage detectors[93]. *SSH* is a single stage detector; it detects faces directly from the early convolutional layers without requiring a proposal stage.

Although *SSH* is a detector, it is more similar to the object proposal algorithms which are used as the first stage in detection pipelines. These algorithms generally regress a fixed set of *anchors* towards objects and assign an objectness score to each of them. *MultiBox* [114] deploys clustering to define anchors. *RPN* [5], on the other hand, defines anchors as a dense grid of boxes with various scales and aspect ratios, centered at every location in the input feature map. *SSH* uses similar strategies, but to localize and at the same time detect, faces.

4.2.3 Scale Invariance and Context Modeling

Being scale invariant is important for detecting faces in unconstrained settings. For generic object detection, [35, 25] deploy feature maps of earlier convolutional layers to detect small objects. Recently, [30] used skip connections in the same way as [75] and employed multiple shared *RPN* and classifier heads from different convolutional layers. For face detection, *CMS-RCNN* [112] used the same idea as [35, 25] and added skip connections to the *Faster RCNN* [5]. [100] creates a pyramid of images and processes each separately to detect faces of different sizes. In contrast, *SSH* is capable of detecting faces at different scales in a single forward pass of the network without creating an image pyramid. We employ skip connections in a similar fashion as [75, 30], and train three detection modules jointly from the convolutional layers with different strides to detect small, medium, and large faces.

In two stage object detectors, context is usually modeled by enlarging the window around proposals [25]. [35] models context by deploying a recurrent neural network. For face detection, *CMS-RCNN* [112] utilizes a larger window with the cost of duplicating the classification head. This increases the memory requirement as well as detection time. *SSH* uses simple convolutional layers to achieve the same larger window effect, leading to more efficient context modeling.

4.3 Proposed Method

SSH is designed to decrease inference time, have a low memory foot-print, and be scale-invariant. *SSH* is a single-stage detector; *i.e.* instead of dividing the detection task into bounding box proposal and classification, it performs classification together with localization from the global information extracted from the convolutional layers. We empirically show that in this way, *SSH* can remove the "head" of its underlying network while achieving state-of-the-art face detection accuracy. Moreover, *SSH* is scale-invariant by design and can incorporate context efficiently.

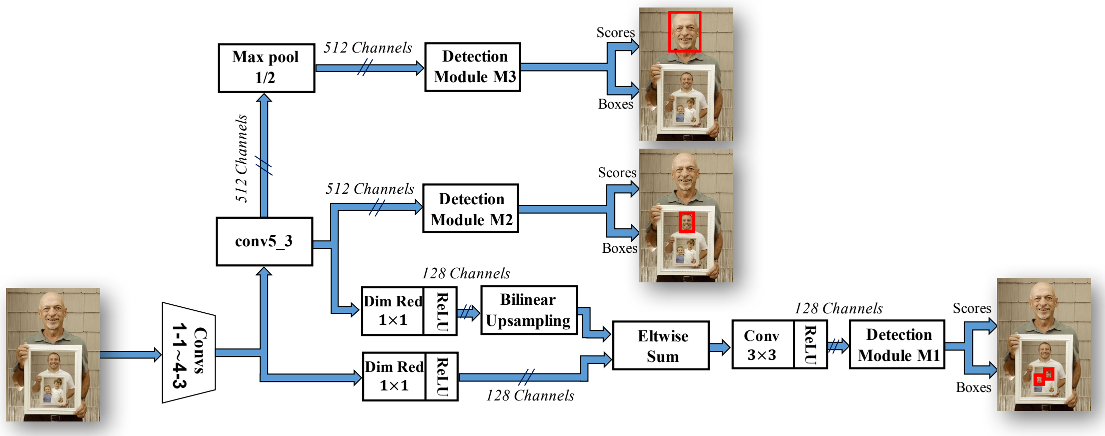


Figure 4.2: The network architecture of *SSH*.

4.3.1 General Architecture

Figure 4.2 shows the general architecture of *SSH*. It is a fully convolutional network which localizes and classifies faces early on by adding a *detection module* on top of feature maps with strides of 8, 16, and 32, depicted as $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$ respectively. The *detection module* consists of a convolutional binary classifier and a regressor for detecting faces and localizing them respectively.

To solve the localization sub-problem, as in [114, 5, 113], *SSH* regresses a set of predefined bounding boxes called *anchor*s, to the ground-truth faces. We employ a similar strategy to the *RPN* [5] to form the anchor set. We define the anchors in a dense overlapping sliding window fashion. At each sliding window location, K anchors are defined which have the same center as that window and different scales. However, unlike *RPN*, we only consider anchors with aspect ratio of one to reduce the number of anchor boxes. We noticed in our experiments that having various aspect ratios does not have a noticeable impact on face detection precision. More formally, if the feature map connected to the

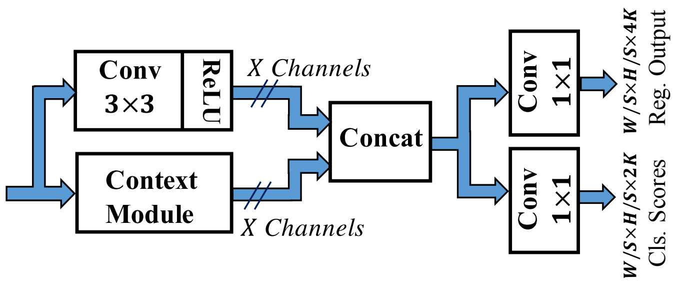


Figure 4.3: *SSH* detection module.

detection module $\mathcal{M}_i$ has a size of $W_i \times H_i$, there would be $W_i \times H_i \times K_i$ anchors with aspect ratio one and scales $\{S_i^1, S_i^2, \dots S_i^{K_i}\}$.

For the detection module, a set of convolutional layers are deployed to extract features for face detection and localization as depicted in Figure 4.3. This includes a simple context module to increase the effective receptive field as discussed in section 4.3.3. The number of output channels of the context module, (*i.e.* "X" in Figures 4.3 and 4.4) is set to 128 for detection module $\mathcal{M}_1$ and 256 for modules $\mathcal{M}_2$ and $\mathcal{M}_3$. Finally, two convolutional layers perform bounding box regression and classification. At each convolution location in $\mathcal{M}_i$, the classifier decides whether the windows at the filter's center and corresponding to each of the scales $\{S_i^k\}_{k=1}^K$ contains a face. A 1×1 convolutional layer with $2 \times K$ output channels is used as the classifier. For the regressor branch, another 1×1 convolutional layer with $4 \times K$ output channels is deployed. At each location during the convolution, the regressor predicts the required change in scale and translation to match each of the *positive* anchors to faces.

4.3.2 Scale-Invariance Design

In unconstrained settings, faces in images have varying scales. Although forming a multi-scale input pyramid and performing several forward passes during inference, as in [100], makes it possible to detect faces with different scales, it is slow. In contrast, *SSH* detects large and small faces simultaneously in a single forward pass of the network. Inspired by [30], we detect faces from three different convolutional layers of our network using detection modules $\mathcal{M}_1, \mathcal{M}_2$, and $\mathcal{M}_3$. These modules have strides of 8, 16, and 32 and are designed to detect small, medium, and large faces respectively.

More precisely, the detection module $\mathcal{M}_2$ performs detection from the *conv5-3* layer in *VGG-16*. Although it is possible to place the detection module $\mathcal{M}_1$ directly on top of *conv4-3*, we use the feature map fusion which was previously deployed for semantic segmentation [75], and generic object detection [30]. However, to decrease the memory consumption of the model, the number of channels in the feature map is reduced from 512 to 128 using 1×1 convolutions. The *conv5-3* feature maps are up-sampled and summed up with the *conv4-3* features, followed by a 3×3 convolutional layer. We used bilinear up-sampling in the fusion process. For detecting larger faces, a max-pooling layer with stride of 2 is added on top of the *conv5-3* layer to increase its stride to 32. The detection module $\mathcal{M}_3$ is placed on top of this newly added layer.

During the training phase, each detection module $\mathcal{M}_i$ is trained to detect faces from a target scale range as discussed in 4.3.4. During inference, the predicted boxes from the different scales are joined together followed by Non-Maximum Suppression (*NMS*) to form the final detections.

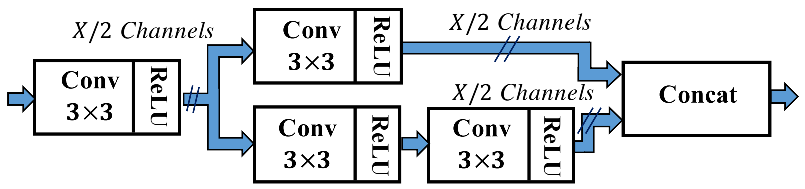


Figure 4.4: *SSH* context module.

4.3.3 Context Module

In two-stage detectors, it is common to incorporate context by enlarging the window around the candidate proposals. *SSH* mimics this strategy by means of simple convolutional layers. Figure 4.4 shows the context layers which are integrated into the detection modules. Since anchors are classified and regressed in a convolutional manner, applying a larger filter resembles increasing the window size around proposals in a two-stage detector. To this end, we use 5×5 and 7×7 filters in our context module. Modeling the context in this way increases the receptive field proportional to the stride of the corresponding layer and as a result the target scale of each detection module. To reduce the number of parameters, we use a similar approach as [115] and deploy sequential 3×3 filters instead of larger convolutional filters. The number of output channels of the detection module (*i.e.* "X" in Figure 4.4) is set to 128 for $\mathcal{M}_1$ and 256 for modules $\mathcal{M}_2$ and $\mathcal{M}_3$. It should be noted that our detection module together with its context filters uses fewer of parameters compared to the module deployed for proposal generation in [5]. Although, more efficient, we empirically found that the context module improves the mean average precision on the *WIDER* validation dataset by more than half a percent.

4.3.4 Training

We use stochastic gradient descent with momentum and weight decay for training the network. As discussed in section 4.3.2, we place three detection modules on layers with different strides to detect faces with different scales. Consequently, our network has three multi-task losses for the classification and regression branches in each of these modules as discussed in Section 4.3.4.1. To specialize each of the three detection modules for a specific range of scales, we only back-propagate the loss for the anchors which are assigned to faces in the corresponding range. This is implemented by distributing the anchors based on their size to these three modules (*i.e.* smaller anchors are assigned to $\mathcal{M}_1$ compared to $\mathcal{M}_2$, and $\mathcal{M}_3$). An anchor is assigned to a ground-truth face if and only if it has a higher IoU than 0.5. This is in contrast to the methods based on *Faster R-CNN* which assign to each ground-truth at least one anchor with the highest IoU. Thus, we do not back-propagate the loss through the network for ground-truth faces inconsistent with the anchor sizes of a module.

4.3.4.1 Loss function

SSH has a multi-task loss. This loss can be formulated as follows:

$$\sum_k \frac{1}{N_k^c} \sum_{i \in \mathcal{A}_k} \ell_c(p_i, g_i) + \\ \lambda \sum_k \frac{1}{N_k^r} \sum_{i \in \mathcal{A}_k} \mathcal{I}(g_i = 1) \ell_r(b_i, t_i) \tag{4.1}$$

where ℓ_c is the face classification loss. We use standard multinomial logistic loss as ℓ_c. The index k goes over the *SSH* detection modules $\mathcal{M} = \{\mathcal{M}_k\}_1^K$ and $\mathcal{A}_k$ represents the set of anchors defined in $\mathcal{M}_k$. The predicted category for the i'th anchor in $\mathcal{M}_k$ and its assigned ground-truth label are denoted as p_i and g_i respectively. As discussed in Section 4.3.2, an anchor is assigned to a ground-truth bounding box if and only if it has an IoU greater than a threshold (*i.e.* 0.5). As in [5], negative labels are assigned to anchors with IoU less than a predefined threshold (*i.e.* 0.3) with any ground-truth bounding box. N_k^c is the number of anchors in module $\mathcal{M}_k$ which participate in the classification loss computation.

ℓ_r represents the bounding box regression loss. Following [9, 11, 5], we parameterize the regression space with a log-space shift in the box dimensions and a scale-invariant translation and use smooth ℓ_1 loss as ℓ_r. In this parametrized space, p_i represents the predicted four dimensional translation and scale shift and t_i is its assigned ground-truth regression target for the i'th anchor in module $\mathcal{M}_k$. $\mathcal{I}(.)$ is the indicator function that limits the regression loss only to the positively assigned anchors, and $N_k^r = \sum_{i \in \mathcal{A}_k} I(g_i = 1)$.

4.3.5 Online hard negative and positive mining

We use online negative and positive mining (*OHEM*) for training *SSH* as described in [14]. However, *OHEM* is applied to each of the detection modules ($\mathcal{M}_k$) separately. That is, for each module $\mathcal{M}_k$, we select the negative anchors with the highest scores and the positive anchors with the lowest scores with respect to the weights of the network at that iteration to form our mini-batch. Also, since the number of negative anchors is more than the positives, following [11], 25% of the mini-batch is reserved for the positive anchors. As empirically shown in Section 4.4.8, *OHEM* has an important role in the success of *SSH* which removes the fully connected layers out of the *VGG-16* network.

4.4 Experiments

4.4.1 Experimental Setup

All models are trained on 4 GPUs in parallel using stochastic gradient descent. We use a mini-batch of 4 images. Our networks are fine-tuned for $21K$ iterations starting from a pre-trained ImageNet classification network. Following [11], we fix the initial convolutions up to *conv3-1*. The learning rate is initially set to 0.004 and drops by a factor of 10 after $18K$ iterations. We set momentum to 0.9, and weight decay to $5e^{-4}$. Anchors with IoU>0.5 are assigned to positive class and anchors which have an IoU<0.3 with all ground-truth faces are assigned to the background class. For anchor generation, we use scales $\{1,2\}$ in $\mathcal{M}_1$, $\{4,8\}$ in $\mathcal{M}_2$, and $\{16,32\}$ in $\mathcal{M}_3$ with a base anchor size of 16 pixels. All anchors have aspect ratio of one. During training, 256 detections per module is selected for each image. During inference, each module outputs 1000 best scoring anchors as detections and *NMS* with a threshold of 0.3 is performed on the outputs of all modules together.

4.4.2 Datasets

WIDER dataset[101]: This dataset contains $32,203$ images with $393,703$ annotated faces, $158,989$ of which are in the train set, $39,496$ in the validation set and the rest are in the test set. The validation and test set are divided into “easy”, “medium”, and “hard” subsets cumulatively (*i.e*. the “hard” set contains all images). This is one of the most challenging public face datasets mainly due to the wide variety of face scales and occlusion. We train all models on the train set of the *WIDER* dataset and evaluate on the validation and test sets. Ablation studies are performed on the the validation set (*i.e*. “hard” subset).

FDDB[116]: *FDDB* contains 2845 images and 5171 annotated faces. We use this dataset only for testing.

Pascal Faces[117]: *Pascal Faces* is a subset of the *Pascal VOC* dataset [118] and contains 851 images annotated for face detection. We use this dataset only to evaluate our method.

4.4.3 WIDER Dataset Result

We compare *SSH* with *HR* [100], *CMS-RCNN* [112], *Multitask Cascade CNN* [119], *LDCF* [120], Faceness [110], and *Multiscale Cascade CNN* [101]. When reporting *SSH* without an image pyramid, we rescale the shortest side of the image up to 1200 pixels while keeping the largest side below 1600 pixels without changing the aspect ratio. *SSH+Pyramid* is our method when we apply *SSH* to a pyramid of input images. Like *HR*, a four level image pyramid is deployed. To form the pyramid, the image is first scaled to have a shortest side of up to 800 pixels and the longest side less than 1200 pixels. Then, we scale the image to have min sizes of $500, 800, 1200$, and 1600 pixels in the pyramid. All modules detect faces on all pyramid levels, except $\mathcal{M}_3$ which is not applied to the

Method	easy	medium	hard
CMS-RCNN [112]	89.9	87.4	62.9
HR(VGG-16)+Pyramid [100]	86.2	84.4	74.9
HR(ResNet-101)+Pyramid [100]	92.5	91.0	80.6
SSH(VGG-16)	91.9	90.7	81.4
SSH(VGG-16)+Pyramid	**93.1**	**92.1**	**84.5**

Table 4.1: Comparison of *SSH* with top performing methods on the validation set of the *WIDER* dataset.

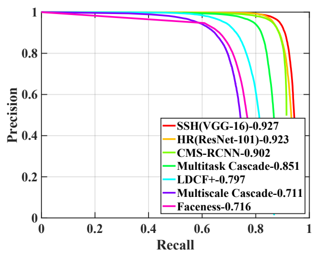
largest level.

Table 4.1 compares *SSH* with best performing methods on the *WIDER* validation set. *SSH* without using an image pyramid and based on the *VGG-16* network outperforms the *VGG-16* version of *HR* by $5.7\%, 6.3\%$, and 6.5% in “easy”, “medium”, and “hard” subsets respectively. Surprisingly, *SSH* also outperforms *HR* based on *ResNet-101* on the whole dataset (*i.e.* “hard” subset) by 0.8. In contrast *HR* deploys an image pyramid. Using an image pyramid, *SSH* based on a light *VGG-16* model, outperforms the *ResNet-101* version of *HR* by a large margin, increasing the state-of-the-art on this dataset by $\sim 4\%$.

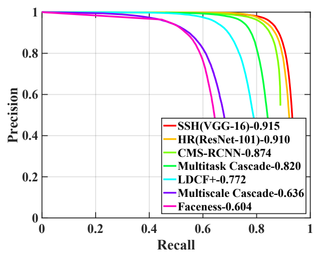
The precision-recall curves on the *test* set is presented in Figure 4.5. We submitted the detections of *SSH* with an image pyramid only once for evaluation. As can be seen, *SSH* based on a headless *VGG-16*, outperforms the prior methods on all subsets, increasing the state-of-the-art by 2.5%.

4.4.4 FDDB and Pascal Faces Results

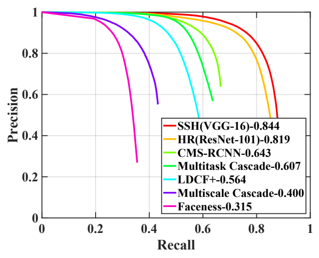
In these datasets, we resize the shortest side of the input to 400 pixels while keeping the larger side less than 800 pixels, leading to an inference time of less than 50 ms/image. We compare *SSH* with *HR*[100], *HR-ER*[100], Conv3D[121], Faceness[110], Faster R-



(a) Easy

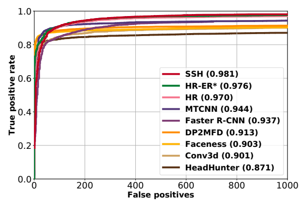


(b) Medium

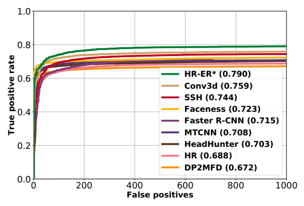


(c) Hard

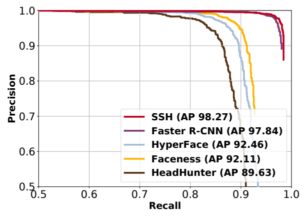
Figure 4.5: Comparison among the methods on the test set of *WIDER* face detection benchmark.



(a) FDDB discrete score.



(b) FDDB continuous score.



(c) Pascal-Faces.

Figure 4.6: Comparison among the methods on FDDB and Pascal-Faces datasets. (*Note that unlike *SSH*, *HR-ER* is also trained on the FDDB dataset in a 10-*Fold Cross Validation* fashion.)

CNN(*VGG-16*)[5], MTCNN[119], DP2MFD[122], and Headhunter[106]. Figures 4.6a and 4.6b show the ROC curves with respect to the discrete and continuous measures on the *FDDB* dataset respectively.

It should be noted that *HR-ER* also uses *FDDB* as a training data in a 10-fold cross validation fashion. Moreover, *HR-ER* and *Conv3D* both generate ellipses to decrease the localization error. In contrast, *SSH* does not use *FDDB* for training, and is evaluated on this dataset out-of-the-box by generating bounding boxes. However, as can be seen, *SSH* outperforms all other methods with respect to the discrete score. Compare to *HR*, *SSH* improved the results by 5.6% and 1.1% with respect to the continuous and discrete scores.

We also compare *SSH* with *Faster R-CNN*(*VGG-16*)[5], *HyperFace*[123], *Headhunter*[106], and *Faceness*[110] on the *Pascal-Faces* dataset. As shown in Figure 4.6c,

Max Size	400×800	600×1000	800×1200	1200×1600
Time	48 ms	74 ms	107 ms	182 ms

Table 4.2: *SSH* inference time with respect to different input sizes

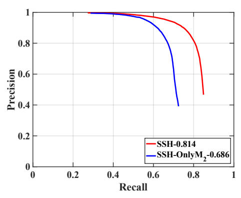
SSH achieves state-of-the-art results on this dataset.

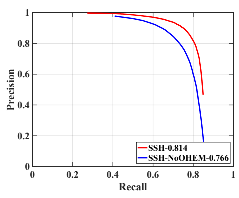
4.4.5 Timing

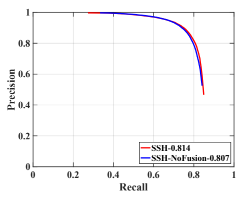
SSH performs face detection in a single stage while removing all fully-connected layers from the *VGG-16* network. This makes *SSH* an efficient detection algorithm. Table 4.2 shows the inference time with respect to different input sizes. We report average time on the *WIDER* validation set. Timing are performed on a *NVIDIA Quadro P6000* GPU. In column with max size $m \times M$, the shortest side of the images are resized to "m" pixels while keeping the longest side less than "M" pixels. As shown in section 4.4.3, and 4.4.4, *SSH* outperforms *HR* on all datasets without an image pyramid. On *WIDER* we resize the image to the last column and as a result detection takes 182 ms/image. In contrast, *HR* has a runtime of 1010 ms/image, more than $5X$ slower. As mentioned in Section 4.4.4, a maximum input size of 400×800 is enough for *SSH* to achieve state-of-the-art performance on *FDDB* and *Pascal-Faces*, with a detection time of 48 ms/image. If an image pyramid is used, the runtime would be dominated by the largest scale.

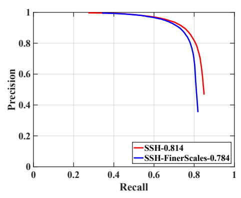
4.4.6 Ablation study: Scale-invariant design

As discussed in Section 4.3.2, *SSH* uses each of its detections modules, $\{\mathcal{M}_i\}_{i=1}^{3}$, to detect faces in a certain range of scales from layers with different strides. To better understand the impact of these design choices, we compare the results of *SSH* with and









(a) Effect of multi-scale design. (b) Effect of Hard Example Mining. (c) Effect of feature fusion. (d) Effect of increasing #anchors.

Figure 4.7: Ablation studies. All experiments are reported on the Wider Validation set.

without multiple detection modules. That is, we remove $\{\mathcal{M}_1, \mathcal{M}_3\}$ and only detect faces with $\mathcal{M}_2$ from *conv5-3* in *VGG-16*. However, for fair comparison, all anchor scales in $\{\mathcal{M}_1, \mathcal{M}_3\}$ are moved to $\mathcal{M}_2$ (*i.e*. we use $\cup_{i=1}^{3}\mathbf{S}_i$ in $\mathcal{M}_2$). Other parameters remain the same. We refer to this simpler method as *"SSH-OnlyM_2"*. As shown in Figure 4.7a, by removing the multiple detection modules from *SSH*, the *AP* significantly drops by $\sim 12.8\%$ on the *hard* subset which contains smaller faces. Although *SSH* does not deploy the expensive head of its underlying network, results suggest that having independent simple detection modules from different layers of the network is an effective strategy for scale-invariance.

4.4.7 Ablation study: The effect of input size

The input size can affect face detection precision, especially for small faces. Table 4.3 shows the *AP* of *SSH* on the *WIDER* validation set when it is trained and evaluated with different input sizes. Even at a maximum input size of 800×1200, *SSH* outperforms *HR-VGG16*, which up-scales images up to 5000 pixels, by 3.5%, showing the effectiveness of our scale-invariant design for detecting small faces.

Max Size	600×1000	800×1200	1200×1600	1400×1800
AP	68.6	78.4	81.4	81.0

Table 4.3: The effect of input size on average precision.

4.4.8 Ablation study: The effect of OHEM

As discussed in Section 4.3.5, we apply hard negative and positive mining (*OHEM*) to select anchors for each of our detection modules. To show its role, we train *SSH*, with and without *OHEM*. All other factors are the same. Figure 4.7b shows the results. Clearly, *OHEM* is important for the success of our light-weight detection method which does not use the pre-trained head of the *VGG-16* network.

4.4.9 Ablation study: The effect of feature fusion

In *SSH*, to form the input features for detection module $\mathcal{M}_1$, the outputs of *conv4-3* and *conv5-3* are fused together. Figure 4.7c, shows the effectiveness of this design choice. Although it does not have a noticeable computational overhead, as illustrated, it improves the *AP* on the *WIDER* validation set.

4.4.10 Ablation study: Selection of anchor scales

As mentioned in Section 4.4.1, *SSH* uses $\mathbf{S}_1 = \{1, 2\}$, $\mathbf{S}_2 = \{4, 8\}$, $\mathbf{S}_3 = \{16, 32\}$ as anchor scale sets. Figure 4.7d compares *SSH* with its slight variant which uses $\mathbf{S}_1 = \{0.25, 0.5, 1, 2, 3\}$, $\mathbf{S}_2 = \{4, 6, 8, 10, 12\}$, $\mathbf{S}_3 = \{16, 20, 24, 28, 32\}$. Although using a finer scale set leads to a slower inference, it also reduces the *AP* due to the increase in the number of *False Positives*.

4.4.11 Qualitative Results

Figure 4.8 shows some qualitative results on the Wider validation set. The colors encode the score of the classifier. Green and blue represent score 1.0 and 0.5 respectively.

4.5 Conclusion

We introduced the *SSH* detector, a fast and lightweight face detector that, unlike two-stage proposal/classification approaches, detects faces in a single stage. *SSH* localizes and detects faces simultaneously from the early convolutional layers in a classification network. *SSH* is able to achieve state-of-the-art results without using the "*head*" of its underlying classification network (*i.e. fc* layers in *VGG-16*). Moreover, instead of processing an input pyramid, *SSH* is designed to be scale-invariant while detecting different face scales in a single forward pass of the network. *SSH* achieves state-of-the-art performance on the challenging *WIDER* dataset as well as *FDDB* and *Pascal-Faces* while reducing the detection time considerably.

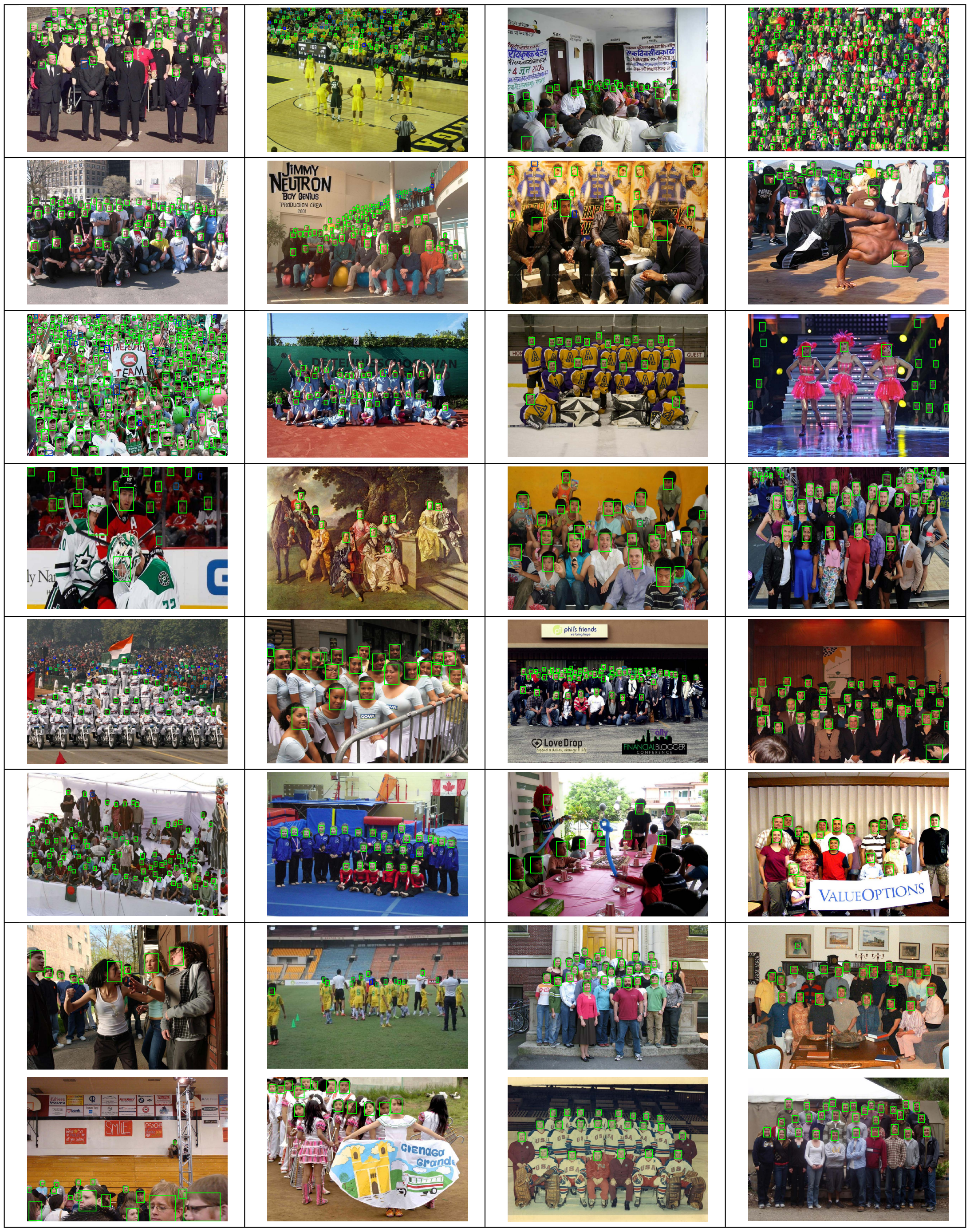


Figure 4.8: Qualitative results of *SSH* on the validation set of the *WIDER* dataset. Green and blue represent a classification score of 1.0 and 0.5 respectively.

Chapter 5: FA-RPN: Floating Region Proposals for Face Detection

5.1 Introduction

Face detection is an important computer vision problem and has multiple applications in surveillance, tracking, consumer-facing devices like iPhones *etc*. Hence, various approaches have been proposed towards solving it [110, 124, 100, 112, 121, 125, 126, 123, 31] and successful solutions have also been deployed in practice. So, expectations from face detection algorithms are much higher and error rates today are quite low. Algorithms need to detect faces which are as small as 5 pixels to 500 pixels in size. As localization is essential for detection, evaluating every small region of the image is important. Face detection datasets can have up to a thousand faces in a single image, which is not common in generic object detection.

Detectors like Faster-RCNN [5] employ a region proposal network (RPN) which places anchor boxes of different sizes and aspect ratios uniformly on the image and classifies them for generating object-like regions. However, RPN only uses a single pixel in the convolutional feature map for evaluating the proposal hypotheses, independent of the size of the object. Therefore, the feature representation in RPN entirely relies on the contextual information encoded in the high-dimensional feature representation generated at the pixel. It does not pool features from the entire extent of an object while generat-

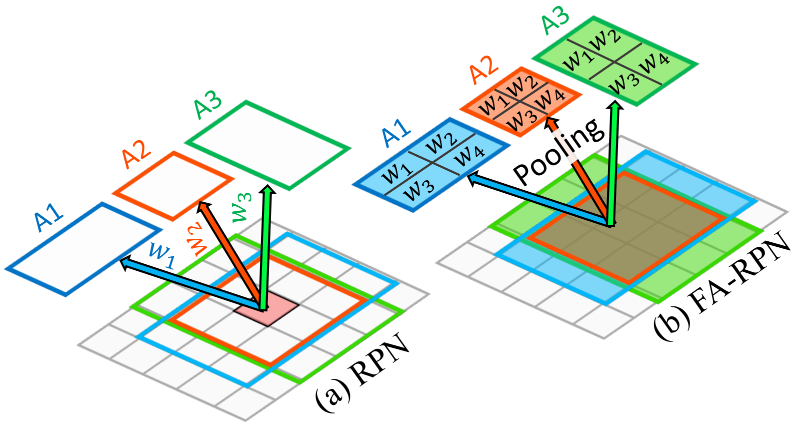


Figure 5.1: Difference between RPN and FA-RPN in terms of weight configuration. For simplicity we show 2x2 pooling for FA-RPN.

ing the feature representation, see Fig. 5.1. Thus, it can miss object regions or generate proposals which are not well localized. Further, it is not possible to iterate and refine the positions of the anchor-boxes as part of the proposal network. If objects of different scale/aspect-ratios are to be learned or if we want to place anchors at sub-pixel resolution, filters specific to each of these conditions need to be added during training. Generating proposals using a pooling-based algorithm can alleviate such problems easily.

There are predominantly two pooling-based methods for the final classification of RoIs in an image - Fast-RCNN [11] and R-FCN [127]. Fast-RCNN projects the region-proposals to the convolutional feature map, and pools the features inside the region of interest (RoI) to a fixed size grid (typically 7×7) and applies two fully connected layers which perform classification and regression. Due to computational constraints, this approach is practically infeasible for proposal generation as one would need to apply it to hundreds of thousands of regions - which is the number of region candidates which are typically evaluated by a region proposal algorithm.

To reduce the dependence on fully connected layers, R-FCN performs local convolutions (7×7) inside an RoI for capturing the spatial extent of each object. Since each of these local filters can be applied to the previous feature-map, we just need to pool the response from the appropriate region corresponding to each local filter. This makes it a good candidate for a pooling-based proposal approach as it is possible to apply it to a large number of RoIs efficiently. However, in high-resolution images, proposal algorithms like RPN evaluate hundreds of thousands of anchors during inference. It is computationally infeasible to perform pooling on that many regions. Luckily, many anchors are not necessary (*e.g.* large anchors which are very close to each other). In this section, we show that careful anchor placement strategies can reduce the number of proposals significantly to the point where a pooling-based algorithm becomes feasible for proposal generation. This yields an efficient and effective objectness detector which does not suffer from the aforementioned problems present in RPN designs.

A pooling-based proposal method based on R-FCN which relies on position sensitive filters is particularly well suited for face detection. While objects deform and positional correspondence between different parts is often lost - faces are rigid, structured and parts have positional semantic correspondence (*e.g.* nose, eyes, lips). Moreover, it is possible to place anchor boxes of different size and aspect ratios without adding more filters. We can also place fractional anchor boxes and perform bilinear interpolation while pooling features for computing objectness. We can further improve the localization performance of the proposal candidates by iteratively pooling again from the generated RoIs and all these design changes can be made during inference! Due to these reasons, we refer to our proposal network as *Floating Anchor Region Proposal Network* (FA-RPN).

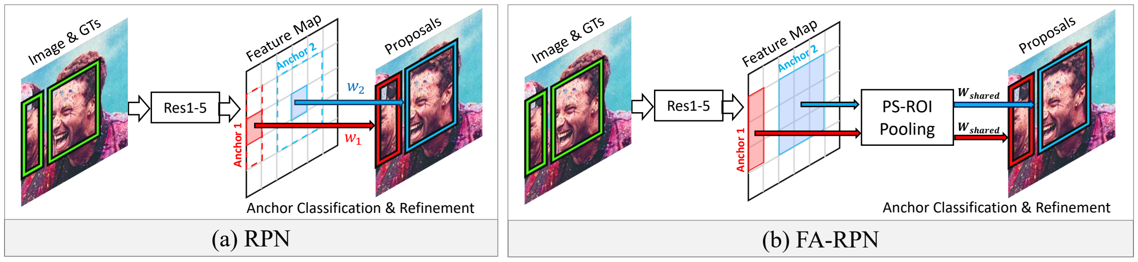


Figure 5.2: We highlight the major differences between RPN (a) and FA-RPN (b) proposals. RPN performs classification on a single pixel in the high-dimensional feature-map and uses different weights for classifying anchor-boxes of different sizes/aspect ratios. FA-RPN proposals, on the other hand, pool features from multiple bins in the image and share weights across objects of different sizes and aspect ratios.

We highlight these advantages in Fig. 5.1 and Fig. 5.2. On the WIDER dataset [101] we show that FA-RPN proposals are better than RPN proposals. FA-RPN also obtains state-of-the-art results on WIDER and PascalFaces which demonstrates its effectiveness for face detection.

5.2 Related Work

Generating class agnostic region proposals has been investigated in computer vision for more than a decade. Initial methods include multi-scale combinatorial grouping [128], constrained parametric min-cuts [2], selective search [78] *etc*. These methods generate region proposals which obtain high recall for objects in a category agnostic fashion. They were also very successful in the pre-deep learning era and obtained state-of-the-art performance even with a bag-of-words model [2]. Using region proposals based on selective search [2], R-CNN [9] was the first deep learning based detector. Unsupervised region proposals were also used in later detectors like Fast-RCNN [11] but since the Faster-RCNN detector [5] generated region proposals using a convolutional neural network, it has become the *de-facto* algorithm for generating region proposals.

To improve RPN, several modifications have been proposed. State-of-the-art detectors can also detect objects in a single step. Detectors like SSH [31], SSD [42], RetinaNet [39], MS-CNN [37] generate multi-scale feature maps to classify and regress anchors placed on these feature-maps. These single-shot detectors are closely related to the region proposal network as they have specific filters to detect objects of different sizes and aspect ratios but also combine feature-maps from multiple layers of the deep neural network. No further refinement is performed after the initial offsets generated by the network are applied. Another class of detectors are iterative, like G-CNN [113], Cascade-RCNN [99], LocNet [26], FPN [30], RFCN-3000 [129], Faster-RCNN [5]. These detectors refine a pre-defined set of anchor-boxes in multiple stages and have more layers to further improve classification and localization of regressed anchors. One should note that even in these networks, the first stage comprises of the region proposal network which eliminates the major chunk of background regions. FA-RPN is closer to this line of work but, in contrast, it supports iterative refinement of *region proposals* during inference.

We briefly review some recent work on face detection. With the availability of large scale datasets like WIDER [101] which contain many small faces in high resolution images, multiple new techniques for face detection have been proposed [110, 124, 100, 112, 121, 125, 126, 123, 130]. A lot of focus has been on scale, combining features of different layers [100, 31, 126, 124, 131, 132] and improving configurations of the region proposal network [126, 124]. For example, in finding tiny faces [100], it is proposed to perform detection on an image pyramid and to have different scale filters for objects of different sizes. SSH [31] and S3FD [124] efficiently utilize the intermediate layers of the network. PyramidBox [133] replaces the context module in SSH by deeper and wider sub-networks to better capture the contextual information for face detection. Recently, even GANs [134] have been used to improve the performance on tiny faces [130].

In face detection, the choice of anchors and their placement on the image is very important [126, 124]. For example, using extra strided anchors were shown to be beneficial [126]. Geometric constraints of the scene have also been used to prune region proposals [135]. Some of these changes require re-training RPN again. In our framework, design decisions such as evaluating different anchor scales, changing the stride of anchors, and adding fractional anchors can simply be made during inference as we share filters for all object sizes and only pooling is performed for them. Moreover, a pooling-based design also provides precise spatial features.

5.3 Background

We provide a brief overview of the R-FCN detector in this section. This detector uses RPN to generate region proposals. It classifies the top 2000 ranked proposals using the R-FCN detector. Classification is performed over all the foreground classes and the background class. The key component in R-FCN is local convolutions. It applies different filters in different sub-regions of an RoI for inferring the spatial extent of an object. These sub-regions may correspond to parts of an object. To accelerate this process over thousands of RoIs, convolution for each part in each object class is performed in the final layer. So, as an example, if there are 21 classes, the last feature-map would contain 21 $\times$ 49 channels. Then, given an RoI, *Position Sensitive RoIPooling* is performed on this feature-map to obtain the effect of local convolutions [127]. We refer the reader to the R-FCN [127] paper for further details on PSRoIPooling. Finally, the response is average pooled and used as the classification score of the object. In Deformable-RFCN [32], the regions for each bin where pooling is performed are also adjusted based on the input feature-map, which is referred to as deformable PSRoIPooling.

5.4 FA-RPN - Floating Anchor Region Proposal Network

In this section, we discuss the training of FA-RPN, which performs iterative classification and regression of anchors placed on an image for generating accurate region proposals. An overview of our approach is shown in Fig. 5.3.

5.4.1 Anchor Placement

In this architecture, classification of anchors is not performed using a single high-dimensional feature vector but by pooling features inside the RoI. Hence, there are no restrictions on how anchors can be placed during training and inference. As long as the convolutional filters can learn objectness, we can apply the model on RoIs of different sizes and aspect ratios, even if the network was not trained explicitly for those particular scales and aspect-ratios.

FA-RPN places anchors of different scales and aspect ratios on a grid, as generated in the region proposal network, and clips the anchors which extend beyond the image. While placing anchors, we vary the spatial stride as we increase the anchor size. Since nearby anchors at larger scales have a very high overlap, including them is not necessary.

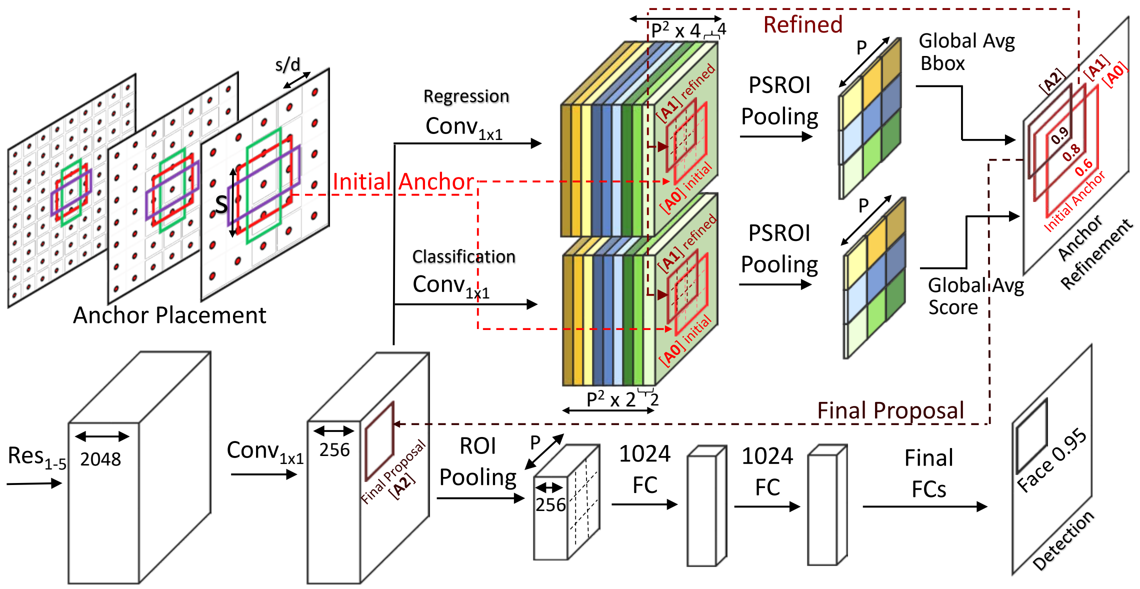


Figure 5.3: FA-RPN framework. FA-RPN uses multi-scale training. At each training iteration, an image scale is randomly selected and *suitable* anchor scales are placed over the image. This set of initial anchors are used to pool objectness scores and localization information from position sensitive filters (for simplicity only the localization branch is depicted in the figure). For improving localization, the top scoring initial anchors are further refined with subsequent poolings. The refinement process for an initial anchor (*[A0]*) is depicted in the figure. This anchor is first refined to *[A1]* based on the network prediction. Another pooling is performed over *[A1]* to form a new prediction for refining it further to the final anchor *[A2]*. Finally, a Faster-RCNN head is used to perform the final classification and regression.

We change the stride of anchor-boxes to $max(c, s/d)$, where s is square-root of the area of an anchor-box, c is a constant and d is the scaling factor, shown in Fig. 5.3. In practice, we set c to 16 and d to 5. This ensures that not too many overlapping anchor-boxes are placed on the image, while ensuring significant overlap between adjacent anchors to cover all objects. Naive placement of anchor boxes of 3 aspect ratios and 5 scales with stride equaling 16 pixels in an 800 $\times$ 1280 image leads to a 2-3$\times$ slow-down when performing inference. With the proposed placement method, we reduce the number of RoIs per image from 400,000 to 100,000 for a 1280 $\times$ 1280 image for the above-mentioned anchor configuration. When we increase the image size, the computation for convolution also increases proportionally, so as long as the time required for pooling is not significant compared to convolution, we will not observe a noticeable difference in performance.

There is no restriction that the stride of anchors should be the same as the stride of the convolutional feature-map. We can even place RoIs between two pixels in the convolutional feature-map without making any architectural change to the network. This allows us to augment the ground-truth bounding boxes as positive RoIs during training. This is unlike RPN, where the maximum overlapping anchor is assigned as positive when no anchor matches the overlap threshold criterion. We show qualitative examples of anchor placement for different scales and aspect ratios in FA-RPN in Fig. 5.3.

5.4.2 Sampling

Since there are hundreds of thousands of anchors which can be placed on an image, we sample anchors during training. We observe that using focal loss [39] reduced recall for RPN (hyper-parameter tuning could be a reason), so we did not use it for FA-RPN. We use the commonly used technique of sampling RoIs for handing class imbalance. In FA-RPN, an anchor-box is marked as positive if its overlap with a ground-truth box is greater than 0.5. An anchor is marked as negative if its overlap is less than 0.4. A maximum of 128 positive and negative anchors are sampled in a batch. Since the probability of a random anchor being an easy sample is high, we also sample 32 anchor-boxes which have an overlap of at least 0.1 with the ground-truth boxes as hard negatives. Just for training FA-RPN proposals, all other RoIs can be ignored. However, for training an end-to-end detector, we also need to score other RoIs in the image. When training an end-to-end detector, we select a maximum of 50,000 RoIs in an image (prioritizing those which have at-least 0.1 overlap with ground-truth boxes first).

5.4.3 Iterative Refinement

The initial set of placed anchors are expected to cover the ground-truth objects present in the image. However, these anchors may not always have an overlap greater than 0.5 with all objects and hence would be given low scores by the classifier. This problem is amplified for small object instances as mentioned in several methods [124, 100]. In this case, no anchor-boxes may have a high score for some ground-truth boxes. Therefore, the ground-truth boxes may not be covered in the top 500 to 1000 proposals generated in the image. In FA-RPN, rather than selecting the top 1000 proposals, we generate 20,000 proposals during inference and then perform pooling again on these 20,000 proposals from the same feature-map (we can also have another convolutional layer which refines the first stage region proposals). The hypothesis is that after refinement, the anchors would be better localized and hence the scores which we obtain after pooling features inside an RoI would be more reliable. Therefore, after refinement, the ordering of the top 1000 proposals would be different because scores are pooled from refined anchor-boxes rather than the anchor-boxes which were placed uniformly on a grid. Since we only need to perform pooling for this operation, it is efficient and can be easily implemented when the number of RoIs is close to 100,000. Note that our method is entirely pooling-based and does not have any fully connected layers like cascade-RCNN [99] or G-CNN [113].

Therefore, it is much more efficient for iterative refinement.

5.4.4 Complexity and Speed

FA-RPN is efficient. Namely, on 800×1280 size images, it takes 50 milliseconds to perform forward propagation for our network on a P6000 GPU. We also discuss how much time it takes to use R-FCN for end-to-end detection. For general object detection, when the number of classes is increased, to say 100, the contribution from the pooling layer also increases. This is because the complexity of pooling is linear in the number of classes. So, if we increase the number of classes to 100, this operation would become 100 times slower and at that stage, pooling will account for a significant portion of the time in forward-propagation. For instance, without our anchor placement strategy, it takes 100 seconds to perform inference for 100 classes in a single image on a V100 GPU. However, as for face detection, we only need to perform pooling for 2 classes and use a different anchor placement scheme, we do not face this problem and objectness can be efficiently computed even with tens of thousands of anchor boxes.

5.4.5 Scale Normalized Training

The positional correspondence of R-FCN is lost when RoI bins become too small. The idea of local convolution or having filters specific to different parts of an object is relevant when each bin corresponds to a unique region in the convolutional feature-map. The position-sensitive filters implicitly assume that features in the previous layer have a resolution which is similar to that after PSRoIPooling. Otherwise, if the RoI is too small, then all the position sensitive filters will pool from more or less the same position, nullifying the hypothesis that these filters are position sensitive. Therefore, we perform scale normalized training [8], which performs selective gradient propagation for RoIs which are close to a resolution of 224 $\times$ 224 and excludes those RoIs which can be observed at a better resolution during training. In this setting, the position-sensitive nature of filters is preserved to some extent, which helps in improving the performance of FA-RPN.

5.5 Datasets

We perform experiments on three benchmark datasets, WIDER [101], AFW [103], and Pascal Faces [117]. The WIDER dataset contains 32,203 images with 393,703 annotated faces, 158,989 of which are in the train set, 39,496 in the validation set, and the rest are in the test set. The validation and test set are divided into “easy”, “medium”, and “hard” subsets cumulatively (i.e. the “hard” set contains all faces and “medium” contains “easy” and “medium”). This is the most challenging public face dataset mainly due to the significant variation in the scale of faces and occlusion. We train all models on the train set of the WIDER dataset and evaluate on the validation set. We mention in our experiments when initialization of our pre-trained model is from ImageNet or COCO. Ablation studies are also performed on the validation set (i.e. “hard” subset which contains the whole dataset). Pascal Faces and AFW have 1335 and 473 faces respectively. We use Pascal Faces and AFW only as test sets for evaluating the generalization of our trained models. When performing experiments on these datasets, we apply the model trained on the WIDER train set out of the box.

5.6 Experiments

We train a ResNet-50 [21] based Faster-RCNN detector with deformable convolutions [32] and SNIP [8]. FA-RPN proposals are generated on the concatenated `Conv4` and `Conv5` features. On WIDER we train on the following image resolutions $(1800, 2800)$, $(1024, 1440)$ and $(512, 800)$. The SNIP ranges we use for WIDER are as follows, [0, 200) for (1800, 2800), [32, 300) for (1024, 1440) and [80, ∞) for (512, 800) as the size of the shorter side of the image is around 1024. We train for 8 epochs with a stepdown at 5.33 epochs. In all experiments, we use a learning rate and weight decay of 0.0005 and train on 8 GPUs. We use the same learning rate and training schedule even when training on 4 GPUs. In all our experiments, we use online hard example mining (OHEM) [14] to train the 2 fully connected layers in our detector. Hard example mining is performed on 900 proposals with a batch size of 512. RoIs which have an overlap greater than 0.5 with ground-truth bounding boxes are marked as positive and anything less than that is labeled as negative. We use Soft-NMS [24] with $\sigma = 0.35$ when performing inference. Since Pascal Faces and AFW contain low-resolution images and also do not contain faces as small as the WIDER dataset, we do not perform inference on the 1800×2800 resolution. All other parameters remain the same as the experiments on the WIDER dataset.

On the WIDER dataset, we remove anchors for different aspect ratios (*i.e.* we only have one anchor per scale with an aspect ratio of 1) and add a 16×16 size anchor for improving the recall for small faces. Note that extreme size anchors are removed during

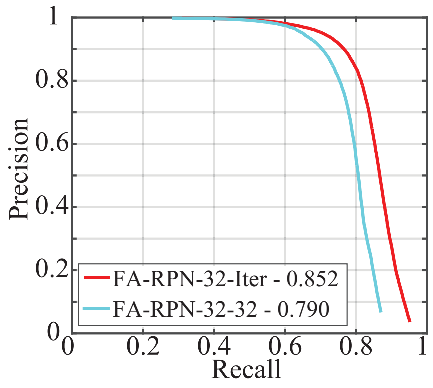
Method	AP
Baseline	87.2
Baseline + SNIP	88.1
Baseline + SNIP + COCO pre-training	89.1
Baseline + SNIP + COCO pre-training + Iteration	89.4

Table 5.1: Ablation analysis with different core-components of our face detector on the hard-set of the WIDER dataset (hard-set contains all images in the dataset).

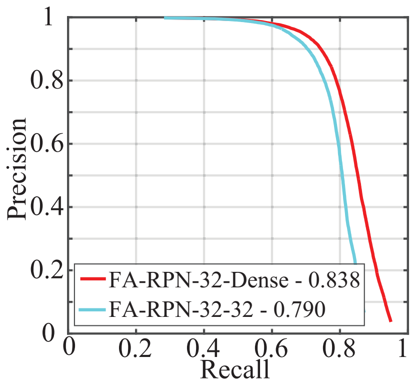
training with SNIP using the same rules which are used for training the detector. With these settings, we outperform state-of-the-art results on the WIDER dataset demonstrating the effectiveness of FA-RPN. However, the objective of this work is not to show that FA-RPN is necessary to obtain state-of-the-art performance. FA-RPN is an elegant and efficient alternative to RPN and can be combined with multi-stage face detection methods to improve performance.

5.6.1 Effect of Multiple Iterations in FA-RPN

We evaluate FA-RPN on WIDER when we perform multiple iterations during inference. Since FA-RPN operates on RoIs rather than classifying single-pixel feature-maps like RPN, we can further refine the RoIs which are generated after applying the regression offsets. As the initial set of anchor boxes are coarse, the RoIs generated after the first step are not very well localized. Performing another level of pooling on the generated RoIs helps to improve recall for our proposals. As can be seen in Table 5.1 and Fig. 5.4a, this refinement step helps to improve precision and recall. We also generate anchors with different strides - 16 and 32 pixels - and show how the final detection performance improves as we iteratively refine proposals.



(a)



(b)

Figure 5.4: Ablation analysis: improving precision at inference time. *FA-RPN-32-32* represents a model which is trained by increasing the stride between anchors to 32 and uses the same stride at inference time. (a) *FA-RPN-32-Iter* is the same model when an additional anchor refinement step is performed at *inference*. (b) *FA-RPN-32-Dense* on the other hand, improves precision by reducing the anchor stride at inference time to our original FA-RPN stride.

5.6.2 Modifying Anchors and Strides during Inference

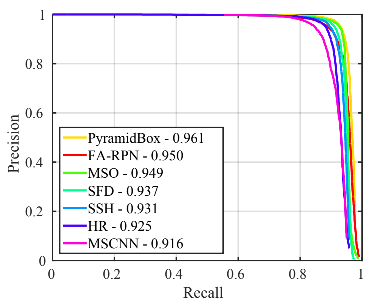
In this section, we show the flexibility of FA-RPN for generating region proposals. We train our network with a stride of 32 pixels and during inference, we generate anchors at a stride of 16 pixels on the WIDER dataset. The result is shown in the right-hand side plot in Fig. 5.4. We notice that the dense anchors improve performance by 3.8%. On the left side of the plot, we show the effect of iterative refinement of FA-RPN proposals. This further provides a boost of 1.4% on top of the denser anchors. This shows that our network is robust to changes in the anchor configuration, and can detect faces even on anchor sizes which were not provided during training. To achieve this with RPN, one would need to re-train it again, while in FA-RPN it is a simple inference time hyper-parameter which can be tuned on a validation set even after the training phase.

5.6.3 Effect of Scale and COCO pre-training on Face Detection

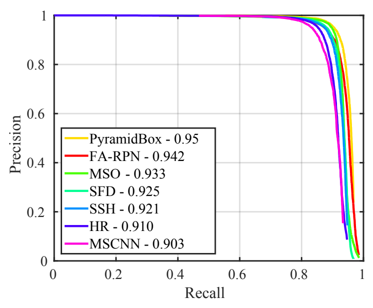
Variation of scale is among the main challenges in detection datasets. Datasets like WIDER consist of many small faces which can be hard to detect for a CNN at the original image scale. Therefore, upsampling images is crucial to obtaining good performance. However, as shown in [8], when we upsample images, large objects become hard to classify and when we downsample images to detect large objects, small objects become harder to classify. Therefore, standard multi-scale training is not effective when using extreme resolutions. In Table 5.1 we show the effect of performing SNIP based multi-scale training in our FA-RPN based Faster-RCNN detector. When performing inference on the same resolutions, we observe an improvement in detection performance on the WIDER dataset by 1%. Note that this improvement is on top of multi-scale inference. We also initialized our ResNet-50 model which was pre-trained on the COCO detection dataset. We show that even pre-training on object detection helps in improving the performance of face detectors by a significant amount, Table 5.1.

5.6.4 Comparison on the WIDER Dataset

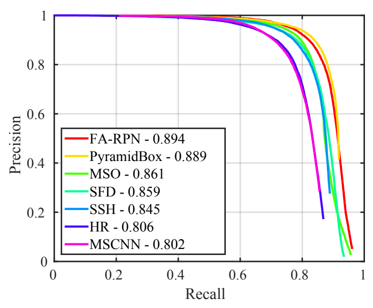
We compare our method with MSCNN [37], HR [100], SSH [31], S3FD [124], MSO [126], and PyramidBox [133] which are the published state-of-the-art methods on the WIDER dataset. Our simple detector outperforms other methods on the "hard" set, which includes *all* the annotations in the WIDER dataset while achieving an average precision of 89.4%. We also perform well in the "easy" and "medium" subsets. The precision-recall plots for each of these cases are shown in Fig. 5.5. Note that we did not



(a) Easy



(b) Medium



(c) Hard (whole dataset)

Figure 5.5: We compare with recently published methods on the WIDER dataset. The plots are for "easy", "medium" and "hard" respectively from left to right. As can be seen, FA-RPN outperforms published baselines on this dataset. Note that, "hard" set contains the whole dataset while "easy" and "medium" are subsets.

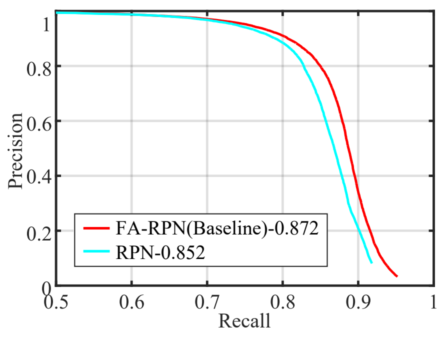


Figure 5.6: Comparison with RPN on the WIDER dataset.

use feature-pyramids or lower layer features from `Conv2` and `Conv3` [31, 124, 100] , enhancing predictions with context [100] or with deeper networks like ResNext-152 [27]/ Xception [28] for obtaining these results. We also compare FA-RPN (baseline version in Table 5.1) with RPN quantitatively and qualitatively in Fig. 5.6 and Fig. 5.7 respectively. These results demonstrate that FA-RPN is competitive with existing proposal techniques as it can lead to a state-of-the-art face detector. We also do not use recently proposed techniques like stochastic face lifting [126], having different filters for different size objects [100] or maxout background loss [124]. Our performance can be further improved if the above mentioned architectural changes are made to our network or better training methods which also fine-tune batch-normalization statistics are used [16, 55].

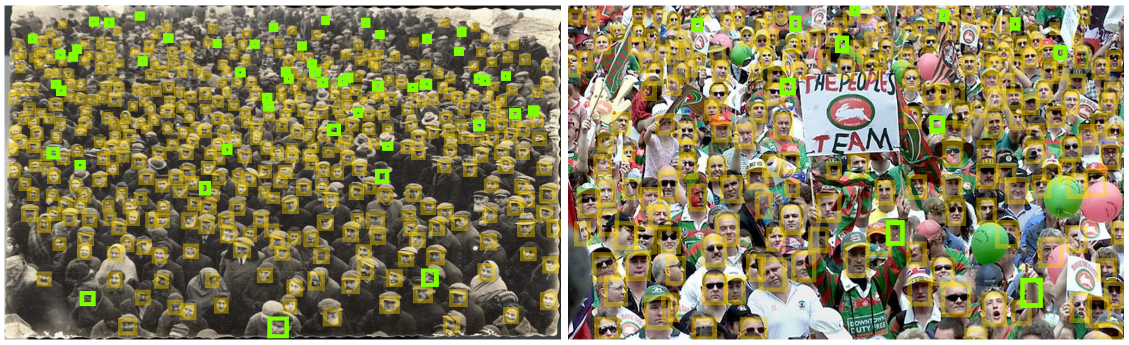


Figure 5.7: Qualitative comparison between RPN and FA-RPN (Baseline). Gold rectangles are those detected by both, greens are detected by FA-RPN but missed by RPN.

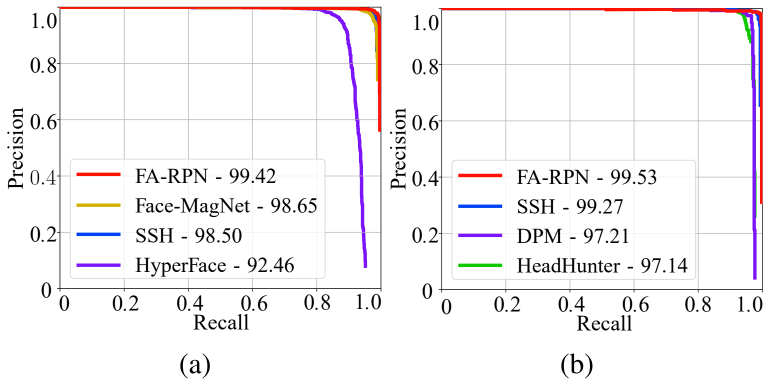


Figure 5.8: Comparison with other methods on (a) Pascal Faces, and (b) AFW datasets.

5.6.5 Pascal Faces and AFW Datasets

To show the generalization of our trained detector, we also apply it out-of-the-box to the Pascal Faces [117] and AFW [103] datasets without fine-tuning. The performance of FA-RPN is compared with SSH [31], Face-Magnet [136], HyperFace [123], HeadHunter [106], and DPM [122] detectors which reported results on these datasets. The results are shown in Fig. 5.8. Compared to WIDER, the resolution of PASCAL images is lower and they do not contain many small images, so it is sufficient to apply FA-RPN to the two lower resolutions in the pyramid. This also leads to faster inference. As can be seen, FA-RPN out-of-the-box generalizes well to these datasets. FA-RPN achieves state-of-the-art result on the PascalFaces and reduces the error rate to 0.68% on this dataset.

5.6.6 Efficiency

Our FA-RPN based detector is efficient and takes less than 0.05 seconds on a 1080Ti GPU to perform inference on an image of size 800×1280. With advances in GPUs over the last few years, performing inference even at very high resolutions is efficient. Our detector takes less than 0.4 seconds to process an image of size 1800×2800 on a 1080Ti GPU. With improved GPU architectures and the use of lower precision like 16 or 8 bits, the speed can be further improved by two to four times (depending on the precision used in inference). As a comparison, the original implementation of SSH [1] takes 0.45s on a Titax X GPU (ours takes 0.41s on the same machine) to process a 1800×2800 pixels image. It should be noted that in high resolutions, the runtime is dominated by convolutional layers and the small difference may be because *e.g.* SSH uses a custom architecture with feature pyramids, we use a standard ResNet50 backbone, SSH is in Caffe, ours is in MxNet,*etc*. SSH has a better runtime at low-resolutions (*e.g.* on the 512x600 resolution, SSH takes 0.05 and FA-RPN takes 0.07 seconds). However, the runtime of current state-of-the-art methods largely depends on the high-resolution scale. The multi-scale inference used in FA-RPN can be further accelerated with AutoFocus [137].

5.6.7 Qualitative Results

Figure 5.9 shows qualitative results on the WIDER validation subset. We picked 20 diverse images to highlight the results generated by FA-RPN. Detections are shown

[1] `http://www.github.com/mahyarnajibi/SSH`

by green rectangles and the brightness encodes the confidence. As can be seen, our face detector works very well in crowded scenes and can find hundreds of small faces in a wide variety of images. This shows that FA-RPN has a high recall and can detect faces accurately. It generalizes well in both indoor and outdoor scenes and under different lighting conditions. Our performance across a wide range of scales is also good without using diverse features from different layers of the network. It is also robust to changes in pose, occlusion, blur, and even works on old photographs.

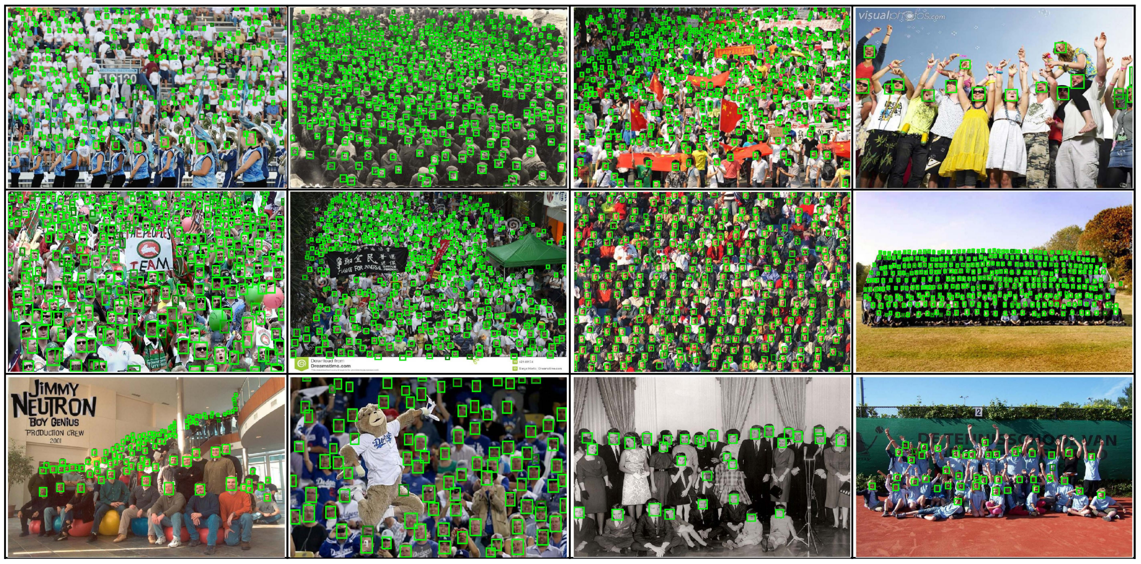


Figure 5.9: Qualitative results on the validation set of the WIDER dataset. Green rectangles show the detection and brightness encodes the detection confidence.

5.7 Conclusion

We introduced FA-RPN, a novel method for generating pooling-based proposals for face detection. We proposed techniques for anchor placement and label assignment which were essential in the design of such pooling-based proposal algorithm. FA-RPN has several benefits like efficient iterative refinement, flexibility in selecting scale and anchor stride during inference, sub-pixel anchor placement *etc*. Using FA-RPN, we obtained state-of-the-art results on the challenging WIDER dataset, showing the effectiveness of FA-RPN for this task. FA-RPN also achieved state-of-the-art results out-of-the-box on datasets like PascalFaces showing its generalizability.

Chapter 6: G-CNN: an Iterative Grid-based Object Detector

6.1 Introduction

Object detection, *i.e.* the problem of finding the locations of objects and determining their categories, is an intrinsically more challenging problem than classification since it includes the problem of object localization. The recent and popular trend in object detection uses a pre-processing step to find a candidate set of bounding-boxes that are likely to encompass the objects in the image. This step is referred to as the bounding-box proposal stage. The proposal techniques are a major computational bottleneck in state-of-the-art object detectors [11]. There have been attempts [77, 138] to take this pre-processing stage out of the loop but they lead to performance degradations.

We show that without object proposals, we can achieve detection rates similar to state-of-the-art performance in object detection. Inspired by the iterative optimization in [139], we introduce an iterative algorithm that starts with a regularly sampled multi-scale grid of boxes in an image and updates the boxes to cover and classify objects. One step regression can-not handle the non-linearity of the mapping from a regular grid to boxes containing objects. Instead, we introduce a piecewise regression model that can learn this non-linear mapping through a few iterations. Each step in our algorithm deals with an easier regression problem than enforcing a direct mapping to actual target locations.

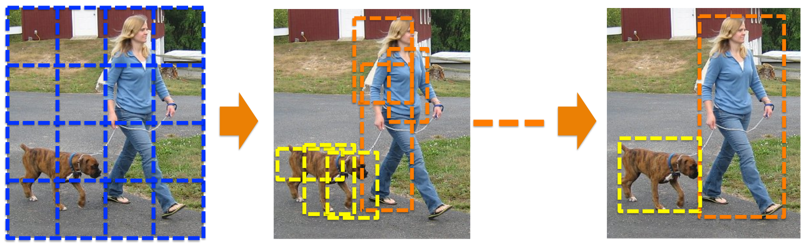


Figure 6.1: This figure shows a schematic illustration of our iterative algorithm ”G-CNN”. It starts with a multi-scale regular grid over the image and iteratively updates the boxes in the grid. Each iteration pushes the boxes toward the objects of interest in the image while classifying their category.

Figure 6.1 depicts an overview of our algorithm. Initially, a multi-scale regular grid is superimposed on the image. For visualization we show a grid of non-overlapping, but in actuality the boxes do overlap. During training, each box is assigned to a ground-truth object by an assignment function based on intersection over union with respect to the ground truth boxes. Subsequently, at each training step, we regress boxes in the grid to move themselves towards the objects in the image to which they are assigned. At test time, for each box at each iteration, we obtain confidence scores over all categories and update its location with the regressor trained for the currently most probable class.

Our experimental results show that G-CNN achieves the state-of-the-art results obtained by Fast-RCNN on PASCAL VOC datasets without computing bounding-box proposals. Our method is about $5X$ faster than Fast R-CNN for detection.

6.2 Related Work

Prior to CNN: For many years the problem of object detection was approached by techniques involving sliding window and classification [140, 141]. Lampert *et al.* [142] proposed an algorithm that goes beyond sliding windows and was guaranteed to reach the global optimal bounding box for an SVM-based classifier. Implicit Shape Models [143, 144] eliminated sliding window search by relying on key-parts of an image to vote for a consistent bounding box that covers an object of interest. Deformable Part-based Models [145] employed an idea similar to Implicit Shape Models, but proposed a direct optimization via latent variable models and used dynamic programming for fast inference. Several extension of DPMs emerged [146, 147] until the remarkable improvements due to the convolutional neural networks was shown by [9].

CNN age: Deep convolutional neural networks (CNNs) are the state-of-the-art image classifiers and successful methods have been proposed based on these networks [1]. Driven by their success in image classification, Girshick *et al*. proposed a multi-stage object detection system, known as R-CNN [9], which has attracted great attention due to its success on standard object detection datasets.

To address the localization problem, R-CNN relies on advances in object proposal techniques. Recently, proposal algorithms have been developed which avoid exhaustive search of image locations [2, 13]. R-CNN uses these techniques to find bounding boxes which include an object with high probability. Next, a standard CNN is applied as feature extractor to each proposed bounding box and finally a classifier decides which object class is inside the box.

The main drawback of R-CNN is the redundancy in computing the features. Generally, around 2K proposals are generated; for each of them, the CNN is applied independently to extract features. To alleviate this problem, in SPP-Net [12] the convolutional layers of the network are applied only once for each image. Then, the features of each region of interest are constructed by pooling the global features which lie in the spatial support of the region. However, learning is limited to fine-tuning the weights of fully connected layers. This drawback is addressed in Fast-RCNN [11] in which all parameters are learned by back propagating the errors through the augmented pooling layer and packing all stages of the system, except generation of the object proposals, into one network.

The generation of object proposals, in CNN-based detection systems has been regarded as crucial. However, after proposing Fast-RCNN, this stage became the bottleneck. To make the number of object proposals smaller, Multibox[148] introduced a proposal algorithm that outputs 800 bounding boxes using a CNN. This increases the size of the final layer of the CNN to 4096x800x5 and introduces a large set of additional parameters. Recently, Faster-RCNN [5] was proposed, which decreased the number of parameters; however it needs to start from thousands of anchor points to propose 300 boxes.

In addition to classification, using a regressor for object detection has been also studied previously. Before proposing R-CNN, Szegedy *et al*. [149], modeled object detection as a regression problem and proposed a CNN-based regression system. More recently, AttentionNet [150] is a single category detection that detects a single object inside an image using iterative regression. For multiple objects, the model is applied as a proposal algorithm to generate thousands of proposals and then is re-applied iteratively on each proposal for single category detection, which makes detection inefficient.

Although R-CNN and its variants attack the problem using a classification approach, they employ regression as a post-processing stage to refine the localization of the proposed bounding boxes.

The importance of the regression stage has not received as much attention as improving the object proposal stage for more accurate localization. The necessity of an object proposal algorithm in CNN based object detection systems has recently been challenged by Lenc *et al*. [138]. Here, the proposals are replaced by a fixed set of bounding boxes. A set with a distribution derived from an object proposal method is selected using a clustering technique. However, for achieving comparable results, even more boxes need to be used compared to R-CNN. Another recent attempt for removing the proposal stage is Redmon *et al*. [77] which conducts object detection in a single shot. However, the considerable gap between the best detection accuracy of these systems and systems with an explicit proposal stage suggests that the identification of good object proposals is critical to the success of these CNN based detection systems.

6.3 G-CNN Object Detector

6.3.1 Network structure

G-CNN trains a CNN to move and scale a fixed multi-scale grid of bounding boxes towards objects. The network architecture for this regressor is shown in Figure 6.2. The backbone of this architecture can be any CNN network (*e.g*. AlexNet [1], VGG [102], *etc*.). As in Fast R-CNN and SPP-Net, a spatial region of interest (ROI) pooling layer is included in the architecture after the convolutional layers. Given the location information of each box, this layer computes the feature for the box by pooling the global features that lie inside the ROI. After the fully connected layers, the network ends with a linear regressor which outputs the change in the location and scale of each current bounding box, conditioned on the assumption that the box is moving towards an object of a class.

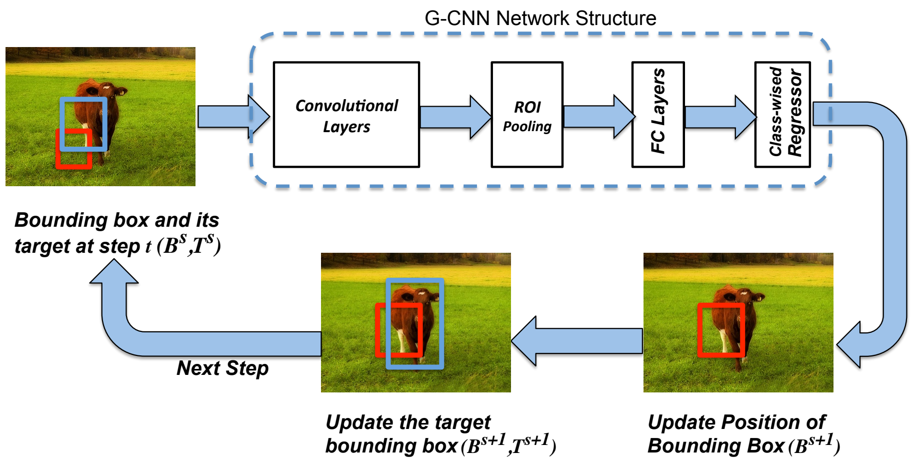

G-CNN Network Structure
Convolutional Layers
ROI Pooling
FC Layers
Class-wised Regressor
Bounding box and its target at step t (B^S, T^S)
Next Step
Update the target bounding box (B^{s+1}, T^{s+1})
Update Position of Bounding Box (B^{s+1})

Figure 6.2: Structure of G-CNN regression network as well as an illustration of the idea behind the iterative training approach. The bounding box at each step is shown by the blue rectangle and its target is represented by a red rectangle. The network is trained to learn the path from the initial bounding box to its assigned target iteratively.

6.3.2 Training the network

Despite the similarities between the Fast R-CNN and G-CNN architectures, the training goals and approaches are different. G-CNN defines the problem of object detection as an iterative search in the space of all possible bounding boxes. G-CNN starts from a fixed multi-scale spatial pyramid of boxes. The goal of learning is to train the network so that it can move this set of initial boxes towards the objects inside the image in S steps iteratively. This iterative behaviour is essential for the success of the algorithm. The reason is the highly non-linear search space of the problem. In other words, although learning how to linearly regress boxes to far away targets is unrealistic, learning small changes in the search space is tractable. Section 6.4.3 shows the importance of this step-wise training approach.

6.3.2.1 Loss function

G-CNN is an iterative method that moves bounding boxes towards object locations in S_{train} steps. For this reason, the loss function is defined not only over the training samples but also over the iterative steps.

More formally, let $\mathcal{B}$ represent the four-dimensional space of all possible bounding boxes represented by the coordinates of their center, their width, and height. $\mathbf{B}_i \in \mathcal{B}$ is the i'th training bounding box. We use the superscript $1 \leq s \leq S_{train}$ to denote the variables in step 's' of the G-CNN training, *i.e.* $\mathbf{B}_i^s$ is the position of the i'th training bounding box in step s.

During training, each bounding box with an IoU higher than a small threshold (0.2) is assigned to one of the ground truth bounding boxes inside its image. The following many-to-one function, $\mathcal{A}$, is used for this assignment.

$$\mathcal{A}(\mathbf{B}_i^s) = arg \max_{\mathbf{G} \in \mathcal{G}_i} IoU(\mathbf{B}_i^1, \mathbf{G}) \tag{6.1}$$

where $\mathcal{G}_i = \{\mathbf{G}_{i1} \in \mathcal{B}, \ldots, \mathbf{G}_{in} \in \mathcal{B}\}$, is the set of ground truth bounding boxes which lie in the same image as $\mathbf{B}_i$. IoU is the intersection over union measure. Note that $\mathbf{B}_i^1$ represents the position of the i'th bounding box in the initial grid. In other words, for each training bounding box, the assignment is done in the initial training step and is not changed during the training.

Since regressing the initial training bounding boxes to their assigned ground truth bounding box can be highly non-linear, we tackle the problem with a piece-wise regression approach. At step s, we solve the problem of regressing $\mathbf{B}_i^s$ to a target bounding box on the path from $\mathbf{B}_i^s$ to its assigned ground truth box. The target bounding box is moved step by step towards the assigned bounding box until it coincides with the assigned ground truth in step S_{train}. The following function is used for defining the target bounding boxes at each step:

$$\Phi(\mathbf{B}_i^s, \mathbf{G}_i^*, s) = \mathbf{B}_i^s + \frac{\mathbf{G}_i^* - \mathbf{B}_i^s}{S_{train} - s + 1} \tag{6.2}$$

where $\mathbf{G}_i^* = \mathcal{A}(\mathbf{B}_i^s)$ represents the assigned ground truth bounding box to $\mathbf{B}_i^s$. That is, at each step, the path from the current representation of the bounding box to the assigned ground truth is divided by the number of remaining steps and the target is defined to be one unit away from the current location.

G-CNN regression network outputs four values for each class, representing the parameterized change for regressing the bounding boxes assigned to that class. Following [9], a log-scale shift in width and height and a scale invariant translation is used to parametrize the relative change for mapping a bounding box to its assigned target. This parametrization is denoted by $\Delta(\mathbf{B}_i^s, \mathbf{T}_i^s)$, where $\mathbf{T}_i^s$ is the assigned target to $\mathbf{B}_i^s$ computed by 6.2.

So the loss function for G-CNN is defined as follows:

$$\begin{aligned} L(\{\mathbf{B}_i\}_{i=1}^N) = \sum_{s=1}^{S_{train}} \sum_{i=1}^{N} \big[I(\mathbf{B}_i^1 \notin \mathcal{B}_{BG}) \times \\ L_{reg}(\delta_{i,l_i}^s - \Delta(\mathbf{B}_i^s, \Phi(\mathbf{B}_i^s, \mathcal{A}(\mathbf{B}_i^s), s)))\big] \end{aligned} \tag{6.3}$$

where δ_{i,l_i}^s is the four-dimensional parameterized output for class l_i representing the relative change in the representation of bounding box $\mathbf{B}_i^s$. l_i is the class label of the assigned ground truth bounding box to $\mathbf{B}_i$. L_{reg} is the regression loss function. The smooth l_1 loss is used as defined in [11]. $I(.)$ is the indicator function which outputs one when its condition is satisfied and zero otherwise. $\mathcal{B}_{BG}$ represents the set of all background bounding boxes.

During training, the representation of bounding box $\mathbf{B}_i$ at step s, $\mathbf{B}_i^s$, can be determined based on the actual output of the network by the following update formula:

$$\mathbf{B}_i^s = \mathbf{B}_i^{s-1} + \Delta^{-1}(\delta_{i,l_i}^{s-1}) \tag{6.4}$$

where Δ^{-1} projects back the relative change in the position and scale from the defined parametrized space into $\mathcal{B}$. However for calculating 6.4, the forward path of the network needs to be evaluated during training, making training inefficient. Instead, we use an approximate update by assuming that in step s, the network could learn the regressor for step $s-1$ perfectly. As a result the update formula becomes $\mathbf{B}_i^s = \Phi(\mathbf{B}_i^{s-1}, \mathbf{G}_i^*, s-1)$. This update is depicted in Figure 6.2.

6.3.2.2 Optimization

G-CNN optimizes the objective function in 6.3 with stochastic gradient descent. Since G-CNN tries to map the bounding boxes to their assigned ground-truth boxes in S_{train} steps, we use a step-wised learning algorithm that optimizes Eq. 6.3 step by step.

To this end, we treat each of the bounding boxes in the initial grid together with its target in each of the steps as an independent training sample *i.e.* for each of the bounding boxes we have S_{train} different training pairs. The algorithm first tries to optimize the loss function for the first step using N_{iter} iterations. Then the training pairs of the second step are added to the training set and training continues step by step. By keeping the samples of the previous steps in the training set, we make sure that the network does not forget what was learned in the previous steps.

The samples for the earlier steps are part of the training set for a longer period of time. This choice is made since the earlier steps determine the global search direction and have a greater impact on the chance that the network will find the objects. On the other hand, the later steps only refine the bounding boxes to decrease localization error. Given that the search direction was correct and a good part of the object is now visible in the bounding box, the later steps solve a relatively easier problem.

Algorithm 2 is the method for generating training samples from each bounding box during each G-CNN step.

6.3.3 Fine-tuning

All models are fine-tuned from pre-trained models on ImageNet. Following [11], we fine-tune all layers except early convolutional layers (*i.e.* *conv2* and up for AlexNet and *conv3_1* and up for VGG16). During training, mini-batches of two images are used. At each step of G-CNN, 64 training samples are selected randomly from all possible samples of the image at the corresponding step.

Algorithm 2: G-CNN Training Algorithm

```
1  for $i = 1$ to $S_{train}$ do
2      TrainTuples $\leftarrow \{\}$ ;
3      for $s = 1$ to $c$ do
4          if $s = 1$ then
5              $\mathbf{B}^1 \leftarrow$ Spatial pyramid grid of boxes ;
6              $\mathbf{G}^* \leftarrow \mathcal{A}(\mathbf{B}^1)$ ;
7          end
8          else
9              $\mathbf{B}^s \leftarrow \mathbf{T}^{s-1}$ ;
10         end
11         $\mathbf{T}^s \leftarrow \Phi(\mathbf{B}^s, \mathbf{G}^*, s)$ ;
12         $\Delta^s \leftarrow \Delta(\mathbf{B}^s, \mathbf{T}^s)$ ;
13         Add $(\mathbf{B}^s, \Delta^s)$ to TrainTuples ;
14     end
15     Train G-CNN $N_{iter}$ iterations with TrainTuples ;
16 end
```

6.3.4 G-CNN Test Network

The G-CNN regression network is trained to detect objects in an iterative fashion from a set of fixed bounding boxes in a multi-scale spatial grid. Likewise at test time, the set of bounding boxes is initialized to boxes inside a spatial pyramid grid. The regressor moves boxes towards objects using the classifier score to determine which class regressor to apply to update the box. The detection algorithm is presented in Algorithm 3.

During the detection phase, G-CNN is run S_{test} times. However, like SPP-Net and Fast R-CNN there is no need to compute activations for all layers at every iteration. During test time, we decompose the network into global and regression parts as depicted in Figure. 6.3. The global net contains all convolutional layers of the network. On the other hand, the regression part consists of the fully connected layers and the regression weights. The input to the global net is the image and the forward path is computed only

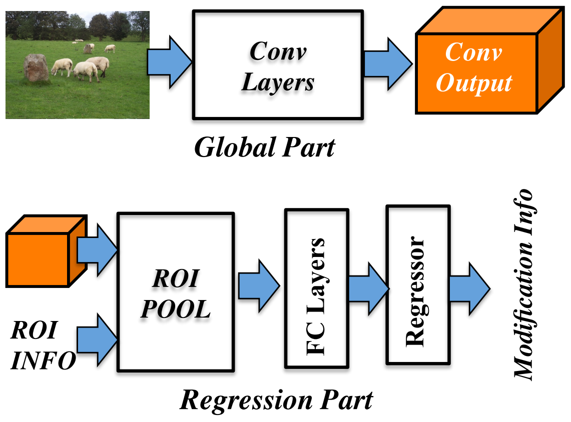


Figure 6.3: Decomposition of the G-CNN network into global (upper) and regression part (lower) for detection after the training phase. Global part is run only once to extract global features but regression part is run at every iteration. This leads to a considerable speed up at test time.

once for each image, outside the detection loop of Algorithm 3. Inside the detection loop, we only operate the regression network, which takes the outputs of the last layer of the global net as input and produces the bounding box modifications.

Algorithm 3: Detection algorithm

```
Input  : The feed-forward G-CNN regression network $f(.)$, The classifier
         function $c(.)$
Output: Final Detections $\mathbf{B}^{S_{test}+1}$
1 $\mathbf{B}^1 \leftarrow$ Spatial pyramid grid of boxes ;
2 for $s = 1$ to $S_{test}$ do
3     $l \leftarrow c(\mathbf{B}^s)$ ;
4     $\delta_l^s \leftarrow f(\mathbf{B}^s)$ ;
5     $\mathbf{B}^{s+1} \leftarrow \mathbf{B}^s + \Delta^{-1}(\delta_l^s)$ ;
6 end
7 return $\mathbf{B}^{S_{test}+1}$ ;
```

This makes the computational cost of the algorithm comparable to Fast R-CNN (without considering the object proposal stage of Fast R-CNN). The global net is called once in both Fast R-CNN and G-CNN. Afterward, Fast R-CNN does $N_{proposal}$ forward calculations of the regression network (where $N_{proposal}$ is the number of generated object proposals for each image). G-CNN, on the other hand, does this forward calculation $S_{test} \times N_{grid}$ times (where N_{grid} is the number of bounding boxes in the initial grid). In section 6.4.2, we show that for $S_{test} = 5$ and $N_{grid} \sim 180$, G-CNN achieves comparable results to Fast R-CNN which uses $N_{proposal} \sim 2K$ object proposals.

6.4 Experiments

6.4.1 Experimental Setup

We report results on the Pascal VOC 2007 and Pascal VOC 2012 datasets. The performance of G-CNN is evaluated with AlexNet [1] as a small and VGG16 [102] as a very deep CNN structure. Following [9], we scale the shortest side of the image to 600 pixels not allowing the longer side of the image to be more than 1000 pixels. However, we always maintain the aspect ratio of the image, so the shortest side might include fewer than 600 pixels. Each model is pre-trained with weights learned from the imagenet dataset.

In all the experiments, the G-CNN regression network is trained on an initial overlapping spatial pyramid with [2,5,10] scales (*i.e.* the bounding boxes in the coarsest level are $(im_{width}/2, im_{height}/2)$ pixels *etc.*). During training, we used [0.9,0.8,0.7] overlap for each spatial scale respectively. By overlap of α we mean that the horizontal and vertical strides are $width_{cell} * (1 - \alpha)$ and $height_{cell} * (1 - \alpha)$ respectively. However, during test time, as will be shown in the following sections, overlaps of [0.7,0.5,0] (non-overlapping grid at the finest scale) is sufficient to obtain results comparable to Fast R-CNN. This leads to a grid of almost 180 initial boxes at test time. The G-CNN regression network is trained for $S = 3$ iterative steps. According to our experiments, no substantial improvement is achieved by training the network for a larger number of steps.

6.4.2 Results on VOC datasets

The goal of G-CNN is to replace object proposals with a fixed multi-scale grid of boxes. To evaluate this, we fix the classifier in Algorithm 3 to the Fast R-CNN classifier and compare our results to the original Fast R-CNN with selective search proposal algorithm.

6.4.2.1 VOC 2007 dataset

Table 6.1 compares the mAP between G-CNN and Fast R-CNN on the VOC2007 *test* set. AlexNet is used as the basic CNN for all methods and models are trained on VOC2007 *trainval* set. G-CNN(3) is our method with three iterative steps during test time. In this version, we used the same grid overlaps used during training. This leads to a set of around 1500 initial boxes. G-CNN(5) is our method when we increase the number of steps at test time to 5 but reduce the overlaps to [0.7,0.5,0] (see 6.4.1). This leads to around 180 boxes per image. According to the result, 180 boxes is enough for G-CNN to surpass the performance of Fast R-CNN, which uses around 2K selective search proposed boxes. In the remainder of this section, we use G-CNN to refer to the G-CNN(5) version of our method.

Table 6.2 shows mAP for various methods trained on VOC2007 trainval set and tested on VOC2007 test set. All methods used VGG16. The results validate our claim that G-CNN effectively moves its relatively small set of boxes toward objects. In other words, there seems to be no advantage to employing the larger set of selective search proposed boxes for detection in this dataset.

VOC 2007	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
FR-CNN [11]	**66.4**	**71.6**	**53.8**	43.3	24.7	69.2	**69.7**	**71.5**	31.1	**63.4**	59.8	62.2	73.1	65.9	**57**	26	**52**	56.4	67.8	**57.7**	57.1
G-CNN(3) [ours]	63.2	68.9	51.7	41.8	**27.2**	69.1	67.7	69.2	31.8	60.6	**60.8**	63.9	**75.5**	67.3	54.9	26.1	51.2	**57.2**	69.6	56.8	56.7
G-CNN(5) [ours]	65	68.5	52	**44.9**	24.5	**69.3**	69.6	68.9	**34.6**	60.3	58.1	**64.6**	75.1	**70.5**	55.2	**28.5**	50.7	56.8	**70.2**	56.1	**57.2**

Table 6.1: Average Precision on VOC 2007 test data. Both Fast R-CNN and our methods use AlexNet CNN structure. Models are trained using VOC 2007 trainval set.

VOC 2007	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
SPPnet BB[12]	73.9	72.3	62.5	51.5	44.4	74.4	73.0	74.4	42.3	73.6	57.7	70.3	74.6	74.3	54.2	34.0	56.4	56.4	67.9	**73.5**	63.1
R-CNN BB[151]	73.4	77.0	63.4	45.4	**44.6**	75.1	78.1	79.8	40.5	**73.7**	62.2	79.4	78.1	73.1	64.2	**35.6**	**66.8**	67.2	70.4	71.1	66.0
FR-CNN[11]	**74.5**	**78.3**	**69.2**	**53.2**	36.6	77.3	78.2	**82.0**	40.7	72.7	**67.9**	**79.6**	79.2	73.0	**69.0**	30.1	65.4	**70.2**	**75.8**	65.8	**66.9**
G-CNN[ours]	68.3	77.3	68.5	52.4	38.6	**78.5**	**79.5**	81	**47.1**	73.6	64.5	77.2	**80.5**	**75.8**	66.6	34.3	65.2	64.4	75.6	66.4	66.8

Table 6.2: Average Precision on VOC 2007 Test data. All reported methods used VGG16. Models are trained using VOC 2007 trainval set.

6.4.2.2 VOC 2012 dataset

The mAP for VOC2012 dataset is reported in Table 6.3. All methods use VGG16 as their backbone. Methods are trained on trainval set and tested on the VOC2012 test set. The results of our method are obtained using the ”comp4” evaluation server with the parameters mentioned in 6.4.1 and the results of other methods are obtained from their papers.

VOC 2012	train	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
R-CNN BB[151]	12	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3	62.4
YOLO[77]	12	71.5	64.2	54.1	35.3	23.3	61.0	54.4	78.1	35.3	56.9	40.9	72.4	68.6	68.0	62.5	26.0	51.9	48.8	68.7	47.2	54.5
FR-CNN[11]	12	80.3	74.7	66.9	46.9	37.7	73.9	68.6	87.7	41.7	71.1	51.1	86.0	77.8	79.8	69.8	32.1	65.5	63.8	76.4	61.7	65.7
FR-CNN[11]	07++12	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2	68.4
G-CNN [ours]	12	82	74	68.2	49.5	38.9	74.4	68.9	85.4	40.6	70.9	50	85.5	77	77.4	67.9	33.7	67.6	60	77.6	60.8	65.5
G-CNN [ours]	07+12	82	76.1	69.3	49.9	40.1	75.2	69.5	86.3	42.3	72.3	50.8	84.7	77.8	77.2	68	38.1	68.4	59.8	79.1	61.9	66.4*

Table 6.3: Average Precision on VOC2012 test data. All reported methods used VGG16. The training set for each image is mentioned in the second column (12 stands for VOC2012 trainval, 07+12 represents the union of the trainval of VOC2007 and VOC2012, and 07++12 is the union of VOC 2007 trainval, VOC 2007 test and VOC 2012 trainval. The * emphasises that our method is trained on fewer data compared to FR-CNN trained on 07++12 training data)

G-CNN obtains almost the same result as Fast R-CNN when both methods are trained on VOC 2012 trainval. Although in this table the best-reported mAP for Fast RCNN is slightly higher than G-CNN, it should be noted that unlike G-CNN, Fast R-CNN used the VOC 2007 test set as part of its training. It is worth noting that all methods except YOLO use proposal algorithms with high computational complexity. Compared to YOLO, which does not use object proposals, our method has a considerably higher mAP. To the best of our knowledge, this is the best-reported result among methods without an object proposal stage.

6.4.3 Stepwise training matters

G-CNN uses a stepwise training algorithm and defines its loss function with this goal. In this section, we investigate the question of how important this stepwise training is and whether it can be replaced by a simpler, single step training approach.

To this end, we compare G-CNN with two simpler iterative approaches in table 6.4. First we consider the iterative version of Fast R-CNN (*IF-RCNN*). In this method, we use the regressor trained with Fast R-CNN in our iterative framework. Clearly, this regressor was not designed for grid-based object detection, but for small post-refinement of proposed objects.

Also, we consider a simpler algorithm for training the regressor for a grid-based detection system. Specifically, we collect all training tuples created in different steps of G-CNN and train our regressor in one step on this training set. So the only difference between G-CNN and this method is stepwise training. We call this method *1Step-Grid*.

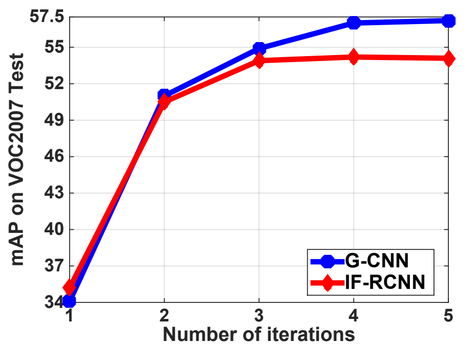


Figure 6.4: Mean average precision on VOC2007 test set vs. number of regression steps for G-CNN and IF-RCNN. Both methods use AlexNet and trained on VOC2007 trainval.

All methods are trained on VOC 2007 trainval set and tested on VOC 2007 test set and AlexNet is used as the core CNN structure. All methods are applied five iterations during test time to the same initial grid. Table 6.4 shows the comparison among the methods and Figure 6.4 compares IF-RCNN and G-CNN for different numbers of iterations.

VOC 2007	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
IF-RCNN	51.3	67.1	51.6	33.7	**26.2**	67.8	66.3	**70.3**	31.5	56.3	55.9	62.6	74.7	64.6	**55.6**	22.2	46.5	54.3	67.4	55	54.1
1Step-Grid	59.6	63.3	**52.4**	40.2	20.9	68.1	67.1	68.6	29.7	59.6	**62.1**	63	70.7	64	53.2	23.4	50.1	56	63.5	53.9	54.5
G-CNN [ours]	**65**	**68.5**	52	**44.9**	24.5	**69.3**	69.6	68.9	**34.6**	**60.3**	58.1	**64.6**	**75.1**	**70.5**	55.2	**28.5**	**50.7**	**56.8**	**70.2**	**56.1**	**57.2**

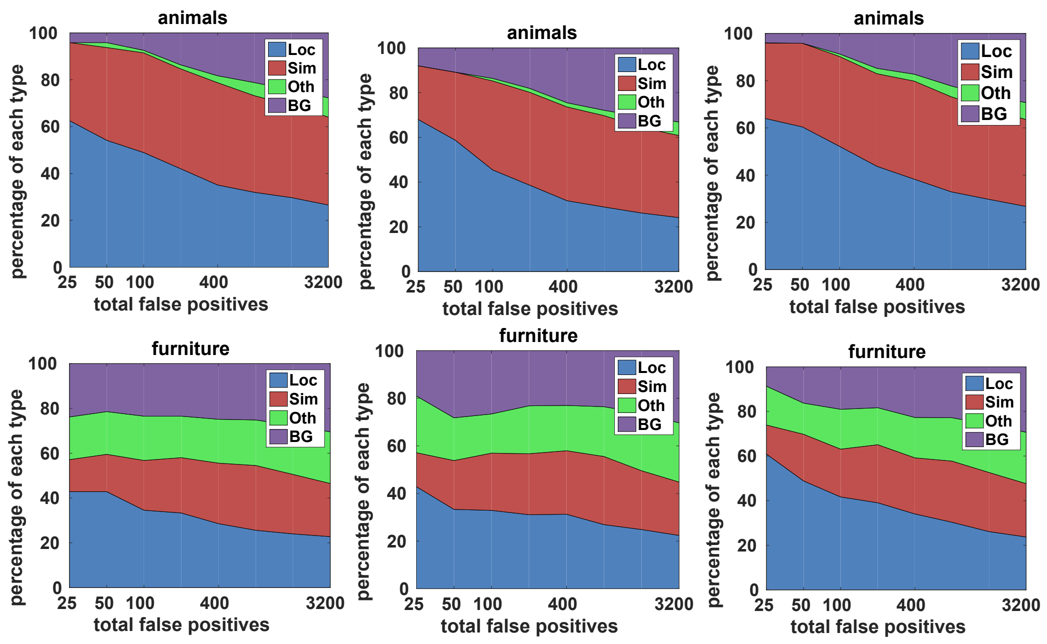
Table 6.4: Comparison among different strategies for grid-based object detection trained on VOC2007 trainval. All methods used AlexNet.

The results show that step-wise training is crucial to the success of G-CNN. Even though the training samples are the same for G-CNN and 1Step-Grid, G-CNN outperforms it by a considerable margin.

6.4.4 Analysis of the detection results

G-CNN removes the proposal stage from CNN-based object detection networks. Since the object proposal stage is known to be important for achieving good localization in CNN-based techniques, we compare the localization of G-CNN with Fast R-CNN.

To this end, we use the powerful tool of Hoeim *et al.* [152]. Figure 6.5 shows the



(a) G-CNN (b) Fast R-CNN (c) 1Step-Grid

Figure 6.5: The distribution of top-ranked types of false positives (FPs). FPs are categorized into four different subcategories. The diagram shows the change in the distribution of these types when more FPs with decreasing scores are considered. *Loc* represents those FPs caused by poor localization (a duplicate detection or detection with IoU between 0.1 and 0.5). *Sim* shows those coming from confusion with one of the similar classes. *BG* stands for FPs on background and *Oth* represents other sources.

distribution of top-ranked false positive rates for *G-CNN*, *Fast R-CNN* and the 1Step-Grid approach defined in the previous subsection. Comparing the distributions for G-CNN and Fast R-CNN, it is clear that removing the proposal stage from the system using our method did not hurt the localization and for the furniture class, it slightly improved the FPs due to localization error. Note that 1Step-Grid is trained on the same set of training tuples as G-CNN. However, the higher rate of false positives due to localization in 1Step-Grid is another indication of the importance of G-CNN's multi-step training strategy.

6.4.5 Qualitative results

Figure 6.6 shows some of the paths found by G-CNN in the space of bounding boxes starting from an initial grid with three scales. This example shows how G-CNN is capable of changing the position and scale of the boxes to fit them to different objects. The first four rows show successful examples while the last ones show failure examples.

6.4.6 Detection run time

Here we compare the detection time of our algorithm with Fast R-CNN. For both methods, we used the truncated SVD technique proposed in [11] and compressed fc6 and fc7 layers by keeping their top 1024 singular values and 256 singular values respectively. Timings are performed on a system with two K40 GPUs. The VGG16 network structure is used for both detection techniques and G-CNN uses the same classifier as Fast R-CNN.

We used Selective Search proposal to generate around 2K bounding boxes as suggested by [11]. This stage takes 1830 ms to complete on average (selective search algorithm is not implemented in GPU mode). Fast R-CNN itself takes 220 ms on average for detecting objects. This leads to a total detection time of 2050 ms/im.

On the other hand, G-CNN does not need any object proposal stage. However, it iterates S=5 times with a grid of around 180 boxes. The global part of the network (See 6.3.4) takes 188 ms for each image. Each iteration of the segmentation network takes 35 ms. The classification network can be run in parallel. This would lead to a detection time of 363 ms/im (around 3 fps) in total.

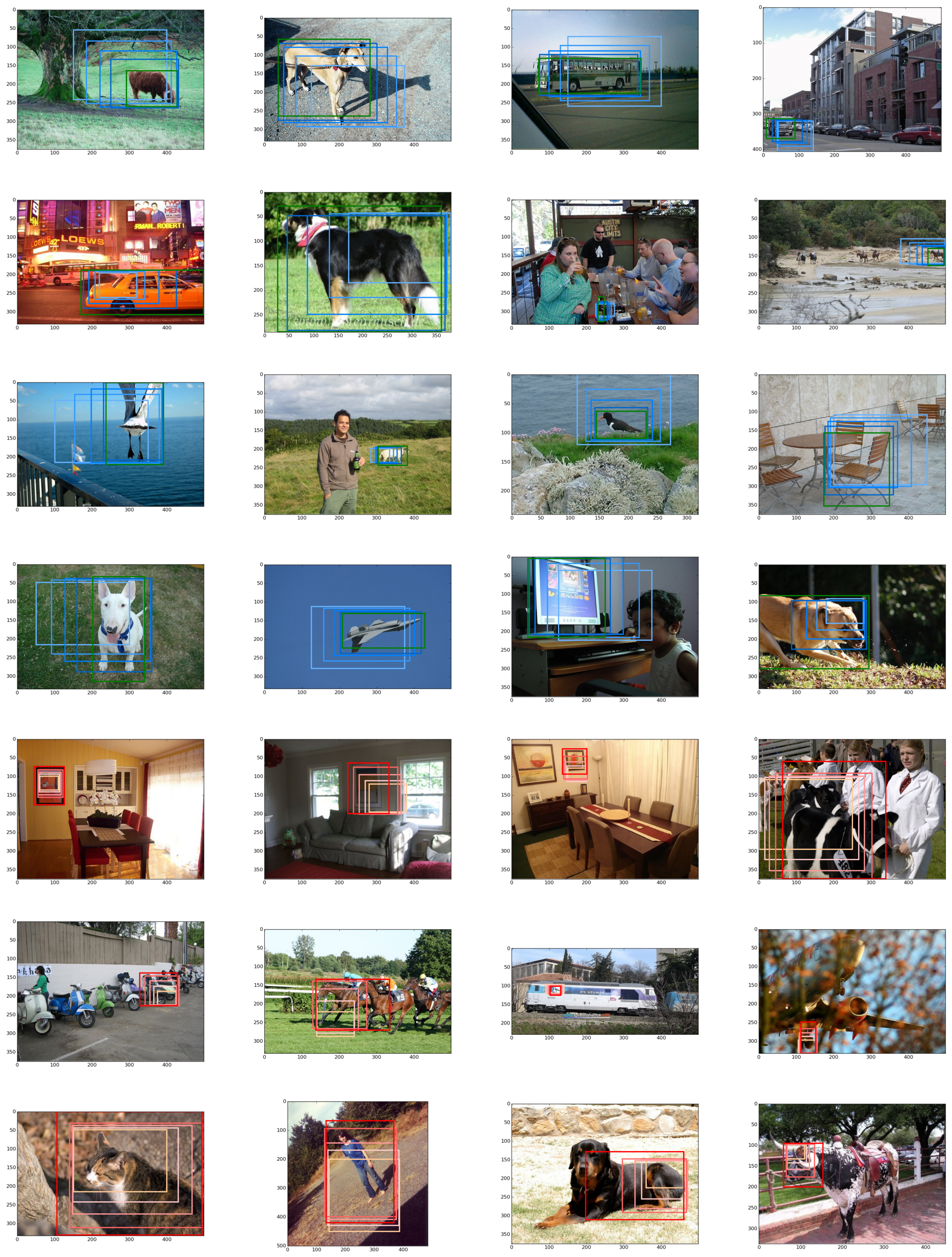


Figure 6.6: A sample of paths G-CNN found towards objects in the VOC2007 test set. The first four rows show some success examples while the last rows show some failure cases. The most common failures of G-CNN can be categorized into the following sub-categories: false firing of the classifier on similar objects (first three failure cases in the fifth row where G-CNN fits into picture frames instead of monitors); bad localization due to high overlapping similar objects (next three examples); false firing of the classifier on small boxes (last two cases in the sixth row); localization error due to hard object pose or small initial box compared to the actual size of the object (examples in the last row).

6.5 Conclusion

We proposed G-CNN, a CNN-based object detection technique which models the problem of object detection as an iterative search in the space of all possible bounding boxes. Our model starts from a grid of fixed boxes regardless of the image content and migrates them to objects in the image. Since this search problem is nonlinear, we proposed a piece-wise regression model that iteratively moves boxes towards objects step by step. We showed how to learn the CNN architecture in a stepwise manner. The main contribution of the proposed technique is removing the object proposal stage from the detection system, which is the current bottleneck for CNN-based detection systems. G-CNN is $5X$ faster than ”Fast R-CNN” and achieves comparable results to state-of-the-art detectors.

Chapter 7: Towards the Success Rate of One: Real-time Unconstrained Salient Object Detection

7.1 Introduction

Saliency detection is the problem of finding the most distinct regions from a visual scene. It attracts a great amount of attention due to its importance in object detection [153], image segmentation [154], image thumb-nailing [155], video summarization [156], *etc*.

Saliency detection has been studied under three different scenarios. Early works attempt to predict human eye-fixation over an image [79], while later works increasingly focus on salient foreground segmentation [157, 158, 159, 160, 161], *i.e.*, predicting a dense, pixel-level binary map to differentiate salient objects from background. However, it cannot separate overlapping salient objects and requires pixel-level annotations that are expensive to acquire for large datasets. Different from salient foreground segmentation, salient object detection aims to locate and draw bounding boxes around salient objects. It only requires bounding box annotations, which significantly reduces the effort for human labeling, and can easily separate overlapping objects. These advantages make the problem of salient object detection more valuable to investigate in terms of applicability to the real world.

With the re-emergence of convolutional neural networks (CNN), computer vision community has witnessed numerous breakthroughs, including salient object detection, thanks to the extraordinary discriminative ability of CNNs [162]. Prior to CNNs, Some works [163, 164, 165, 166] have proposed heuristics to detect single salient object in an image, while others [167, 168] rank a fixed-sized list of bounding boxes which might contain salient objects without determining the exact detections. However, most of these methods do not solve the *existence* problem, *i.e.*, determining whether any salient objects exist in an image at all, and simply rely on external binary classifiers to address this problem. Recently, saliency detection based on deep networks has achieved state-of-the-art performance. Zhang *et al.* [162] propose to use the MultiBox proposal network [148] to generate hundreds of candidate bounding boxes that are further ranked to output a compact set of salient objects. A probabilistic approach is proposed to filter and re-rank candidate boxes as a substitution for non-maxima suppression (NMS). To accurately localize salient objects, [162] requires a large number of class-agnostic proposals covering the whole image (see Figure 7.1). However, its precision and recall significantly drop if one only uses tens of boxes. The reason is that generic object proposals have very low success rate of locating an object, *i.e.*, only few of them tightly enclose the ground-truth objects, while most are redundant. Even though additional refinement steps are applied [162], there are still a lot of false positives (see Figure 7.1). The additional steps add more overhead and make this framework infeasible for real-time applications.

In this section, we address this problem by moving towards the success rate of one, *i.e.*, generating the exact number of boxes for salient objects without object proposals.

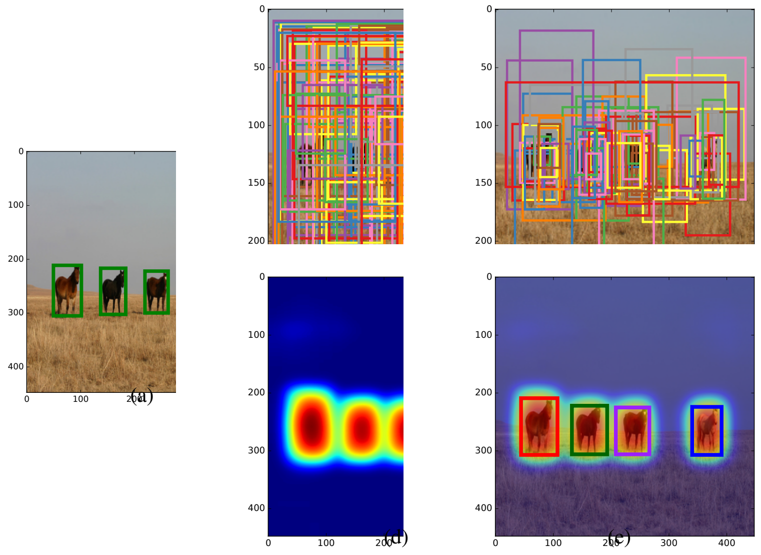


Figure 7.1: (a) Ground-truth image (b) Output of MultiBox [148] (c) Remaining MultiBox bounding boxes after pruning steps in [162] (d) Saliency map predicted by our method without pixel-level labeling (e) Bounding boxes generated by our method from (d) without proposals.

We present an end-to-end deep network for real-time salient object detection, dubbed as RSD. Rather than generating lots of candidate boxes and filtering them, our network directly predicts a saliency map with Gaussian distributions centered at salient objects, and infers bounding boxes from these distributions. Our network consists of two branches trained with multi-task loss to perform saliency map prediction, salient object detection and subitizing simultaneously, all in a single pass within a unified framework. Notably, our RSD with VGG16 achieves more than 100 fps on a single GPU during inference, significantly faster than existing CNN-based approaches. To the best of our knowledge, this is the first work on real-time non-redundant bounding box prediction for simultaneous salient object detection, saliency map estimation and subitizing, without object proposals. We also show the possibility of generating accurate saliency maps without pixel-level annotations, formulating it as a weakly-supervised approach that is more practical than fully-supervised approaches.

Our contributions are summarized as follows. First, we present a unified deep network performing salient object detection, saliency map prediction and subitizing simultaneously in a single pass. Second, our network is trained with Gaussian distributions centered at ground-truth salient objects that are considered to be more informative and discriminative than bounding boxes to distinguish multiple salient objects. Third, our approach outperforms state-of-the-art methods using object proposals by a large margin, and also produces comparable results on salient foreground segmentation datasets, even though we do not use any pixel-level annotations. Finally, our network achieves $100+$ fps during inference and is applicable to real-time systems.

7.2 Related Works

Salient object detection aims to mark important regions by rectangles in an image. Early works assume that there is only one dominant object in an image and utilize various hand-crafted features to detect salient objects [81, 166]. Salient objects are segmented out by a CRF model [81] or bounding box statistics learned from a large image database [166]. Some works [167, 168] demonstrate the ability of generating multiple overlapping bounding boxes in a single scene by combining multiple image features. Recently, Zhang *et al.* [162] apply deep networks with object proposals to achieve state-of-the-art results. However, these methods are not scalable for real-time applications due to the use of sliding windows, complex optimization or expensive box sampling process.

Object proposal have been used widely in object detection, which are either generated from grouping superpixels [128, 169, 2] or sliding windows [170, 13]. However, it is a bottleneck to generate a large number of proposals for real-time detection [11, 9]. Recently, deep networks are trained to generate proposals in an end-to-end manner to improve efficiency [148, 5]. While methods such as [42, 77, 113, 31] instead adopt regular grid structures, they still rely on filtering a set of refined boxes. Different from previous methods, our approach does not use any proposals.

Object subitizing addresses the object *existence* problem by learning an external binary classifier [171, 166]. Zhang *et al.* [172] present a salient object subitizing model to remove detected boxes in images with no salient object. While the method in [162] addresses *existence* and *localization* problems at the same time, it still requires generating proposals recursively, which is inefficient.

Saliency map prediction produces a binary mask to segment salient objects from background. While both bottom-up methods using low-level image features [173, 174, 174, 175, 176] and top-down methods [81, 166] have been proposed for decades, many recent works utilize deep neural networks for this task [161, 158, 160, 159, 177, 178, 179]. Li *et al.* [158] propose a model for visual saliency using multi-scale deep features computed by CNNs. Wang *et al.* [160] develop two deep neural networks to learn local features and global contrast with geometric features to predict saliency score of each region. In [161], both global and local context are combined into a single deep network, while a fully convolutional network is applied in [177]. Note that existing methods heavily rely on pixel-level annotations [161, 159, 177] or external semantic information, *i.e.*, superpixels [178], which is not feasible for large-scale problems, where human labeling is extremely sparse. In contrast, our approach, as a weakly-supervised approach, only requires bounding box annotations and produces promising results as a free by-product, along with salient object detection and subitizing.

7.3 Proposed Approach

Existing detection methods based on CNNs and object proposals [148, 11, 9, 5, 162] convert the problem of selecting candidate locations in an image in the spatial domain to a parameter estimation problem, *e.g.*, finding independent numbers as the coordinates of the bounding boxes. They use as many as billions of parameters in fully connected (*fc*) layers [11, 9], which is computationally expensive and increases the possibility of over-fitting on small datasets. In contrast, our RSD approach discards proposals and directly solves the problem in the spatial domain. It reduces the number of parameters from billions to millions and achieves real-time speed. We predict a rough saliency map, from which we infer the exact number of boxes as the ground-truth objects based on the guidance of the subitizing output of our network. This unified framework addresses three closely related problems, saliency map prediction, subitizing and salient object detection, without allocating separate resources for each.

7.3.1 Network architecture

Our network is composed of the following components (see Figure 7.2). Images first go through a series of convolutional layers that can be any widely used models, such as VGG16 and ResNet-50. Specifically, we use the convolutional layers *conv1_1* through *conv5_3* from VGG16 [102], and *conv1* through *res4f* from ResNet-50 [21]. These layers capture low-level cues and high-level visual semantics. Two branches are connected

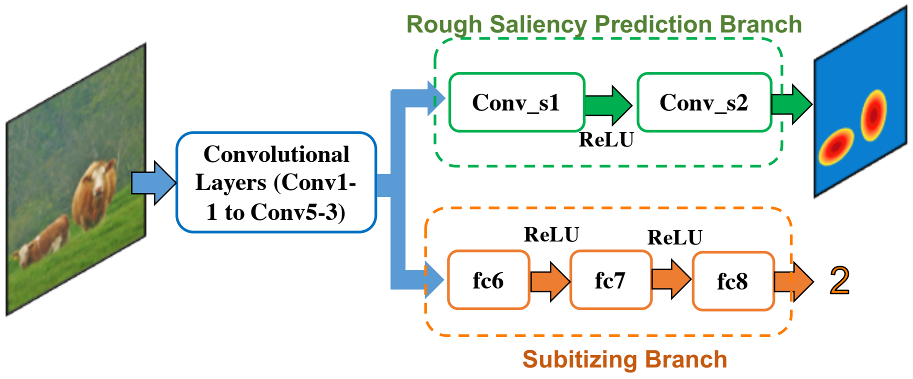

Rough Saliency Prediction Branch
Conv_s1
ReLU
Conv_s2
Convolutional Layers (Conv1-1 to Conv5-3)
ReLU
ReLU
fc6
fc7
fc8
2
Subitizing Branch

Figure 7.2: Our RSD network based on VGG16.

to the feature maps from the last convolutional layer: saliency map prediction branch and subitizing branch. The saliency map prediction branch consists of two convolutional layers, *conv_s1* and *conv_s2*, to continue processing the image in the spatial domain and produce a rough saliency map. The layer *conv_s1* has 80 3×3 filters to produce intermediate saliency maps conditioned on different latent distributions of the objects (*e.g.*, latent object categories). For instance, each of the 80 filters can be seen as a way to generate a rough saliency map for a specific type of category. The layer *conv_s2* summarizes these conditional maps into a single saliency map by a 1×1 filter followed by a sigmoid function. The subitizing branch predicts the number of salient objects that can be 0, 1, 2, or $3+$. It contains the final *fc* layers for VGG16, and all the remaining convolutional layers followed by a global average pooling layer and a single *fc* layer for ResNet-50.

7.3.2 Ground-truth preparation

The ground-truth for salient object detection only contains a set of numbers defining coordinates of bounding boxes tightly enclosing the objects. Although we can generate a binary mask based on these coordinates, *i.e.*, 1 inside the bounding boxes and 0 elsewhere, it cannot easily separate overlapping objects or encode non-rigid boundaries well if we use rectangular boxes.

To address this problem, we propose to generate Gaussian distributions to represent salient objects, and use images with Gaussian distributions as ground-truth saliency maps. Given a ground-truth bounding box B_i for an image with width W and height H, let (cx_i, cy_i, w_i, h_i) represent the coordinates of its center, width, and height. If the network has the stride of s at the beginning of the saliency map prediction branch (*e.g.*, 16 for VGG16), the ground-truth saliency map $\mathcal{M}_g$ is an image of size $\lfloor W/s \rfloor \times \lfloor H/s \rfloor$, where $\lfloor . \rfloor$ is the floor function. Its (x, y)-th element is then defined as

$$\mathcal{M}_g(x, y) = \sum_{i=1}^{N} e^{-\frac{1}{2}(\mathbf{v}_{xy}-\mu_i)^T \mathbf{\Sigma_i} (\mathbf{v}_{xy}-\mu_i)} \mathbb{1}_{\mathbf{v}_{xy} \in \mathcal{R}_{B_i}}, \tag{7.1}$$

where $\mathbf{v}_{xy} = [x, y]^T$ is the location vector, and $\mu_i = [\lfloor cx_i/s \rfloor, \lfloor cy_i/s \rfloor]^T$ is the mean value. N is the number of ground-truth bounding boxes in the image. $\mathcal{R}_{B_i}$ represents the ROI inside bounding box B_i. $\mathbb{1}$ is an indicator function. The covariance matrix $\mathbf{\Sigma}_n$ can be represented as

$$\mathbf{\Sigma}_i = \begin{bmatrix} \lfloor \frac{w_i}{s} \rfloor^2/4, 0 \\ 0, \lfloor \frac{h_i}{s} \rfloor^2/4 \end{bmatrix}. \tag{7.2}$$

By (7.1), we represent each bounding box as a normalized 2D Gaussian distribution, located at the center of the bounding box, with the co-variance determined by the bounding box's height and width and truncated at the box boundary. As shown in Figure 7.3, the Gaussian shape ground-truth provides better separation for multiple objects compared to rectangular bounding boxes. It also naturally acts as spatial weighting to the

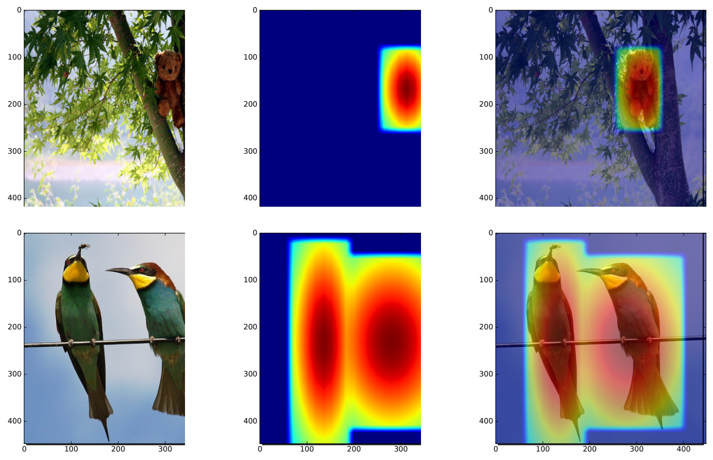


Figure 7.3: Samples of generated ground-truth saliency maps.

ground-truth, so that the network learns to focus more on the center of objects instead of being distracted by background.

7.3.3 Multi-task loss

Our network predicts a saliency map from an image and performs subitizing as well. During training, the network tries to minimize the difference between the ground-truth map and the predicted saliency map. Although Euclidean loss is widely used to measure the pixel-wise distance, it pushes gradients towards 0 if the values of most pixels are 0, which is the case in our application when there are images with no salient object. Therefore, we use a weighted Euclidean loss to better handle this scenario, defined as

$$\ell_{sal}(\mathbf{x}, \mathbf{g}) = \frac{1}{2d} \sum_{i=1}^{d} \alpha^{\mathbb{1}_{g_i > 0.5}} (x_i - g_i)^2, \tag{7.3}$$

where $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{g} \in \mathbb{R}^d$ are the vectorized predicted and ground-truth saliency maps, respectively. x_i and g_i represent the corresponding i-th element. α is a constant weight set to 5 in all our experiments. Essentially, the loss ℓ_{sal} assigns more weight to the pixels with the ground-truth value higher than 0.5, compared to those with the value close to 0. In this way, the problem of gradient vanishing is alleviated since the loss focuses more on pixels belonging to the real salient objects and is not dominated by background pixels. As a classifier, the subitizing branch minimizes the multinomial logistic loss $\ell_{sub}(y, n)$ between the ground-truth number of objects n, and the predicted number of objects y. The two losses are combined as our final multi-task loss

$$\ell(\mathbf{x}, \mathbf{g}, y, n) = \ell_{sal}(\mathbf{x}, \mathbf{g}) + \lambda \ell_{sub}(y, n), \tag{7.4}$$

where λ is a weighting factor to balance the importance of the two losses. We empirically set $\lambda = 0.25$ to make the magnitude of the loss values comparable.

7.3.4 Training

The loss in (7.4) defines a multi-task learning problem previously studied by other vision applications [11, 5]. It reduces required resources by sharing weights between different tasks, and acts as a regularization to avoid over-fitting. We use standard SGD with momentum and weight decay for learning the parameters of the network.

To ensure fair comparison, we adopt the same two-stage training scheme suggested by [162]. In the first stage, we initialize the network using the weights trained on ImageNet [180] for classification and fine-tune it on the ILSVRC2014 detection dataset [181] by treating all objects as salient. In the second stage, we continue fine-tuning the network on the SOS dataset [172] for salient object subitizing and detection. Although all the images in SOS are annotated for subitizing, some are not labled for detection. Therefore, we do not back-propagate gradients to our saliency map prediction branch for these images labeled as containing salient objects but without bounding box annotations. The loss function to fine-tune on the SOS dataset $\ell(\mathbf{x}, \mathbf{g}, y, n)$ is denoted as $\mathbb{1}_{n_{box} \geq n} \ell_{sal}(\mathbf{x}, \mathbf{g}) + \lambda \ell_{sub}(y, n)$, where n_{box} indicates the number of bounding box annotations in the image.

7.3.5 Bounding box generation

Our method leverages the saliency prediction branch and subtizing branch to infer the correct number and location of bounding boxes. Given the output of the subitizing branch n_{sub} and the rough saliency prediction map $\mathcal{M}$, the goal is to find K Gaussians $\mathcal{N}(\mu_k, \sigma_k^2), k = 1, \ldots, K$ that align with the predicted saliency map and are supported by the subitizing output, which can be formulated as

$$\begin{aligned} \arg\min_{\{\mu_k\},\{\sigma_k\},K} \; & \ell_s\Big(\sum_{k=1}^{K} \mathcal{N}(\mu_k, \sigma_k^2), \mathcal{M}\Big) \\ & + \mathbb{1}_{n_{sub} < n_M} \mathbb{1}_{\theta_{sub} > \theta_c} \ell_c(K, n_{sub}), \end{aligned} \tag{7.5}$$

where ℓ_s captures the discrepancy between the predicted saliency map and the generated Gaussian map. ℓ_c measures the disagreement between the subitizing branch and the number of Gaussians, from which boxes' locations can be inferred. n_M is the maximal possible output of the subitizing branch, *i.e.*, maximal number of salient objects. θ_{sub} is the confidence score of the subitizing branch, and θ_c is a fixed confidence threshold that will be discussed later. In other words, if $n_{sub} = n_M$ or θ_{sub} is lower than the threshold, we rely only on the predicted saliency map to determine the number and locations of salient objects. Since solving (7.5) directly is intractable, we propose a feasible and efficient greedy algorithm to approximate it, which predicts the center and scale of boxes, while optimizing the objective function. f $n_{sub} = 0$, our method does not generate any bounding boxes; otherwise it generates either single or multiple objects.

7.3.5.1 Single salient object detection

If $n_{sub} = 1$ and the confidence of subitizing branch c is larger than a pre-defined threshold θ_c, we think there is only a single object. We convert the saliency map $\mathcal{M}$ to a binary map $\mathcal{M}_b$ using θ_c, and then perform contour detection using the fast Teh-Chin chain approximation [182] on $\mathcal{M}_b$ to detect connected components $\mathcal{C}$ and infer bounding boxes $\mathcal{B}$. We define the ROI of box B_i on the original map as $\mathcal{M}_{\mathcal{R}_{B_i}}$, from which the maximal value is assigned as its score S_{B_i}. The one with the highest score is selected as the salient object. The entire process is summarized in Algorithm 4.

7.3.5.2 Multiple salient object detection

If $n_{sub} > 1$, there may be multiple salient objects. When the subitizing branch outputs n_M, or its confidence score $\theta_{sub} < \theta_c$, we rely on the predicted saliency map $\mathcal{M}$ to find as many reliable peaks as possible. Therefore, our method is able to detect arbitrary number of salient objects (see Table 7.1). Otherwise, we try to find at least n_{sub} reliable peaks. A multi-level thresholding scheme is proposed for robust peak detection and bal-

Algorithm 4: Single bounding box generation

```
Input  : $\mathcal{M}, \theta_c$ ;
         $\mathcal{M}$ is the predicted saliency map;
         $\theta_c$ is a fixed confidence threshold;
1  procedure SINGLEDETECT($\mathcal{M}$, $\theta_c$)
2      $\mathcal{M}_b \leftarrow \texttt{threshold}(\mathcal{M}, \theta_c)$ ;
3      $\mathcal{C} \leftarrow \texttt{detectContours}(\mathcal{M}_b)$ ;
4      $\mathcal{B} \leftarrow \texttt{generateBoxes}(\mathcal{C})$ ;
5      for $B_i \in \mathcal{B}$ do
6          $\mathcal{R}_{B_i} \leftarrow$ ROI of box $B_i$ ;
7          $S_{B_i} \leftarrow \max \mathcal{M}_{\mathcal{R}_{B_i}}$ ;
8      end
9      return $B = \arg\max_{B_i} S_{B_i}$ ;
10 end
```

ancing the losses, ℓ_s and ℓ_c, in (7.5). Starting from a high threshold, a peak $P_i = [x_i, y_i]^T$ is discovered from $\mathcal{M}$ following similar steps in Algorithm 4. Peaks are continuously identified and added to the set of peaks $\mathcal{P}$ by reducing the threshold and repeating the process until the cardinality of $\mathcal{P}$ reaches or exceeds n_{sub}. Note that the predicted number of boxes depends on both the subitizing and saliency map prediction branches, which could be less or more than n_{sub}, if no threshold can separate n_{sub} reliable peaks or more peaks are detected in different thresholds.

After the initial set of peaks are determined, peaks with low confidence are treated as noise and removed. Then we try to find separating lines $\mathcal{L}$ to isolate remaining peaks into different non-overlapping regions. Each line perpendicular to the line segment connecting a pair of peaks is associated with a score. The score is the maximal value of the pixels this line passes on $\mathcal{M}$. The one with the minimal score is selected as the separating line of the two peaks. In this way, we ensure that the separating line passes through the boundaries between objects rather than the objects themselves. These lines $\mathcal{L}$ divide $\mathcal{M}$

Algorithm 5: Multiple bounding box generation

```
Input  : $\mathcal{M}$, $\boldsymbol{\Theta}$, $\theta_c$ ;
         $\mathcal{M}$ is the predicted saliency map;
         $\boldsymbol{\Theta}$ is the set of fixed thresholds for peak detection;
         $\theta_c$ is a fixed confidence threshold;
1  procedure MULTIDETECT ($\mathcal{M}$, $\theta_c$)
2      $\mathcal{P} \leftarrow \varnothing, \mathcal{B} \leftarrow \varnothing$;
3      while $|\mathcal{P}| \leq n_{sub}$ do
4          for $\theta_i \in \boldsymbol{\Theta}$ do
5              $\mathcal{M}_b \leftarrow$ threshold($\mathcal{M}, \theta_i$);
6              $\mathcal{C} \leftarrow$ detectContours($\mathcal{M}_b$);
7              for $C_j \in \mathcal{C}$ do
8                  $\mathcal{P} \leftarrow \mathcal{P} \cup \arg\max \mathcal{M}_{C_j}$;
9              end
10         end
11     end
12     $\mathcal{P} \leftarrow \mathcal{P} \setminus P_i$ where $\mathcal{M}(P_i) < \theta_c, \forall i$;
13     $\mathcal{L} \leftarrow$ findSeparatingLines $(P_i, P_j), \forall i \neq j$;
14     for $P_i \in \mathcal{P}$ do
15         $\mathcal{R}_{P_i} \leftarrow$ ROI formed by $\mathcal{L}$ containing $P_i$;
16         $\mathcal{B} \leftarrow \mathcal{B} \cup$ SINGLEDETECT($\mathcal{M}_{\mathcal{R}_{P_i}}, \theta_c$);
17     end
18     return $\mathcal{B}$;
19 end
```

into different regions. Finally, for each peak $P_i \in \mathcal{P}$, we apply Algorithm 4 to its corresponding region on the saliency map to obtain a bounding box. Algorithm 5 summarizes the process.

7.4 Experiments

7.4.1 Experimental setup

Datasets. We evaluate our salient object detection method on four datasets, MSO[172], PASCAL-S[183], MSRA[81] and DUT-O[174]. The MSO dataset is the test set of the SOS dataset annotated for salient object detection. It contains images of multiple salient objects and many background images with no salient object. PASCAL-S is a subset of

PASCAL VOC 2010 validation set [118] annotated for saliency segmentation problem. It contains images with multiple salient objects and 8 subjects decided on the saliency of each object segment. As suggested by [183], we define salient objects as those having a saliency score of at least 0.5, *i.e.*, half of the subjects believe that the object is salient, and consider the surrounding rectangles as the ground-truth. We also compare the performance of our method and existing methods for subitizing on this dataset. The MSRA and DUT-O datasets only contain images of single salient object. For every image in the MSRA and DUT-O datasets, five raw bounding box annotations are provided, which are later converted to one ground-truth following the same protocol in [162]. We use only the SOS dataset for training and others for evaluation. To verify that our RSD can also generate accurate pixel-wise saliency map, we additionally compare our method with existing methods on ESSD [184] and PASCAL-S [183] datasets.

Parameters and settings. In Algorithm 4 and 5, we set $\theta_c = 0.7$ as our strong evidence threshold, $\boldsymbol{\Theta} = [0.95, 0.9, 0.8, 0.6]$ as our peak detection thresholds, and use vertical and horizontal lines as our separating lines. In our real-time network based on VGG16, we use an image size of 224×224 and for our network based on ResNet-50 we use 448×448 instead. We smooth predicted saliency maps by a Gaussian filter before converting them to binary maps. We use a Gaussian kernel with standard deviation of 10 for 448×448 input and 2 for 224×224 input. In the first training step, we use Xavier initialization for *conv_s1* and *conv_s2* and Gaussian initializer for the final *fc* layer in the subitizing branch. For fine-tuning on SOS, we use a momentum of 0.9, weight decay of $5e^{-4}$, and learning rates of $1e^{-4}$ and $1e^{-5}$ for our VGG16 and ResNet-50 based methods, respectively. All

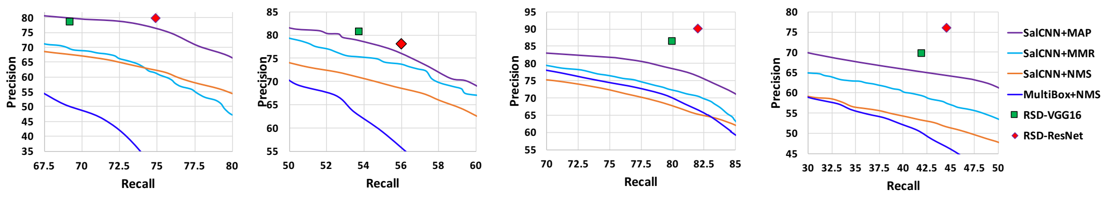


Figure 7.4: Comparisons of precision/recall by different methods on MSO, PASCAL-S, MSRA, and DUT-O (from left to right) datasets. We let others methods to generate different number of boxes by varying the threshold for confidence scores of boxes and present the performance change as precision-recall curves. The IoU threshold for evaluation is set to 0.5.

timings are measured on an NVIDIA Titan X Pascal GPU, and a system with 128GB RAM and Intel Core i7 6850K CPU.

7.4.2 Results

Salient Object detection. We compare RSD with several existing methods including the state-of-the-art approach in [162], which are SalCNN+MAP, SalCNN+NMS, SalCNN with Maximum Marginal Relevance (SalCNN+MMR), and MultiBox [148] with NMS. Unlike our RSD that generates the exact number of bounding boxes as salient objects, other methods have free parameters to determine the number of selected bounding boxes from hundreds of proposals, which greatly changes their performance. For fair comparison, we change these free parameters and show their best results with our performance point in Figure 7.4. It should be noted that we use the same set of parameters on all datasets, while for other methods different parameters lead to their best performance on different datasets.

On the MSO and PASCAL-S datasets that contain multiple salient objects, our RSD-ResNet produces the best results at the same precision or recall rate. RSD-VGG achieves comparable precision/recall as the state-of-the-art methods while being nearly

Method	RSD-ResNet/RSD-VGG	SalCNN[162]+MAP	SalCNN+MMR	SalCNN+NMS
False Positives	**36/30**	54/40	95/50	53/34
F1 Score (1 $\sim$ 3 objects)	**79.2/77.4**	78.9/77.0	71.6/72.6	72.5/70.7
F1 Score (4+ objects)	**57.5**/26.8	55.2/**50.9**	46.1/47.7	47.7/48.5

Table 7.1: Number of false positives in images containing no salient objects and F1 score for different number of ground-truth objects in the MSO dataset. For fair comparison, results of other methods are obtained at the same recall rate of RSD-ResNet and RSD-VGG respectively (seperated by "/").

$12\times$ faster. Although our subitizing branch has a range of three, Table 7.1 shows that our RSD-ResNet also achieves the best results on images with $4+$ objects based on the predicted saliency map. On the MSRA and DUT-O datasets that contain single salient object in an image, both of our RSD-VGG and RSD-ResNet outperform the state-of-the-arts by a large margin. Notably, our RSD-ResNet achieves nearly 15% and 10% absolute improvement in precision at the same recall rate on the MSRA and DUT-O datasets, respectively, which clearly indicates that our method, without any object proposals, is more powerful and robust even when it is allowed to generate only a single bounding box.

Object subitizing. We evaluate the subitizing performance of our RSD on the MSO dataset. First, we compare our RSD with state-of-the-art methods in terms of solving the *existence* problem in Table 7.1. While our parameters are fixed, we vary the parameters of other methods on different datasets to match their performance. For example, we tune the parameters of other methods when comparing with our RSD-ResNet, so that they can achieve the same recall as ours. Then we compare the number of false positives in the background images. We do the same thing for the comparison with our RSD-VGG as well.

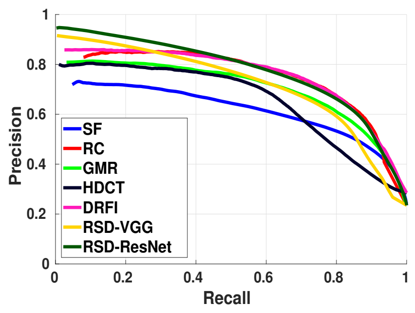
For predicting *existence*, both our RSD-ResNet and RSD-VGG produce fewer false positives when there is no salient object. Additionally, we compare the counting perfor-

Method	ResNet [21]	RSD-ResNet	VGG [102]	RSD-VGG
Accuracy	83.33	**86.19**	83.25	83.97

Table 7.2: The accuracy of the counting branch and comparison with the baselines.

mance of RSD with two baselines using vanilla ResNet-50 and VGG16 in Table 7.2. For fair comparison, we use exactly the same training scheme and initialization for all networks. Our RSD method successfully produces better accuracy compared with vanilla ResNet-50 and VGG16, verifying that the multi-task training facilitates the subitizing branch to learn a better classifier by utilizing the information from saliency map prediction.

Saliency map prediction. In real world scenarios, pixel-level annotations are difficult to collect. It is challenging to generate precise saliency maps without such detailed labeling. As a weakly-supervised approach only using bounding boxes for salient foreground segmentation, we will show that our RSD still generates accurate saliency map that aligns well with multiple salient objects in the scene. Figure 7.5 compares our RSD against five unsupervised salient object segmentation algorithms, RC [175], SF [173], GMR [174], PCAS [185], GBVS [186] and three supervised methods, HDCT [187], DRFI [188], GBVS+PatchCut [189]. While we focus on the comparison with supervised approaches, the comparison with those unsupervised methods is included for completeness and provides the context of how our approach fits into the pixel-level saliency prediction task. We also evaluate the performance using precision-recall curves. Specifically, the precision and recall are computed by binarizing the grayscale saliency map using varying thresholds [190, 173, 174, 184] and comparing the binary mask against the ground-truth. Our RSD approach is surprisingly good considering that it only uses rough Gaussian maps as



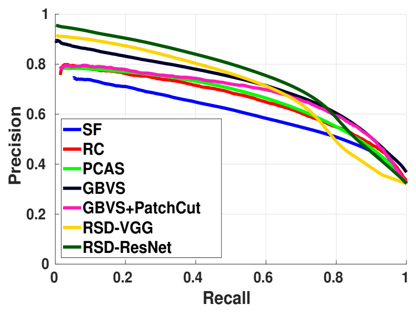
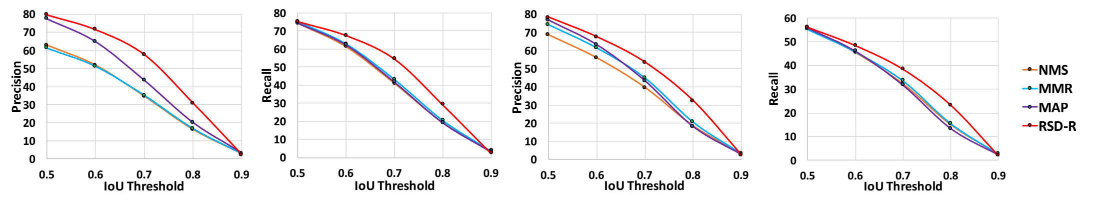


Figure 7.5: The pixel-wise saliency map prediction performance on the ESSD [184] (left) and PASCAL-S [183] (right) datasets.

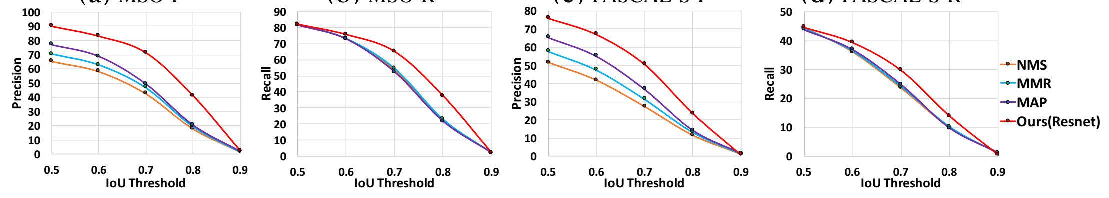
ground-truth. In particular, the RSD-ResNet approach produces comparable results with the fully-supervised methods in terms of precision/recall, making it readily applicable for salient foreground segmentation without any pixel-level annotations.

7.4.3 Ablation study

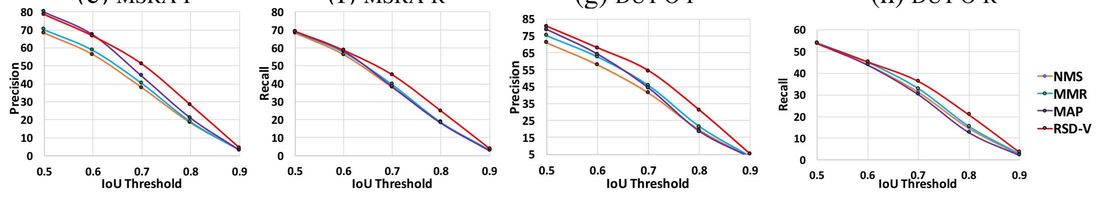
Localization. Although we do not use proposals and pruning stage like NMS, our straightforward bounding box generation algorithm generates good results. Moreover, bounding boxes generated by our method align with the ground-truth better compared to existing approaches, leading to the best precision and recall, as shown in Figure 7.6. In this experiment, we let other methods to pick their parameters to get the same recall as ours at IoU$= 0.5$, and then change the IoU threshold to evaluate the performance change. Notably, if we have a more strict IoU criteria, such as 0.8, RSD still maintains a relatively high precision and recall, while the precision and recall of all the other methods greatly drop. At this IoU, even our fast RSD-VGG is able to outperform the state-of-the-art methods on all datasets by an average margin of around 10% in terms of both precision and recall. The results clearly demonstrate that our network successfully predicts an accurate



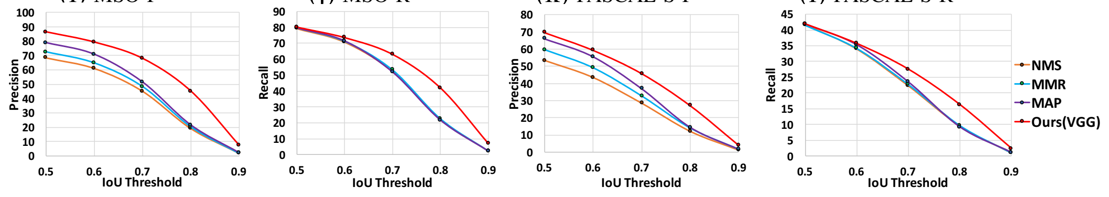
(a) MSO-P (b) MSO-R (c) PASCAL-S-P (d) PASCAL-S-R



(e) MSRA-P (f) MSRA-R (g) DUT-O-P (h) DUT-O-R



(i) MSO-P (j) MSO-R (k) PASCAL-S-P (l) PASCAL-S-R



(m) MSRA-P (n) MSRA-R (o) DUT-O-P (p) DUT-O-R

Figure 7.6: Comparison of RSD-ResNet (top), RSD-VGG (bottom) with other methods in terms of precision at the same recall and recall at the same precision under different localization thresholds. “P” stands for precision and “R” stands for recall.

saliency map and easily generates only a few bounding boxes tightly enclosing the correct salient objects. Some qualitative results are presented in Figure 7.7. Our RSD approach clearly outperforms SalCNN+MAP in generating better bounding boxes that more tightly enclose the ground-truth.

Object size. The behavior of detection methods usually differs when dealing with small and large objects. To better understand how our method works compared to existing methods, we further analyze its performance with respect to different sizes of objects. Objects with an area larger than 200×200 pixels are counted as *large* objects. For MSO

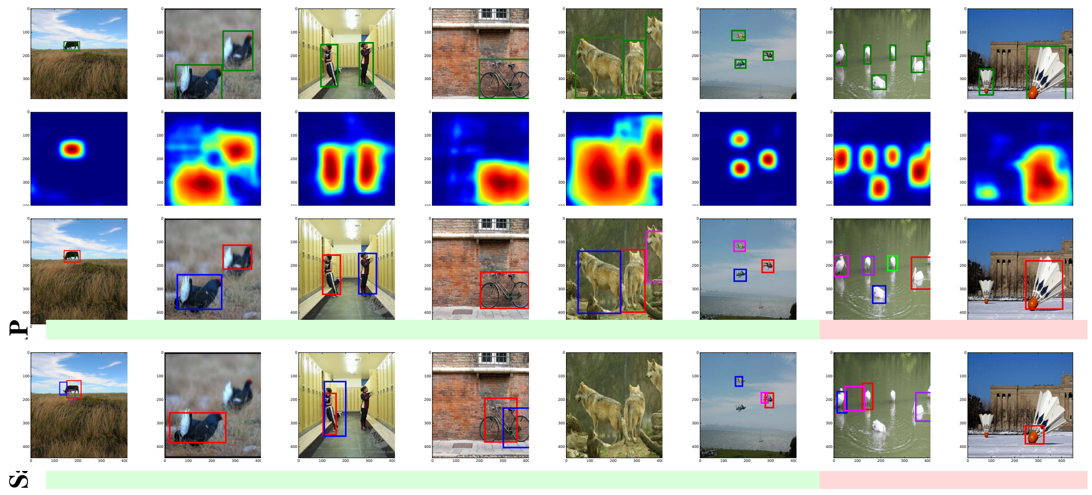


Figure 7.7: Qualitative results of our RSD-ResNet and SalCNN+MAP. Note that our method locates multiple salient object in the scene more accurately than SalCNN+MAP. The last two columns show hard examples where both SalCNN+MAP and ours cannot locate all the salient objects.

and DUT-O datasets, the ground-truth boxes with an area less than 75×75 pixels are defined as *small* objects. We increase this size to 125×125 pixels for the MSRA dataset to obtain a statistically reliable subset for performance estimation since the salient objects in this dataset are generally larger. We evaluate the precision and recall on small and large objects separately and show the results in Table 7.3.

Method	MSO Dataset						MSRA Dataset						DUT-O Dataset						PASCAL-S Dataset					
	Precision			Recall			Precision			Recall			Precision			Recall			Precision			Recall		
	S	S_s	S_l	S	S_s	S_l	S	S_s	S_l	S	S_s	S_l	S	S_s	S_l	S	S_s	S_l	S	S_s	S_l	S	S_s	S_l
SalCNN[162]+NMS	63.0	23.3	76.1	74.2	42.7	81.4	65.4	11.4	78.6	81.9	**74.0**	83.4	51.7	19.6	70.0	44.5	25.9	46.7	68.8	28.1	82.2	55.7	17.4	73.8
SalCNN+MMR	61.4	23.7	72.6	74.9	38.5	84.9	70.6	15.9	78.9	82.0	71.9	84.3	57.9	24.4	69.9	44.2	24.0	48.1	74.2	35.9	81.7	55.1	16.0	74.2
SalCNN+MAP	77.5	43.8	79.2	74.1	40.6	84.6	77.1	20.7	79.4	81.5	72.7	83.7	65.5	31.6	70.3	43.7	23.3	**47.3**	76.8	28	82.1	55.8	9.7	77.3
RSD-ResNet	**79.7**	**69.1**	**81.8**	**74.9**	**49.0**	**85.5**	**90.1**	**39.9**	**85.2**	**82.0**	67.8	**87.1**	**76.0**	**61.2**	**80.0**	**44.6**	**25.6**	45.3	**78.2**	**72.7**	**82.1**	**56.0**	**16.7**	**77.8**

Table 7.3: The precision/recall of our RSD-ResNet and other methods on different datasets and objects of different sizes. The IoU threshold is 0.5. S, S_s and S_l denote all objects, objects of small and large size.

Our RSD-ResNet clearly outperforms all the compared methods, achieving the best performance on the MSO dataset for both small and large objects. It also produces the best recall at the same precision for large objects on MSRA dataset and small objects on DUT-O dataset, indicating that it discovers objects of different sizes well under various conditions. At the same recall, our RSD-ResNet greatly improves the precision, especially for small objects that are difficult to locate by object proposal based approaches.

7.4.4 Run-time efficiency

By directly generating the saliency map through network forward without proposals, our approach is extremely efficient for salient object detection during inference. We compare the run-time speed of SalCNN [162] and our approach in Table 7.4. With ResNet, our approach achieves nearly 20 fps, while SalCNN only runs at 10 fps. With VGG16, our method achieves an impressive speed at 120 fps, $12\times$ faster than SalCNN, and readily applicable to real-time scenarios. This experiment confirms that we successfully improve the run-time speed of the network by removing the bottleneck of proposal generation and refinement.

Method	RSD-VGG-S	RSD-VGG-M	RSD-ResNet-S	RSD-ResNet-M	SalCNN+MAP	SalCNN+NMS	SalCNN+MMR
Speed	**120.48**	113.63	19.05	18.65	10.64	10.72	10.71

Table 7.4: The run-time speed (in fps) of our RSD and compared methods during inference. Our methods with suffix “S” are for single salient object detection, while the ones with suffix “M” are for multiple salient object detection.

7.5 Conclusion

We have presented a real-time unconstrained salient object detection framework using deep convolutional neural networks, named RSD. By eliminating the steps of proposing and refining thousands of candidate boxes, our network learns to directly generates the exact number of salient objects. Our network performs saliency map prediction without pixel-level annotations, salient object detection without object proposals, and salient object subitizing simultaneously, all in a single pass within a unified framework. Extensive experiments show that our RSD approach outperforms existing methods on various datasets for salient object detection and subitizing, and produces comparable results for salient foreground segmentation. In particular, our approach based on VGG16 network achieves more than 100 fps on average on GPU during inference time, which is $12\times$ faster than the state-of-the-art approach, while being more accurate.

Chapter 8: Conclusion and Future Work

In this dissertation, we studied the shortcomings that arise when deep convolutional neural networks are applied to the tasks of object and face detection. We first focused on the problem of lack of scale-invariance in these networks and discussed how multi-scale pyramids can be deployed efficiently to reduce this problem. In chapter 2, we proposed "SNIPER", an efficient algorithm for multi-scale training that sampled low-resolution chips from an image pyramid to accelerate training by a factor of 3 times. While SNIPER makes training efficient, it is not applicable during inference given its dependency on ground-truth annotations for chip sampling. Chapter 3 introduced "AutoFocus", a novel approach for performing efficient multi-scale inference for object detection. With predicting where to zoom, we demonstrated that a considerable speedup over a baseline is possible without any drop in performance.

We then considered the problem of face detection. Compared to generic objects, faces are rigid and images with hundreds of faces are common. In chapter 4, we presented a fast and lightweight face detector that can detect faces of various scales efficiently even in crowded scenes. We then introduced a method in chapter 5 which takes into account the spatial relationships between parts of an object. We also showed the possibility of densifying detections at inference time without requiring retraining of the model to deal with crowded scenes.

In the last part of the dissertation, we turned our attention to the localization subproblem of the detection task. In chapter 6, we proposed a novel way to model object detection as a search in the space of all possible bounding boxes. In this way, we could replace the slow localization part of the detection framework with the same model used for classifying objects. Finally, in chapter 7, we showed how the inference speed can be further improved to hundreds of images per second when it comes to tasks like saliency detection where a few objects are usually present in each scene.

Going forward, efficient detection in 3D environments, as a fundamental building block of real-time applications such as autonomous driving, is an important topic to be studied further. The trade-off between accuracy and speed becomes steeper when it comes to scenes with thousands of 3D points and the requirement of processing tens of scenes per second. Moreover, new modalities (*e.g.* point clouds) demand novel approaches to efficiently process data without degrading the performance.

Aside from 3D, I believe there is still room to improve the efficiency of 2D object detection systems. In the past couple of years, there has been an ongoing effort to develop more efficient CNN architectures for visual recognition. However, besides the main computation, there remain heuristic steps in instance recognition pipelines that slow down the systems (*e.g.* applying non-maximum suppression to thousands of proposal boxes). Replacing these heuristics to simplify the design has the potential to improve efficiency even further.

Lastly, the robustness of detection systems needs to be considered more thoroughly. Currently, it is possible to come up with imperceivable adversarial noises to fool even the strongest detectors available. Although the applicability of the current adversarial attacks to commercialized visualized systems is still open to debate, they can eventually limit the deployment of such systems for high-stake applications. Besides, improving the robustness of detection pipelines has the potential to act as a strong regularization to increase the generalization of the models.

Bibliography

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[2] Jasper RR Uijlings, Koen EA van de Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[3] Maurizio Corbetta and Gordon L Shulman. Control of goal-directed and stimulus-driven attention in the brain. *Nature reviews neuroscience*, 3(3):201, 2002.

[4] Ronald A Rensink. The dynamic representation of scenes. *Visual cognition*, 7(1-3):17–42, 2000.

[5] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[6] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

[7] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. *CVPR*, 2018.

[8] Bharat Singh and Larry S Davis. An analysis of scale invariance in object detection-snip. *CVPR*, 2018.

[9] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jagannath Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, pages 580–587, 2014.

[10] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[11] Ross Girshick. Fast r-cnn. In *ICCV*, pages 1440–1448, 2015.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, pages 346–361, 2014.

[13] C Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *ECCV*, pages 391–405, 2014.

[14] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 761–769, 2016.

[15] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2881–2890, 2017.

[16] Chao Peng, Tete Xiao, Zeming Li, Yuning Jiang, Xiangyu Zhang, Kai Jia, Gang Yu, and Jian Sun. Megdet: A large mini-batch object detector. *CVPR*, 2018.

[17] Yuxin Wu and Kaiming He. Group normalization. *arXiv:1803.08494*, 2018.

[18] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 4898–4906, 2016.

[19] Ivan Krasin, Tom Duerig, Neil Alldrin, Vittorio Ferrari, Sami Abu-El-Haija, Alina Kuznetsova, Hassan Rom, Jasper Uijlings, Stefan Popov, Shahab Kamali, Matteo Malloci, Jordi Pont-Tuset, Andreas Veit, Serge Belongie, Victor Gomes, Abhinav Gupta, Chen Sun, Gal Chechik, David Cai, Zheyun Feng, Dhyanesh Narayanan, and Kevin Murphy. Openimages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://storage.googleapis.com/openimages/web/index.html*, 2017.

[20] Sharan Narang, Gregory Diamos, Erich Elsen, Paulius Micikevicius, Jonah Alben, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *ICLR*, 2018.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[23] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[24] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. Soft-nms—improving object detection with one line of code. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5562–5570. IEEE, 2017.

[25] Sergey Zagoruyko, Adam Lerer, Tsung-Yi Lin, Pedro O Pinheiro, Sam Gross, Soumith Chintala, and Piotr Dollár. A multipath network for object detection. *arXiv preprint arXiv:1604.02135*, 2016.

[26] Spyros Gidaris and Nikos Komodakis. Locnet: Improving localization accuracy for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 789–798, 2016.

[27] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.

[28] François Chollet. Xception: Deep learning with depthwise separable convolutions. *CVPR*, 2017.

[29] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[30] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, pages 936–944, 2017.

[31] Mahyar Najibi, Pouya Samangouei, Rama Chellappa, and Larry Davis. SSH: Single stage headless face detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4875–4884, 2017.

[32] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. *ICCV*, 2017.

[33] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ICML*, 2015.

[34] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[35] Sean Bell, C Lawrence Zitnick, Kavita Bala, and Ross Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2874–2883, 2016.

[36] Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan Yuille. The role of context for object detection and semantic segmentation in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 891–898, 2014.

[37] Zhaowei Cai, Quanfu Fan, Rogerio S Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *European Conference on Computer Vision*, pages 354–370. Springer, 2016.

[38] Fan Yang, Wongun Choi, and Yuanqing Lin. Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2129–2137, 2016.

[39] Tsung-Yi Lin, Priyal Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[40] Vinay Praneeth Boda and Prakash Narayan. Sampling rate distortion. *IEEE Transactions on Information Theory*, 63(1):563–574, 2017.

[41] Vinay Praneeth Boda and Prakash Narayan. Universal sampling rate distortion. *IEEE Transactions on Information Theory*, 2018.

[42] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37. Springer, 2016.

[43] Xingyu Zeng, Wanli Ouyang, Junjie Yan, Hongsheng Li, Tong Xiao, Kun Wang, Yu Liu, Yucong Zhou, Bin Yang, Zhe Wang, et al. Crafting gbd-net for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[44] John Aloimonos, Isaac Weiss, and Amit Bandyopadhyay. Active vision. *International journal of computer vision*, 1(4):333–356, 1988.

[45] John M Findlay and Iain D Gilchrist. *Active vision: The psychology of looking and seeing*. Oxford University Press, 2003.

[46] Helga Kolb. Simple anatomy of the retina. *Webvision: The organization of the retina and visual system*, pages 13–36, 1995.

[47] Hans Strasburger, Ingo Rentschler, and Martin Jüttner. Peripheral vision and pattern recognition: A review. *Journal of vision*, 11(5):13–13, 2011.

[48] Wolf Kienzle, Matthias O Franz, Bernhard Schölkopf, and Felix A Wichmann. Center-surround patterns emerge as optimal predictors for human saccade targets. *Journal of vision*, 9(5):7–7, 2009.

[49] Bruno G Breitmeyer and Leo Ganz. Implications of sustained and transient channels for theories of visual pattern masking, saccadic suppression, and information processing. *Psychological review*, 83(1):1, 1976.

[50] David E Irwin, Joseph S Brown, and Jun-shi Sun. Visual masking and visual integration across saccadic eye movements. *Journal of Experimental Psychology: General*, 117(3):276, 1988.

[51] Ethel Matin. Saccadic suppression: a review and an analysis. *Psychological bulletin*, 81(12):899, 1974.

[52] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *ICML Workshop on Abstraction in Reinforcement Learning*, 2016.

[53] Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. *AAAI*, 2018.

[54] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. *CVPR*, 2018.

[55] Bharat Singh, Mahyar Najibi, and Larry S Davis. SNIPER: Efficient multi-scale training. *NeurIPS*, 2018.

[56] Andrew Witkin. Scale-space filtering: A new approach to multi-scale description. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'84.*, volume 9, pages 150–153. IEEE, 1984.

[57] Mingfei Gao, Ruichi Yu, Ang Li, Vlad I Morariu, and Larry S Davis. Dynamic zoom-in network for fast object detection in large images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[58] Guanglu Song, Yu Liu, Ming Jiang, Yujie Wang, Junjie Yan, and Biao Leng. Beyond trade-off: Accelerate fcn-based face detector with higher accuracy. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7756–7764, 2018.

[59] Kai Chen, Jiaqi Wang, Shuo Yang, Xingcheng Zhang, Yuanjun Xiong, Chen Change Loy, and Dahua Lin. Optimizing video object detection via a scale-time lattice. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7814–7823, 2018.

[60] Mohammad Javad Shafiee, Brendan Chywl, Francis Li, and Alexander Wong. Fast yolo: A fast you only look once system for real-time embedded object detection in video. *arXiv preprint arXiv:1709.05943*, 2017.

[61] Mason Liu and Menglong Zhu. Mobile video object detection with temporally-aware feature maps. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[62] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. No-scope: optimizing neural network queries over video at scale. *Proceedings of the VLDB Endowment*, 10(11):1586–1597, 2017.

[63] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.

[64] Lubomir Bourdev and Jonathan Brandt. Robust object detection via soft cascade. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 236–243. IEEE, 2005.

[65] Qiang Zhu, Mei-Chen Yeh, Kwang-Ting Cheng, and Shai Avidan. Fast human detection using a cascade of histograms of oriented gradients. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1491–1498. IEEE, 2006.

[66] Cha Zhang and Paul A Viola. Multiple-instance pruning for learning efficient cascade detectors. In *Advances in neural information processing systems*, pages 1681–1688, 2008.

[67] Paul Viola, Michael J Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63(2):153–161, 2005.

[68] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[69] Piotr Dollár, Zhuowen Tu, Pietro Perona, and Serge Belongie. Integral channel features. *BMVC*, 2009.

[70] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2010.

[71] Piotr Dollár, Serge J Belongie, and Pietro Perona. The fastest pedestrian detector in the west. In *BMVC*, volume 2, page 7. Citeseer, 2010.

[72] Rodrigo Benenson, Markus Mathias, Radu Timofte, and Luc Van Gool. Pedestrian detection at 100 frames per second. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2903–2910. IEEE, 2012.

[73] Piotr Dollár, Ron Appel, and Wolf Kienzle. Crosstalk cascades for frame-rate pedestrian detection. In *Computer Vision–ECCV 2012*, pages 645–659. Springer, 2012.

[74] Piotr Dollár, Ron Appel, Serge Belongie, and Pietro Perona. Fast feature pyramids for object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(8):1532–1545, 2014.

[75] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[76] Yu Liu, Hongyang Li, Junjie Yan, Fangyin Wei, Xiaogang Wang, and Xiaoou Tang. Recurrent scale approximation for object detection in cnn. In *IEEE international conference on computer vision*, volume 5, 2017.

[77] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[78] Joao Carreira and Cristian Sminchisescu. Constrained parametric min-cuts for automatic object segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3241–3248. IEEE, 2010.

[79] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on pattern analysis and machine intelligence*, 20(11):1254–1259, 1998.

[80] Xiaodi Hou and Liqing Zhang. Saliency detection: A spectral residual approach. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[81] Tie Liu, Zejian Yuan, Jian Sun, Jingdong Wang, Nanning Zheng, Xiaoou Tang, and Heung-Yeung Shum. Learning to detect a salient object. *IEEE Transactions on Pattern analysis and machine intelligence*, 33(2):353–367, 2011.

[82] Mahyar Najibi, Fan Yang, Qiaosong Wang, and Robinson Piramuthu. Towards the success rate of one: Real-time unconstrained salient object detection. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1432–1441. IEEE, 2018.

[83] Abel Gonzalez-Garcia, Alexander Vezhnevets, and Vittorio Ferrari. An active search strategy for efficient object class detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3022–3031, 2015.

[84] Stefan Mathe, Aleksis Pirinen, and Cristian Sminchisescu. Reinforcement learning for visual object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2894–2902, 2016.

[85] Aleksis Pirinen and Cristian Sminchisescu. Deep reinforcement learning of region proposal networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6945–6954, 2018.

[86] Zequn Jie, Xiaodan Liang, Jiashi Feng, Xiaojie Jin, Wen Lu, and Shuicheng Yan. Tree-structured reinforcement learning for sequential object localization. In *Advances in Neural Information Processing Systems*, pages 127–135, 2016.

[87] Yongxi Lu, Tara Javidi, and Svetlana Lazebnik. Adaptive object detection using adjacency and zoom prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2351–2359, 2016.

[88] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[89] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[90] David Marr and Ellen Hildreth. Theory of edge detection. *Proc. R. Soc. Lond. B*, 207(1167):187–217, 1980.

[91] Krystian Mikolajczyk and Cordelia Schmid. Scale & affine invariant interest point detectors. *International journal of computer vision*, 60(1):63–86, 2004.

[92] Peter J Burt and Edward H Adelson. The laplacian pyramid as a compact image code. In *Readings in Computer Vision*, pages 671–679. Elsevier, 1987.

[93] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[94] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[95] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Light-head r-cnn: In defense of two-stage object detector. *arXiv preprint arXiv:1711.07264*, 2017.

[96] Shifeng Zhang, Longyin Wen, Xiao Bian, Zhen Lei, and Stan Z Li. Single-shot refinement neural network for object detection. *CVPR*, 2018.

[97] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 734–750, 2018.

[98] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. *arXiv preprint arXiv:1811.11168*, 2018.

[99] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. *CVPR*, 2018.

[100] Peiyun Hu and Deva Ramanan. Finding tiny faces. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1522–1530. IEEE, 2017.

[101] Shuo Yang, Ping Luo, Chen-Change Loy, and Xiaoou Tang. Wider face: A face detection benchmark. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5525–5533, 2016.

[102] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[103] Xiangxin Zhu and Deva Ramanan. Face detection, pose estimation, and landmark localization in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2879–2886. IEEE, 2012.

[104] Haoxiang Li, Gang Hua, Zhe Lin, Jonathan Brandt, and Jianchao Yang. Probabilistic elastic part model for unsupervised face detector adaptation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 793–800, 2013.

[105] Haoxiang Li, Zhe Lin, Jonathan Brandt, Xiaohui Shen, and Gang Hua. Efficient boosted exemplar-based face detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1843–1850, 2014.

[106] Markus Mathias, Rodrigo Benenson, Marco Pedersoli, and Luc Van Gool. Face detection without bells and whistles. In *European Conference on Computer Vision*, pages 720–735. Springer, 2014.

[107] Dong Chen, Shaoqing Ren, Yichen Wei, Xudong Cao, and Jian Sun. Joint cascade face detection and alignment. In *European Conference on Computer Vision*, pages 109–122. Springer, 2014.

[108] Bin Yang, Junjie Yan, Zhen Lei, and Stan Z Li. Aggregate channel features for multi-view face detection. In *IEEE International Joint Conference on Biometrics (IJCB)*, pages 1–8. IEEE, 2014.

[109] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, and Gang Hua. A convolutional neural network cascade for face detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5325–5334, 2015.

[110] Shuo Yang, Ping Luo, Chen-Change Loy, and Xiaoou Tang. From facial parts responses to face detection: A deep learning approach. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3676–3684, 2015.

[111] Bin Yang, Junjie Yan, Zhen Lei, and Stan Z Li. Convolutional channel features. In *Proceedings of the IEEE international conference on computer vision*, pages 82–90, 2015.

[112] Chenchen Zhu, Yutong Zheng, Khoa Luu, and Marios Savvides. Cms-rcnn: contextual multi-scale region-based cnn for unconstrained face detection. In *Deep Learning for Biometrics*, pages 57–79. Springer, 2017.

[113] Mahyar Najibi, Mohammad Rastegari, and Larry S Davis. G-cnn: an iterative grid based object detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2369–2377, 2016.

[114] Christian Szegedy, Scott Reed, Dumitru Erhan, Dragomir Anguelov, and Sergey Ioffe. Scalable, high-quality object detection. *arXiv preprint arXiv:1412.1441*, 2014.

[115] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[116] Vidit Jain and Erik G Learned-Miller. Fddb: A benchmark for face detection in unconstrained settings. *UMass Amherst Technical Report*, 2010.

[117] Junjie Yan, Xuzong Zhang, Zhen Lei, and Stan Z Li. Face detection by structural models. *Image and Vision Computing*, 32(10):790–799, 2014.

[118] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, 2015.

[119] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016.

[120] Eshed Ohn-Bar and Mohan M Trivedi. To boost or not to boost? on the limits of boosted trees for object detection. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 3350–3355. IEEE, 2016.

[121] Yunzhu Li, Benyuan Sun, Tianfu Wu, and Yizhou Wang. Face detection with end-to-end integration of a convnet and a 3d model. In *European Conference on Computer Vision*, pages 420–436. Springer, 2016.

[122] Rajeev Ranjan, Vishal M Patel, and Rama Chellappa. A deep pyramid deformable part model for face detection. *arXiv preprint arXiv:1508.04389*, 2015.

[123] Rajeev Ranjan, Vishal M Patel, and Rama Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[124] Shifeng Zhang, Xiangyu Zhu, Zhen Lei, Hailin Shi, Xiaobo Wang, and Stan Z Li. S3fd: Single shot scale-invariant face detector. *ICCV*, 2017.

[125] Xudong Sun, Pengcheng Wu, and Steven CH Hoi. Face detection using deep learning: An improved faster rcnn approach. *arXiv preprint arXiv:1701.08289*, 2017.

[126] Chenchen Zhu, Ran Tao, Khoa Luu, and Marios Savvides. Seeing small faces from robust anchor's perspective. *CVPR*, 2018.

[127] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[128] Pablo Arbeláez, Jordi Pont-Tuset, Jonathan T Barron, Ferran Marques, and Jitendra Malik. Multiscale combinatorial grouping. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 328–335, 2014.

[129] Bharat Singh, Hengduo Li, Abhishek Sharma, and Larry S Davis. R-fcn-3000 at 30fps: Decoupling detection and classification. *CVPR*, 2018.

[130] Yancheng Bai, Yongqiang Zhang, Mingli Ding, and Bernard Ghanem. Finding tiny faces in the wild with generative adversarial network. *CVPR. IEEE*, 2018.

[131] Cheng Chi, Shifeng Zhang, Junliang Xing, Zhen Lei, Stan Z Li, and Xudong Zou. Selective refinement network for high performance face detection. *arXiv preprint arXiv:1809.02693*, 2018.

[132] Jian Li, Yabiao Wang, Changan Wang, Ying Tai, Jianjun Qian, Jian Yang, Chengjie Wang, Jilin Li, and Feiyue Huang. Dsfd: dual shot face detector. *arXiv preprint arXiv:1810.10220*, 2018.

[133] Xu Tang, Daniel K Du, Zeqiang He, and Jingtuo Liu. Pyramidbox: A context-assisted single shot face detector. *ECCV*, 2018.

[134] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[135] Sikandar Amin and Fabio Galasso. Geometric proposals for faster r-cnn. In *Advanced Video and Signal Based Surveillance (AVSS), 2017 14th IEEE International Conference on*, pages 1–6. IEEE, 2017.

[136] Pouya Samangouei, Mahyar Najibi, Larry Davis, and Rama Chellappa. Face-magnet: Magnifying feature maps to detect small faces. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 122–130, 2018.

[137] Mahyar Najibi, Bharat Singh, and Larry S Davis. Autofocus: Efficient multi-scale inference. *ICCV*, 2019.

[138] Karel Lenc and Andrea Vedaldi. R-CNN minus R. *CoRR*, abs/1506.06981, 2015.

[139] Joao Carreira, Pulkit Agrawal, Katerina Fragkiadaki, and Jitendra Malik. Human pose estimation with iterative error feedback. *arXiv preprint arXiv:1507.06550*, 2015.

[140] Andrea Vedaldi, Varun Gulshan, Manik Varma, and Andrew Zisserman. Multiple kernels for object detection. In *ICCV*, pages 606–613, 2009.

[141] Antonio Torralba, Kevin P Murphy, and William T Freeman. Contextual models for object detection using boosted random fields. In *Advances in neural information processing systems*, pages 1401–1408, 2004.

[142] Christoph H Lampert, Matthew B Blaschko, and Thomas Hofmann. Beyond sliding windows: Object localization by efficient subwindow search. In *CVPR*, pages 1–8, 2008.

[143] Bastian Leibe, Ales Leonardis, and Bernt Schiele. *An implicit shape model for combined object categorization and segmentation*. Springer, 2006.

[144] Subhrajyoti Maji and Jagannath Malik. Object detection using a max-margin hough transform. In *CVPR*, pages 1038–1045, 2009.

[145] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *CVPR*, pages 1–8, 2008.

[146] Pedro F Felzenszwalb, Ross B Girshick, and David McAllester. Cascade object detection with deformable part models. In *CVPR*, pages 2241–2248, 2010.

[147] Hossein Azizpour and Ivan Laptev. Object detection using strongly-supervised deformable part models. In *ECCV*, pages 836–849, 2012.

[148] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov. Scalable object detection using deep neural networks. In *CVPR*, pages 2147–2154, 2014.

[149] Christian Szegedy, Alexander Toshev, and Dumitru Erhan. Deep neural networks for object detection. In *Advances in Neural Information Processing Systems*, pages 2553–2561, 2013.

[150] Donggeun Yoo, Sunggyun Park, Joon-Young Lee, Anthony S Paek, and In So Kweon. Attentionnet: Aggregating weak directions for accurate object detection. In *ICCV*, pages 2659–2667, 2015.

[151] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Region-based convolutional networks for accurate object detection and segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 38(1):142–158, 2016.

[152] Derek Hoiem, Yodsawalai Chodpathumwan, and Qieyun Dai. Diagnosing error in object detectors. In *ECCV*, pages 340–353, 2012.

[153] Zhixiang Ren, Shenghua Gao, Liang-Tien Chia, and Ivor Wai-Hung Tsang. Region-based saliency detection and its application in object recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 24(5):769–779, 2014.

[154] Michael Donoser, Martin Urschler, Martin Hirzer, and Horst Bischof. Saliency driven total variation segmentation. In *ICCV*, pages 817–824, 2009.

[155] Luca Marchesotti, Claudio Cifarelli, and Gabriela Csurka. A framework for visual saliency detection with applications to image thumbnailing. In *ICCV*, pages 2232–2239, 2009.

[156] Denis Simakov, Yaron Caspi, Eli Shechtman, and Michal Irani. Summarizing visual data using bidirectional similarity. In *CVPR*, pages 1–8, 2008.

[157] Gayoung Lee, Yu-Wing Tai, and Junmo Kim. Deep saliency with encoded low level distance map and high level features. *CVPR*, 2016.

[158] Guanbin Li and Yizhou Yu. Visual saliency based on multiscale deep features. In *CVPR*, pages 5455–5463, 2015.

[159] Nian Liu and Junwei Han. Dhsnet: Deep hierarchical saliency network for salient object detection. In *CVPR*, pages 678–686, 2016.

[160] Lijun Wang, Huchuan Lu, Xiang Ruan, and Ming-Hsuan Yang. Deep networks for saliency detection via local estimation and global search. In *CVPR*, pages 3183–3192, 2015.

[161] Rui Zhao, Wanli Ouyang, Hongsheng Li, and Xiaogang Wang. Saliency detection by multi-context deep learning. In *CVPR*, pages 1265–1274, 2015.

[162] Jianming Zhang, Stan Sclaroff, Zhe Lin, Xiaohui Shen, Brian Price, and Radomír Měch. Unconstrained salient object detection via proposal subset optimization. In *CVPR*, 2016.

[163] Ye Luo, Junsong Yuan, Ping Xue, and Qi Tian. Saliency density maximization for object detection and localization. In *ACCV*, pages 396–408, 2010.

[164] Bongwon Suh, Haibin Ling, Benjamin B Bederson, and David W Jacobs. Automatic thumbnail cropping and its effectiveness. In *UIST*, pages 95–104, 2003.

[165] Roberto Valenti, Nicu Sebe, and Theo Gevers. Image saliency by isocentric curvedness and color. In *ICCV*, pages 2185–2192. IEEE, 2009.

[166] Peng Wang, Jingdong Wang, Gang Zeng, Jie Feng, Hongbin Zha, and Shipeng Li. Salient object detection for searched web images via global saliency. In *CVPR*, pages 3194–3201, 2012.

[167] Jie Feng, Yichen Wei, Litian Tao, Chao Zhang, and Jian Sun. Salient object detection by composition. In *ICCV*, pages 1028–1035, 2011.

[168] Parthipan Siva, Chris Russell, Tao Xiang, and Lourdes Agapito. Looking beyond the image: Unsupervised learning for object saliency and detection. In *CVPR*, pages 3238–3245, 2013.

[169] João Carreira and Cristian Sminchisescu. CPMC: automatic object segmentation using constrained parametric min-cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(7):1312–1328, 2012.

[170] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. Measuring the objectness of image windows. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(11):2189–2202, 2012.

[171] Christian Scharfenberger, Steven Lake Waslander, John S. Zelek, and David A. Clausi. Existence detection of objects in images for robot vision using saliency histogram features. In *CRV*, pages 75–82, 2013.

[172] Jianming Zhang, Shugao Ma, Mehrnoosh Sameki, Stan Sclaroff, Margrit Betke, Zhe L. Lin, Xiaohui Shen, Brian L. Price, and Radomír Mech. Salient object subitizing. In *CVPR*, pages 4045–4054, 2015.

[173] Federico Perazzi, Philipp Krähenbühl, Yael Pritch, and Alexander Hornung. Saliency filters: Contrast based filtering for salient region detection. In *CVPR*, pages 733–740, 2012.

[174] Chuan Yang, Lihe Zhang, Huchuan Lu, Xiang Ruan, and Ming-Hsuan Yang. Saliency detection via graph-based manifold ranking. In *CVPR*, pages 3166–3173, 2013.

[175] Ming Cheng, Niloy J Mitra, Xumin Huang, Philip HS Torr, and Song Hu. Global contrast based salient region detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(3):569–582, 2015.

[176] Xiaohui Li, Huchuan Lu, Lihe Zhang, Xiang Ruan, and Ming-Hsuan Yang. Saliency detection via dense and sparse reconstruction. In *ICCV*, pages 2976–2983, 2013.

[177] Linzhao Wang, Lijun Wang, Huchuan Lu, Pingping Zhang, and Xiang Ruan. Saliency detection with recurrent fully convolutional networks. In *ECCV*, pages 825–841, 2016.

[178] Xi Li, Liming Zhao, Lina Wei, Ming-Hsuan Yang, Fei Wu, Yueting Zhuang, Haibin Ling, and Jingdong Wang. Deepsaliency: Multi-task deep neural network model for salient object detection. *IEEE Trans. Image Processing*, 25(8):3919–3930, 2016.

[179] Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, and Yi Ma. Single-image crowd counting via multi-column convolutional neural network. In *CVPR*, pages 589–597, 2016.

[180] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[181] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[182] C-H Teh and Roland T. Chin. On the detection of dominant points on digital curves. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(8):859–872, 1989.

[183] Yin Li, Xiaodi Hou, Christian Koch, James M Rehg, and Alan L Yuille. The secrets of salient object segmentation. In *CVPR*, pages 280–287, 2014.

[184] Qiong Yan, Li Xu, Jianping Shi, and Jiaya Jia. Hierarchical saliency detection. In *CVPR*, 2013.

[185] Lingyun Zhang, Matthew H Tong, Tim K Marks, Honghao Shan, and Garrison W Cottrell. Sun: A bayesian framework for saliency using natural statistics. *Journal of Vision*, 8(7):32–32, 2008.

[186] Jonathan Harel, Christof Koch, Pietro Perona, et al. Graph-based visual saliency. In *NIPS*, volume 1, page 5, 2006.

[187] Jiwhan Kim, Dongyoon Han, Yu-Wing Tai, and Junmo Kim. Salient region detection via high-dimensional color transform. In *CVPR*, pages 883–890, 2014.

[188] Huaizu Jiang, Jingdong Wang, Zejian Yuan, Yang Wu, Nanning Zheng, and Shipeng Li. Salient object detection: A discriminative regional feature integration approach. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2083–2090, 2013.

[189] Jimei Yang, Brian Price, Scott Cohen, Zhe Lin, and Ming-Hsuan Yang. Patchcut: Data-driven object segmentation via local shape transfer. In *CVPR*, pages 1770–1778, 2015.

[190] Radhakrishna Achanta, Sheila Hemami, Francisco Estrada, and Sabine Susstrunk. Frequency-tuned salient region detection. In *CVPR*, pages 1597–1604, 2009.