ABSTRACT

Title of dissertation: DETECTING OBJECTS AND ACTIONS WITH DEEP LEARNING

Bharat Singh, Doctor of Philosophy, 2018

Dissertation directed by: Professor Larry S. Davis
University of Maryland, College Park

Deep learning based visual recognition and localization is one of the pillars of computer vision and is the driving force behind applications like self-driving cars, visual search, video surveillance, augmented reality, to name a few. This thesis identifies key bottlenecks in state-of-the-art visual recognition pipelines which use convolutional neural networks and proposes effective solutions to push their limits. A few shortcomings of convolutional neural networks are, lack of scale invariance which poses a challenge for tasks like object detection, fixed structure of the network which restricts their usage when presented with new class labels, and difficulty in modeling long range spatial/temporal dependencies. We provide evidence of these problems and then design effective solutions to overcome them.

In the first part, an analysis of different techniques for recognizing and detecting objects under extreme scale variation is presented. Since small and large objects are difficult to recognize at smaller and larger scales of an image pyramid respectively, we present a novel training scheme called Scale Normalization for Image Pyramids (SNIP) which selectively back-propagates the gradients of object instances of different sizes as a function of the image scale. As SNIP ignores gradients of objects at extreme resolutions, following up on this idea, we developed SNIPER (Scale Normalization for Image Pyramids with Efficient

Re-sampling), an algorithm for performing efficient multi-scale training for instance level visual recognition tasks. Instead of processing every pixel in an image pyramid, SNIPER processes context regions (512x512 pixels) around ground-truth instances at the appropriate scale. For background sampling, these context-regions are generated using proposals extracted from a region proposal network trained with a short learning schedule. Hence, the number of chips generated per image during training adaptively changes based on the scene complexity. SNIPER brings training of instance level recognition tasks like object detection closer to the protocol for image classification and suggests that the commonly accepted guideline that it is important to train on high resolution images for instance level visual recognition tasks might not be correct.

Next, we present a real-time large-scale object detector (R-FCN-3000) for detecting thousands of classes where objectness detection and classification are decoupled. To obtain the detection score for an RoI, we multiply the objectness score with the fine-grained classification score. We show that the objectness learned by R-FCN-3000 generalizes to novel classes and the performance increases with the number of training object classes - supporting the hypothesis that it is possible to learn a universal objectness detector. Because of generalized objectness, we can train object detectors for new classes, just with classification data, without even requiring bounding boxes.

Finally, we present a multi-stream bi-directional recurrent neural network for action detection. This was the first deep learning based system which could perform action localization in long videos and it could do it just with RGB data, without requiring any skeletal models or performing intermediate tasks like pose-estimation. Our system uses a tracking algorithm to locate a bounding box around the person, which provides a frame of reference for appearance and motion while suppressing background noise that is not within the bounding box. We train two additional streams on motion and appearance cropped to the tracked bounding box, along with full-frame streams. To model long-term temporal dy-

namics within and between actions, the multi-stream CNN is followed by a bi-directional Long Short-Term Memory (LSTM) layer. We show that our bi-directional LSTM network utilizes about 8 seconds of the video sequence to predict an action label and outperforms state-of-the-art methods on multiple benchmarks.

DETECTING OBJECTS AND ACTIONS WITH DEEP LEARNING

by

Bharat Singh

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
2018

Advisory Committee:
Professor Larry S. Davis, Chair/Advisor
Professor Rama Chellappa
Professor David Jacobs
Professor Tom Goldstein
Professor Ramani Duraiswami

Acknowledgments

I would like to thank my advisor Prof. Larry Davis for supporting me during my stay at University of Maryland. It would have been impossible for me to collaborate with so many bright people on diverse projects if the open research environment in his lab (and UMD in general) did not exist. I also learned to write clear and to the point papers after going over so many edits I received over the course of five years.

I would also like to thank Vlad, who advised me for a major period of my PhD and patiently listened to my immature ideas. With the interactions I had with him, I gained invaluable research experience and most importantly, understood how to reject bad ideas. The students at University of Maryland were the most important resources during my stay. Due credit goes to Abhishek Sharma who introduced me to deep learning and discussed important papers in computer vision and machine learning. I will cherish the discussions we had throughout my life. I also had lots of fun working on conference deadlines with Xintong, Zhe, Mahyar, Navaneeth, Hengduo, Xiyang and Seyed. Having people like Varun (both tall and small), Venkat and Zuxuan in the lab was nice as well.

My mentors during my internships at MERL (Mike Jones, Tim Marks, Oncel Tuzel) and NEC (Samuel Schulter) have played a pivotal role in my research career. Samuel took the pain of explaining every small detail in object detection to me and I would always be grateful to him for that.

I am also grateful to all my committee members. Tom and David's classes helped me in establishing a strong foundation in computer vision and machine learning. It was also fun to collaborate with Tom on research projects.

My PhD was partially supported by the Intelligence Advanced Research Projects Activity (IARPA) grants D17PC00345, D11PC20071 and Office of Naval Research grant N000141612713. I appreciate their support for my research work.

My parents, brother and grandfather were always supportive of me to pursue higher studies abroad. Any number of words are not sufficient to express my gratitude towards them.

Finally, I would like to thank the clusters, Euclid, Deepthought2, Bluecrab, Vulcan and AWS which worked day and night long, without complaining. These are remarkable people and having access to them in a university was a delight.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Visual recognition is a fundamental problem in computer vision and has wide ranging applications like autonomous driving, visual search, surveillance, augmented reality, to name a few. Modern recognition pipelines primarily rely on data driven model based approaches, which are instantiated in the form of convolutional neural networks [54]. These networks distill information from high dimensional input spaces by applying multiple non-linear operators in an iterative fashion to compress the input to a low dimensional semantic space. For different problems like object detection, semantic segmentation, action detection, problem specific architectural changes are made to enable appropriate information flow through the network cascade for generating meaningful semantic representations towards the end.

A major bottleneck in convolutional neural networks is that they are not invariant to scale. For example, if we change the input size of the image or video, the semantic features which will be generated at the end of the network would be different. Another bottleneck is that the models are not flexible. By flexible, it means that after training the network on 1,000 classes if we want to add another class, we need to re-train the network again. This makes it impractical to use them in many applications where online learning is needed. Another failure mode arises while modeling long range spatial/temporal dependencies. This is because these networks aggregate contextual information by applying successive convolution and pooling operations to increase their receptive field. Although in theory their receptive field grows, in practice it does not happen [65]. This is similar to the visual cor-

tex and if we want to focus on two small objects which are far away simultaneously, it gets very difficult. In our mind, we retain information about other parts of the image or video in our memory cells which helps us in processing long term contextual information. In this thesis, we provide evidence of these problems by designing controlled experiments and then propose effective solutions to alleviate them.

In the first part of this thesis, we show that convolutional neural networks are brittle and do not even generalize to minor changes, like upsampling artifacts. Therefore, it is essential to ensure that the data distribution during training and inference is as close as possible. By performing an empirical analysis on tasks like object detection, we also show that convolutional neural networks have a limited receptive field, just like our visual cortex or depth of field in a camera because of which they have difficulty in recognizing objects of different sizes. Based on this observation, we propose a novel training algorithm called scale normalization for image pyramids, which selectively back propagates gradients of objects of different sizes as a function of the image scale. Since we can ignore gradients at extreme resolutions of the image pyramid, we can also process the image pyramid much more efficiently during training.

In the second part, to generalize convolutional neural network based object detectors to novel classes, we propose to decouple detection and classification. The idea is to learn a generalized objectness measure which is trained on thousands of classes and can be used to localize objects agnostic of their class. To perform detection, we multiply the generalized objectness measure with the boundary agnostic classification scores. As the classification layer can be trained without bounding boxes and deep features learned from the large bank of classes can be used as a feature representation for classification, we only need to learn a linear classifier for novel classes which can be trained in seconds.

Finally, we propose to employ LSTM layers to model long term temporal dependencies for tasks like actions detection. These layers have an explicit memory cell which helps in

retaining contextual information efficiently. To model long term spatial dependencies, we propose a multi-stream network which captures object centric and contextual information. We show that the memory cells and the multi-stream structure are indeed effective via well designed ablation studies on action detection benchmarks. Finally, we quantitatively evaluate the extent of temporal information retained by LSTM layers for performing action detection.

Chapter 2: An Analysis of Scale Invariance in Object Detection - SNIP

An analysis of different techniques for recognizing and detecting objects under extreme scale variation is presented. Scale specific and scale invariant design of detectors are compared by training them with different configurations of input data. By evaluating the performance of different network architectures for classifying small objects on ImageNet, we show that CNNs are not robust to changes in scale. Based on this analysis, we propose to train and test detectors on the same *scales* of an image-pyramid. Since small and large objects are difficult to recognize at smaller and larger scales respectively, we present a novel training scheme called Scale Normalization for Image Pyramids (SNIP) which selectively back-propagates the gradients of object instances of different sizes as a function of the image scale. On the COCO dataset, our single model performance is 45.7% and an ensemble of 3 networks obtains an mAP of 48.3%. We use off-the-shelf ImageNet-1000 pre-trained models and only train with bounding box supervision. Our submission won the Best Student Entry in the COCO 2017 challenge.

2.1 Introduction

Deep learning has fundamentally changed how computers perform image classification and object detection. In less than five years, since AlexNet [51] was proposed, the top-5 error on ImageNet classification [17] has dropped from 15% to 2% [41]. This is super-human level performance for image classification with 1000 classes. On the other hand,

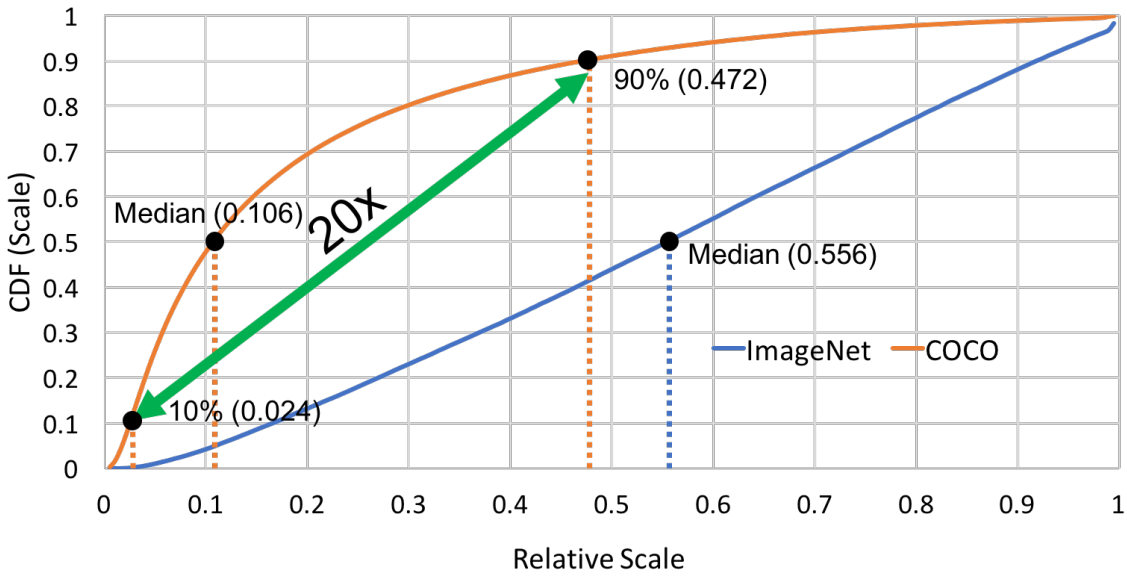

Figure 2.1: Fraction of RoIs in the dataset vs scale of RoIs relative to the image.

the mAP of the best performing detector [44] (which is only trained to detect 80 classes) on COCO [60] is only 62% – even at 50% overlap. Why is object detection so much harder than image classification?

Large scale variation across object instances, and especially, the challenge of detecting very small objects stands out as one of the factors behind the difference in performance. Interestingly, the median scale of object instances relative to the image in ImageNet (classification) vs COCO (detection) are 0.554 and 0.106 respectively. Therefore, most object instances in COCO are smaller than 1% of image area! To make matters worse, the scale of the smallest and largest 10% of object instances in COCO is 0.024 and 0.472 respectively (resulting in scale variations of almost 20 times!); see Fig. 2.1. This variation in scale which a detector needs to handle is enormous and presents an extreme challenge to the scale invariance properties of convolutional neural networks. Moreover, differences in the scale of object instances between classification and detection datasets also results in a large *domain-shift* while fine-tuning from a pre-trained classification network. In this chapter, we first provide evidence of these problems and then propose a training scheme called

Scale Normalization for Image Pyramids which leads to a state-of-the-art object detector on COCO.

To alleviate the problems arising from scale variation and small object instances, multiple solutions have been proposed. For example, features from the layers near to the input, referred to as shallow(er) layers, are combined with deeper layers for detecting small object instances [4, 32, 57, 63, 93], dilated/deformable convolution is used to increase receptive fields for detecting large objects [14, 15, 82, 114], independent predictions at layers of different resolutions are used to capture object instances of different scales [6, 55, 113], context is employed for disambiguation [24, 117, 118], training is performed over a range of scales [14, 15, 34] or, inference is performed on multiple scales of an image pyramid and predictions are combined using non-maximum suppression [5, 14, 15, 83].

While these architectural innovations have significantly helped to improve object detection, many important issues related to training remain unaddressed:

- Is it critical to upsample images for obtaining good performance for object detection? Even though the typical size of images in detection datasets is 480x640, why is it a common practice to up-sample them to 800x1200? Can we pre-train CNNs with smaller strides on low resolution images from ImageNet and then fine-tune them on detection datasets for detecting small object instances?

- When fine-tuning an object detector from a pre-trained image classification model, should the resolution of the training object instances be restricted to a tight range (from 64x64 to 256x256) after appropriately re-scaling the input images, or should all object resolutions (from 16x16 to 800x1000, in the case of COCO) participate in training after up-sampling input images?

We design controlled experiments on ImageNet and COCO to seek answers to these questions. In Section 2.3, we study the effect of scale variation by examining the perfor-

mance of existing networks for ImageNet classification when images of different scales are provided as input. We also make minor modifications to the CNN architecture for classifying images of different scales. These experiments reveal the importance of up-sampling for small object detection. To analyze the effect of scale variation on object detection, we train and compare the performance of scale-specific and scale invariant detector designs in Section 2.5. For scale-specific detectors, variation in scale is handled by training separate detectors - one for each scale range. Moreover, training the detector on similar scale object instances as the pre-trained classification networks helps to reduce the domain shift for the pre-trained classification network. But, scale-specific designs also reduce the number of training samples per scale, which degrades performance. On the other hand, training a single object detector with all training samples makes the learning task significantly harder because the network needs to learn filters for detecting object instances over a wide range of scales.

Based on these observations, in Section 2.6 we present a novel training paradigm, which we refer to as Scale Normalization for Image Pyramids (SNIP), that benefits from reducing scale-variation during training but without paying the penalty of reduced training samples. Scale-invariance is achieved using an image-pyramid (instead of a scale-invariant detector), which contains normalized input representations of object instances in one of the scales in the image-pyramid. To minimize the domain shift for the classification network during training, we only back-propagate gradients for RoIs/anchors that have a resolution close to that of the pre-trained CNN. Since we train on each scale in the pyramid with the above constraint, SNIP effectively utilizes all the object instances available during training. The proposed approach is generic and can be plugged into the training pipeline of different problems like instance-segmentation, pose-estimation, spatio-temporal action detection - wherever the "objects" of interest manifest large scale variations.

Contrary to the popular belief that deep neural networks can learn to cope with large

variations in scale given enough training data, we show that SNIP offers significant improvements (3.5%) over traditional object detection training paradigms. Our ensemble with a Deformable-RFCN backbone obtains an mAP of 69.7% at 50% overlap, which is an improvement of 7.4% over the state-of-the-art on the COCO dataset.

2.2 Related Work

Scale space theory [61, 109] advocates learning representations that are invariant to scale and the theory has been applied to many problems in the history of computer vision [7, 9, 33, 53, 57, 64, 77]. For problems like object detection, pose-estimation, instance segmentation etc., learning scale invariant representations is critical for recognizing and localizing objects. To detect objects at multiple scales, many solutions have been proposed.

The deeper layers of modern CNNs have large strides (32 pixels) that lead to a very coarse representation of the input image, which makes small object detection very challenging. To address this problem, modern object detectors [9, 14, 82] employ dilated/atrous convolutions to increase the resolution of the feature map. Dilated/deformable convolutions also preserve the weights and receptive fields of the pre-trained network and do not suffer from degraded performance on large objects. Up-sampling the image by a factor of 1.5 to 2 times during training and up to 4 times during inference is also a common practice to increase the final feature map resolution [14, 15, 34]. Since feature maps of layers closer to the input are of higher resolution and often contain complementary information (wrt. conv5), these features are either combined with shallower layers (like conv4, conv3) [4, 57, 78, 78] or independent predictions are made at layers of different resolutions [6, 63, 113]. Methods like SDP [113], SSH [69] or MS-CNN [6], which make independent predictions at different layers, also ensure that smaller objects are trained on higher resolution layers (like conv3) while larger objects are trained on lower resolution layers (like

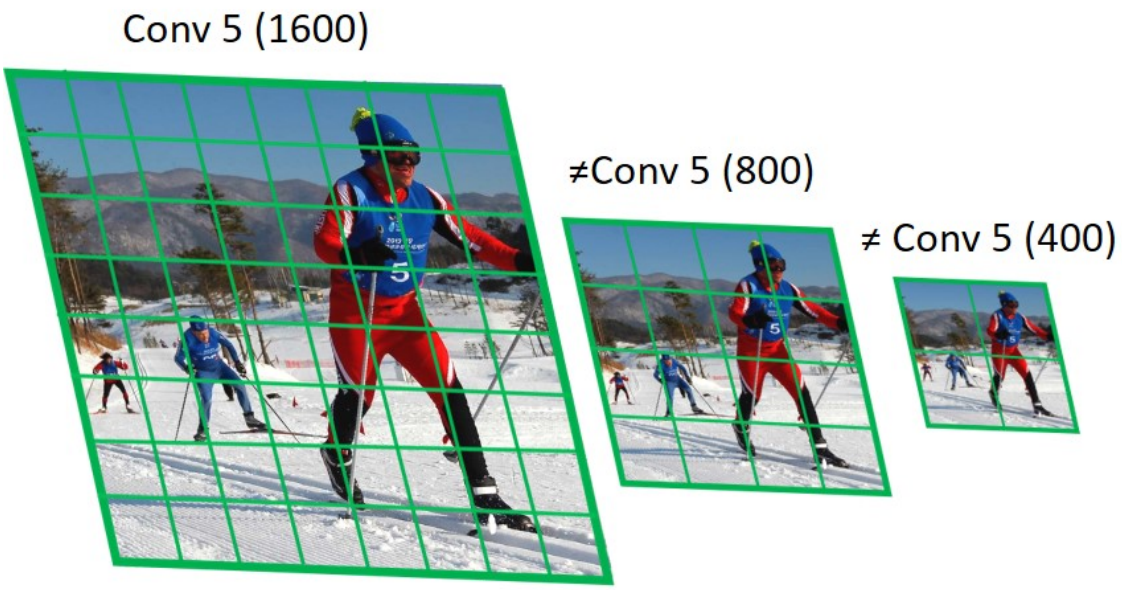

Figure 2.2: The same layer convolutional features at different scales of the image are different and map to different semantic regions in the image at different scales.

conv5). This approach offers better resolution at the cost of high-level semantic features which can hurt performance.

Methods like FPN, Mask-RCNN, RetinaNet [32, 57, 59], which use a pyramidal representation and combine features of shallow layers with deeper layers at least have access to higher level semantic information. However, if the size of an object was 25x25 pixels then even an up-sampling factor of 2 during training,will scale the object to only 50x50 pixels. Note that typically the network is pre-trained on images of resolution 224x224. Therefore, the high level semantic features (at conv5) generated even by feature pyramid networks will not be useful for classifying small objects (a similar argument can be made for large objects in high resolution images). Hence, combining them with features from shallow layers would not be good for detecting small objects, see Fig. 2.2. Although feature pyramids efficiently exploit features from all the layers in the network, they are not an attractive alternative to an image pyramid for detecting very small/large objects.

Recently, a pyramidal approach was proposed for detecting faces [42] where the gradients of all objects were back-propagated after max-pooling the responses from each scale. Different filters were used in the classification layers for faces at different scales. This approach has limitations for object detection because training data per class in object detection is limited and the variations in appearance, pose etc. are much larger compared to face detection. We observe that adding scale specific filters in R-FCN for each class hurts performance for object detection. In [83], an image pyramid was generated and max-out [29] was used to select features from a pair of scales closer to the resolution of the pre-trained dataset during inference. A similar inference procedure was also proposed in SPPNet and Fast-RCNN [26, 33]: however, standard multi-scale training (described in Section 5) was used. We explore the design space for *training* scale invariant object detectors and propose to selectively back-propagate gradients for samples close to the resolution of the pre-trained network.

2.3 Image Classification at Multiple Scales

In this section we study the effect of domain shift, which is introduced when different resolutions of images are provided as input during training and testing. We perform this analysis because state-of-the-art detectors are typically trained at a resolution of 800x1200 pixels [1], but inference is performed on an image pyramid, including higher resolutions like 1400x2000 for detecting small objects [5, 14, 15].

Naïve Multi-Scale Inference: Firstly, we obtain images at different resolutions, 48x48, 64x64, 80x80, 96x96 and 128x128, by down-sampling the original ImageNet database. These are then up-sampled to 224x224 and provided as input to a CNN architecture trained on 224x224 size images, referred to as CNN-B (see Fig. 2.3). Fig. 2.4 (a) shows the top-1

[1] original image resolution is typically 480x640

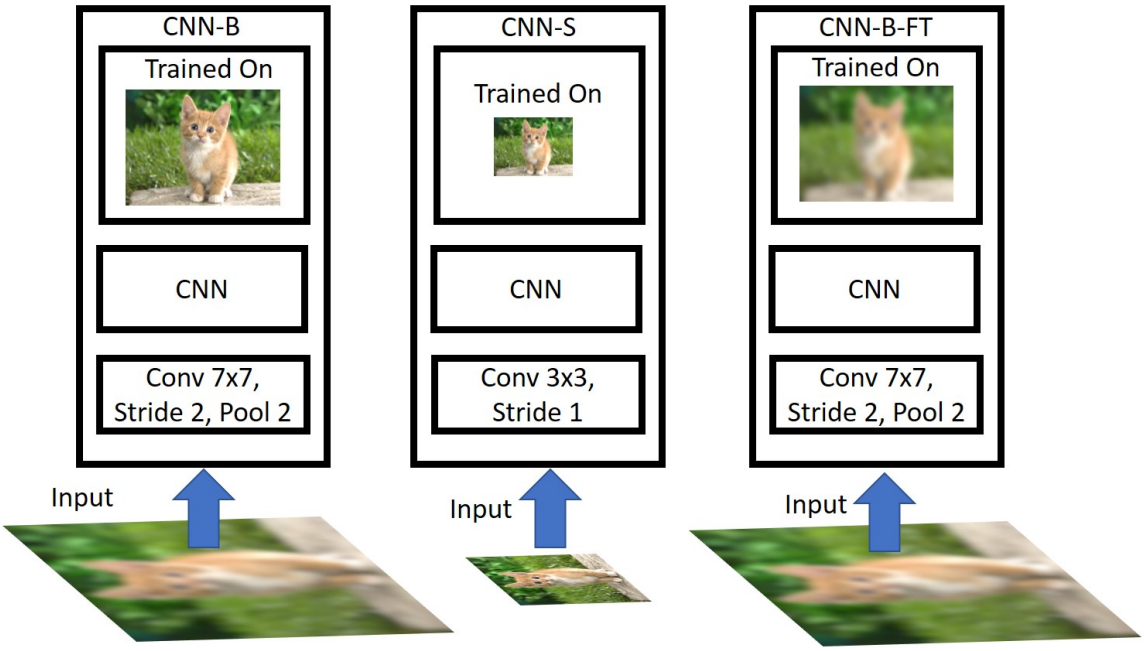

Figure 2.3: Both CNN-B and CNN-B-FT are provided an upsampled low resolution image as input. CNN-S is provided a low resolution image as input. CNN-B is trained on high resolution images. CNN-S is trained on low resolution images. CNN-B-FT is pre-trained on high resolution images and fine-tuned on upsampled low-resolution images. ResNet-101 architecture is used.

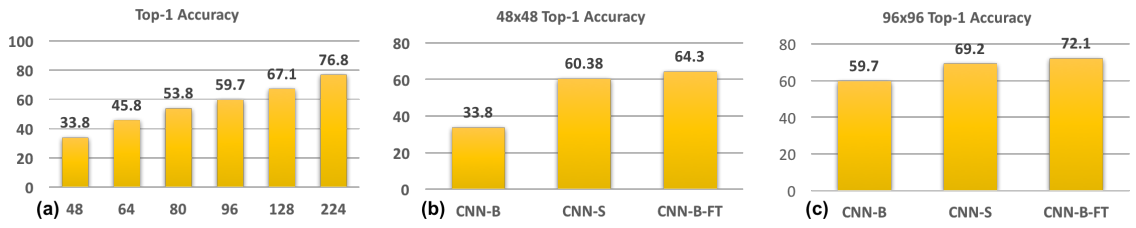

Figure 2.4: All figures report accuracy on the validation set of the ImageNet classification dataset. We upsample images of resolution 48,64,80 etc. and plot the Top-1 accuracy of the pre-trained ResNet-101 classifier in figure (a). Figure (b,c) show results for different CNNs when the original image resolution is 48,96 pixels respectively.

accuracy of CNN-B with a ResNet-101 backbone. We observe that as the difference in resolution between training and testing images increases, so does the drop in performance. Hence, testing on resolutions on which the network was not trained is clearly sub-optimal, at least for image classification.

Resolution Specific Classifiers: Based on the above observation, a simple solution for improving the performance of detectors on smaller objects is to pre-train classification networks with a different stride on ImageNet. After-all, network architectures which obtain best performance on CIFAR10 [50] (which contains small objects) are different from ImageNet. The first convolution layer in ImageNet classification networks has a stride of 2 followed by a max pooling layer of stride 2x2, which can potentially wipe out most of the image signal present in a small object. Therefore, we train ResNet-101 with a stride of 1 and 3x3 convolutions in the first layer for 48x48 images (CNN-S, see Fig. 2.3), a typical architecture used for CIFAR. Similarly, for 96x96 size images, we use a kernel of size 5x5 and stride of 2. Standard data augmentation techniques such as random cropping, color augmentation, disabling color augmentation after 70 epochs are used to train these networks. As seen in Fig. 2.4, these networks (CNN-S) perform significantly better than CNN-B. Therefore, it is tempting to pre-train classification networks with different architectures for low resolution images and use them for object detection for low resolution objects.

Fine-tuning High-Resolution Classifiers: Yet another simple solution for small object detection would be to fine-tune CNN-B on up-sampled low resolution images to yield CNN-B-FT (Fig. 2.3). The performance of CNN-B-FT on up-sampled low-resolution images is better than CNN-S, Fig. 2.4. This result empirically demonstrates that the filters learned on high-resolution images can be useful for recognizing low-resolution images as well. Therefore, instead of reducing the stride by 2, it is better to up-sample images 2 times and then fine-tune the network pre-trained on high-resolution images.

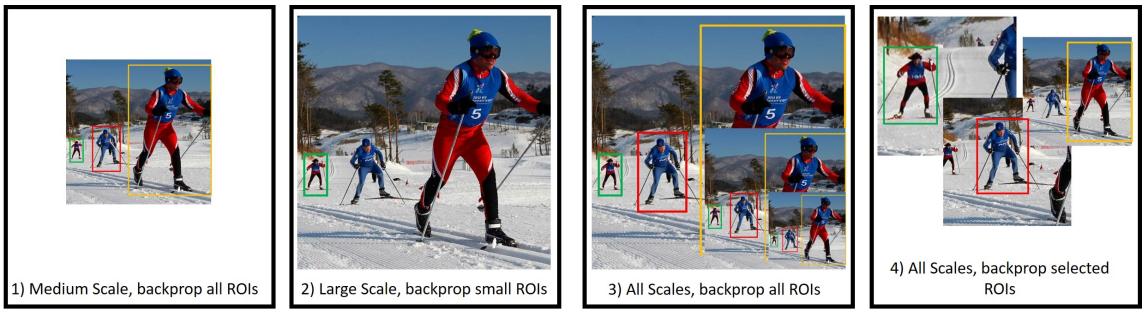

Figure 2.5: Different approaches for providing input for training the classifier of a proposal based detector.

While training object detectors, we can either use different network architectures for classifying objects of different resolutions or use the a single architecture for all resolutions. Since pre-training on ImageNet (or other larger classification datasets) is beneficial and filters learned on larger object instances help to classify smaller object instances, up-sampling images and using the network pre-trained on high resolution images should be better than a specialized network for classifying small objects. Fortunately, existing object detectors up-sample images for detecting smaller objects instead of using a different architecture. Our analysis supports this practice and compares it with other alternatives to emphasize the difference.

2.4 Background

In the next section, we discuss a few baselines for detecting small objects. We briefly describe the Deformable-RFCN [15] detector which will be used in the following analysis. D-RFCN obtains the best single model results on COCO and is publicly available, so we use this detector.

Deformable-RFCN is based on the R-FCN detector [14]. It adds deformable convolutions in the conv5 layers to adaptively change the receptive field of the network for creating scale invariant representations for objects of different scales. At each convolutional feature

map, a lightweight network predicts offsets on the 2D grid, which are spatial locations at which spatial sub-filters of the convolution kernel are applied. The second change is in Position Sensitive RoI Pooling. Instead of pooling from a fixed set of bins on the convolutional feature map (for an RoI), a network predicts offsets for each position sensitive filter (depending on the feature map) on which Position Sensitive RoI (PSRoI)-Pooling is performed.

For our experiments, proposals are extracted at a single resolution (after upsampling) of 800x1200 using a publicly available Deformable-RFCN detector. It has a ResNet-101 backbone and is trained at a resolution of 800x1200. 5 anchor scales are used in RPN for generating proposals [5]. For classifying these proposals, we use Deformable-RFCN with a ResNet-50 backbone without the Deformable Position Sensitive RoIPooling. We use Position Sensitive RoIPooling with bilinear interpolation as it reduces the number of filters by a factor of 3 in the last layer. NMS with a threshold of 0.3 is used. Not performing end-to-end training along with RPN, using ResNet-50 and eliminating deformable PSRoI filters reduces training time by a factor of 3 and also saves GPU memory.

2.5 Data Variation or Correct Scale?

The study in section 2.3 confirms that differences in resolutions between the training and testing phase leads to a significant drop in performance. Unfortunately, this difference in resolution is part of the current object detection pipeline - due to GPU memory constraints, training is performed on a lower resolution (800x1200) than testing (1400x2000) (note that original resolution is typically 640x480). This section analyses the effect of image resolution, the scale of object instances and variation in data on the performance of an object detector. We train detectors under different settings and evaluate them on 1400x2000 images for detecting small objects (less than 32x32 pixels in the COCO dataset) only to

tease apart the factors that affect the performance. The results are reported in Table 2.1.

Training at different resolutions: We start by training detectors that use all the object instances on two different resolutions, 800x1400 and 1400x2000, referred to as 800_{all} and 1400_{all}, respectively, Fig 2.5.1. As expected, 1400_{all} outperformed 800_{all}, because the former is trained and tested on the same resolution i.e. 1400x2000. However, the improvement is only marginal. Why? To answer this question we consider what happens to the medium-to-large object instances while training at such a large resolution. They become too big to be correctly classified! Therefore, training at higher resolutions scales up small objects for better classification, but blows up the medium-to-large objects which degrades performance.

Scale specific detectors: We trained another detector ($1400_{<80px}$) at a resolution of 1400x2000 while ignoring all the medium-to-large objects ($>$ 80 pixels, in the original image) to eliminate the deleterious-effects of extremely large objects, Fig 2.5.2. Unfortunately, it performed significantly worse than even 800_{all}. What happened? We lost a significant source of variation in appearance and pose by ignoring medium-to-large objects (about 30% of the total object instances) that hurt performance more than it helped by eliminating extreme scale objects.

Multi-Scale Training (MST): Lastly, we evaluated the common practice of obtaining scale-invariant detectors by using randomly sampled images at multiple resolutions during training, referred to as MST [2] , Fig 2.5.3. It ensures training instances are observed at many different resolutions, but it also degraded by extremely small and large objects. It performed similar to 800_{all}. We conclude that it is important to train a detector with appropriately scaled objects while capturing as much variation across the objects as possible. In the next section we describe our proposed solution that achieves exactly this and show that it outperforms current training pipelines.

[2]MST also uses a resolution of 480x800

$1400_{<80px}$	800_{all}	1400_{all}	MST	SNIP
16.4	19.6	19.9	19.5	21.4

Table 2.1: mAP on Small Objects (smaller than 32x32 pixels) under different training protocols. MST denotes multi-scale training as shown in Fig. 2.5.3. R-FCN detector with ResNet-50 (see Section 4).

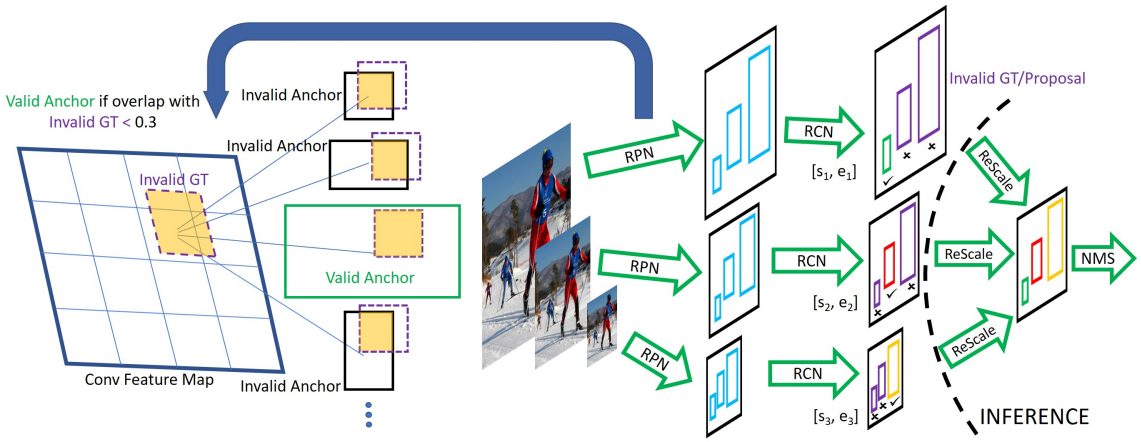

Figure 2.6: SNIP training and inference is shown. Invalid RoIs which fall outside the specified range at each scale are shown in purple. These are discarded during training and inference. Each batch during training consists of images sampled from a particular scale. Invalid GT boxes are used to invalidate anchors in RPN. Detections from each scale are rescaled and combined using NMS.

2.6 Object Detection on an Image Pyramid

Our goal is to combine the best of both approaches i.e. train with maximal variations in appearance and pose while restricting scale to a reasonable range. We achieve this by a novel construct that we refer to as Scale Normalization for Image Pyramids (SNIP). We also discuss details of training object detectors on an image pyramid within the memory limits of current GPUs.

2.6.1 Scale Normalization for Image Pyramids

SNIP is a modified version of MST where only the object instances that have a resolution close to the pre-training dataset, which is typically 224x224, are used for training

the detector. In multi-scale training (MST), each image is observed at different resolutions therefore, at a high resolution (like 1400x2000) large objects are hard to classify and at a low resolution (like 480x800) small objects are hard to classify. Fortunately, each object instance appears at several different scales and some of those appearances fall in the desired scale range. In order to eliminate extreme scale objects, either too large or too small, training is only performed on objects that fall in the desired scale range and the remainder are simply ignored during back-propagation. Effectively, SNIP uses all the object instances during training, which helps capture all the variations in appearance and pose, while reducing the *domain-shift* in the scale-space for the pre-trained network. The result of evaluating the detector trained using SNIP is reported in Table 2.1 - it outperforms all the other approaches. This experiment demonstrates the effectiveness of SNIP for detecting small objects. Below we discuss the implementation of SNIP in detail.

For training the classifier, all ground truth boxes are used to assign labels to proposals. We do not select proposals and ground truth boxes which are outside a specified size range at a particular resolution during training. At a particular resolution i, if the area of an RoI $ar(r)$ falls within a range $[s_i^c, e_i^c]$, it is marked as valid, else it is invalid. Similarly, RPN training also uses all ground truth boxes to assign labels to anchors. Finally, those anchors which have an overlap greater than 0.3 with an invalid ground truth box are excluded during training (i.e. their gradients are set to zero). During inference, we generate proposals using RPN for each resolution and classify them independently at each resolution as shown in Fig 2.6. Similar to training, we do not select detections (not proposals) which fall outside a specified range at each resolution. After classification and bounding-box regression, we use soft-NMS [5] to combine detections from multiple resolutions to obtain the final detection boxes, refer to Fig. 2.6.

The resolution of the RoIs after pooling matches the pre-trained network, so it is easier for the network to learn during fine-tuning. For methods like R-FCN which divide RoIs

into sub-parts and use position sensitive filters, this becomes even more important. For example, if the size of an RoI is 48 pixels (3 pixels in the conv5 feature map) and there are 7 filters along each axis, the positional correspondence between features and filters would be lost.

2.6.2 Sampling Sub-Images

Training on high resolution images with deep networks like ResNet-101 or DPN-92 [10] requires more GPU memory. Therefore, we crop images so that they fit in GPU memory. Our aim is to generate the minimum number of chips (sub-images) of size 1000x1000 which cover all the small objects in the image. This helps in accelerating training as no computation is needed where there are no small objects. For this, we generate 50 randomly positioned chips of size 1000x1000 per image. The chip which covers the maximum number of objects is selected and added to our set of training images. Until all objects in the image are covered, we repeat the sampling and selection process on the remaining objects. Since chips are randomly generated and proposal boxes often have a side on the image boundary, for speeding up the sampling process we snap the chips to image boundaries. We found that, on average, 1.7 chips of size 1000x1000 are generated for images of size 1400x2000. This sampling step is not needed when the image resolution is 800x1200 or 480x640 or when an image does not contain small objects. Random cropping is not the reason why we observe an improvement in performance for our detector. To verify this, we trained ResNet-50 (as it requires less memory) using un-cropped high-resolution images (1400x2000) and did not observe any change in mAP.

Method	AP	AP^S	AP^M	AP^L
Single scale	34.5	16.3	37.2	47.6
MS Test	35.9	19.5	37.3	48.5
MS Train/Test	35.6	19.5	37.5	47.3
SNIP	37.8	21.4	40.4	50.1

Table 2.2: MS denotes multi-scale. Single scale is (800,1200). R-FCN detector with ResNet-50 (as described in Section 4).

2.7 Datasets and Evaluation

We evaluate our method on the COCO dataset. COCO contains 123,000 images for training and evaluation is performed on 20,288 images in test-dev. Since recall for proposals is not provided by the evaluation server on COCO, we train on 118,000 images and report recall on the remaining 5,000 images (commonly referred to as minival set). Unless specifically mentioned, the area of small objects is less than 32x32, medium objects range from 32x32 to 96x96 and large objects are greater than 96x96.

2.7.1 Training Details

We train Deformable-RFCN [15] as our detector with 3 resolutions, (480, 800), (800, 1200) and (1400,2000), where the first value is for the shorter side of the image and the second one is the limit on the maximum size of a side. Training is performed for 7 epochs for the classifier while RPN is trained for 6 epochs. Although it is possible to combine RPN and RCN using alternating training which leads to slight improvement in accuracy [57], we train separate models for RPN and RCN and evaluate their performance independently. This is because it is faster to experiment with different classification architectures after proposals are extracted. We use a warmup learning rate of 0.0005 for 1000 iterations after which it is increased to 0.005. Step down is performed at 4.33 epochs for RPN and

5.33 epochs otherwise. For our baselines which did not involve SNIP, we also evaluated their performance after 8 or 9 epochs but observed that results after 7 epochs were best. For the classifier (RCN), on images of resolution (1400,2000), the valid range in the original image (without up/down sampling) is [0, 80], at a resolution of (800,1200) it is [40, 160] and at a resolution of (480,800) it is [120, ∞]. We have an overlap of 40 pixels over adjacent ranges. These ranges were design decisions made during training, based on the consideration that after re-scaling, the resolution of the valid RoIs does not significantly differ from the resolution on which the backbone CNN was trained. Since in RPN even a one pixel feature map can generate a proposal we use a validity range of [0,160] at (800,1200) for valid ground truths for RPN. For inference, the validity range for each resolution in RCN is obtained using the minival set. Training RPN is fast so we enable SNIP after the first epoch. SNIP doubles the training time per epoch, so we enable it after 3 epochs for training RCN.

2.7.2 Improving RPN

In detectors like Faster-RCNN/R-FCN, Deformable R-FCN, RPN is used for generating region proposals. RPN assigns an anchor as positive only if overlap with a ground truth bounding box is greater than 0.7 [3]. We found that when using RPN at conv4 with 15 anchors (5 scales - 32, 64, 128, 256, 512, stride 16, 3 aspect ratios), only 30% of the ground truth boxes match this criterion when the image resolution is 800x1200 in COCO. Even if this threshold is changed to 0.5, only 58% of the ground truth boxes have an anchor which matches this criterion. Therefore, for more than 40% of the ground truth boxes, an anchor which has an overlap less than 0.5 is assigned as a positive (or ignored). Since we sample the image at multiple resolutions and back-propagate gradients at the relevant resolution

[3] If there does not exist a matching anchor, RPN assigns the anchor with the maximum overlap with ground truth bounding box as positive.

Method	AR	AR^{50}	AR^{75}	0-25	25-50	50-100
Baseline	57.6	88.7	67.9	67.5	90.1	95.6
+ Improved	61.3	89.2	69.8	68.1	91.0	96.7
+ SNIP	64.0	92.1	74.7	74.4	95.1	98.0
DPN-92	65.7	92.8	76.3	76.7	95.7	98.2

Table 2.3: For individual ranges (like 0-25 etc.) recall at 50% overlap is reported because minor localization errors can be fixed in the second stage. First three rows use ResNet-50 as the backbone. Recall is for 900 proposals, as top 300 are taken from each scale.

Method	Backbone	AP	AP^{50}
D-RFCN [5, 15]	ResNet-101	38.4	60.1
Mask-RCNN [32]	ResNext-101 (seg)	39.8	62.3
D-RFCN [5, 15]	ResNet-101 (6 scales)	40.9	62.8
G-RMI [44]	Ensemble	41.6	62.3
D-RFCN	DPN-98	41.2	63.5
D-RFCN + SNIP (RCN)	DPN-98	44.2	65.6
D-RFCN + SNIP (RCN+RPN)	DPN-98	44.7	66.6
Faster-RCNN + SNIP (RPN)	ResNet-101	43.1	65.3
Faster-RCNN + SNIP (RPN+RCN)	ResNet-101	44.4	66.2
D-RFCN + SNIP	ResNet-101 (ResNet-101 proposals)	43.4	65.5
	DPN-98 (with flip)	45.7	67.3
	Ensemble	**48.3**	**69.7**

Table 2.4: Comparison with state-of-the-art detectors. (seg) denotes that segmentation masks were also used. We train on train+val and evaluate on test-dev. Unless mentioned, we use 3 scales and DPN-92 proposals. Ablation for SNIP in RPN and RCN is shown.

only, this problem is alleviated to some extent. We also concatenate the output of conv4 and conv5 to capture diverse features and use 7 anchor scales. A more careful combination of features with predictions at multiple layers like [32, 57] should provide a further boost in performance.

2.7.3 Experiments

First, we evaluate the performance of SNIP on classification (RCN) under the same settings as described in Section 4. In Table 2.2, performance of the single scale model,

multi-scale testing, and multi-scale training followed by multi-scale testing is shown. We use the best possible validity ranges at each resolution for each of these methods when multi-scale testing is performed. Multi-scale testing improves performance by 1.4%. Performance of the detector deteriorates for large objects when we add multi-scale training. This is because at extreme resolutions the receptive field of the network is not sufficient to classify them. SNIP improves performance by 1.9% compared to standard multi-scale testing. Note that we only use single scale proposals common across all three scales during classification for this experiment.

For RPN, a baseline with the ResNet-50 network was trained on the conv4 feature map. Top 300 proposals are selected from each scale and all these 900 proposals are used for computing recall. Average recall (averaged over multiple overlap thresholds, just like mAP) is better for our improved RPN, as seen in Table 3.1. This is because for large objects ($>$ 100 pixels), average recall improves by 10% (not shown in table) for the improved baseline. Although the improved version improves average recall, it does not have much effect at 50% overlap. Recall at 50% is most important for object proposals because bounding box regression can correct minor localization errors, but if an object is not covered at all by proposals, it will clearly lead to a miss. Recall for objects greater than 100 pixels at 50% overlap is already close to 100%, so improving average recall for large objects is not that valuable for a detector. Note that SNIP improves recall at 50% overlap by 2.9% compared to our improved baseline. For objects smaller than 25 pixels, the improvement in recall is 6.3%. Using a stronger classification network like DPN-92 also improves recall. In last two rows of Table 3.3, we perform an ablation study with our best model, which uses a DPN-98 classifier and DPN-92 proposals. If we train our improved RPN without SNIP, mAP drops by 1.1% on small objects and 0.5% overall. Note that AP of large objects is not affected as we still use the classification model trained with SNIP.

Finally, we compare with state-of-the-art detectors in Table 3.3. For these experiments,

we use the deformable position sensitive filters and Soft-NMS. Compared to the single scale deformable R-FCN baseline shown in the first line of Table 3.3, multi-scale training and inference improves overall results by 5% and for small objects by 8.7%! This shows the importance of an image pyramid for object detection. Compared to the best single model method (which uses 6 instead of 3 scales and is also trained end-to-end) based on ResNet-101, we improve performance by 2.5% overall and 3.9% for small objects. We observe that using better backbone architectures further improves the performance of the detector. When SNIP is not used for both the proposals and the classifier, mAP drops by 3.5% for the DPN-98 classifier, as shown in the last row. For the ensemble, DPN-92 proposals are used for all the networks (including ResNet-101). Since proposals are shared across all networks, we average the scores and box-predictions for each RoI. During flipping we average the detection scores and bounding box predictions. Finally, Soft-NMS is used to obtain the final detections. Iterative bounding-box regression is not used. All pre-trained models are trained on ImageNet-1000 and COCO segmentation masks are not used. Faster-RCNN was not used in the ensemble. On 100 images, it takes 90 seconds for to perform detection on a Titan X GPU using a ResNet-101 backbone. Speed can be improved with end-to-end training (we perform inference for RPN and RCN separately).

We also conducted experiments with the Faster-RCNN detector with deformable convolutions. Since the detector does not have position-sensitive filters, it is more robust to scale and performs better for large objects. Training it with SNIP still improves performance by 1.3%. Note that we can get an mAP of 44.4% with a single head faster-RCNN without using any feature-pyramid!

2.8 Conclusion

We presented an analysis of different techniques for recognizing and detecting objects under extreme scale variation, which exposed shortcomings of the current object detection training pipeline. Based on the analysis, a training scheme (SNIP) was proposed to tackle the wide scale spectrum of object instances which participate in training and to reduce the domain-shift for the pre-trained classification network. Experimental results on the COCO dataset demonstrated the importance of scale and image-pyramids in object detection. Since we do not need to back-propagate gradients for large objects in high-resolution images, it is possible to reduce the computation performed in a significant portion of the image. We would like to explore this direction in future work.

Chapter 3: SNIPER: Efficient Multi-Scale Training

We present *SNIPER*, an algorithm for performing efficient multi-scale training in instance level visual recognition tasks. Instead of processing every pixel in an image pyramid, SNIPER processes context regions around ground-truth instances (referred to as *chips*) at the appropriate scale. For background sampling, these context-regions are generated using proposals extracted from a region proposal network trained with a short learning schedule. Hence, the number of chips generated per image during training adaptively changes based on the scene complexity. SNIPER only processes 30% more pixels compared to the commonly used single scale training at 800x1333 pixels on the COCO dataset. But, it also observes samples from extreme resolutions of the image pyramid, like 1400x2000 pixels. As SNIPER operates on resampled low resolution chips (512x512 pixels), it can have a batch size as large as 20 on a single GPU even with a ResNet-101 backbone. Therefore it can benefit from batch-normalization during training without the need for synchronizing batch-normalization statistics across GPUs. SNIPER brings training of instance level recognition tasks like object detection closer to the protocol for image classification and suggests that the commonly accepted guideline that it is important to train on high resolution images for instance level visual recognition tasks might not be correct. Our implementation based on Faster-RCNN with a ResNet-101 backbone obtains an mAP of 47.6% on the COCO dataset for bounding box detection and can process 5 images per second during inference with a single GPU.

3.1 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 [12]. Naturally, it is difficult for us to focus on objects of different scales simultaneously [84]. 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 [32, 82] are trained on a multi-scale image pyramid [62, 96]. 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 was proposed that while performing multi-scale training, it is beneficial to ignore gradients of objects which are of extreme resolutions[96]. For example, when constructing an image pyramid of 3 scales, we should ignore gradients of large objects at large resolutions and small objects at small resolutions. If this is the case, an intuitive question which arises is, do we need to process the entire image at a 3x resolution? Wouldnt 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 computa-

tion, 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 training method proposed in [96] 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.

3.2 Background

Deep learning based object detection algorithms have primarily evolved from the R-CNN detector [27], which started with object proposals generated with an unsupervised algorithm [101], resized these proposals to a canonical 224x224 size image and classified them using a convolutional neural network [54]. This model is scale invariant (with the assumption that CNNs can classify images of a fixed resolution), 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 [26] 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 con-

volutional 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 [26, 33]. 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.

The recently proposed scale normalized training scheme in [96] 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 crop-

ping 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 this chapter) from regions which are likely to cover false positives. To find such regions, we train a lightweight RPN network with a very 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 [121] as well). After adding negative chip sampling, the performance of SNIPER matches the scale normalized training scheme in [96], 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.

3.3 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.

3.3.1 Chip Generation

SNIPER generates chips $\mathscr{C}^i$ at multiple scales $\{s_1, s_2, .., s_i, ..s_n\}$ in the image. For each scale, the image is first resized to width (W_i) and height (H_i). On this canvas, $K \times K$ pixel

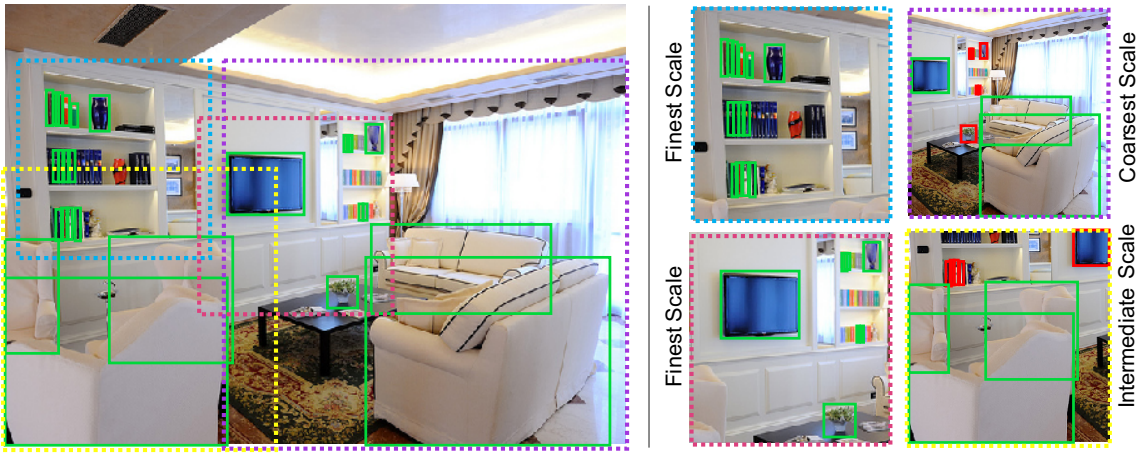

Figure 3.1: SNIPER Positive chip selection . SNIPER adaptively samples context regions (aka chips) based on the presence of objects inside the image. Left side: The image, ground-truth boxes (represented by solid green lines), and the chips in the original image scale(represented by dotted lines). Right 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 in red rectangles.

chips are placed at equal intervals of d pixels (we set d to 32 in this chapter). This leads to a two-dimensional array of chips at each scale.

3.3.2 Positive Chip Selection

For each scale, there is a desired area range $\mathscr{R}^i = [r^i_{min}, r^i_{max}]$, $i \in [1,n]$ which determines which ground-truth boxes/proposals are participate in training for each scale i. The valid list of ground-truth bounding boxes which lie in $\mathscr{R}^i$ are referred to as $\mathscr{G}^i$. Then, chips are greedily selected so that maximum number of valid ground-truth boxes ($\mathscr{G}^i$) 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 $\mathscr{C}^i_{pos}$. For each ground-truth bounding box, there always exists a chip which covers it. Since consecutive $\mathscr{R}^i$ 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

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 3.1. The left side of the figure shows the image with the ground-truth boxes represented by green bounding boxes. The colored dotted rectangles 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.

3.3.3 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 $\mathscr{C}^i_{pos}$. Then, for each scale i, we greedily select all the chips which cover at least M proposals in $\mathscr{R}^i$. This generates a set of negative chips for each scale per image, $\mathscr{C}^i_{neg}$. 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. $\bigcup_{i=1}^{n} \mathscr{C}^i_{neg}$. Figure 3.2 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 $\mathscr{C}^i_{pos}$ 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.

3.3.4 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 ground-truth boxes based on $\mathscr{R}^i$. Instead, proposals which do not fall in $\mathscr{R}^i$ are ignored during training.

Figure 3.2: 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).

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 [82], as in Fast-RCNN. We did not use OHEM [92] 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 [96].

3.3.5 Benefits

For training, we randomly sample chips from the whole dataset for generating a batch. On average, we generate ~ 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 per-

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

formed 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 or training on 128 GPUs 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 [110] 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 [65].

In very large datasets like *OpenImagesV4* 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 3x. 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!

3.4 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.* $\mathscr{R}^i$) are set to ($0,80^2$), (32^2, 150^2), and (120^2, inf) for each of the scales respectively. 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 stepdowns. Therefore, training RPN for 2 epochs is 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

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

as described in [70]. 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 [34, 35] and MobileNetV2. 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 [89], 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 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.

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

3.4.1 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 3.4.2, 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.

3.4.2 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 3.3.3, 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 3.2 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

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 3.2: 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.

when were interested in generating 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 in this chapter we use the aforementioned three scales (See Section 3.4 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 3.2 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. This shows the effectiveness of SNIPER in multi-scale training.

3.4.3 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.

3.4.4 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 [96]. For combining the detections, we use Soft-NMS [5]. We do not perform flipping [117], iterative bounding box regression [25] or mask tightening [62].

3.4.5 Comparison with State-of-the-art

It is difficult to fairly compare different detectors as they differ in backbone architectures (like ResNet [34], ResNext [111], Xception [11]), pre-training data (*e.g.* ImageNet-5k, JFT [37], OpenImages [49]), different structures in the underlying network (*e.g* multi-scale features [58, 69], deformable convolutions [15], heavier heads [75], anchor sizes, path aggregation [62]), test time augmentations like flipping, mask tightening, iterative bounding box regression etc.

Therefore, we compare our results with SNIP[96], which is a recent method for training object detectors on an image pyramid. The results are presented in Table 3.3. Without using batch normalization [45], SNIPER achieves comparable results. While [96] 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 [96], 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 3x faster during training. To the best of our knowledge, for a Faster-RCNN architecture with a ResNet-101 backbone (with deformable convolu-

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 3.3: 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.

tions), 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 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 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 [32], just that we not use FPN 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.

3.5 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 [62, 75, 120] 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 [75]. Like [96], 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 [4, 68, 114] and scale [6, 58, 69, 113]. Considerable importance has been given to leveraging features of different layers of the network and designing architectures for explicitly encoding context/multi-scale information [63, 69, 117, 118] for classification. Our results highlight that context may not be very important for training high performance object detectors.

3.6 Conclusion and Future Work

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 resampled 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. In future, we would like to accelerate multi-scale inference, because a significant portion of the background can be eliminated without performing expensive computation. It would also be interesting to evaluate at what chip resolution does context start to hurt the performance of object detectors.

Chapter 4: R-FCN-3000 at 30fps: Decoupling Detection and Classification

We propose a modular approach towards large-scale real-time object detection by decoupling objectness detection and classification. We exploit the fact that many object classes are visually similar and share parts. Thus, a universal objectness detector can be learned for class-agnostic object detection followed by fine-grained classification using a (non)linear classifier. Our approach is a modification of the R-FCN architecture to learn shared filters for performing localization across different object classes. We trained a detector for 3000 object classes, called R-FCN-3000, that obtains an mAP of 34.9% on the ImageNet detection dataset. It outperforms YOLO-9000 by 18% while processing 30 images per second. We also show that the objectness learned by R-FCN-3000 generalizes to novel classes and the performance increases with the number of training object classes - supporting the hypothesis that it is possible to learn a universal objectness detector.

4.1 Introduction

With the advent of Deep CNNs [36, 51], object-detection has witnessed a quantum leap in the performance on benchmark datasets. It is due to the powerful feature learning capabilities of deep CNN architectures. Within the last five years, the mAP scores on PASCAL [21] and COCO [60] have improved from 33% to 88% and 37% to 73% (at 50% overlap), respectively. While there have been massive improvements on standard

benchmarks with tens of classes [15, 26, 27, 32, 82], little progress has been made towards real-life object detection that requires real-time detection of thousands of classes. Some recent efforts [39, 81] in this direction have led to large-scale detection systems, but at the cost of accuracy. We propose a solution to the large-scale object detection problem that outperforms YOLO-9000 [81] by 18% and can process 30 images per second while detecting 3000 classes, referred to as R-FCN-3000.

R-FCN-3000 is a result of systematic modifications to some of the recent object-detection architectures [14, 15, 59, 63, 80] to afford real-time large-scale object detection. Recently proposed fully convolutional class of detectors [14, 15, 59, 63, 80] compute per-class objectness score for a given image. They have shown impressive accuracy within limited computational budgets. Although fully-convolutional representations provide an efficient [44] solution for tasks like object detection [15], instance segmentation [56], tracking [22], relationship detection [119] etc., they require class-specific sets of filters for each class that prohibits their application for large number of classes. For example, R-FCN [14]/ Deformable-R-FCN [15] requires 49/197 position-specific filters for each class. RetinaNet [59] requires 9 filters for each class for each convolutional feature map. Therefore, such architectures would need hundreds of thousands of filters for detecting 3000 classes, which will make them extremely slow for practical purposes.

The key insight behind the proposed R-FCN-3000 architecture is to decouple *objectness detection* and classification of the detected object so that the computational requirements for localization remain constant as the number of classes increases - see Fig. 4.1. We leverage the fact that many object categories are visually similar and share parts. For example - different breeds of dogs all have common body parts; therefore, learning a different set of filters for detecting each breed is overkill. So, R-FCN-3000 performs object detection (with position-sensitive filters) for a fixed number of *super-classes* followed by fine-grained classification (without position-sensitive filters) within each super-class. The

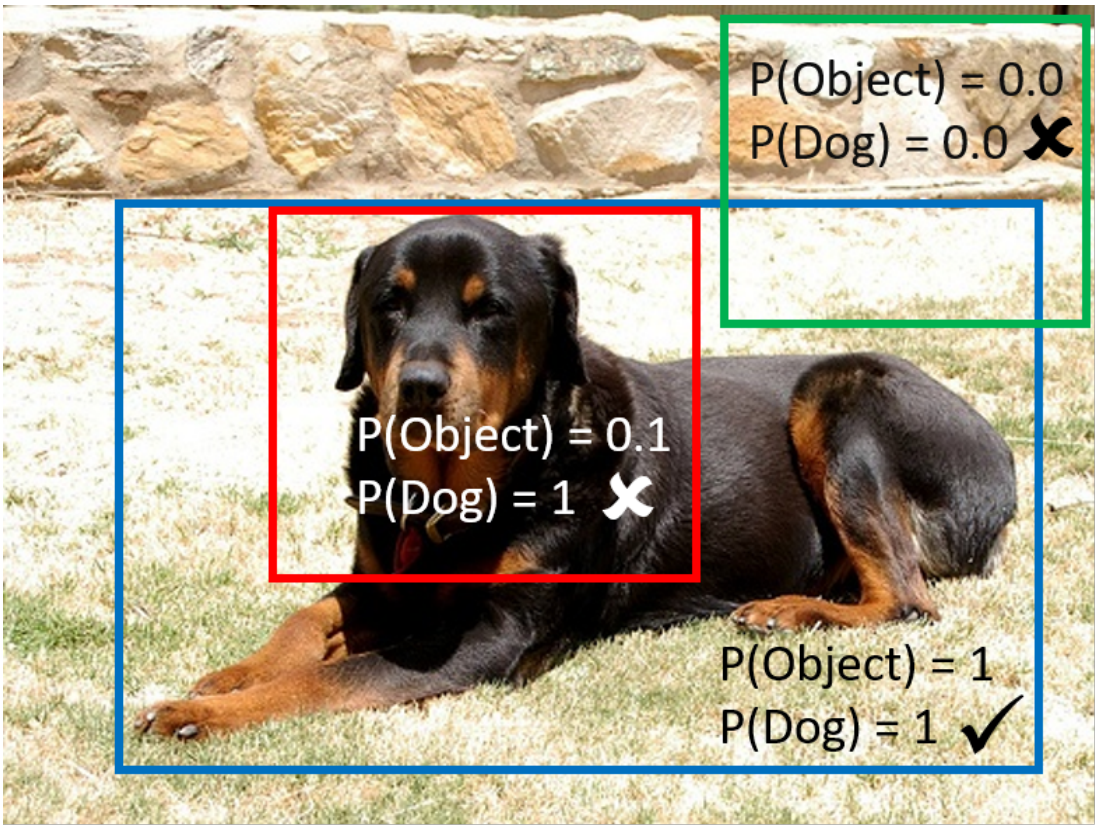

Figure 4.1: We propose to decouple classification and localization by independently predicting objectness and classification scores. These scores are multiplied to obtain a detector.

super-classes are obtained by clustering the deep semantic features of images (2048 dimensional features of ResNet-101 in this case); therefore, we do not require a semantic hierarchy. The fine-grained class probability at a given location is obtained by multiplying the super-class probability with the classification probability of the fine-grained category within the super-class.

In order to study the effect of using super-classes instead of individual object categories, we varied the number of super-classes from 1 to 100 and evaluated the performance on the ImageNet detection dataset. Surprisingly, the detector performs well even with one super-class! This observation indicates that position-sensitive filters can potentially learn to detect universal objectness. It also reaffirms a well-researched concept from the past [1, 2, 101] that objectness is a generic concept and a universal objectness detector can be learned. Indeed, the very first application of deep-learning for object detection [27] used Selective-Search [101] to obtain class-agnostic object proposals and classified them using a deep CNN - fine-tuned AlexNet in this case. R-FCN-3000 exploits the powerful hierarchical representation capacity of deep CNNs to significantly improve universal objectness prediction. Thus, for performing object detection, it suffices to multiply the objectness score of an RoI with the classifiation probability for a given class. This results in a fast detector for thousands of classes, as per-class position sensitive filters are no longer needed. On the PASCAL-VOC dataset, with only our objectness based detector, we observe a 1.5% drop in mAP compared to the deformable R-FCN [15] detector with class-specific filters for all 20 object classes. R-FCN-3000, trained for 3000 classes, obtains an 18% improvement in mAP over the current state-of-the-art large scale object detector (YOLO-9000) on the ImageNet detection dataset. Finally, we also evaluate the generalizability of our objectness detector on *unseen* classes (a zero-shot setting for localization) and observe that the generalization error decreases as we train the objectness detector on larger numbers of classes.

4.2 Related Work

Large scale localization using deep convolutional networks was first performed in [91, 95] which used regression for predicting the location of bounding boxes. Later, RPN [82] was used for localization in ImageNet classification [34]. However, no evaluations were performed to determine if these networks generalize when applied on detection datasets without specifically training on them. Weakly-supervised detection has been a major focus over the past few years for solving large-scale object detection. In [39], knowledge of detectors trained with bounding boxes was transferred to classes for which no bounding boxes are available. The assumption is that it is possible to train object detectors on a fixed number of classes. For a class for which supervision is not available, transformations are learned to adapt the classifier to a detector. Multiple-instance learning based approaches have also been proposed which can leverage weakly supervised data for adapting classifiers to detectors [40]. Recently, YOLO-9000 [81] jointly trains on classification and detection data. When it sees a classification image, classification loss is back-propagated on the bounding box which has the highest probability. It assumes that the predicted box is the ground truth box and uses the difference between other anchors and the predicted box as the objectness loss. YOLO-9000 is fast, as it uses a lightweight network and uses 3 filters per class for performing localization. For performing good localization, just 3 priors are not sufficient.

For classifying and localizing a large number of classes, some methods leverage the fact that parts can be shared across objects categories [73, 74, 88, 99]. Sharing filters for object parts reduces model complexity and also reduces the amount of training data required for learning part-based filters. Even in traditional methods, it has been shown that when filters are shared, they are more generic [99]. However, current detectors like Deformable-R-FCN [15], R-FCN [14], RetinaNet [59] do not share filters (in the final classification layer) across

object categories: because of this, inference is slow when they are applied on thousands of categories. Taking motivation from prior work on sharing filters across object categories, we propose an architecture where filters can be shared across some object categories for large scale object detection.

The extreme version of sharing parts is objectness, where we assume that all objects have something in common. Early in this decade (if not before), it was proposed that objectness is a generic concept and it was demonstrated that only a very few category agnostic proposals were sufficient to obtain high recall [1, 2, 8, 101]. With a bag-of-words feature-representation [53] for these proposals, better performance was shown compared to a sliding-window based part-based-model [23] for object detection. R-CNN [27] used the same proposals for object detection but also applied per-class bounding-box regression to refine the location of these proposals. Subsequently, it was observed that per-class regression was not necessary and a class-agnostic regression step is sufficient to refine the proposal position [14]. Therefore, if the regression step is class agnostic, and it is possible to obtain a reasonable objectness score, a simple classification layer should be sufficient to perform detection. We can simply multiply the objectness probability with the classification probability to make a detector! Therefore, in the extreme case, we set the number of super-classes to one and show that we can train a detector which obtains an mAP which is very close to state-of-the-art object detection architectures [14].

4.3 Background

This section provides a brief introduction of Deformable R-FCN [15] which is used in R-FCN-3000. In R-FCN [14], *Atrous* convolution [9] is used in the conv5 layer to increase the resolution of the feature map while still utilizing the pre-trained weights from the ImageNet classification network. In Deformable-R-FCN [15], the atrous convolution is

replaced by a deformable convolution structure in which a separate branch predicts offsets for each pixel in the feature map, and the convolution kernel is applied after the offsets have been applied to the feature-map. A region proposal network (RPN) is used for generating object proposals, which is a two layer CNN on top of the conv4 features. Efficiently implemented local convolutions, referred to as position sensitive filters, are used to classify these proposals.

4.4 Large Scale Fully-Convolutional Detector

This section describes the process of training a large-scale object detector. We first explain the training data requirements followed by discussions of some of the challenges involved in training such a system - design decisions for making training and inference efficient, appropriate loss functions for a large number of classes, mitigating the domain-shift which arises when training on classification data.

4.4.1 Weakly Supervised vs. Supervised?

Obtaining an annotated dataset of thousands of classes is a major challenge for large scale detection. Ideally, a system that can learn to detect object instances using partial image level tags (class labels) for the objects present in training images would be preferable because large-scale training data is readily available on the internet in this format. Since the setting with partial annotations is very challenging, it is commonly assumed that labels are available for all the objects present in the image. This is referred to as the *weakly supervised* setting. Unfortunately, explicit boundaries of objects or atleast bounding-boxes are required as supervision signal for training accurate object detectors. This is the *supervised* setting. The performance gap between supervised and weakly supervised detectors is large - even 2015 object detectors [34] were better by 40% on the PASCAL VOC 2007 dataset

compared to recent weakly supervised detectors [18]. This gap is a direct result of insufficient learning signal coming from weak supervision and can be further explained with the help of an example. For classifying a dog among 1000 categories, only body texture or facial features of a dog may be sufficient and the network need not learn the visual properties of its tail or legs for correct classification. Therefore, it may never learn that legs or tail are parts of the dog category, which are essential to obtain accurate boundaries.

On one hand, the huge cost of annotating bounding boxes for thousands of classes under settings similar to popular detection datasets such as PASCAL or COCO makes it prohibitively expensive to collect and annotate a large-scale detection dataset. On the other hand, the poor performance of weakly supervised detectors impedes their deployment in real-life applications. Therefore, we ask - is there a middle ground that can alleviate the cost of annotation while yielding accurate detectors? Fortunately, the ImageNet database contains around 1-2 objects per image; therefore, the cost of annotating the bounding boxes for the objects is only a few seconds compared to several minutes in COCO [60]. It is because of this reason that the bounding boxes were also collected while annotating ImageNet! A potential downside of using ImageNet for training object detectors is the loss of variation in scale and context around objects available in detection datasets, but we do have access to the bounding-boxes of the objects. Therefore, a natural question to ask is, how would an object detector perform on "detection" datasets if it were trained on classification datasets with bounding-box supervision? We show that careful design choices with respect to the CNN architecture, loss function and training protocol can yield a large-scale detector trained on the ImageNet classification set with significantly better accuracy compared to weakly supervised detectors.

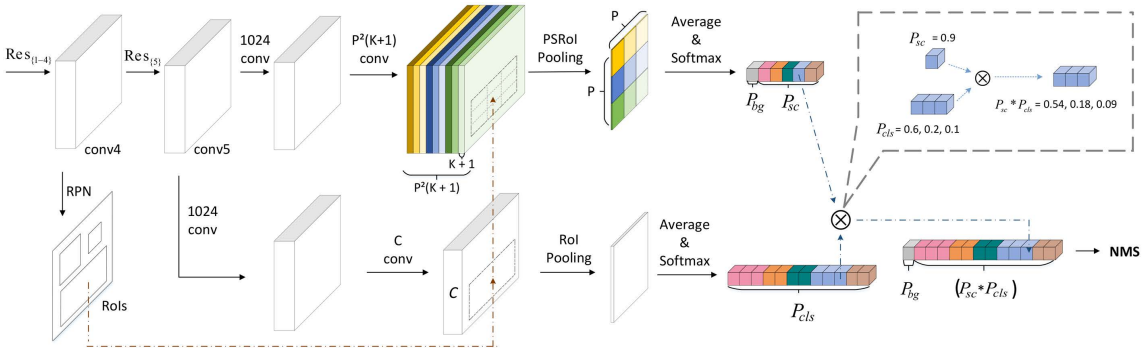

Figure 4.2: R-FCN-3000 first generates region proposals which are provided as input to a super-class detection branch (like R-FCN) which jointly predicts the detection scores for each super-class (sc). A class-agnostic bounding-box regression step refines the position of each RoI (not shown). To obtain the semantic class, we do not use position-sensitive filters but predict per class scores in a fully convolutional fashion. Finally, we average pool the per-class scores inside the RoI to get the classification probability. The classification probability is multiplied with the super-class detection probability for detecting 3000 classes. When K is 1, the super-class detector predicts objectness.

4.4.2 Super-class Discovery

Fully convolutional object detectors learn class-specific filters based on scale & aspect-ratio [59] or in the form of position sensitive filters [14, 15] for each class. Therefore, when the number of classes become large, it becomes computationally in-feasible to apply these detectors. Hence, we ask is it necessary to have sets of filters for each class or can they be shared across visually similar classes? In the extreme case - can detection be performed using just a foreground/background detector and a classification network? To obtain visually similar sets of objects for which position-sensitive filters can be shared, objects should have similar visual appearances. We obtain the j^{th} object-class representation, x_j, by taking the average of 2048-dimensional feature-vectors (x_j^i), from the final layer of ResNet-101, for the all the samples belonging to the j^{th} object-class in the ImageNet classification dataset (validation set). Super-classes are then obtained by applying K-means clustering on $\{x_j : j \in \{1, 2, \ldots C\}\}$, where C is the number of object-classes, to obtain K super-class clusters.

4.4.3 Architecture

First, RPN is used for generating proposals, as in [15]. Let the set of individual object-classes the detector is being trained on be $\mathscr{C}, |\mathscr{C}| = C$, and the set of super-classes (SC) be $\mathscr{K}, |\mathscr{K}| = K$. For each super-class k, suppose we have $P \times P$ position-sensitive filters, as shown in Fig 4.2. On the conv5 feature, we first apply two independent convolution layers as in R-FCN for obtaining detection scores and bounding-box regression offsets. On each of these branches, after a non-linearity function, we apply position sensitive filters for classification and bounding-box regression. Since we have K super-classes and $P \times P$ filters per super-class, there are $(K+1) \times P \times P$ filters in the classification branch (1 more for background) and $P \times P \times 4$ filters in the bounding-box regression branch as this branch is class-agnostic. After performing position-sensitive RoI pooling and averaging the predictions in each bin, we obtain predictions of the network for classification and localization. To get the super-class probability, softmax function over K super-classes is used and predictions from the localization branch are directly added to get the final position of the detection. These two branches help detect the super-classes which are represented by each cluster k. For obtaining fine-grained class information, we employ a two layer CNN on the conv5 feature map, as shown in Fig . 4.2. A softmax function is used on the output of this layer for obtaining the final class probability. The detection and classification probabilities are multiplied to obtain the final detection score for each object-class. This architecture is shown in Fig. 4.2. Even though there are other challenges such as entailment, cover, equivalence etc. [16, 66] which are not correctly modelled with the softmax function, the Top-1 accuracy even on the ImageNet-5000 classification dataset is greater than 67% [10]. So, we believe these are not the bottlenecks for detecting a few thousand classes.

4.4.4 Label Assignment

Labels are assigned exactly the same way as fast-RCNN [26] for the K super-classes on which detection is performed. Let C be the total number of object-classes and let k_i and c_j denote the i^{th} super-class and j^{th} sub-class in k_i, then $k_i = \{c_1, c_2, ..., c_j\}$ and $\sum_{i=1}^{K} |k_i| = C$. For detecting super-class k_i, an RoI is assigned as positive for super-class k_i if it has an intersection over union (IoU) greater than 0.5 with any of the ground truth boxes in k_i, otherwise it is marked as background (label for background class $K+1$ is set to one). For the classification branch (to get the final 3000 classes), only positive RoIs are used for training, i.e. only those which have an IoU greater than 0.5 with a ground truth bounding box. The number of labels for classification is C instead of $K+1$ in detection.

4.4.5 Loss Function

For training the detector, we use online hard example mining (OHEM) [92] as done in [15] and smooth L1 loss for bounding box localization [26]. For fine-grained classification we only use a softmax loss function over C object-classes for classifying the positive bounding boxes. Since the number of positive RoIs are typically small compared to the number of proposals, the loss from this branch is weighted by a factor of 0.05, so that these gradients do not dominate network training. This is important as we train RPN layers, R-FCN classification and localization layers, and fine-grained layers together in a multi-task fashion, so balancing the loss from each branch is important.

4.5 Experiments

In this section, we describe the implementation details of the proposed large-scale object detector and compare against some of the weakly supervised large-scale object detec-

Dataset (ImageNet)	Images	Object Instances
Detection	400,000	764,910
CLS 194	87,577	100,724
CLS 500	121,450	141,801
CLS 1000	191,463	223,222
CLS 2000	403,398	462,795
CLS 3000	925,327	1,061,647

Table 4.1: The number of images and object instances in the ImageNet Detection and different versions of our ImageNet classification (CLS) training set.

tors in terms of speed and accuracy.

4.5.1 Training Data

We train on the ImageNet classification dataset which contains bounding boxes for 3,130 classes. Each class contains at least 100 images in the training set. In the complete dataset, there are 1.2 million images. The detection test set (ILSVRC 2014) of ImageNet contains 194 classes out of the 3,130 classes present in the classification set. Therefore, we present our results on the 194 classes in our experiments (6 classes did not have bounding boxes in the ImageNet classification dataset which were present in the ImageNet detection test set). We also perform experiments on the PASCAL VOC 2007+2012 object detection dataset. We evaluate our models on the VOC 2007 test set.

4.5.2 Implementation Details

For fast training and inference, we train on images of resolution (375x500), where the smaller side is at least 375 pixels and the larger side is a maximum of 500 pixels. Three anchor scales of (64,128,256) pixels are used. At each anchor scale, there are 3 aspect ratios of (1:2), (1:1) and (2:1) for the anchor boxes, hence there are a total of 9 anchors in RPN. We train for 7 epochs. A warm-up learning rate of 0.00002 is used for first 1000 iterations

Method	LSDA [39]	SKT [97]	KDT [101]	Ours
mAP	18.1	20.0	34.3	43.3

Table 4.2: Comparison of our decoupled R-FCN trained on classification data with bounding-box supervision vs. weakly-supervised methods that use a knowledge transfer approach to exploit information from detectors pre-trained on 100 classes on the ImageNet detection set.

and then it is increased to 0.0002. The learning rate is dropped by a factor of 10 after 5.33 epochs. Training is performed on 2 Nvidia P6000 GPUs. When increasing the number of classes beyond 194, we first select classes with the least number of samples (each class still has at least 100 samples) from the classification set. This is done for accelerating our ablation experiments on 500, 1000 and 2000 classes. Statistics of the detection set and classification set with different numbers of classes are shown in Table 4.1. For our analysis, we first train a region proposal network on 3,130 classes separately and extract proposals on the training and test set. Then, deformable R-FCN is trained like fast-RCNN with different numbers of clusters and classes. Multi-scale inference is performed at two scales, (375,500) and (750,1000) and predictions of the two scales are combined using NMS. In all our experiments, a ResNet-50 backbone network is used. On the PASCAL VOC dataset we train under the same settings as [15].

Figure 4.3: The mAP on the 194 classes in the ImageNet detection set is shown as we vary the number of clusters (super-classes). This is shown for 194 class and 1000 class detectors. We also plot the mAP for different number of classes for an objectness based detector.

4.5.3 Comparison with Weakly Supervised Detectors

First, to calibrate our results with existing methods and to highlight the improvement by training on classification data with bounding-box supervision, we compare our method with knowledge transfer based weakly supervised methods. Methods like LSDA [39] and Semantic Knowledge Transfer (SKT) [97] assume that detectors for 100 classes (trained on the ImageNet detection dataset) are available and use semantic similarity between weakly supervised classes and strongly supervised classes to leverage information learned from pre-trained detectors. They evaluate on the remaining 100 classes in the ImageNet detection set. Contemporary work [100] (KDT) also employs a knowledge transfer based approach, albeit with a modern Inception-ResNet-v2 based Faster-RCNN detector. Since these methods leverage classification data and also detection data for other classes, these can be considered as a very loose upper-bound on what a *true weakly-supervised detector*, which does not have any access to bounding boxes, would achieve. Our single scale ResNet-50 based model trained on 194 classes obtains an mAP of 40.5% [1] and after multi-scale testing (2 scales), we obtain an mAP of 43.3%.

We also provide some statistics on the number of images and object instances in the ImageNet detection and ImageNet classification set in Table 4.1. Weakly supervised methods like LSDA [39], SKT [97], KDT [101] use detectors trained on 400,000 instances present in 200,000 images from the detection dataset. This acts as a prior which is used as a basis for adapting the classification network.

[1] 2 classes did not have bounding box annotations in the ImageNet classification training set, so results are on 98 out of 100 classes

Clusters	1	5	25	100	1000
mAP	36	36.7	37.1	37.3	-
Time(ms)	33	33	34	51	231

Table 4.3: The mAP scores for different number of clusters for the 1000 class detector and run-time(in milli-seconds)/image.

Clusters	20	50	100	200	1000
mAP	35.6	35.6	35.6	35.7	36.0
Time(ms)	1	1.5	1.8	2.6	10.1

Table 4.4: The mAP for different number of super-classes in NMS for the 1000 class objectness based detector and the NMS run-time (in milli-seconds).

4.5.4 Speed and Performance

In Table 4.3, we present the speed accuracy trade-off when increasing the number of clusters for the 1000 class detector. The 100 class clustering based detector is 66% slower than the objectness based detector. It was infeasible to train the original detector with 1000 classes, so we only present the run time for this detector. All the speed results are on a P6000 GPU. We also present results when we use different numbers of clusters during NMS. In this process, NMS is performed for a group of visually similar classes together, instead of each class separately. We use the same clustering based grouping of classes. The clusters used during NMS can be different from those which are used when grouping classes for R-FCN as this is only done for accelerating the post-processing step. We present the runtime for NMS (on GPU) for different numbers of clusters in Table 4.4. Note that 10 ms is 33% of the runtime of our detector, and this is only for 1000 classes. Therefore, performing NMS on visually similar classes is a simple way to speed up inference without taking a significant hit in average precision. As mentioned in the title, our 3000 class detector can be applied to more than 30 images per second (on a resolution of 375x500 pixels, minimum side 375, maximum side 500) on an Nvidia P6000 GPU.

Figure 4.4: The objectness, classification and final detection scores against various transformations such as combinations of scaling and translation are shown. These scores are generated by forward propagating an ideal bounding-box RoI (in green) and a transformed bounding-box RoI (in red) through the R-FCN (objectness) and classification branch of the network. The selectiveness of the detector in terms of objectness is clearly visible against the various transformations that lead to poor detection.

4.6 Discussion

In order to better understand the behaviour of the proposed object detection system, we evaluate it while varying the number of clusters and classes under different training and testing dataset conditions. Lastly, we also conduct experiments with unseen classes during training to assess the generalizability capacity of the proposed detector beyond the training classes.

4.6.1 Impact of Number of Classes and Clusters

We present results as we increase the number of classes on the ImageNet detection test set which contains 194 classes in Fig. 4.3 (c). In this experiment, we only use one cluster, hence the position sensitive RoI filters only predict objectness and perform bounding-box regression. The drop in performance typically reduces as we increase the number of classes. For example, there is a drop of 2% as the number of classes is increased from 200 to 500, but from 1000 to 2000, the performance drop is only 0.3%. Even with 3,000 classes, we obtain an mAP of 34.9% which is 15% better than YOLO-9000 which obtains an mAP of 19.9%. Performance of YOLO-9000 drops to 16% when it is evaluated on classes which are not part of the detection set (these are majority of the classes which it detects). Therefore,

we perform better by 19% on classes which are not part of the detection set compared to YOLO-9000. Although we detect 3,000 instead of 9,000 classes, our performance is more than 2 times better than YOLO-9000. Qualitative results for the R-FCN-3000 detector are also shown in Fig. 4.5 on some images from COCO.

To assess the effect of the number of super-classes on performance, we varied the number of super-classes and report the results. All results use a single-scale inference. Fig. 4.3 (a) reports mAP for training/testing on 194 classes from the ImageNet detection dataset and Fig. 4.3 (b) reports mAP for the same 194 classes while training with 1,000 object classes. The drop in performance is merely 1.7% for a detector with only one super-class as compared to 100 super-classes for 194 class training. More interestingly, as the number of training classes are increased to 1,000, the drop is only 1.3%, which is counter-intuitive because one would expect that using more super-classes would be helpful as we increase the number of object classes. In light of these observations, we can conclude that more crucial to R-FCN is learning an *objectness* measure instead of class-specific objectness.

4.6.2 Are Position-Sensitive Filters Per Class Necessary?

To further verify our claim that detection can be modelled as a product of objectness and classification probability, we conduct more experiments on the PASCAL VOC dataset. We train a deformable R-FCN detector, as the baseline, with a ResNet-50 backbone that uses deformable position sensitive filters and obtains an mAP of 79.5%. After training a decoupled network which predicts objectness and performs classification on RoIs, we observe a similar pattern even on this dataset. At a 0.5 overlap, the performance only drops by 1.9% and at 0.7 by 1.5%, Table 4.6. This confirms that our proposed design changes to R-FCN are effective and only marginally deteriorate the mAP of the detector. We show a few visual examples of objectness and classification scores predicted by our class-agnostic

detector in Fig 4.4.

Based on these results, we explore some other alternatives of R-FCN for estimating objectness. First, we just use RPN scores as the objectness measure and classify the proposals with our network (which is a linear classifier). Then, we add a bounding box regression step on the proposals, as they are already class agnostic. These two baselines are significantly worse than R-FCN. The mAP of only RPN is very poor at an overlap of 0.7. Although bounding-box regression provides a boost of 35% at 0.7 overlap, the performance is still 15% worse than R-FCN. Since RPN uses an overlap of 0.7 for assigning positives and 0.3 for assigning negatives, we decided to change these two thresholds to 0.5 and 0.4 respectively, like [59]. We train two versions of RPN, on conv4 and conv5 and present the results. These results show that performance with RPN also improves after changing the overlap criterion and with better features, so other objectness measures could also be an alternative for R-FCN. Results for these experiments are presented in Table 4.5.

Figure 4.5: Detections for classes in the ImageNet3K dataset which are typically not found in common object detection datasets are shown.

Figure 4.6: Objectness scores on images containing unseen object-classes from the ImageNet detection dataset.

$Ov^{0.5}$	$Ov^{0.7}$	C4	C5	BBR	$mAP^{0.5}$	$mAP^{0.7}$
×	✓	✓	×	×	47.3	12.7
×	✓	✓	×	✓	65.1	47.8
✓	×	✓	×	×	52.0	16.0
✓	×	✓	×	✓	66.8	49.7
✓	×	×	✓	×	70.6	44.1
✓	×	×	✓	✓	74.1	56.9

Table 4.5: Results for different versions of RPN scores used for objectness are reported. C4 and C5 denote if RPN is applied on Conv4 or Conv5 feature-map. $Ov^{0.5}$, $Ov^{0.7}$ denotes if the overlap for assigning positives in RPN is 0.5 or 0.7. BBR denotes if bounding box regression of deformable R-FCN is used or not.

Method	$mAP^{0.5}$	$mAP^{0.7}$
D-R-FCN (decoupled)	77.6	63.8
D-R-FCN	79.5	65.3

Table 4.6: Results of D-R-FCN and our decoupled version where the R-FCN classification branch only predicts objectness.

4.6.3 Generalization of Objectness on Unseen Classes

We evaluate the generalization performance of our objectness detector on a held out set of 20 classes. In this experimental setting, we train two objectness detectors - **OB** (objectness baseline), which includes the 20 object classes during training and **GO** (generalized objectness), which does not. For both the settings, the same classifier is used with different objectness detectors. OB and the classifier are trained on 194, 500, 1000, 2000 and 3130 classes and GO on 174, 480, 980, 1980 and 3110 classes. While going from 194 to 500 classes, the number of classes increase significantly but the number of samples do not (see Table 4.1); therefore, the mAP of OB drops by 1.8%. Since more samples help in improving the objectness measure for GO, the performance drop is only marginal (Fig 4.7). Increasing the number of classes to 1000 and 2000 improves the mAP of GO, implying that the improvement in objectness can overshadow the performance drop caused by increasing the number of classes. Fig 4.7 clearly shows that the initial gap of 9.7% in the performance

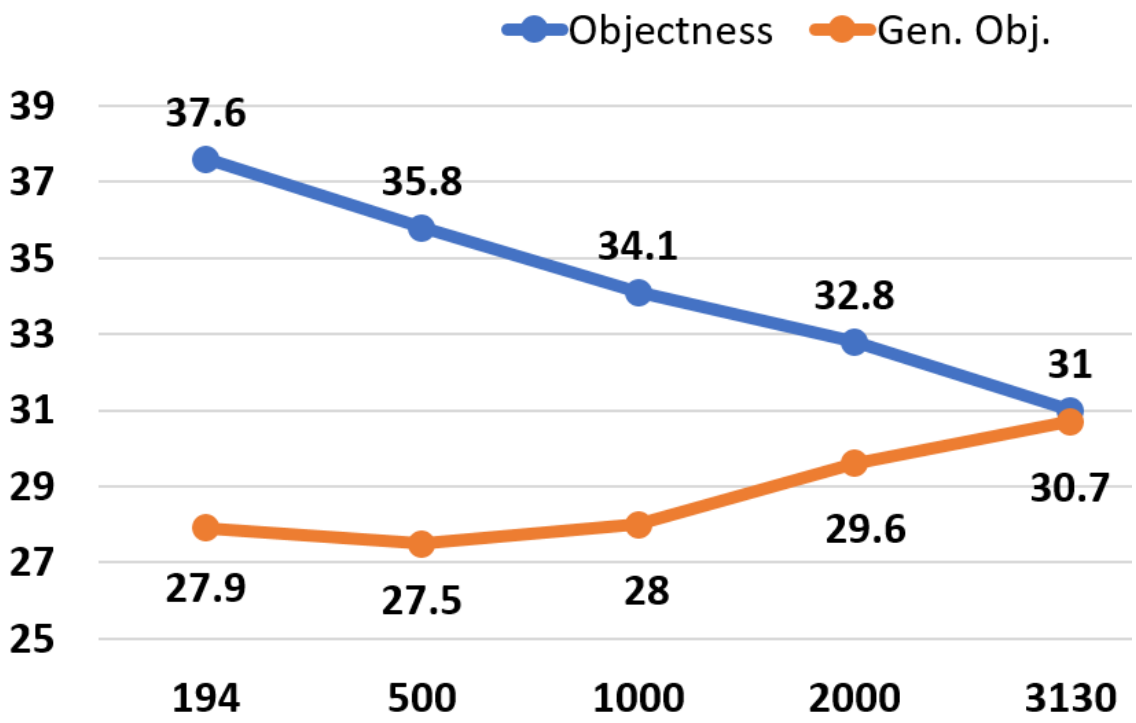

Figure 4.7: The mAP scores on a held out set of 20 classes for Generalized Objectness and Objectness baseline.

Classes	Objectness	Generalized Objectness
20	31	30.7
194	34.9	32

Table 4.7: mAP of Objectness and Generalized Objectness on held out classes in the ImageNet detection set.

drops to 0.3% as the number of classes increase. We also compared **OB** with **GO** when we remove all the 194 classes in ImageNet detection set and present the results in Table 4.7. Note that the performance drop is 3% even after removing 10% of the instances in the dataset (all of which belong to the classes in the test set). It strongly indicates that objectness learned on thousands of classes generalizes to novel unseen classes as well. A few qualitative results for such cases are shown in Fig. 4.6. Note that we did not train on any of these images!

4.7 Conclusion

A modular framework for real-time large-scale object detection is proposed that exploits the visual similarities and part sharing across different object categories. The proposed framework is trained on 3000 classes from the ImageNet classification dataset with bounding-box supervision. R-FCN-3000 outperformed the previous state-of-the-art large-scale detector (YOLO-9000) by 18%, while running at 30fps. We demonstrate that the proposed framework can potentially predict a universal objectness score by using only one set of filters for object vs. background detection. It resulted in a marginal drop of less than 2% compared to the detector which performed *detection* for each class on the PASCAL VOC dataset. Finally, we also show that the objectness learned generalizes to unseen classes and the performance increases with the number of training object classes. It bolsters the hypothesis of the universality of objectness.

This chapter presents significant improvements for large-scale object detection but many questions still remain unanswered. Some promising research questions are - How can we accelerate the classification stage of R-FCN-3000 for detecting 100,000 classes? A typical image contains a limited number object categories - how to use this prior to accelerate inference? What changes are needed in this architecture if we also need to detect objects and their parts? Since it is expensive to label each object instance with all valid classes in every image, can we learn robust object detectors if some objects are not labelled in the dataset?

Chapter 5: A Multi-Stream Bi-Directional Recurrent Neural Network for Fine-Grained Action Detection

We present a multi-stream bi-directional recurrent neural network for fine-grained action detection. Recently, two-stream convolutional neural networks (CNNs) trained on stacked optical flow and image frames have been successful for action recognition in videos. Our system uses a tracking algorithm to locate a bounding box around the person at each frame of the video, which provides a frame of reference for appearance and motion trajectories and also suppresses background noise that is not within the bounding box. In addition to training two streams (flow and image appearance) on full-frame video, we train two additional streams on flow and appearance cropped to the tracked bounding box. Whereas recent methods use stacked optical flow as input to the flow stream of a network, our flow streams takes as input pixel trajectories, in which the displacement values corresponding to a moving scene point are input at the same pixel location across several frames. To model long-term temporal dynamics within and between actions, the multi-stream CNN is followed by a bi-directional Long Short-Term Memory (LSTM) layer. In our experiments, we demonstrate that the LSTM layer improves performance of the multi-stream network by a large margin. We also show that our bi-directional LSTM network utilizes about 8 seconds of the video sequence to predict an action label. We test on two action detection datasets: the MPII Cooking 2 Dataset, and a new Shopping Dataset that we introduce with this chapter. The results demonstrate that our method significantly out-

performs state-of-the-art action detection methods on both datasets.

5.1 Introduction

In this chapter, we present a novel method for detecting actions in videos. *Action detection* refers to the problem of localizing temporally and spatially every occurence of each action from a known set of action classes in a long video sequence. This is in contrast to most of the previous work in video activity analysis, which has focused on the problem of *action recognition* (also called action classification). In action recognition, a temporally segmented clip of a video is given as input, and the task is to classify it as one of N known actions. For action recognition, temporal localization is not required, as each video clip is trimmed to contain precisely the full duration (from start to finish) of one action. Furthermore, action recognition algorithms do not need to consider the case that a presented clip might not contain any of the known actions. In general, the problem of action detection is more difficult than action recognition. However, it is worth overcoming that difficulty, because action detection is also much more relevant to real-world applications.

In this work, we will focus on fine-grained action detection. We use the term *fine-grained* in the same sense as [86] to indicate that the differences among the set of actions to be detected is low. For example, in a cooking scenario, detecting actions from a set that includes similar actions such as chopping, grating, and peeling constitutes fine-grained action detection.

We propose a novel method for fine-grained action detection in long video sequences, based on a Multi-Stream Bi-Directional Recurrent Neural Network (MSB-RNN). We call our neural network *multi-stream* because it begins with a convolutional neural network (CNN) that has four streams: two different streams of information (motion and appearance) for each of two different spatial frames (full-frame and person-centric). The video

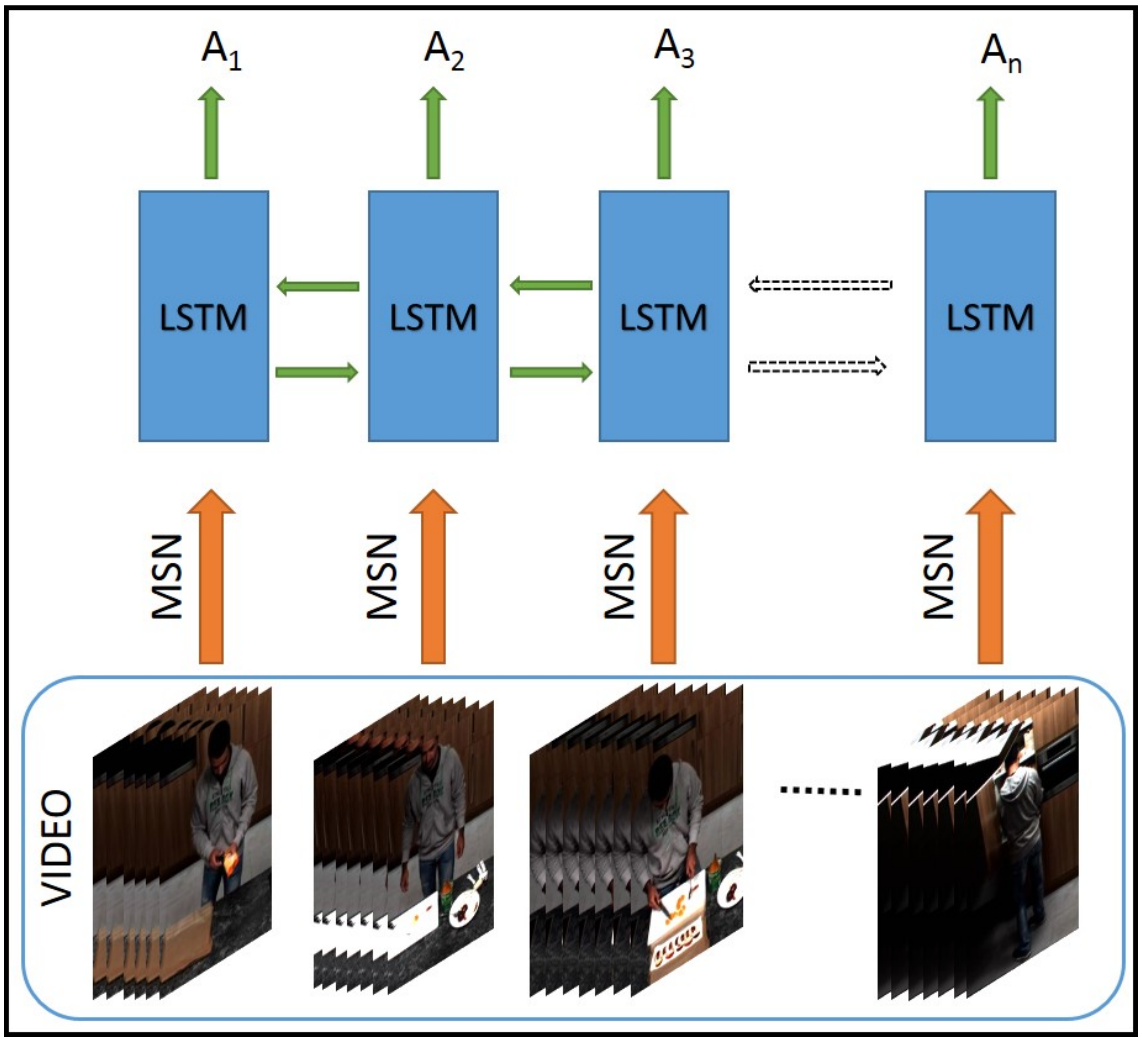

Figure 5.1: Framework for our approach. Short chunks of a video are given to a multi-stream network (MSN) to create a representation for the clip. This representation is then given to a bi-directional LSTM which is used to predict the action label, A_i. Two streams of the multi-stream network compute CNN features on pixel level trajectories and RGB channels respectively. Using a tracker, we use two more streams which only look at a zoomed in region of the video.

that is input to the network is split into a sequence of brief (6-frame-long) chunks. The multi-stream network output is a sequence of high-level representations of these chunks. These are input to bi-directional long short-term memory (LSTM) [31, 38] units to analyze long-term temporal dynamics. Previous deep learning approaches use features that are computed over the full spatial extent of the video frame. We show the importance of using a tracked bounding box around the person to compute features relative to the location of the person, in addition to full-frame features, to provide both location-independent and location-dependent information. Unlike some previous work that represents motion information using a sequence of flow fields [94], we instead use a sequence of corresponding pixel displacements that we call *pixel trajectories*, as illustrated in Figure 5.3. The advantage of pixel trajectories is that the displacements for a moving point in the scene are represented at the same pixel location across several frames. We analyze the relative importance of each of these components using two different datasets. The first is the MPII Cooking 2 Dataset [87], and the second is a new dataset we introduce containing overhead videos of people shopping from grocery-store shelves. We collected this new dataset because there are very few existing datasets that consist of long videos that each contain multiple temporally labeled actions. Our results on the MPII Cooking 2 Dataset represent a significant improvement over the previous state of the art.

Our work contains four novel contributions:

- We demonstrate the effectiveness of a bi-directional LSTM for the action-detection task. It should be noted that although LSTMs have been used before for action recognition and sentence generation, we are the first to use LSTMs for action detection. Furthermore, since our LSTM layer is trained on full-length videos containing multiple actions (not just trimmed clips of individual actions), it can learn interactions among temporally neighboring actions.

- We train a multi-stream convolutional network that consists of two 2-stream networks, demonstrating the importance of using both full-frame and person-centric cropped video.
- We use pixel trajectories rather than stacked optical flow as input to the motion streams, leading to a significant improvement in results.
- We introduce a new action detection dataset.

5.2 Related Work

Early work that can be considered action detection includes methods that detect walking people by analyzing simple appearance and motion patterns [13, 103]. Several algorithms have been proposed since then for detecting simple actions using space time interest points [116], multiple instance learning [43], or part-based models [46, 98]. By adding another dimension (time) to object proposals, action proposals have also been used for detection [47, 115].

Until recently, the standard pipeline for most video analysis tasks such as action recognition, event detection, and video retrieval was to compute hand-crafted features such as Histogram of Oriented Gradients (HOG), Motion Boundary Histogram (MBH), and Histogram of Optical Flow (HOF) along improved dense trajectories [105], create a Fisher vector for each video clip, then perform classification using support vector machines. In fact, shallow architectures using Fisher vectors still give state-of-the-art results for action/activity recognition [76, 87, 106]. Wang et al. [106] showed that results improved when hand-crafted features were replaced by deep features that were computed by convolutional neural networks from an input of images and stacked optical flow along trajectories. In [94], a two-stream network was proposed in which video frames and stacked optical flow

fields (computed over a few frames) were fed to a deep neural network for action recognition. A similar architecture was used for spatial localization of actions [28] in short video clips. However, these networks did not learn long-term sequence information from videos.

Since recurrent neural networks can learn long-term sequence information in a data-driven fashion, they have recently gained traction in the action recognition community [3, 19, 71]. In [3], a 3D convolutional neural network followed by an LSTM classifier was successful at classifying simple actions. LSTMs have shown improved performance over a two-stream network for action recognition [19, 71]. Recently, bi-directional LSTMs were also successful in skeletal action recognition [20]. However, even after using LSTMs, deep learning methods perform only slightly better than fisher vectors built on hand-crafted features for many action recognition tasks [71].

Although substantial progress has been made in action recognition/classification) [76, 87, 94, 106], not as much work has been done in action detection (spatio-temporal localization of actions in longer videos). Recently, it was shown that improving tracking helps action detection in sports videos [108]. In this method, track proposals are generated and then hand-crafted features are computed over these tracks. Since spatial localization is often not too difficult for fine-grained action detection in indoor videos, this approach is closely related to computing dense trajectory features over tracked regions (similar to hand-trajectories in [87]). Using annotations for the objects being interacted with [72, 87] or enforcing the grammar of the high level activity being performed [52, 79] is generally helpful, though these approaches may require learning extra detectors for objects and having prior knowledge about high-level activities.

For fine-grained action detection, extracting trajectories from spatio-temporal regions of interest or using hand-trajectories has shown significantly improved performance [72, 87]. In recent work in generating sentences from images, LSTM networks with attention models [67, 112] learn to focus on salient regions in an image to generate captions for

the image. Since motion and actor location are important clues for knowing where an action is happening, we were inspired by these methods to add our network's two person-centric streams, which capture information from regions of video that are salient due to actor motion.

5.3 Approach

Our framework is shown in Fig. 5.1. First, we train 4 independent convolutional neural networks, each based on the VGG architecture [95], to perform the task of action classification when given as input a single small chunk (6 consecutive frames) of video. Two networks (one each for images and motion trajectories) are trained on chunks of full-frame video, so that the spatial context of the action being performed is preserved. The other two networks (one each for images and motion trajectories) are trained on frames that have been cropped to a tracked bounding box. These cropped frames provide an action with a reference frame, which helps in classifying them. After these four networks have been trained, we learn a fully-connected projection layer on top of all all four fc7 layer outputs, to create a joint representation for these independent streams. This multi-stream network is shown in Figure 5.2. This multi-stream network is provided with full-length video (arranged as a temporal sequence of 6-frame chunks), and the corresponding temporal sequence of outputs of the projection layer is then fed into an LSTM network running in two directions. We use a fully-connected layer on top of the LSTM hidden state, followed by a softmax layer, to obtain an intermediate score corresponding to each action. Finally, the scores for each LSTM are averaged to get action-specific scores.

There are multiple components in an action detection pipeline that are critical for achieving good performance. In this task, we need a model that captures both spatial and long-term temporal information that are present in a video. Person tracks (bounding boxes)

provide a reference frame that make many actions easier to learn by removing location variation from the input representation. Some actions, however, are location dependent. For scenes shot from a static camera such as the ones in our testing datasets; this means these actions always happen at the same location in the image. For example, washing/rinsing would be done near the sink, and opening a door would most likely be performed near a refrigerator or a cupboard. For these reasons, we train two separate deep networks each on pixel trajectories and image frames. The first network is trained on the entire frame to preserve the global spatial context. The second network is trained on cropped boxes from the tracker to reduce background noise and to provide a person-centric reference frame for trajectories and image regions. To capture short-term temporal information, we use pixel trajectories, in which each moving scene point is in positional correspondence with itself across several frames. This alignment enables pixel trajectories to capture much richer motion information than stacked optical flow fields. Since actions can be of any duration, the method uses LSTMs to learn the duration and long-term temporal context of actions in a data-driven fashion. Our results demonstrate that LSTMs are quite effective in learning long-term temporal context for fine-grained action detection.

5.3.1 Tracking for Fine-Grained Action Detection

To provide a bounding box around the person for the location-independent image and motion trajectory streams, any good person tracking algorithm could be used. In this paper, we use a simple state-based tracker to spatially localize actions in a video. Keeping the size of the tracked bounding box fixed, we update its position so that the magnitude of flow inside the box is maximized. If the magnitude is below a threshold, the location is not updated (when the person is not moving, the bounding box is stationary). Location of the bounding box is updated only after a video chunk (6 frames) is processed and

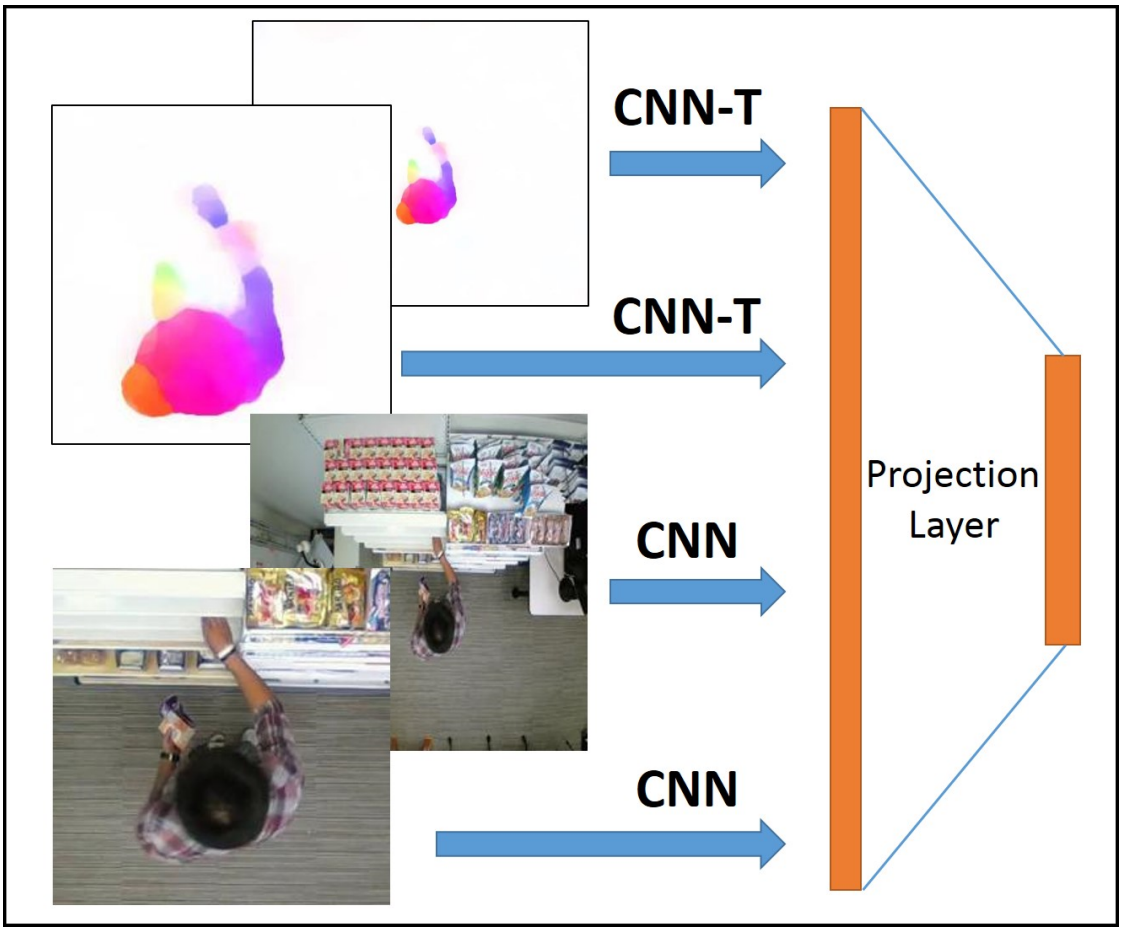

Figure 5.2: Figure depicting the multi-stream network. We use two different streams of information (motion and appearance) for each of two different spatial croppings (full-frame and person-centric) to analyze short segments of video. One network (CNN-T) computes features on pixel level trajectories, while the other one computes features on RGB channels.

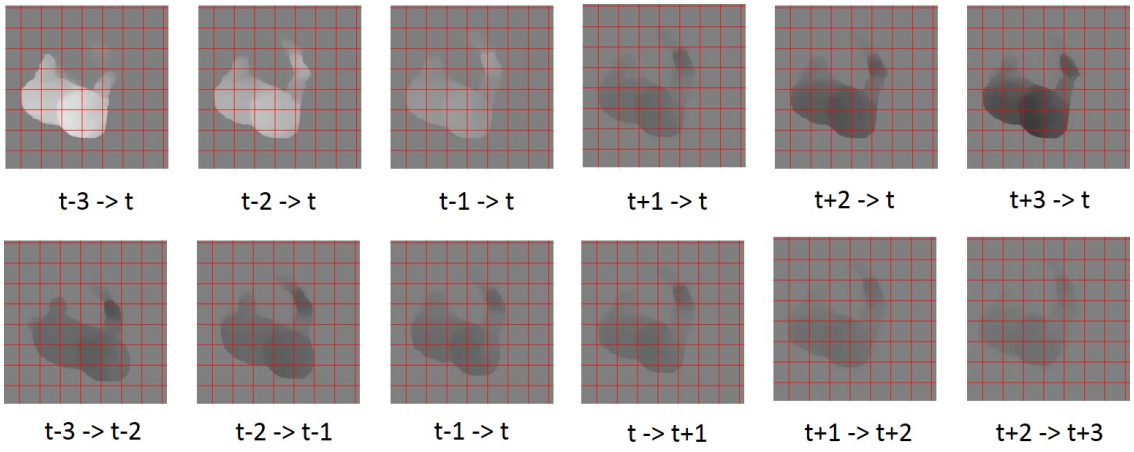

Figure 5.3: The first row shows y-component of optical flow with respect to the center frame in a small video chunk. Note that only the intensity of the images changes, while the spatial layout of the image stays the same. Thus, only a single convolution layer in time is sufficient for learning motion features for a pixel. The second row shows stacked optical flow, where correspondence between pixels is lost. For e.g. the boundary of the head is not at the same grid line in the second row for all frames.

flow/appearance features are computed relative to it. Such an approach can be effectively applied whenever the camera is stationary and we have a reasonable estimate about the size of the actor. This is a practical assumption for many videos taken at retail stores, individual homes, or in a surveillance setting where fine grained action detection is likely to be used. For more difficult tracking situations, a more sophisticated tracker would be needed.

5.3.2 Training of Flow Networks

Stacking optical flow as an input to the deep network has been a standard practice in the literature to train motion based networks [94, 106, 107]. However, in stacked optical flow, the motion vectors corresponding to a particular moving point in the scene (e.g., the tip of a finger) change their pixel location from one frame to the next. Thus, the convolutional neural network needs to learn the spatial movement of optical flow for classifying an action. The complete motion information could be learned by this model at a higher layer, but that

would require more parameters and data to learn. An alternate representation for motion in a sequence of frames is to compute flow from a central frame t to each of the K previous and K subsequent frames. This representation, which we call *pixel trajectories*, is illustrated and compared with stacked optical flow in Figure 5.3. In all $2K$ frames of a pixel trajectory, the flow values from each point to the corresponding point in frame t are all located at the point's location in frame t. As shown in Figure 5.3, in pixel trajectories, only the intensity of the optical flow image changes (its location is fixed). Thus, the network can learn a temporal filter for each pixel more easily than from stacked flow fields.

Now, for each pixel in frame t, we have the complete motion information in a short window of time. To learn motion patterns for each pixel, a $1\times 2K$ convolutional kernel can produce a feature map for movement of each pixel. In contrast, a network layer that inputs stacked optical flow (using, e.g., a $3\times 3\times 2K$ kernel on stacked optical flow) will not be able to learn motion patterns for pixels that have a displacement of more than 3 pixels over $2K$ frames using the first convolutional layer. A similar method was mentioned in [94], but there it yielded slightly worse performance than stacked optical flow, likely because it was applied on unconstrained videos where trajectories are less reliable. For fine grained action detection with a stationary camera, however, we demonstrate that pixel trajectories perform better than stacked flow for both datasets (see Table 5.3).

5.3.3 Training on Long Sequences using Bi-Directional LSTM Network

We provide a brief background of Recurrent Neural Networks and Long Short-Term Memory (LSTM) cells [38]. Given an input sequence, $\mathbf{x} = (x_1, \ldots, x_T)$, a Recurrent Neural Network (RNN) uses a hidden state representation $\mathbf{h} = (h_1, \ldots, h_T)$ so that it can map the input $\mathbf{x}$ to the output sequence $\mathbf{y} = (y_1, \ldots, y_T)$. To compute this representation, it iterates

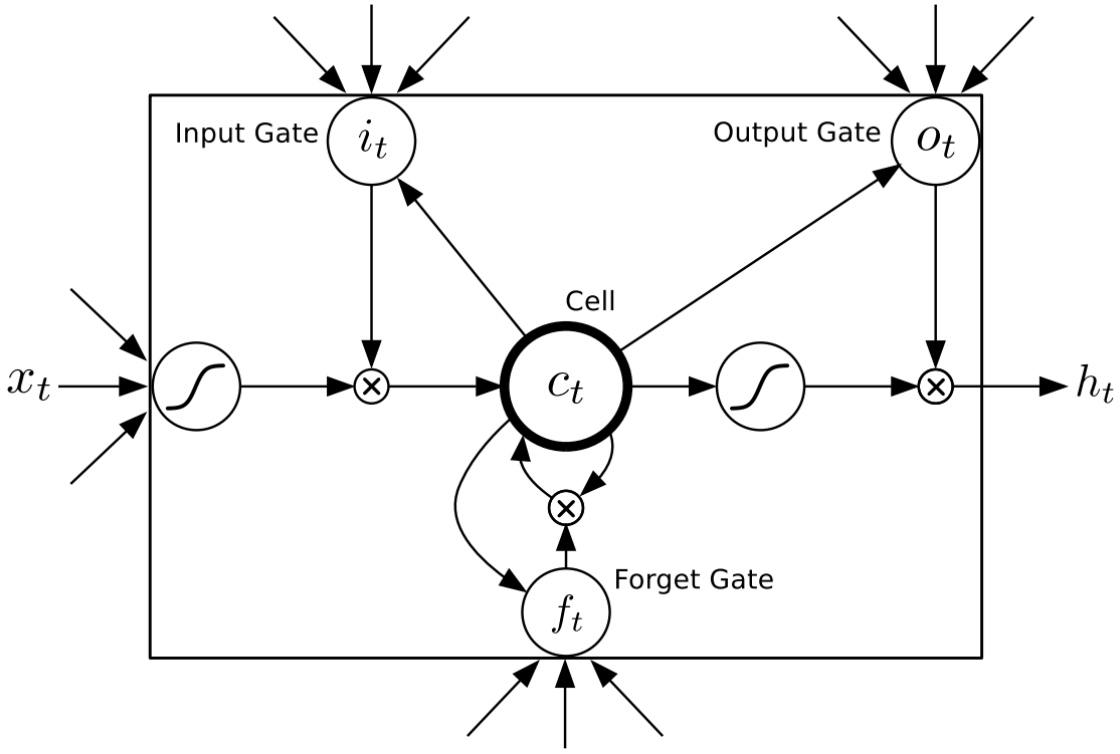

Figure 5.4: Structure of an LSTM cell [30]

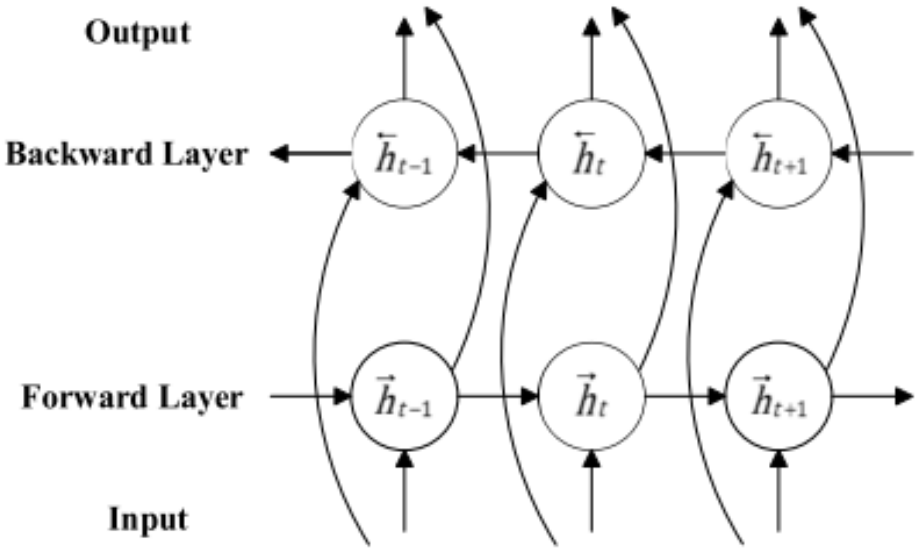

Figure 5.5: Connections depicting architecture of a bi-directional LSTM [30].

through the following recurrence equations:

$$h_t = g(W_{xh}x_t + W_{hh}h_{t-1} + b_h), y_t = g(W_{hz}h_t + b_z)$$

where g is an activation function, W_{xh} is the weight matrix which maps the input to the hidden state, W_{hh} is the transition matrix between hidden states at two adjacent time steps, W_{hz} is a matrix which maps the hidden state h to the output y, and b_h and b_z are bias terms. Unlike hidden Markov models (HMMs), which use discrete hidden state representations, recurrent neural networks use a continuous-space representation for the hidden states. However, it is difficult to train RNNs to learn long-term sequence information, as training is performed by unrolling the network using back-propagation through time. This leads to either a vanishing or exploding gradients problem [38]. To avoid this problem, an LSTM unit (illustrated in Figure 5.4) has a memory cell c_t and a forget gate f_t that helps it to learn when to retain the previous state and when to forget it. This enables an LSTM network to learn long-term temporal information. The weight update equations for an LSTM cell are as follows:

$$\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
g_t &= \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\
c_t &= f_t c_{t-1} + i_t\ g_t \\
h_t &= o_t\ \tanh(c_t)
\end{aligned}$$

where σ is a sigmoid function, tanh is the hyperbolic tangent function, and i_t, f_t, o_t, and c_t are the *input gate*, *forget gate*, *output gate*, and *cell* activation vectors respectively. The

Figure 5.6: Images for different actions in the Shopping Dataset. We show images corresponding to different actions like, 'retract from shelf', 'inspect a product', 'hand in shelf', 'inspect shelf'

forget gate f_t decides when (and which) information should be cleared from the memory cell c_t. The input gate i_t decides when (and which) new information should be incorporated into the memory. The tanh layer g_t generates a candidate set of values which will be added to the memory cell if the input gate allows it. Based on the output of the forget gate f_t, input gate i_t, and the new candidate values g_t, the memory cell c_t is updated. The output gate o_t controls which information in the memory cell should be used as a representation for the hidden state. Finally, the hidden state is represented as a product between a function of the memory cell state and the output gate.

Recently, LSTM architectures for Recurrent Neural Networks have been successful for tasks such as sentence generation from images [19], video to text [102], and in the speech recognition community [30]. Bi-directional recurrent neural networks [90] can also capture long temporal context from both past and future sequences. However, the performance of LSTM networks is still close to fisher vector built over improved dense trajectories for tasks like action recognition [19, 71]. In action recognition datatsets (e.g., UCF 101), video clips are temporally trimmed to start and end precisely at the start and end times of each action, and are generally short in length (e.g., from 2–20 seconds). Hence, in the action recognition task, there is not enough long-term context to be learned in a data-driven fashion. This long-term context could include properties such as the expected duration of an action, which action follows or precedes an action, or even long-term motion patterns that extend beyond

action boundaries. In an action recognition setting, an LSTM network has little access to the longer-term temporal context. However, in fine grained action detection, videos are typically on the order of minutes or hours. Thus, LSTM networks are more suited towards this task as they are designed to model long-term temporal dynamics in a sequence.

Bi-directional LSTM networks [31], illustrated in Figure 5.5, also integrate information from the future as well as the past to make a prediction for each chunk in the video sequence. Therefore, they are expected to be better at predicting the temporal boundaries of an action as compared to a one-directional LSTM. In this work, the forward and backward LSTM networks each give soft-max scores for each action class, and we average these softmax predictions of the two LSTM networks to obtain the score for each action. While training these networks on long sequences, back-propagation through time can only be done up to a fixed number of steps, using a short sequence of chunks. To preserve long-term context, we retain the hidden state of the last element in the previous sequence when training on the subsequent sequence.

5.4 Results

5.4.1 Datasets

We evaluate our method on two datasets: the MPII Cooking 2 Dataset [87], and a Shopping Dataset that we collected and are releasing to the community with the publication of this paper. The MPII Cooking 2 Dataset consists of 273 video sequences that vary in length from 40 seconds to 40 minutes, with a total of 2.8 million frames. The videos are labeled with the start and end times of fine-grained actions from 67 action classes. Actions such as “smell,” “screw open,” and “take out” may be as brief as one half of a second, while other actions such as “grate,” “peel,” and “stir” can last as long as a few minutes. There is also significant intra-class variation in the duration of an action.

Our new Shopping Dataset consists of 96 two-minute videos, shot by a static overhead HD camera, of people shopping from grocery-store shelving units that we set up in a lab space. There are 32 subjects, each of whom is in 3 videos collected on different days. Videos are labeled with the start and end times of fine-grained actions from 5 different action classes: "Reach to Shelf," "Retract from Shelf," "Hand in Shelf," "Inspect Product," and "Inspect Shelf." We divide this dataset into three partitions: 60 training videos, 9 validation videos, and 27 test videos. For each subject, all three videos of that subject are in only one of the three partitions. Although the number of videos in this dataset is less than in MPII Cooking 2, there are many action instances per video, so the number of frames per action class is high ($\sim$ 30,000). In this dataset also, the duration of an action ranges from one-half of a second to on the order of a minute. We show examples of frames this dataset in Figure 5.6. See the Supplementary Materials for sample video from the Shopping Dataset.

5.4.2 Implementation Details

We sample each video at 15 frames per second. We then extract optical flow between each frame (sampled every 6 frames) and its 6 neighboring frames ($K = 3$ each to the left and right). This provides trajectories for each pixel. Epic flow is used to compute optical flow [85], as it gives reliable flow even for large movements. We then use our tracker to obtain bounding boxes for each video. Finally, all full size image frames and cropped frames are re-sized to 256$\times$256. Pixel trajectories are resized to 224$\times$224. For training of frame-based networks (the appearance stream), we fine-tune VGG net [95] using Caffe [48]. For each 6-frame chunk of video, we use one image frame for the appearance stream. We encode two sets of 6 optical flow fields (one stack each for x- and y-direction) as pixel trajectories for the motion stream. While training flow networks, we change the

conv_1 filter to a $1 \times 2K$ kernel, which only performs convolution in time. We project the fc7 features of multi-stream networks using a fully connected layer to a 200 dimensional vector. This 200 dimensional vector is given to two LSTM networks (one forward and one backward in time) with 60 hidden units each. Finally, a softmax classifier is trained on each LSTM's hidden layer, and softmax predictions of both LSTM networks are averaged to get the action scores for each class. While training LSTMs for detection, we use the entire video sequence, so this also includes the background class. We use the same architecture for both the datasets. Since the four networks that make up our multi-stream network cannot all fit in GPU (Tesla K40) memory at once, we train each network independently. To train the LSTM networks, we use the implementation provided in [19].

Since mean average precision (mAP) is the standard measure used to evaluate action detection in past work, we need to produce a ranked list of action clips, along with a start frame, an end frame, and a score associated with each clip. Mid-point hit criterion is used to evaluate detection as done in [87]. This means that the mid-point of the detected interval should lie within the ground-truth interval in the test video. If a second detection fires within the same ground-truth interval, that second detection is considered a false positive. To obtain segments for each action class, we start with an initial threshold. We apply this threshold to the output score (average of the two LSTM softmax outputs) that was assigned to each 6-frame chunk of video by our MSB-RNN network. We group the above-threshold chunks into connected components, each of which represents one detection, which we refer to as a clip (defined by its start and end time). This initial threshold will give us some number of detections. If the number of detections is less than m for a class, we lower the threshold until we get m unique clips. To get the next set of clips, we lower the threshold until we get $2m$ unique clips. If a new action clip intersects with any clip in the previous set, we discard the new clip. We keep on doubling the size of the next set until we obtain 2500 unique clips. In our experiments, m was set to 5. Each clip consists of some number of

consecutive 6-frame chunks of video, each of which is assigned an output score (average of the two LSTM softmax outputs) by our MSB-RNN system. We assign a score to each clip by max-pooling the output scores of all of the chunks in the clip. Since the validation set in the MPII Cooking 2 Dataset does not contain every class in the validation set, we adopt this method because it enables us to obtain a ranked list of detections without requiring us to select detection thresholds for each action class. We use the same process on the Shopping Dataset. Since labels are available at a per-frame level, we replicate the labels for a chunk 6 times, to get per-frame labels.

5.4.3 Experiments

In Table 5.1 we show that our MSB-RNN obtains an mAP of 41.2% on the MPII Cooking 2 Dataset, outperforming the previous state-of-the-art's mAP of 34.5%. Note that the 34.5% reported in [87] is a very strong baseline. Dense trajectories are still known to give state-of-the-art performance in fine-grained action recognition and detection, and [87] uses a combination of dense trajectories along with the additional hand-centric color-SIFT and Hand-Trajectories features. Our implementation of the two-stream network [94] (just the two full-frame streams, without our person-centric streams, and without the LSTMs) yields an mAP of 30.18% on this dataset, which is only slightly better than the performance of using improved dense trajectories alone.

Pixel Trajectories In Table 5.3, we compare the effectiveness of variations of the cropped-frame (person-centric) appearance stream ("Frame") and the motion stream (person-centric) using pixel trajectories ("Trajectories"). We evaluate mAP for two versions of each network: when the stream is followed by a unidirectional LSTM layer, and when the LSTM is omitted and replaced by a softmax layer. Using pixel trajectories instead of stacked optical flow improves performance both with and without LSTM, and on both the MPII Cooking

Method	mAP
Hand-cSIFT [87]	10.5%
Hand-trajectories [87]	21.3%
Hand-cSIFT+Hand-trajectories [87]	26.0%
Dense Trajectories [87, 104]	29.5%
Two-Stream Network [94]	30.18%
DT+Hand-trajectories+cSIFT [87]	34.5%
MSB-RNN	41.2%

Table 5.1: Comparison of performance of our MSB-RNN system with previous action detection methods on the MPII Cooking 2 dataset. Mean Average Precision (mAP) is reported.

2 (MPII 2) and Shopping (Shop) datasets, making pixel trajectories a clear winner over stacked optical flow for this task. For all three types of streams on both datasets, the LSTM layer produces a large improvement.

Action	Two Stream	MSN	LSTM $\leftarrow$	LSTM $\rightarrow$	BLSTM
Reach To Shelf	75.95%	80.8%	84.39%	84.86%	89.74%
Retract From Shelf	74.53%	77.71%	81.45%	84.61%	90.47%
Hand In Shelf	52.48%	54.13%	59.69%	68.73%	65.56%
Inspect Product	67.56%	75.68%	79.29%	78.6%	82.7%
Inspect Shelf	55.52%	57.09%	70.57%	70.31%	73.09%
Mean	65.21%	69.08%	75.08%	77.42%	80.31%

Table 5.2: Results for each action class with different network configurations on the Shopping Dataset.

Multi-Stream Network

In table 5.4 we compare the performance of our Multi-Stream network (using both full-frame and person-centric bounding boxes) with that of a two-stream network (full-frame only). Including a person-centric reference frame yields improved performance on both datasets. We report results for indivisual actions for the Shopping dataset in Table 5.2.

LSTM

Tables 5.4 and 5.2 also compare the perfomance of our Multi-Stream network followed by a forward uni-directional LSTM, a backward uni-directional LSTM, and the bi-directional LSTM (which is our complete MSB-RNN system). Each of the uni-directional

Method	MPI	MPI_{lstm}	Shop	Shop_{lstm}
Stacked OF	21.31%	27.36%	55.29%	71.70%
Trajectories	22.35%	29.51%	57.88%	73.06%
Frame	24.72%	28.77%	40.02%	63.26%

Table 5.3: Evaluating individual components of our MSB-RNN system. Mean average Precision (mAP) is reported. For both datasets, MPII Cooking 2 and Shopping dataset, pixel trajectories outperform stacked flow (both with and without a subsequent LSTM layer). For all three stream types and both datasets, incorporating the LSTM layer greatly improves performance.

Method	MPII 2	Shop
Two-Stream [94]	30.18%	65.29%
Multi-Stream	33.38%	69.27%
Multi-Stream LSTM $\rightarrow$	38.03%	77.24%
Multi-Stream LSTM $\leftarrow$	37.43%	75.08%
MSB-RNN	41.22%	80.31%

Table 5.4: Performance comparison of multi-stream vs. two-stream network. Performance when Multi-stream network is followed by each uni-directional LSTM or by their bi-directional combination (MSB-RNN). mAP is reported.

LSTMs provides a significant boost, and including bi-directional LSTMs (MSB-RNN) yields a substantial improvement, because it provides more temporal context than a uni-directional LSTM. The results in these tables and Table 5.3 clearly show that the LSTM layer is the most important factor contributing to our system's improved performance over previous methods.

These observations led us to explore in more detail why using an LSTM layer improves performance by such a large margin. We conducted two experiments to analyze the contributions of LSTM layer to our system.

How much memory does an LSTM have?

In the first experiment, we keep the model fixed and analyze how long does an LSTM remember. For this experiment, we clear the memory of the LSTM network at different time steps using a continuation sequence indicator. A continuation sequence indicator is 0 at the beginning of a sequence and 1 otherwise. Thus, we can set every kth indicator to 0

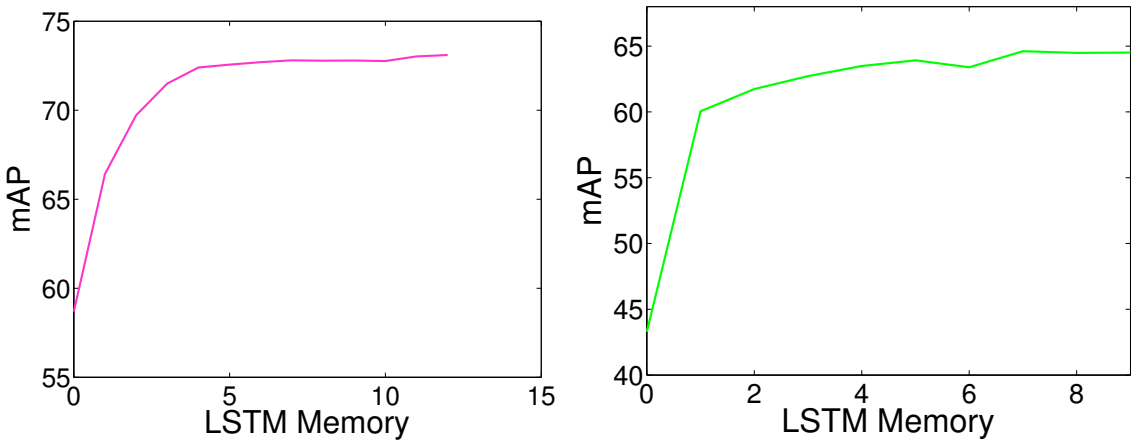

Figure 5.7: Average Precision (AP) for frame and trajectory network with restricted memory is plotted at inference for the Shopping Dataset. We observe that the LSTM network can remember as long as previous 10 sequences for making a prediction. The left plot corresponds to the flow network and the right one to the frame network

for clearing the memory, if we are interested in an LSTM which remembers history from the past k elements in the sequence. However, this would abruptly reduce the memory of the element at the beginning of the sequence to zero. To avoid this problem, we generate k different continuation indicator sequences, shifted by one element, if we want to limit the memory of the LSTM network to k time steps. Thus, when a prediction is made for an element, we would chose the output from the sequence whose continuation indicator was set to 0, k time steps before. In Fig. 5.7 we plot the mAP when LSTM is used on top of frame or flow features on the Shopping Dataset. We observe that performance increases as we reduce the artificially imposed memory limit for LSTM. It is quite encouraging that the LSTM network can remember as far as 10 elements in a sequence. Thus, a bi-directional LSTM would look at 20 elements in a sequence while making a prediction for an element in the sequence. In the context of a video, where each chunk going into an LSTM comprises 6 frames of a video (sampled at 15 frames per second), this sequence length would correspond to 8 seconds. Thus, the bi-directional LSTM improves action detection performance by a large margin, by incorporating information from about 8 seconds of temporal

context. Many actions last less than 8 seconds, and actions that last longer than that are likely to have a recurring pattern that can be captured in 8 seconds.

Learning Transitions between actions

The first experiment (above) demonstrated that an LSTM can remember long-term temporal information. In the second experiment, we explore whether the LSTM can also learn information in the transitions between different actions in a video sequence. Recent works train an LSTM network on trimmed video sequences [19, 71]. Thus, they would not learn long-term context that extends beyond the start or end of an action. Therefore, we conducted our second experiment, in which the continuation indicators are set to 0 (while training only) whenever an action starts or ends. This simulates training on trimmed video sequences, instead of a continuous video sequence that includes many actions. We observe that training on trimmed clips drops the performance from 77.24% to 75.51% on the Shopping Dataset and from 38.03% to 36.22% on the MPII Cooking 2 dataset (for single direction LSTM). This confirms our hypothesis that training networks on long video sequences is beneficial as compared to training on temporally clipped videos of individual actions.

5.5 Conclusion

In this chapter, we showed that using a multi-stream network which augments features inside a bounding-box surrounding the actor is useful in fine-grained detection. When pixel trajectories are reliable, they give better results compared to stacked-optical flow as they have correspondence. To capture long term temporal dynamics within and between actions, a bi-directional LSTM was found to be very effective. We also provided an analysis of how long an LSTM network can remember information in the action detection scenario. Finally, our results represent a significant step forward in accuracy on a difficult publicly available

dataset (MPII Cooking 2) as well as on a new dataset that we are releasing.

Chapter 6: Conclusion and Future Directions

This thesis highlighted important bottlenecks in convolutional neural network based visual recognition algorithms and proposed effective solutions to alleviate them. This was done by effective use of image pyramids, reformulating the object detection problem and the use of alternate architectures to model long range spatial and temporal dependencies.

Going forward, I believe more fundamental and radically different approaches are needed to obtain a quantum leap in performance. This could be achieved by rethinking the optimization of deep neural networks, as back-propagation often leads us to a local minima. As of today, pre-training is essential to obtain good performance on visual recognition tasks, and training on less data leads us to a local minima. My belief is that a deep representation is important but the amount of data required to learn that deep representation need not be in the order of millions of images. The power of deep learning lies in its cascade and if we can efficiently search through the weight space through alternate optimization techniques, it could transform the shape of artificial intelligence in the coming decade.

Bibliography

[1] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. Measuring the objectness of image windows. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2189–2202, 2012.

[2] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 33(5):898–916, 2011.

[3] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt. Sequential deep learning for human action recognition. In *Human Behavior Understanding*, pages 29–39. Springer, 2011.

[4] 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.

[5] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S. Davis. Soft-nms – improving object detection with one line of code. *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[6] 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.

[7] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[8] 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.

[9] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018.

[10] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks. In *Advances in Neural Information Processing Systems*, pages 4470–4478, 2017.

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

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

[13] Ross Cutler and Larry S Davis. Robust real-time periodic motion detection, analysis, and applications. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):781–796, 2000.

[14] 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.

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

[16] Jia Deng, Nan Ding, Yangqing Jia, Andrea Frome, Kevin Murphy, Samy Bengio, Yuan Li, Hartmut Neven, and Hartwig Adam. Large-scale object classification using label relation graphs. In *European Conference on Computer Vision*, pages 48–64. Springer, 2014.

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[18] Ali Diba, Vivek Sharma, Ali Pazandeh, Hamed Pirsiavash, and Luc Van Gool. Weakly supervised cascaded convolutional networks. *arXiv preprint arXiv:1611.08258*, 2016.

[19] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *arXiv preprint arXiv:1411.4389*, 2014.

[20] Yong Du, Wei Wang, and Liang Wang. Hierarchical recurrent neural network for skeleton based action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1110–1118, 2015.

[21] 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.

[22] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Detect to track and track to detect. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3038–3046, 2017.

[23] 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.

[24] Spyros Gidaris and Nikos Komodakis. Object detection via a multi-region and semantic segmentation-aware cnn model. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[25] 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.

[26] Ross Girshick. Fast r-cnn. *ICCV*, 2015.

[27] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[28] Georgia Gkioxari and Jitendra Malik. Finding action tubes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 759–768, 2015.

[29] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *ICML*, 2013.

[30] Alan Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE, 2013.

[31] Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. Bidirectional lstm networks for improved phoneme classification and recognition. In *Artificial Neural Networks: Formal Models and Their Applications–ICANN 2005*, pages 799–804. Springer, 2005.

[32] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *ICCV*, 2017.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European Conference on Computer Vision*, pages 346–361. Springer, 2014.

[34] 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.

[35] 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.

[36] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

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

[38] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[39] Judy Hoffman, Sergio Guadarrama, Eric S Tzeng, Ronghang Hu, Jeff Donahue, Ross Girshick, Trevor Darrell, and Kate Saenko. Lsda: Large scale detection through adaptation. In *Advances in Neural Information Processing Systems*, pages 3536–3544, 2014.

[40] Judy Hoffman, Deepak Pathak, Eric Tzeng, Jonathan Long, Sergio Guadarrama, Trevor Darrell, and Kate Saenko. Large scale visual recognition through adaptation using joint representation and multiple instance learning. *The Journal of Machine Learning Research*, 17(1):4954–4984, 2016.

[41] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

[42] Peiyun Hu and Deva Ramanan. Finding tiny faces. *CVPR*, 2016.

[43] Yuxiao Hu, Liangliang Cao, Fengjun Lv, Shuicheng Yan, Yihong Gong, and Thomas S Huang. Action detection in complex scenes with spatial and temporal ambiguities. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 128–135. IEEE, 2009.

[44] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. *CVPR*, 2016.

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

[46] Abhishek Jain, Arpan Gupta, Mikel Rodriguez, and Larry S Davis. Representing videos using mid-level discriminative patches. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2571–2578. IEEE, 2013.

[47] Manan Jain, Jan Van Gemert, Hervé Jégou, Patrick Bouthemy, and Cees GM Snoek. Action localization with tubelets from motion. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 740–747. IEEE, 2014.

[48] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[49] 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.

[50] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[51] 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.

[52] Hilde Kuehne, Ali Arslan, and Thomas Serre. The language of actions: Recovering the syntax and semantics of goal-directed human activities. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 780–787. IEEE, 2014.

[53] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pages 2169–2178. IEEE, 2006.

[54] 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.

[55] Jianan Li, Xiaodan Liang, ShengMei Shen, Tingfa Xu, Jiashi Feng, and Shuicheng Yan. Scale-aware fast r-cnn for pedestrian detection. *arXiv preprint arXiv:1510.08160*, 2015.

[56] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. Fully convolutional instance-aware semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[57] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. *CVPR*, 2016.

[58] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, volume 1, page 4, 2017.

[59] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017.

[60] 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.

[61] Tony Lindeberg. Scale-space theory in computer vision, 1993.

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

[63] 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.

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

[65] 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.

[66] Bill MacCartney and Christopher D Manning. An extended model of natural logic. In *Proceedings of the eighth international conference on computational semantics*, pages 140–156. Association for Computational Linguistics, 2009.

[67] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*, pages 2204–2212, 2014.

[68] 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.

[69] Mahyar Najibi, Pouya Samangouei, Rama Chellappa, and Larry Davis. SSH: Single stage headless face detector. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2017.

[70] 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.

[71] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. *arXiv preprint arXiv:1503.08909*, 2015.

[72] Bingbing Ni, Vignesh R Paramathayalan, and Philippe Moulin. Multiple granularity analysis for fine-grained action detection. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 756–763. IEEE, 2014.

[73] David Novotny, Diane Larlus, and Andrea Vedaldi. I have seen enough: Transferring parts across categories. 2016.

[74] Patrick Ott and Mark Everingham. Shared parts for deformable part-based models. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1513–1520. IEEE, 2011.

[75] 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.

[76] Xiaojiang Peng, Limin Wang, Xingxing Wang, and Yu Qiao. Bag of visual words and fusion methods for action recognition: Comprehensive study and good practice. *arXiv preprint arXiv:1405.4506*, 2014.

[77] Pietro Perona and Jitendra Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on pattern analysis and machine intelligence*, 12(7):629–639, 1990.

[78] Pedro O Pinheiro, Tsung-Yi Lin, Ronan Collobert, and Piotr Dollár. Learning to refine object segments. In *European Conference on Computer Vision*, pages 75–91. Springer, 2016.

[79] Hamed Pirsiavash and Deva Ramanan. Parsing videos of actions with segmental grammars. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 612–619. IEEE, 2014.

[80] 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.

[81] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[82] 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.

[83] Shaoqing Ren, Kaiming He, Ross Girshick, Xiangyu Zhang, and Jian Sun. Object detection networks on convolutional feature maps. *IEEE transactions on pattern analysis and machine intelligence*, 39(7):1476–1481, 2017.

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

[85] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. *arXiv preprint arXiv:1501.02565*, 2015.

[86] Marcus Rohrbach, Sikandar Amin, Mykhaylo Andriluka, and Bernt Schiele. A database for fine grained activity detection of cooking activities. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1194–1201. IEEE, 2012.

[87] Marcus Rohrbach, Anna Rohrbach, Michaela Regneri, Sikandar Amin, Mykhaylo Andriluka, Manfred Pinkal, and Bernt Schiele. Recognizing fine-grained and composite activities using hand-centric features and script data. *arXiv preprint arXiv:1502.06648*, 2015.

[88] Ruslan Salakhutdinov, Antonio Torralba, and Josh Tenenbaum. Learning to share visual appearance for multiclass object detection. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1481–1488. IEEE, 2011.

[89] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *arXiv preprint arXiv:1801.04381*, 2018.

[90] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45(11):2673–2681, 1997.

[91] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[92] 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.

[93] Abhinav Shrivastava, Rahul Sukthankar, Jitendra Malik, and Abhinav Gupta. Beyond skip connections: Top-down modulation for object detection. *arXiv preprint arXiv:1612.06851*, 2016.

[94] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, pages 568–576, 2014.

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

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

[97] Yuxing Tang, Josiah Wang, Boyang Gao, Emmanuel Dellandréa, Robert Gaizauskas, and Liming Chen. Large scale semi-supervised object detection using visual and semantic knowledge transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2119–2128, 2016.

[98] Yicong Tian, Rahul Sukthankar, and Mubarak Shah. Spatiotemporal deformable part models for action detection. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2642–2649. IEEE, 2013.

[99] Antonio Torralba, Kevin P Murphy, and William T Freeman. Sharing features: efficient boosting procedures for multiclass object detection. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–II. IEEE, 2004.

[100] Jasper Uijlings, Stefan Popov, and Vittorio Ferrari. Revisiting knowledge transfer for training object class detectors. *arXiv preprint arXiv:1708.06128*, 2017.

[101] 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.

[102] Subhashini Venugopalan, Huijuan Xu, Jeff Donahue, Marcus Rohrbach, Raymond Mooney, and Kate Saenko. Translating videos to natural language using deep recurrent neural networks. *arXiv preprint arXiv:1412.4729*, 2014.

[103] Paul Viola, Michael J Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 734–741. IEEE, 2003.

[104] Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Dense trajectories and motion boundary descriptors for action recognition. *International journal of computer vision*, 103(1):60–79, 2013.

[105] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 3551–3558. IEEE, 2013.

[106] Limin Wang, Yu Qiao, and Xiaoou Tang. Action recognition with trajectory-pooled deep-convolutional descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4305–4314, 2015.

[107] Limin Wang, Yuanjun Xiong, Zhe Wang, and Yu Qiao. Towards good practices for very deep two-stream convnets. *arXiv preprint arXiv:1507.02159*, 2015.

[108] Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Learning to track for spatio-temporal action localization. *arXiv preprint arXiv:1506.01929*, 2015.

[109] 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.

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

[111] 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.

[112] Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.

[113] 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.

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

[115] Gang Yu and Junsong Yuan. Fast action proposals for human action detection and search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1302–1311, 2015.

[116] Junsong Yuan, Zicheng Liu, and Ying Wu. Discriminative subvolume search for efficient action detection. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2442–2449. IEEE, 2009.

[117] 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.

[118] 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.

[119] Hanwang Zhang, Zawlin Kyaw, Jinyang Yu, and Shih-Fu Chang. Ppr-fcn: Weakly supervised visual relation detection via parallel pairwise r-fcn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4233–4241, 2017.

[120] 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.

[121] C Lawrence Zitnick and Piotr Dollár. Edge boxes: Locating object proposals from edges. In *European Conference on Computer Vision*, pages 391–405. Springer, 2014.